Implementing SOA in an ESB Framework

Miikka Lötjönen

***Abstract*—this study is an introduction to Service Oriented Architecture (SOA) and Enterprise Service Bus (ESB). It starts from business needs; why are SOA and ESB needed in the first place. Main concepts, related technologies and benefits of SOA will be introduced. Two typical software engineering problems are taken as practical examples that could be solved with SOA – in this case ESB-driven Web Services. For selecting the ESB framework, an evaluation of the ESB products on the market is conducted. The approach is practical: two implementations will be done with the selected frameworks.**

***Index Terms*— Service Oriented Architecture (SOA), Web Services (WS), Enterprise Service Bus (ESB), Simple Object Access Protocol (SOAP)**

GLOSSARY

EJB	Enterprise Java Bean
ESB	Enterprise Service Bus
EAI	Application Integration
J2EE	Java 2 Enterprise Edition
JBI	Java Business Integration
JCA	J2EE Connector Architecture (also J2CA)
JMS	Java Message Service
JNDI	Java Naming and Directory Interface
MOM	Message Oriented Middleware
PTP	Point to Point (also P2P)
QoS	Quality of Service
RPC	Remote Procedure Call
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
UDDI	Universal Description, Discovery and Integration
W3C	World Wide Web Consortium
WS	Web Service
WSDL	Web Service Description Language
XML	eXtensible Markup Language
XSLT	eXtensible Stylesheet Language Transformation

Concepts to be explained:

Adaptor
Asynchronous
Auditing
Bean (EJB, XML)
Bridge
Broker
Channel
Cluster
Container (EJB, ESB)
Endpoint
Legacy Application
Listener
Loose Coupling
Message
Orchestration
Persistence
Point-to-Point
Publish-and-Subscribe
Routing
Scalability
Schema
Service
Store-and-Forward
Synchronous
Transformation
XPath

1. INTRODUCTION

1.1 Description of the Environment

As a motivation to this study, we begin by introducing the company and the various challenges it is facing. For security reasons, the company and its projects that are discussed are not mentioned by name. The company is simply referred to as “the company” and projects are named “Project D” and “Project M”. Everything is based on reality, but some parts of the environment and the projects are deliberately simplified and generalized, to keep the focus in the essential things.

Although the aim is to solve the problems of this company, the discussion is kept in a generic enough level to be applied to other environments.

The company in its current form is the result of a recent merger that united many smaller companies working in the same business area. This has created a highly heterogenic environment; the company is distributed geographically and organizationally. Although the company is working with one name, it is divided into 21 sites in 8 countries, all of which have their own projects, processes, technologies, tools and methods of working. Also the projects inside one site are somewhat independent of each other, but sometimes would benefit from reuse between them.

From the technological point of view, the types of variation between the sites are numerous. Within the company, various types of architectures, platforms, frameworks, applications, tools, and programming languages are used. In most cases, there are no common interfaces to help inter-operation between the systems. The situation is acknowledged and the direction is towards integration, but the change is slow. Concrete actions are needed to help this process and leverage the existing solutions. Integration is the actual issue that this

study addresses; two example scenarios are taken into closer inspection.

1.2 Project D Overview and Problems

The company is in service business; both of the discussed projects will result in a service for end users. The end product of project D is a java-based application, running on an application server. The importance of the service that this application is providing is growing in various other projects of the company, and it is necessary to make it more accessible from various different systems. The goal is also to make servers easily replicable, add a possibility for clients to use multiple servers, and extend to support command line interface. The main problem is therefore accessibility.

The problem is that if this kind of extensions to support new ways to access the system is implemented in the traditional methods, each additional access method will produce extra cost, extra work and extra maintenance. In the worst case, the new applications must be implemented as separate applications. New interfaces and data exchange methods have to be introduced. Maintenance costs will grow as new components with dependencies to the old ones are introduced.

All of the things mentioned above add to the complexity of the product. A good solution would be if it was possible to use the same application for everything, and interact with many different clients through the same interfaces. Even a better solution would be if these interfaces were standard-compliant. Much of the effort could be saved through using standard off-the-shelf tools to handle the interaction between the server and clients.

1.3 Project M Overview and Problems

The corporation has an online portal with all the applications bundled into the same platform. Now a new project created in a subsidiary of the corporation needs to connect to this system, to be able to access customer information etc.

The portal is running on a commercial J2EE application server that the Project M will not be using, and there are no ways to access the databases from the outside. To add a bit more challenge, the operation platform for the new project is yet to be defined. An inter-organizational, platform independent, and very scalable A2A solution is needed. The main problem in Project M is therefore integration.

1.4 Field of study

For both of these tasks, Service Oriented Architecture (SOA) seems to offer a good starting point, and it is actually the basis of the corporate-wide reference architecture.

SOA is and architectural style whose goal is to achieve loose coupling among interacting software clients. (viite jostain ns. Arvostetusta julkasusta) The main business drivers of SOA are flexibility and efficiency. Flexibility comes mostly from the abstraction that an SOA offers. Efficiency is the result of using standards-based approach and reuse. SOA and the related concepts and technologies are introduced in chapter 4.

A Web Service is of the possibilities to implement SOA. Enterprise Service Bus (ESB) is a Service Oriented Architecture that can be implemented with an ESB framework.

In both projects it was decided that a Web Service would be implemented, but the developers weren't yet assigned for this. After some discussions it became my responsibility to learn about the subject, make a decision about the framework to be used, and then develop the services.

2. Objectives and Scope of the Study

2.1 Objectives

The primary objective is to investigate and report the possibilities of Web services, SOA and ESB. The secondary objective is to present practical solutions to practical problems: 1. ESB framework selection, and 2. implementations of two example scenarios.

Besides this paper to be published, the study process will also produce source code architectural descriptions as deliverables to the company.

2.2 Research questions

1. What are the needs and problems of the current environment (i.e. architectures, tools and methods currently used in the projects)?
2. How can the current environment be improved through SOA and ESB?
3. What are the requirements for an ESB framework?
4. Which ESB framework would be the most suitable?

2.3 Scope

The research paper started with a short introduction to the projects and their needs, followed by some discussion about the reasons that drive this change towards SOA. The reasons will be further discussed and later on used as bases when defining criteria for the evaluation.

Due to the broad range of possible themes, neither SOA nor Web Services will be covered extensively. There will be an introduction to both themes that will serve as the motivation for doing things "the SOA way" in these two example projects. Enabling technologies such as XML and SOAP will be discussed in the detail that is necessary to describe the overall functionality of Web Services.

ESB will be introduced through its relation to SOA. ESB frameworks will be introduced from two viewpoints; first describing them in generic terms, and then finding out how they fulfil the company-specific and project-specific needs.

The details of the ESB frameworks are left outside the scope. Although the practical part includes digging pretty deep into the products, the findings are covered only in the extent that is needed to make the conclusions about which software package to use. The documentation and examples that are available online, will be referred to, but not described as a whole.

Although the research will result in source code and architectural descriptions for both example projects, these will not be provided except for generic principles behind them.

3. Research Methods and Structure of the Report

The structure of the paper is briefly described, linking it to the objectives of the study and research methods.

The paper will be a constructive study that addresses both theoretical and practical issues. It is roughly divided into literature study and empirical work, but due to the practical nature of the subject, the themes will undoubtedly overlap.

The approach to the theme is practical. During the study process I came to realize that there is a great deal of general information about SOA and software architectures. Also there are plenty of documents in practical level, such as tutorials on how to implement web services.

If you are new to the field, many times the generic descriptions are too general to serve as understandable introductions, and the technical details are too narrow and detailed to provide the overall view. I figured that there is a gap to fill in. I wanted to present the big picture, starting from ideological principles, presenting the essential things that are needed to know and wrapping the presentation up by implementing useful things following these principles.

3.1 Literature Study

The literature study is an overview of SOA, ESB and related technologies. Chapter 4 serves as introductory material to the problem domain, introduces the challenges in traditional methods and explains the benefits of SOA. The following chapter 5 is a short market overview of SOA product support.

Now that some background information has been provided, chapter 6 gives an overview of Enterprise Service Bus and the implementing products. The relation to SOA is explained, together with illustrations and examples. The chapter prepares for the practical part of the study, by giving criteria and reasoning for ESB product selection. Two candidates will be selected for further inspection.

Addressed research questions:

- Question 2 in chapter 4.2
- Question 3 in chapter 7.1

3.2 Empirical Work

The case company, projects and problems have already been analyzed in chapter 1. Most of the actual empirical material is presented in chapter 7 in the form of evaluation between two ESB framework candidates. Most of the discussion is based on the example scenarios that are implemented.

Addressed research questions:

- Question 1 in chapter 1
- Question 4 in chapter 7

4. Service Oriented Architecture Overview

4.1 Introduction

Service oriented architecture is actually nothing new. It is more of a collection of principles and a way of thinking than a real technology. (He 2003) explains SOA quite well, with examples.

List basic concepts, benefits and drawbacks.

(Doernhoefer 2005) is a very high-level description about SOA and online resources concerning SOA. Another good source of basic information is IBM's (Gottschalk et al. 2002).

4.2 Why SOA?

- Company-wide benefits
- Project D benefits
- Project M benefits
- (Moad 2005)describes the benefits of SOA.
- (Linthicum 2003) describes application integration through Web Services
- (Leymann, Roller & Schmidt 2002) introduces SOA from the business perspective

4.3 SOA and Patterns

Many features of a SOA can be described by using design patterns. There are many types of patterns, for different uses. (Hohpe, Woolf 2005) introduces an extensive set of Enterprise Integration Patterns that are the basis for many SOA/ESB concepts, for example messaging patterns.

4.4 SOA and Java Enterprise Edition

SOA is often mentioned in java context. This is only because of Java's popularity as a programming language. SOA itself is an architectural style, and is not bound to any programming language or protocol. As mentioned before, SOA doesn't even have to be implemented with an object oriented language; internal details of the services are ignored. Focus is on component level, interfaces, architectures and integration approach.

4.5 SOA and Web Services

- A short introduction to Web Services, referencing to (Alesso, Smith 2005).
- A bit more technical point of view in (Farrell, Kreger 2002).
- Mentioning Semantic Web as described in (Daconta, Orbst & Smith 2003) and (Korotkiy, Top 2006).

4.6 SOA and Supporting Standards

Extensible Markup Language (XML) has become very popular as a data exchange language, and it is not a big surprise that it is also an essential part of many SOA implementations. XML is a simple, text-based, generic use markup language that is used to describe data through user defined tags. XML is a cross-platform, software and hardware independent tool for transmitting information. The universal nature of XML is also the biggest strength of the language; it can essentially describe anything you want.

The grammar that defines the allowed tags and their

relations is described in separate documents, Either Data Type Definitions (DTD) or XML Schema Definitions (XSD) that are defined in (W3 Consortium 2006). XML The focus is on Schema because it is more expressive and the current W3C recommendation. Schemas can be used as sort of agreements between remote applications, and also the validity of the messages can be verified by checking the received XML message against its Schema.

XML helps in transfer of arbitrary data, but still it is just a data storage format. A communications protocol is needed to provide the actual transfer services. Also this can be implemented in XML: Simple Object Access Protocol (SOAP) is a commonly used communications protocol that delivers XML-encoded SOAP messages on top of HTTP protocol. It is a platform and language independent way to communicate between applications and send messages.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-
envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soa
p-encoding">
<soap:Header>
  ...
  ...
</soap:Header>
<soap:Body>
  ...
  ...
  <soap:Fault>
    ...
    ...
  </soap:Fault>
</soap:Body>
</soap:Envelope>
```

For describing web services, there is a separate language, Web Services Description Language (WSDL), which is also XML-based. Due to limited space, I will not introduce more code examples. An example of a WSDL and related SOAP envelopes can be found from (Prud'hommeaux 2001).

For locating the services, there is a protocol called UDDI (Universal Description, Discovery and Integration).

4.7 SOA Criticism

SOA is a nowadays a fairly well-known way of thinking, but it does not solve all the problems of developing complex systems.

The first obstacle is that it takes time to learn it. While learning, the solutions may be far from perfect; it is easy to misunderstand or misuse the tools. When starting from an object-oriented viewpoint, designing a SOA requires a slight change of mindset, and the process can be time consuming.

It is not self-evident where to use SOA approach and where not. In some cases, such as some time-critical real-time systems, it may not work at all.

Reusable services sound very good in principle, but in practice, the maintenance may turn out to be more of a challenge than expected, especially if the services go through lots of changes.

5. SOA Product Support

It is possible to implement an SOA from scratch, but there are also standard based tools to help with the process.

5.1 J2EE Application Servers

General description. Motivation is that these will be used in the project M, and also that these products are common platforms for SOA development. Include references to all the company websites (currently not listed in the references).

Most of the major vendors of application servers and similar products have introduced or announced to introduce their ESB solution.

- BEA Weblogic
- IBM WebSphere (referencing (Cuomo 2005))
- Oracle Application Server
- Visual Studio .NET
- JBoss
- Caucho Resin
- Apache Tomcat

5.2 Middleware

- What is the purpose of middleware?
- Some architectural diagrams
- ESB frameworks, general description about what ESB is. Based on (Chappel 2004).
- Apache Axis, implementation of the Simple Object Access Protocol (SOAP).

5.3 Other tools

While the commercial development software packages normally come with a extensive set of tools, the situation with open source development is a bit different.

Usually things can be done with very basic tools but the workload can be dramatically decreased with proper tools. There is no point in writing hundreds of lines of XML with notepad, if there is a good XML editor available, or better yet, a tool that will write these XML files for you.

XML Spy is a great tool when writing XML code. It simplifies work and takes care of many routine tasks, such as creating schemas for XML-documents.

Eclipse is a popular, free and powerful integrated development environment (IDE) that has a wide and active community extending the functionality by developing Eclipse plug-ins. The number of plug-ins is impressive, with new ones being developed all the time. From the ESB point of view, Eclipse is a good platform also; many vendors have their future tools implemented as Eclipse plug-ins. For example Mule IDE, ServiceMix IDE, and even the upcoming BEA Weblogic Workshop 9.2 are Eclipse plug-ins.

6. ESB Frameworks

6.1 What is an ESB?

ESB is not an easy term to define. Just like with SOA, the definitions vary a lot from person to person, and from vendor

to vendor. (Silver 2004) discusses this confusion.

In most publications, ESB is seen as a lightweight alternative to monolithic and centralized Enterprise Application Integration (EAI) broker architecture. An ESB framework is an example of Message Oriented Middleware (MOM).

According to (Silver 2004), an ESB has these four essential capabilities:

1. Universal connectivity of services via XML messaging, interconnecting requesters and providers across diverse platforms and data models, providing a common backbone for requests, messages, and events.
2. Vendor-independent communications standards, such as SOAP and Java Messaging Service (JMS).
3. Quality of service features, including reliable delivery, transaction management, and scalable performance.
4. Service mediation features, providing loose coupling between requesters and providers.

A list of common characteristics (adapted from (Chappel 2004)) follows:

- Pervasiveness, serves for building integration solutions that can span through the whole organization
- Highly distributed, event driven SOA.
- Selective deployment of integration components. Adapters, distributed data transformation services, and content-based routing services can be selectively deployed when and where they are needed
- Security and reliability are basic functionalities of ESB messaging
- Orchestration and process flow, for managing both local and remote services.
- Autonomous yet federated managed environment, brought by loose coupling.
- Incremental adoption. An ESB can be used for small projects by separately adding them to the ESB.
- XML as a native data type
- Real-time insight. For example monitoring the data.

6.2 Why ESB

- General Description of ESB Pattern, using (Newcomer, Lomow 2004) chapter 4.
- Benefits

6.3 Available ESB Framework Products

- Commercial Products
  - BEA Aqualogic Service Bus
  - Oracle ESB
  - IBM WebSphere ESB (referencing (Cuomo 2005))
  - Fiorano ESB (50 000€ / processor)
  - Sonic ESB (10 000€ / processor)
  - Cape Clear ESB
- Open Source Products
  - SymphonySoft Mule
  - Apache ServiceMix (JBI)
  - Project Open ESB
  - Sun Java Enterprise Service Bus (JBI)

These are just examples of ESB offerings, and for sure not the only possibilities. In addition, for example Microsoft Visual Studio .NET and BizTalk can also be used in a fashion similar to ESB.

The power of commercial ESB products is that they can be integrated easily to the other products of the same vendor. One might imagine that this leads to faster learning due to the similarity of concepts, easier coordination of projects' components because of well-defined and well-known interfaces, and more automated and streamlined development because of well-thought integration between the tools.

The definite downside of the commercial products is that they are costly. When making decisions about integrating all the systems of the company into an ESB solution, investing tens of thousands of euros for an ESB framework may be justified, but for a proof-of-concept application or a single task, it does not seem feasible. This is why all commercial products are excluded from this evaluation.

6.4 Selection of Two Candidates for Evaluation

Based on previous discussion, two ESB frameworks are selected for further evaluation. Since the two example implementations are pilot projects by nature, the company does not want to invest in the ESB framework, especially since good open source candidates are available. In this case there are at least two good free options, SymphonySoft Mule and Apache ServiceMix.

Mule is based on Universal Messaging Objects (UMO), while ServiceMix is slightly more Java-oriented with its Java Business Integration (JBI) approach. Both frameworks are feature-rich, and include support for various types of messaging through generic endpoints. Both are java frameworks, but as ESB products they support applications written in any other language. Essentially, they are made for the same purpose, but solve the problems in slightly different ways. They can also be made to co-operate; Mule has a JBI binding that enables it to interact with ServiceMix JBI container, and also Mule code can be reused in ServiceMix just like any other JBI components.

In addition to these two open source products, a commercial option may be evaluated later on. The company is evaluating BEA Weblogic to be the future development platform, and considering this, it may make sense to invest in BEA Aqualogic Service Bus. Since Weblogic already supports many features that ESB frameworks offer, there is no hurry to do this either. Also open-source frameworks can be used with Weblogic. ESB products are mostly used with existing applications. This means that if the need arises, migration to

use the Aqualogic ESB should not be difficult. The benefits of an ESB increase when the framework is used throughout the company.

6.5 ESB Frameworks Feature Comparison

In terms of technology support, the two are almost identical. No critical differences can be found. Both either currently support or will support soon most of the major technologies.

Based on (ServiceMix.org 2006) some differences can be found, though. The biggest difference is in the approach: Mule existed well before ServiceMix, which started as a Java Business Integration (JBI) container and extended into a full ESB.

Mule started before JBI even existed, and a new API called UMO (Universal Messaging Objects) was introduced. Mule's architectural design is based on a services container and configuration of message endpoints.

Mule is endpoint centric, while ServiceMix also offers integration to Apache Geronimo Application server.

In practice, many differences will probably come up due to the different approach. But on paper, both products have a similar feature set.

7. EMPIRICAL EVALUATION OF SELECTED ESB FRAMEWORKS

7.1 Basis for Evaluation

Criteria	**Reason**
Compatibility with current technology ("legacy systems")	This is actually the reason why ESB is taken into use
Compatibility with the technology of future products	To not start using something that cannot be used with future products
Learning curve	Many people will need to learn to use the framework, and it should not cause much extra work. Quality of documentation is also a thing worth noting.
Extendibility and flexibility	The technologies of future projects are yet to be undefined, and the framework should stretch to fulfil the future needs
Robustness	The services that will be running on the ESB are critical for the success of the company. In many cases, 99,999% uptime is required. The ESB must be stable enough to meet these requirements.
Performance	Many of the systems that the company is developing are performance-critical, both in terms of throughput and response time. The ESB should not add too much extra overhead.

Performance and robustness testing would require so much effort that it can not be done in the timeframe of this paper. They are important things, but for now it is assumed that the serious candidates (that are in production use in many companies) will meet the requirements.

7.2 Experience from practice

As mentioned before, starting learning SOA and ESB without any training, just reading online materials is not necessarily easy. Both from ideological and practical perspectives, it is a bit of a challenge at first.

My first attempt designing a SOA ended up with nothing but a series of synchronous Web services that were point-to-point remote procedure calls (RPC) in nature, and actually the system didn't benefit at all from the ESB. It didn't perform content based routing, message brokering, or in fact any of the advanced features that make ESB worthwhile.

Also the first steps writing XML Schemas, WSDL files and ESB configuration files by hand can prove to be trickier than it sounds when started from zero. Once you start understanding how everything works, it becomes more routine and you learn things that save time.

7.3 Suitability for Project D

Due to schedule-related reasons, ESB could not be tested with project D.

7.4 Suitability for Project M

Some preliminary testing has been done with project M and it seems that both of the frameworks would be equally functional, with Mule maybe being a bit easier to implement.

7.5 Subjective Evaluation

ServiceMix with its JBI support is a natural step for companies that have their current products running with Java. Mule is configured by its own configuration files, making it a bit "less standard", but in the other hand, this approach makes the service orchestration almost invisible. No special annotations or any other ESB-specific things need to be programmed in java code. This is closer to the "ESB spirit" of favouring configuration over programming. The framework is the invisible glue that connects the endpoints together, but is separated from the implementation.

In the end, both candidates are viable solutions; they have been proven to do their work well in practice. In terms of development work, they are both in a mature state, and still evolving rapidly. Their support for different technologies is similar; both of them can be used with the existing software in the company. One might even say that it is a matter of taste which one to start using.

An important thing in this case is the learning curve. Neither of them is really difficult to learn, but it takes time, and many questions arise on the way. Mule has already been successfully used in one branch of the company, so if problems with Mule occur, there is "support personnel" that may be able to help in the same building.

- Ease of use, learning curve
- Intuitive interfaces

- Programming model

8. Results

Answers to research questions are still mostly open at this point of the study when not all the necessary practical parts have been done.

8.1 Selection of the Framework with Rationale

To sum up chapter 7, a final selection of a framework will be made.

Mule is selected as the framework for doing the new proof-of-concept implementations. This is partly because of its features, but mostly because there is more Mule-experience in the company. Since I started experimenting with Web services in Mule framework, I got familiarized with it first.

8.2 Two Example Implementations Using an ESB Framework

This chapter will introduce the implementations as UML diagrams and architectural drawings in the detail that is needed to give a practical view of the benefits and functionality of ESB frameworks. Some excerpts of source code will be generalized and explained.

9. Summary and Conclusions

Conclusion will gather the lessons learned during the study and sum up the results from Chapter 8.

References

Alesso, H.P. & Smith, C.F. 2005, *Developing Semantic Web Services*, 1st edition edn, A K Peters, Canada.

Chappel, D. 2004, *Enterprise Service Bus*, 1st edition edn, O'Reilly, United States of America.

Cuomo, G.(. 2005, "IBM SOA "on the edge"", *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*ACM Press, , pp. 840.

Daconta, M.C., Orbst, L.J. & Smith, K.T. 2003, *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management*, 1st edition edn, Wiley Publishing, United States of America.

Doernhoefer, M. 2005, "Surfing the net for software engineering notes", *SIGSOFT Softw.Eng.Notes*, vol. 30, no. 6, pp. 5-13.

Farrell, J.A. & Kreger, H. 2002, "Web services management approaches", *New Developments in Web Services and E-commerce*, vol. 41, no. 2, pp. 212-227.

Gottschalk, K., Graham, S., Kreger, H. & Snell, J. 2002, "Introduction to Web services architecture", *New Developments in Web Services and E-Commerce*, vol. 41, no. 2, pp. 170-178.

He, H. 2003, 2003-09-30-last update, *What Is Service-Oriented Architecture* [Homepage of www.xml.com], [Online]. Available: http://www.xml.com/pub/a/ws/2003/09/30/soa.html [2006, 2006-04-22 19:22] .

Hohpe, G. & Woolf, B. 2005, *Enterprise Integration Patterns*, 5th edition edn, Pearson Education, USA.

Korotkiy, M. & Top, J. 2006, "Onto-SOA: From Ontology-enabled SOA to Service-enabled Ontologies", *Telecommunications, 2005. AICT-ICIW '06. International Conference on Internet and Web Applications and Services/Advanced International Conference on*, vol. vol. 00, no. -, pp. 124-131.

Leymann, F., Roller, D. & Schmidt, M.T. 2002, "Web services and business process management", *New Developments in Web Services and E-commerce*, vol. 41, no. 2, pp. 198-211.

Linthicum, D.S. 2003, *Next Generation Application Integration - From Simple Information to Web Services*, 2nd edition edn, Addison-Wesley, United States of America.

Moad, J. 2005, "Software architecture", *Managing Automation*, vol. 20, no. 5, pp. 28-30.

Newcomer, E. & Lomow, G. 2004, *Understanding SOA with Web Services*, 1st edition edn, Addison-Wesley, United States of America.

Prud'hommeaux, E. 2001, 2001-03-26 11:12:20-last update, *Annotated WSDL Examples* [Homepage of W3 Consortium], [Online]. Available: http://www.w3.org/2001/03/14-annotated-WSDL-examples.html [2006, 2006-04-22 23:22] .

ServiceMix.org 2006, 2006-01-01 00:00:00-last update, *How does ServiceMix compare to Mule?* [Homepage of The Apache Software Foundation], [Online]. Available: http://servicemix.org/How+does+ServiceMix+compare+to+Mule [2006, 2006-04-21 13:12] .

Silver, B. 2004, *Enterprise Service Bus Technology for Real-World Solutions*, Bruce Silver Associates.

W3 Consortium 2006, 2006-02-05 16:44:46-last update, *Extensible Markup Language (XML)* [Homepage of W3C], [Online]. Available: http://www.w3.org/XML/ [2006, 2006-03-16 12:31] .

List of Figures