Evaluating .NET-Based Enterprise Service Bus Solutions

Clermond de Hullu

October, 2012

Department of Computer Science
Chair Software Engineering

Supervisors
Dr. Luís Ferreira Pires
Martijn van Tongeren

Academic Year
2011/2012
# Table of Contents

1 Introduction  
1.1 Motivation ........................................... 1  
1.2 Objective ............................................ 2  
1.3 Approach ............................................. 2  
1.4 Outline ............................................... 3  

2 Background  
2.1 Service-Oriented Architecture ...................... 4  
2.2 Enterprise Service Bus ............................... 5  
2.3 Implementation techniques .......................... 6  

3 ESB Capability Model  
3.1 Categories ............................................ 8  
3.2 Communication ....................................... 9  
3.3 Integration .......................................... 13  
3.4 Service Interaction ................................... 14  
3.5 Message Processing .................................. 18  
3.6 Business Processes .................................. 21  
3.7 Quality of Service ................................... 23  
3.8 Security .............................................. 26  
3.9 Management .......................................... 28  

4 Existing Solutions  
4.1 Microsoft BizTalk Server 2010 ...................... 29  
4.2 Neudesic Neuron ESB 2.6 ........................... 32  
4.3 Windows Azure AppFabric Service Bus ............ 33  
4.4 Custom ESB ......................................... 34  
4.5 Product Selection .................................... 34  

5 Capability Evaluation  
5.1 Evaluation Criteria ................................... 36
INTRODUCTION

This chapter presents the motivation, the objective, the approach and the structure of our research.

Section 1.1 discusses our motivation.
Section 1.2 presents the objective of our research.
Section 1.3 explains our approach by identifying the steps that were taken to achieve the objective.
Section 1.4 ends this chapter by elaborating on the structure of our research.

1.1 Motivation

Executives across all industries are demanding more value from their strategic business processes. An important aspect that helps improve the value of those business processes is the ability to flexibly and rapidly create and change those processes. To do this, it is important that different software applications that support these processes are able to communicate with each other. However, not all software applications are built with interoperability in mind. So, allowing all those different applications to interoperate can be both time-consuming and costly due to different transmission protocols and data exchange formats.

The last few years have seen some significant technology trends to solve this problem, such as Service-Oriented Architecture (SOA), Enterprise Application Integration (EAI), Business-to-Business (B2B), and Web services. These technologies have attempted to address the challenges of improving the results and increasing the value of integrated business processes [1].

SOA is the current state-of-the-art in IT application architecture that addresses the challenges of increasing the value of integrated business processes. Every major software vendor, including IBM, Microsoft, Oracle, and SAP has embraced SOA and is investing large amounts of money to redesign their products according to the SOA principles. SOA does not prescribe any specific technology for its implementation and can be implemented using Web services, an Enterprise Service Bus
(ESB), an application server platform, or other middleware. The ESB draws the best traits from these and other technology trends [1].

An ESB is an integration infrastructure to facilitate SOA and can be used to connect and coordinate the interaction of a significant number of diverse applications. However, there are a lot of different vendors and different ESB architectures, so choosing a suitable ESB for an integration task becomes a complex decision.

There are many comparisons between Java-based ESBs [2] [3], but at the time of writing we have found no comparisons between .NET-based ESBs. Our research provides a capability model which can be used to evaluate ESBs, and also gives an overview and evaluation of existing .NET-based solutions.

1.2 Objective

The main objective of our research is twofold:
(1) to provide an overview of the state-of-the-art commercially available .NET-based ESB solutions and
(2) to perform an evaluation to understand the benefits and drawbacks of these solutions.

1.3 Approach

The following steps were taken to achieve our objective.

1. **Literature study.**
   We perused existing literature to acquire general knowledge about SOAs and ESBs.

2. **Capability model.**
   We developed a capability model based on the knowledge obtained from the literature study.

3. **Study of existing solutions.**
   We researched existing solutions and provided an overview of them.
   
   (a) **Selection of solutions.**
   We selected two state-of-the-art commercially available ESBs for the evaluation, and in addition we also selected two existing techniques that can
be used as building blocks for the development of a custom ESB. We selected these techniques to stress the benefits of a commercially available solution over that of a custom ESB implementation.

4. Evaluation based on the capability model.
   We evaluated the solutions based on our capability model.

5. Case study.
   We carried out a case study to test, evaluate and demonstrate the solutions that we had chosen.

1.4 Outline

The outline of the remainder of our thesis is as follows:

Chapter 2 gives the general background information which is necessary to understand our research.

Chapter 3 defines an ESB capability model which gives an overview of some of the ESB’s capabilities, and can be used to compare different ESBs.

Chapter 4 gives an overview of the existing ESBs and the techniques which can be used as building blocks to implement a custom ESB.
Three solutions, based on this overview, have been selected to be evaluated in more detail.

Chapter 5 carries out the evaluation of the three selected solutions by analyzing them using the capabilities specified in our capability model.

Chapter 6 describes the case study which we have used to test, evaluate and demonstrate the selected products.

Chapter 7 summarises our conclusions.
BACKGROUND

This chapter contains the background information which we collected from the literature study and helps understand our research.
Section 2.1 provides basic information on SOAs.
Section 2.2 gives a definition for an ESB and supplies information about them.
Section 2.3 gives information on the technologies discussed in our research.

2.1 Service-Oriented Architecture

Service-oriented architecture is an approach which makes software resources in an enterprise available and discoverable as well-defined services. It applies concepts from areas like Object-Oriented development, Component-Based Design, and Enterprise Application Integration technology.

Services in an SOA have the following characteristics [4]:

• They are defined by explicit, implementation-independent interfaces.

• They are loosely bound and invoked through communication protocols that stress location transparency and interoperability.

• Finally they encapsulate reusable business functions.

Currently SOA services are mainly implemented as Web services using a standard description language (WSDL), a discovery service (UDDI) and a messaging protocol (SOAP). These technologies contributed to the maturity of the interoperability of Web services.

After a function has been encapsulated and defined as a service in an SOA, it can be used and reused by one or more systems that participate in the architecture.

The encapsulation of services by interfaces and their invocation through location-transparent, interoperable protocols are the basic means by which SOA enables flexibility and reusability.
2.2 Enterprise Service Bus

The term ESB was first used by Sonic in 2002 [5] to refer to their SonixXQ product, which was an XML-enabled Message-Oriented Middleware (MOM) that was later re-branded as Sonic ESB. A few months later, Gartner called ESB a strategic investment and then soon many systems from integration servers to messaging products were re-branded as ESBs.

In our research we see an ESB as an integration infrastructure which is used to facilitate SOA. An ESB combines service hosting, message transformation, protocol bridging and intelligent routing to connect and coordinate the interaction of a significant number of diverse applications across extended enterprises [1].

This is done by providing a set of rules and principles for integrating numerous applications together over a bus-like infrastructure (as shown in Figure 2.1). ESB products support developers to build SOAs, but vary strongly in their operations and capabilities.

Message-Oriented Middleware

At the core of the ESB communications layer is a MOM infrastructure that is capable of supporting a variety of industry-standard access mechanisms and protocols. MOM can be seen as a multiprotocol messaging bus that supports asynchronous delivery of messages with configurable QoS options [1].

ESB Service Container

The highly distributed nature of the integration capabilities of the ESB is largely due to the facilities of the ESB service container. A service container is a remote
process that can host software components and also implements the abstract endpoints through which services can be reached.

The ESB container provides a variety of facilities such as location, routing, service invocation and management, so a service does not have to be concerned with these issues. The container also provides facilities for lifecycle management such as start-up, shut-down, and resource clean-up. Connection management facilities provide the technical details of the binding to the underlying MOM, plus additional fault tolerance and retry logic. [1]

2.3 Implementation techniques

This sections discusses the implementation techniques mentioned in our research.

2.3.1 Microsoft Message Queuing

Microsoft Message Queuing (MSMQ) technology enables applications running at different times to communicate across heterogeneous networks and systems that may be temporarily offline. This technology allows applications to send messages to and read messages from queues.

Message Queuing provides guaranteed delivery, efficient routing, security and priority-based messaging. It can be used to implement solutions for both asynchronous and synchronous scenarios that require high performance [6].

2.3.2 Windows Communication Foundation

Windows Communication Foundation (WCF) is a framework for building service-oriented applications and it addresses a range of problems for communicating applications. The three most important aspects of WCF are:

1. Unification of the original .NET Framework communication technologies.

2. Interoperability with applications built on other technologies.

3. Explicit support for service-oriented development.

Because it unifies different approaches to communication, WCF implements Web services technologies defined by the WS-* specifications to allow more than just basic communication. Figure 2.2 shows how these specifications address several different areas, including basic messaging, security, reliability, transactions, and working with service metadata.
WCF unifies different approaches to communication and it can simplify the creation of distributed applications on Windows operating systems. In addition it provides broad interoperability with other platforms because it implements SOAP and the most important WS-* specifications, along with RESTful communication. It also gives developers an environment for developing and deploying service-oriented applications [7] because it offers explicit support for a service-oriented approach.

2.3.3 Windows Workflow Foundation

Writing scalable and understandable software can be a difficult thing to do because the approaches which help applications scale tend to break them apart, dividing their logic into separate chunks that can be hard to understand. Yet writing unified logic that lives in a single executable can make scaling difficult. The primary goal of Windows Workflow Foundation (WF) is to keep the application’s logic unified, making it more understandable, while still letting the application scale.

By supporting logic created using workflows, WF provides a foundation for creating unified and scalable applications. Along with this, WF can also simplify other development challenges, such as coordinating parallel work and tracking a program’s execution.

WF is a Microsoft technology that provides an API, an in-process workflow engine, and a rehostable designer to implement long-running processes as workflows within .NET applications [8]. The current version of WF was released as part of the .NET Framework version 4 and we will refer to it in our research as (WF4).
ESB Capability Model

In this chapter we define an ESB capability model which categorizes the ESB’s capabilities and which can also be used to compare different ESBs.

This chapter is structured as follows:
Section 3.1 gives an overview of our categorisation.
Section 3.2 deals with Communication.
Section 3.3 discusses Integration.
Section 3.4 addresses Service Interaction.
Section 3.5 deals with Message Processing.
Section 3.6 considers Business Processes.
Section 3.7 discusses Quality of Service.
Section 3.8 deals with Security.
Section 3.9 discusses the aspects involved in Management.

3.1 Categories

Figure 3.1 depicts the categories that we have derived from existing literature [9] [4] [1] [10] [11]. These categories were derived by analyzing the existing categorizations for suitability and by grouping their capabilities to obtain a categorization within them which contains as little overlap as possible.

Table 3.2 provides a short summary of the derived categories, some of which are further divided into subcategories. At the lowest level the category or subcategory corresponds to some concrete capability which is used as evaluation criteria in Chapter 5. We have selected, where feasible, the most popular and platform-independent capabilities, but in some cases the subcategories contain too many capabilities to mention all of them. In case a selection had to be made, we preferred capabilities that are most relevant in a .NET environment. For example, we selected the C# programming language in the Language Adapter subcategory instead of trying to list all possible programming languages.

The diagrams in this chapter indicate mandatory and optional components.
In our research, the capabilities part of the ESB definition mentioned in Section 2.2 (i.e., service hosting, message transformation, protocol bridging and intelligent routing) are marked mandatory, since these capabilities characterise ESBs.

![ESB capability categories](image)

**Figure 3.1: ESB capability categories**

<table>
<thead>
<tr>
<th>Capability Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication</td>
<td>Communication protocols, message exchange patterns and routing mechanisms that are available to connect service requesters and service providers to an ESB.</td>
</tr>
<tr>
<td>Integration</td>
<td>Mechanisms available to integrate existing techniques and applications with an ESB.</td>
</tr>
<tr>
<td>Service Interaction</td>
<td>Basic requirements necessary for interaction between service requesters and service providers once a network connection has been made.</td>
</tr>
<tr>
<td>Message Processing</td>
<td>Ways available for transforming and validating messages.</td>
</tr>
<tr>
<td>Quality of Service</td>
<td>Possible quality of service options.</td>
</tr>
<tr>
<td>Security</td>
<td>Possible security options.</td>
</tr>
<tr>
<td>Business Processes</td>
<td>How an ESB can be used to define business processes, business events and business rules.</td>
</tr>
<tr>
<td>Management</td>
<td>Possible management capabilities of an ESB.</td>
</tr>
</tbody>
</table>

### 3.2 Communication

To enable interaction between service requesters and providers (that is, to receive and deliver messages), the ESB must connect to requesters and providers using communication protocols and message exchange patterns.
3.2.1 Communication Protocols

The ESB itself does not actually provide the communication protocols, but instead leverages one or more features of the underlying MOM which is at the core of the ESB communications layer.

Table 3.2 lists popular communication protocols an ESB can support to interconnect service requesters and providers.

Table 3.2: Capabilities of the Communication Protocols category

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTP</td>
<td>FTP is a commonly used protocol for exchanging files over any TCP/IP based network to manipulate files on another computer on that network.</td>
</tr>
<tr>
<td>HTTP</td>
<td>A communication protocol used to transfer information on intranets and the World Wide Web.</td>
</tr>
<tr>
<td>HTTPS</td>
<td>HTTPS is a combination of the HTTP protocol with the SSL/TLS protocol. It provides encrypted communication to prevent eavesdropping and secure identification of a network Web server.</td>
</tr>
<tr>
<td>POP3</td>
<td>POP is an application layer Internet protocol used to retrieve e-mail from a remote server.</td>
</tr>
<tr>
<td>SMTP</td>
<td>SMTP is an Internet protocol for e-mail transmission across IP networks.</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>TCP/IP is the set of communications protocols used for the Internet and other similar networks.</td>
</tr>
</tbody>
</table>
3.2.2 Routing

When the ESB receives a message from an endpoint it should route the message to the next node, or back to the requester. The output message may be the same message that was received or the service may augment or modify parts of the message, or create a whole new response message. The operation of identifying and locating the next service in the chain, the binding to the service, and its invocation are the tasks which are carried out by the ESB. The find/bind/invoke operations should not be defined through programming language code, but through configuration and deployment tools.

Table 3.3 lists the two most popular routing mechanisms.

Table 3.3: Capabilities of the Routing category

<table>
<thead>
<tr>
<th>Routing Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content-Based Routing</td>
<td>Content-based routing seeks to route messages, not in terms of a specified destination, but in terms of the actual content of the message itself.</td>
</tr>
<tr>
<td>Itinerary-Based Routing</td>
<td>Itinerary-based routing routes messages based on a message itinerary. The itinerary represents a set of discrete message routing operations.</td>
</tr>
</tbody>
</table>

3.2.3 Message Exchange Patterns

A Messaging Exchange Pattern (MEP) is a network-oriented architectural pattern that describes how service requesters and service providers connect and communicate with each other. Communication protocols inherently support one or more MEPS due to the characteristics of the underlying communication mechanisms. The most common MEPS are synchronous and asynchronous request-response, one-way, and publish-subscribe. For example, HTTP is a request-response pattern protocol, and the User Datagram Protocol (UDP) follows a one-way pattern. An ESB should adapt the communication between different MEPS.

Table 3.4 lists the most popular MEPS.
Table 3.4: Capabilities of the Message Exchange Patterns category

<table>
<thead>
<tr>
<th>Capability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous Invocations</td>
<td>Asynchronous invocations are non-blocking, allowing the invoking program flow to continue processing.</td>
</tr>
<tr>
<td>Request-Response</td>
<td>Request-response, also known as request-reply, is a MEP in which a requester sends a request message to a replier system, which receives and processes the request, ultimately returning a message in response.</td>
</tr>
<tr>
<td>One-way</td>
<td>One-way is a MEP where a message is sent and no response is returned.</td>
</tr>
<tr>
<td>Publish-Subscribe Engine</td>
<td>Publish-subscribe is a MEP where senders of messages, called publishers, do not send the messages directly to specific receivers, called subscribers, but through a messaging middleware system. Published messages are characterized into classes, without knowledge of what, if any, subscribers are interested in the messages. Subscribers express interest in one or more classes of messages, and only receive messages in which they are interested without knowledge of what, if any, publishers are generating the messages.</td>
</tr>
</tbody>
</table>

3.2.4 Protocol Transformations

Services can use different protocols to communicate. The ESB should facilitate service interoperation, even when they use different protocols to communicate.

Table 3.5 lists the identified capability of the Protocol Transformation category.

Table 3.5: Capabilities of the Protocol Transformation category

| Protocol Transformation | Ability to transparently transform one type of communication protocol to another. |

12
3.3 Integration

Which applications can communicate with an ESB depend on the applications accessibility options and the supported integration mechanisms of the ESB. An ESB extends its MOM core to flexibly connect to applications in an adaptable and non-intrusive way, through protocol adapters and messaging patterns.

![Integration Diagram]

Figure 3.3: Integration capability category

Adapters are useful for adjustments between different technologies and for extracting data and transactional information from the applications. The adapters belong to one of the following four types:

1. **Application adapters.** Allow existing applications to connect and communicate with the ESB.

2. **Technology adapters.** Allow systems built with different technologies (e.g., databases and message queues) to communicate with the ESB.

3. **Language adapters.** Allow programs written in different languages (e.g., Java and C#) to communicate with the ESB.

4. **Custom adapters.** Allow users to write their own adapters because it is not feasible for an ESB to offer adapters for all existing applications, programming languages and technologies.

Table 3.6 lists some identified capabilities of the Integration category.
Table 3.6: Some capabilities of the Integration category

<table>
<thead>
<tr>
<th>Framework for Custom Adapters</th>
<th>Framework for development of custom adapters.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Adapters</td>
<td></td>
</tr>
<tr>
<td>Excel</td>
<td>Retrieves data from Microsoft Excel files.</td>
</tr>
<tr>
<td>SharePoint</td>
<td>Natively interoperates with Microsoft SharePoint.</td>
</tr>
<tr>
<td>Technology Adapters</td>
<td></td>
</tr>
<tr>
<td>Web Services</td>
<td>Natively interoperates with Web services.</td>
</tr>
<tr>
<td>WCF Adapters</td>
<td>Natively interoperates with services built on WCF.</td>
</tr>
<tr>
<td>Microsoft SQL Server</td>
<td>Natively interoperates with Microsoft SQL Server.</td>
</tr>
<tr>
<td>MSMQ</td>
<td>Natively interoperates with MSMQ.</td>
</tr>
<tr>
<td>Silverlight</td>
<td>Natively interoperates with Silverlight.</td>
</tr>
<tr>
<td>Language Adapters</td>
<td></td>
</tr>
<tr>
<td>C#</td>
<td>Natively interoperates with programs written with the C# programming language.</td>
</tr>
</tbody>
</table>
3.4.1 Service Protocols

Service Protocols define the message structure and rules governing the processing along the message path between message participants. The most popular service protocol is the use of REST-style messages encoded in XML format. However, XML messages with SOAP as the message format are also still very popular.

Table 3.7 lists the identified capabilities of the Service Protocols category.

<table>
<thead>
<tr>
<th>Capability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAP</td>
<td>SOAP is a protocol for exchanging structured information via Web services.</td>
</tr>
<tr>
<td>REST</td>
<td>REST is an architectural style for service interactions that takes advantage of the technologies and protocols of the World Wide Web. REST describes how distributed data objects, or resources, can be defined and addressed, stressing the easy exchange of information and scalability.</td>
</tr>
</tbody>
</table>

3.4.2 Service Registry

Typical enterprise SOA solutions rely on a variety of services. Invoking these services requires knowledge of their location (i.e., the service endpoint address). In the simplest case, it is possible to hard-code endpoint addresses in the implementations. However, this approach introduces tight coupling between the implemen-
tations and their location (location coupling), and requires modifications to imple-
mentations when a service endpoint address changes.

The most flexible and maintainable solution for this problem is to use a service
registry as an intermediary. Service registries hide the concrete service descrip-
tions. Functionalities of a service registry include inquiring (search and lookup
entries) and publication (publish, delete, and update registry-related information).
The re-use of services greatly depends on the ability to describe and publish the
offered functionality of the services to potential service requesters.

Table 3.8 lists the two most popular service registries.

Table 3.8: Capabilities of the Service Registry category

<table>
<thead>
<tr>
<th>UDDI</th>
<th>Infrastructure that supports the description, publication, and discovery of service providers; the services that they offer; and the technical details for accessing those services.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ebXML</td>
<td>Infrastructure that supports information sharing using standard protocols. It provides a secure approach for publishing business service information. Specifically, ebXML enables the storage, classification and retrieval of e-business structures.</td>
</tr>
</tbody>
</table>

3.4.3 Interface Definition

A service provider needs a way to describe the functionality it offers to potential
service consumers and it is preferable to use a standard notation that allows mul-
tiple service consumers to interact with the service.

Table 3.9 lists the most popular capability of the Interface Definition category.
Table 3.9: Capabilities of the Interface Definition category

| WSDL                  | An XML-based language that is used to describe the functionality offered by a Web service. A WSDL description of a Web service provides a machine-readable description of how the service can be called, what parameters it expects, and what data structures it returns. |

3.4.4 WS-* Specifications

In order to guarantee interoperability the interaction between service requesters and providers should be supported by standards. There are a variety of specifications associated with Web services, and these specifications may complement, overlap, and compete with each other. We have divided the available standards over the identified categories of the capability model. So, this category only includes the standards that include information about the interaction with Web services.

Table 3.10 lists the specifications that an ESB can support to describe specific aspects of the interaction between service requesters and service providers.
Table 3.10: Capabilities of the WS-* Specifications category

<table>
<thead>
<tr>
<th>Specification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS-Policy</td>
<td>A specification that allows Web services to advertise their policies (on security, quality of service, etc.) and for Web service consumers to specify their policy requirements.</td>
</tr>
<tr>
<td>WS-Discovery</td>
<td>A technical specification that defines a multicast discovery protocol to locate services on a local network. The actual communication between nodes is done using Web services standards, notably SOAP over UDP.</td>
</tr>
<tr>
<td>WS-Addressing</td>
<td>A specification of a transport-neutral mechanism that allows Web services to communicate addressing information. It consists of a standardized way of including message routing data within SOAP headers.</td>
</tr>
<tr>
<td>WS-MetadataExchange</td>
<td>A Web services protocol specification designed to work in conjunction with WS-Addressing, WSDL and WS-Policy to allow retrieval of metadata about a Web services endpoint.</td>
</tr>
</tbody>
</table>

3.5 Message Processing

A message may require transformation because of one of the following:
the structure of the received message is not in the expected format,
additional information is needed,
or the message must be converted, e.g. from XML to a non-standard format.
This section describes this process by introducing the VETO pattern [1].

Table 3.11 lists the identified capabilities of the Message Processing category.
3.5.1 VETO

VETO is a common integration pattern [1] that stands for Validate, Enrich, Transform, Operate (see Figure 3.6). The VETO pattern can ensure that consistent, validated data will be routed throughout the ESB.

Validate

The Validate step is usually the first part of any ESB process and can be accomplished in a number of ways. The simplest form of validation is to verify that an incoming message contains a well-formed XML document and conforms to a particular schema or to a WSDL document that describes the message. This requires that the service uses a validating parser and that it always has the up-to-date XML schema available for a particular message type. The schema can be kept in a directory service and managed remotely by the management infrastructure of the ESB. If the target data is not in XML format, or if there is no schema available, then a custom service can be used to validate the incoming message.
**Enrich**

The Enrich step involves adding data to a message to make it more meaningful and useful to a target service or application. The Enrich service could be implemented to invoke another service to look up additional data, or it could access a database to get the information necessary for its operation.

**Transform**

The Transform step converts the message to a target format. This often involves converting the data structure to an internal canonical format, or converting from the canonical format to the target format of the Operate step. The target system may have its own built-in validation rules requiring that the transformation step modifies the incoming data in order to prevent the target system from rejecting the message.

**Operate**

The Operate step invokes the target service or interacts in some way with the target application. If the operation is supported by a service that is written to conform to the agreed-upon canonical data format, the transformation converts the incoming message to the canonical format, if it is not already in that format. If the target operation is supported by an enterprise application that requires its own data format, then the transformation step converts the message to the target format required by the application.
Table 3.11: Capabilities of the Message Processing category

<table>
<thead>
<tr>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSD Schema Message Validation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Enrichment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
</tr>
<tr>
<td>Web Service</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>XSLT</td>
</tr>
<tr>
<td>Schema Mapping GUI</td>
</tr>
</tbody>
</table>

### 3.6 Business Processes

An SOA requires that services encapsulate reusable business functions that are loosely bound. So, it is imperative that there should be a mechanism to enable these services to be linked together to create higher-level business processes.

An ESB can provide such a mechanism to enable integration architects to create flows between business services that can be modified and changed dynamically by changing business rules.

![Figure 3.7: Business Processes capability category](image)
3.6.1 Service Orchestrations

Service orchestrations provide an open, standards-based approach for connecting services together to create higher-level business processes.

Orchestrations describe how services can interact with each other at the message level, including the business logic and an execution order of the interactions. These interactions may span applications and/or organizations, and result in a long-running, transactional, multi-step process model.

Table 3.12 lists the identified capabilities of the Service Orchestrations category.

<table>
<thead>
<tr>
<th>Capability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long Running Orchestrations</td>
<td>Orchestrations that can take days to complete.</td>
</tr>
<tr>
<td>Web Service Generation</td>
<td>Ability to generate and publish Web services from orchestrations.</td>
</tr>
<tr>
<td>Tracking and Debugging Flows</td>
<td>GUI tool to allow for tracking and debugging of process flows.</td>
</tr>
<tr>
<td>Process Visualization</td>
<td>GUI tool to visually enable the creation of orchestrations.</td>
</tr>
<tr>
<td>WS-BPEL</td>
<td>A standard executable language for specifying business processes involving activities performed by invoking (orchestrating) Web services. Processes in BPEL exchange information by using exclusively Web service interfaces.</td>
</tr>
</tbody>
</table>

3.6.2 Business Events

It is important for businesses to provide both technical and non-technical users with end-to-end visibility into the business process on a near real-time basis. This enables organizations to make timely and well-formed decisions to improve business agility and customer satisfaction. The ESB can support developers in publishing information to the right users at the right time by allowing events to be publishing from within orchestrations.

Table 3.13 lists the identified capabilities of the Business Events category.
Table 3.13: Capabilities of the Business Events category

<table>
<thead>
<tr>
<th>Publish-Subscribe Mechanism</th>
<th>Mechanism to publish events that occur within orchestrations and the ability to subscribe to those events.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Management GUI</td>
<td>GUI to manage publications and subscriptions of business events.</td>
</tr>
</tbody>
</table>

3.6.3 Business Rules

Business rules guide the decision-making behaviour of all business processes. They are the conditional criteria against which factual variables are evaluated to determine an appropriate business action. In conventional application design, business rules are coded as procedural implementation methods. Embedding business rules within applications has always been problematic because it compromises the effectiveness and versatility of the application, and consequently the business process that the application is meant to support.

Isolating business rules entirely from procedural code, or any process implementation mechanism, would dramatically improve a business’ ability to manage and adapt its business processes in response to new requirements or business conditions.

Table 3.14 lists the identified capabilities of the Business Rules category.

Table 3.14: Capabilities of the Business Rules category

<table>
<thead>
<tr>
<th>Rule Authoring/Definitions</th>
<th>GUI to be able to author business rules.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Versioning</td>
<td>Ability to deploy new versions of business rules, ability to have several versions that can be deployed.</td>
</tr>
<tr>
<td>API</td>
<td>Published API for interacting with Business Rules from external applications.</td>
</tr>
</tbody>
</table>

3.7 Quality of Service

Quality of Service (QoS) options are related to the quality of the experience associated with interactions with a Web service and they specify the requirements that are associated with the overall reliability of services.
3.7.1 Reliability

The benefits of a loosely coupled messaging framework comes at the cost of a loss of control over the actual communications process. After a service transmits a message, it has no immediate way of knowing:

- Whether the message has successfully arrived at its intended destination.
- Whether the message failed to arrive and therefore requires a retransmission.
- Whether a series of messages arrived in the sequence in which they were intended to.

Reliable messaging addresses these concerns by establishing a measure of quality assurance and it also provides guaranteed notification of delivery success or failure.

Table 3.15 lists the capabilities of the Reliability category which have been identified.

Table 3.15: Capabilities of the Reliability category

<table>
<thead>
<tr>
<th>Capability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS-ReliableMessaging</td>
<td>WS-ReliableMessaging describes a protocol that allows messages to be delivered reliably between distributed applications in the presence of software component, system, or network failures.</td>
</tr>
<tr>
<td>Failed Message Routing</td>
<td>When a message fails on a receive port it is routed to a location where additional action can be taken.</td>
</tr>
</tbody>
</table>

3.7.2 Transactions

An ESB can coordinate distributed transactions in which multiple services participate. When multiple distributed services need to participate in a transaction some
entity typically has to coordinate the transaction. The ESB can do this rather than forcing the clients to do it.

Table 3.16 lists the identified capability of the Transactions category.

Table 3.16: Capabilities of the Transactions category

<table>
<thead>
<tr>
<th>Capability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS-AtomicTransaction</td>
<td>Defines a specific set of protocols that plug into WS-Coordination to implement the traditional two-phase atomic ACID transaction protocols. It centers around short-lived operations, or in other words, processes in which the success or failure of a transaction should be known rapidly.</td>
</tr>
</tbody>
</table>

3.7.3 Performance

An ESB is most valuable if it is the mediator for all service requesters-providers conversations. However, this introduces some overhead. The amount of overhead incurred by invoking a service via an ESB versus invoking it directly depends on the architecture of the specific ESB. An ESB can increase its performance by supporting load balancing of services which will allow it to cope with higher workloads by distributing services evenly amongst servers.

Table 3.17 lists the identified capabilities of the Performance category.

Table 3.17: Capabilities of the Performance category

<table>
<thead>
<tr>
<th>Capability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Balancing</td>
<td>Ability to deploy multiple instances of a service and use a load balancer to dispatch requests and spread out the service request traffic.</td>
</tr>
<tr>
<td>Message Throttling</td>
<td>Configuration to allow only a specific number of messages to reach the service in a specific period of time.</td>
</tr>
</tbody>
</table>

3.7.4 Availability

Service availability is important in an SOA because there are likely to be dependencies of multiple applications on a single service. If that single service becomes unavailable or unresponsive, every application relying upon that service becomes unavailable. A successful SOA implementation must be able to assure a level of availability to every service and every application that depends on that service.
Table 3.18 lists the identified capability of the Availability category.

Table 3.18: Capabilities of the Availability category

| High Availability | High availability of a service regardless of the status of the hosting or dependent servers on which it runs. |

3.7.5 Scalability

To cope with high workloads an ESBs can support different scalability options.

Table 3.19 lists the identified capabilities of the Scalability category.

Table 3.19: Capabilities of the Scalability category

| Vertical Scaling | Scaling vertically increases the performance by using multiple processors and more memory. |
| Horizontal Scaling | Scaling horizontally increases the performance and fault-tolerance by running multiple ESB instances on different machines. |

3.8 Security

Security aspects like authentication and authorization can be centralized in the ESB. Even if a service does not have a built-in authentication and authorization mechanism, an ESB can require this in the service interface it exposes to potential service requesters.

Table 3.20 lists the identified capabilities of the Security category.

Figure 3.9: Security capability category
Table 3.20: Capabilities of the Security category

<table>
<thead>
<tr>
<th>Capability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS-Security</td>
<td>WS-Security describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication.</td>
</tr>
<tr>
<td>Content Encryption/Decryption</td>
<td>Support for encryption of message contents.</td>
</tr>
<tr>
<td>Content Based Authentication and Authorizations</td>
<td>Authentication or authorization based on the content of the messages.</td>
</tr>
<tr>
<td>Digital Signatures</td>
<td>Ability to use digital signatures to grant permissions.</td>
</tr>
<tr>
<td>Password Synchronization</td>
<td>Mechanism to provide for password synchronization (ESB connects to other systems to synchronize credentials across multiple systems).</td>
</tr>
<tr>
<td>Non-Repudiation</td>
<td>Ensure that a transferred message has been sent and received by the parties claiming to have sent and received the message.</td>
</tr>
<tr>
<td>Single SignOn</td>
<td>Service that allows administrators to map a user account to one or more alternative Windows or non-Windows accounts. These accounts are mapped per application so that they can be used to securely access applications that require credentials other than those originally provided by the end-user.</td>
</tr>
</tbody>
</table>
3.9 Management

The ESB can provide a management application to manage, monitor and configure its features.

![Management capability category](image)

**Figure 3.10: Management capability category**

**Table 3.21: Capabilities of the Management category**

<table>
<thead>
<tr>
<th>Capability</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logging</td>
<td>Logging of messages and ease of access to these messages.</td>
</tr>
<tr>
<td>Alerts</td>
<td>Issue alerts based on certain parameters.</td>
</tr>
<tr>
<td>Message Tracking</td>
<td>Tool to track messages as they flow through the ESB.</td>
</tr>
<tr>
<td>Exception Reporting</td>
<td>Generate reports based on error messages severity, either related to orchestrations or server statuses.</td>
</tr>
<tr>
<td>Performance Monitoring</td>
<td>Tool for monitoring system performance.</td>
</tr>
<tr>
<td>Real-Time Process Monitoring</td>
<td>Tool to visually monitor processes in real-time.</td>
</tr>
<tr>
<td>End Point Monitoring</td>
<td>Provide real-time information about the current state of the services hosted and used by the ESB.</td>
</tr>
</tbody>
</table>
Existing Solutions

This chapter provides an overview of the available .NET-based ESBs and discusses techniques that can be used as building blocks to develop a custom ESB.

Section 4.1 treats Microsoft’s BizTalk Server 2010.
Section 4.2 discusses Neudesic’s Neuron ESB 2.6.
Section 4.3 handles Microsoft’s AppFabric Service Bus.
Section 4.4 deals with the techniques that can be used as building blocks for the development of a custom ESB.
Finally, Section 4.5 provides an overview of the smaller ESBs and discusses the solutions which we selected to be evaluated.

4.1 Microsoft BizTalk Server 2010

BizTalk Server 2010 [12] [13] is Microsoft’s integration and connectivity server which is a solution to facilitate the connection of disparate systems. BizTalk Server includes over 25 multi-platform adapters [14], an extensive messaging infrastructure and also provides connectivity between core systems both inside and outside an organization.

Its key capabilities are:

- **Messaging.** BizTalk Server enables the efficient processing of incoming and outgoing messages. This capability provides connectivity to disparate systems and trading partners through a variety of file formats and adapters and is secured by message-level security.

- **Orchestration.** BizTalk orchestration provides transactional and non-transactional message processing through centrally managed business processes. Orchestrations enable the automation and standardization of complex processes that are composed in a process diagram and executed by the BizTalk Orchestration engine at runtime.

- **Business Rules Framework.** The Business Rules Framework enables the cre-
ation of business rules that define the logic for a given business process. The business rules abstracts the business process logic out of an orchestration which enables updates to be made to the business process logic without requiring any recoding of the orchestration.

• **Business-to-Business integration.** Electronic transactions with trading partners can play a vital role in enterprise business processes. BizTalk Server enables business-to-business integration by providing native support for Electronic Data Interchange (EDI) data (including both X12 and EDIFACT, and HIPAA support) and Applicability Statement 2 (AS2) data for EDI over the Internet.

• **BizTalk RFID.** BizTalk RFID provides a device management and event processing platform that enables the development, deployment, and management of Radio Frequency ID (RFID) and sensor solutions.

• **Business Activity Monitoring (BAM).** BAM provides real-time monitoring and archived statistical data of BizTalk processes and it enables business users to gain end-to-end visibility of these processes.

• **Management and operations.** BizTalk Server provides management of the BizTalk Server runtime environment, including application management, application deployment, host management, and process execution tracking and reporting.

• **Tools.** BizTalk Server provides a number of tools that help configure, design, deploy, manage, and view its processes and capabilities. These tools come in a variety of forms; some are integrated into Microsoft Visual Studio 2010, some are add-ins to the Microsoft Management Console (MMC), while others are Web-based.

Shortly after the release of Microsoft BizTalk Server, Microsoft also released the BizTalk ESB Toolkit [15]. This toolkit extends the functionality of BizTalk Server to provide a range of new capabilities focused on building robust, connected, service-oriented applications. It also incorporates itinerary-based service invocation for lightweight service composition, dynamic resolution of endpoints, Web service and WS-* integration, fault management and reporting, and integration with third-party SOA governance solutions.
At the core of BizTalk Server are the Messaging Engine and the Orchestration Engine, which provide the underlying architecture for integrating and exchanging messages between various services. The BizTalk Orchestration Engine coordinates and schedules message processing, and performs complex logic on messages as they pass through a defined workflow.

BizTalk Server implements the publish-subscribe model to route messages. Figure 4.1 shows how BizTalk Server implements the publish-subscribe model.

In this model the message providers, also called the publishers, submit messages to a central store (the MessageBox database). The subscribers, which can be send ports or orchestrations, subscribe to specific messages. After the MessageBox receives a message of interest, this message is then delivered to all of its subscribers.

BizTalk Server provides a rich set of development tools for designing, organizing, and building the various elements of a BizTalk application. The BizTalk project system is part of Visual Studio 2010 and provides developers with a fully integrated design experience to create parts of a BizTalk application or an entire business solution.

BizTalk Server 2010 has an extensive licensing model which offers licenses varying in price from $2,528 (August 2012), with very limited capabilities, up to $44,228

Figure 4.1: The BizTalk Server publish-subscribe model [16]
for enterprise licenses which allow the users to host unlimited applications in BizTalk Server.

### 4.2 Neudesic Neuron ESB 2.6

Neuron ESB is a .NET-based ESB, developed by a Microsoft partner, whose scope includes the following elements [17]:

- Interconnection of applications, services, and legacy systems.
- Intelligent routing.
- EAI functionality.
- Centralized management.
- Service Level Agreement monitoring and enforcement.
- Deployment.
- Compliance auditing.

Neuron ESB specifies an architecture for combining these elements, defines metadata for expressing configuration and policies, and includes framework software that is implemented using .NET components.

The Neuron ESB software consists of infrastructure services and .NET assemblies which are modular, replaceable, and extensible. Working in concert, they provide a connection and integration backbone for the enterprise, and an operating environment for services. The infrastructure services that are shipped with Neuron ESB are implemented using .NET components, but customers are free to change the implementation of any service.

Figure 4.2 illustrates the Neuron ESB architecture. This ESB provides a common messaging fabric that interconnects several categories of endpoints:

- Business endpoints include applications, services, and legacy systems.
- Integration services perform useful EAI functions that can be applied to messages as they are en route.
- Infrastructure services implement the ESB, providing endpoint connections, security enforcement, configuration management, message routing, and operations monitoring.
The core messaging system is responsible for connecting endpoints and routing messages between them. Applications share messages through the publish-subscribe communication.

Neuron ESB provides a powerful management tool named ESB Explorer. Using ESB explorer, administrators can view, configure, monitor, and control the complete ESB with a single tool.

Neuron ESB is priced per processor at about $24,000 (August 2012). For a more detailed price the Neuron ESB help desk has to be contacted.

4.3 Windows Azure AppFabric Service Bus

The Windows Azure Service Bus is a highly scalable developer-oriented service running in Microsoft data centers [18]. The primary purpose of this Service Bus is to "relay" messages through the cloud to services running on-premises behind firewalls and NAT devices on-premises. The Service Bus relies on Access Control for securing access to endpoints through a claims-based security model. Together these services provide a development framework required by most cloud applications today.

The Service Bus provides both "relayed" and "brokered" messaging capabilities. In the relayed messaging pattern, the relay service supports direct one-way mes-
saging, request/response messaging, and peer-to-peer messaging.

Brokered messaging provides durable, asynchronous messaging components such as queues, topics, and subscriptions, with features that support publish-subscribe and temporal decoupling: senders and receivers do not have to be online at the same time, but the messaging infrastructure reliably stores messages until the receiving party is ready to receive them.

Because Azure ESB is a cloud platform the pricing model is very different compared to applications like BizTalk Server and Neuron ESB. The pricing model for the Azure ESB does not involve up-front costs and only resources which have been used must be paid. Some resources like virtual PCs are paid for on a per-hour usage basis, whereas others like storage, SQL databases and bandwidth are charged on a per-GB/month usage basis, and some other resources are charged with per-transaction costs.

4.4 Custom ESB

The .NET framework itself already contains all the elements necessary to build a custom ESB. The most important building block for building an ESB is the Windows Communication Foundation (WCF), this is because it provides a broad set of network capabilities which can be used to interconnect systems. An important advantage of WCF as the base of an ESB is the extensive API it provides. WCF significantly reduces the amount of code that has to be written for basic infrastructure mechanisms (e.g., reliable messaging, transactions and security). It is also possible to implement an ESB without WCF, but not having to write and maintain tens of thousands of lines of code is a compelling reason to use WCF.

Windows Workflow Foundation 4 (WF4) is also part of the .NET framework and can be used as the second building block for the development of a custom ESB. WF4 can be used in combination with WCF to provide the framework for defining workflows that can be updated at runtime and can be used as service orchestrations, long-running business processes or perform message processing.

4.5 Product Selection

Besides the previously discussed ESBs and techniques there are many smaller and less mature solutions. Below is a list of those solutions that we have researched:

- Rhino Service Bus [19]
Most of these ESBs have in common that they are based on MSMQ or WCF, and provide a reliable publish-subscribe mechanism that is not supported by WCF by default. However, none of these products comes close to the maturity, documentation quality and wide range of capabilities supported by BizTalk Server and Neuron ESB. There are also a couple of Java based ESBs which provide a .NET programming interface (e.g., Fiorano ESB and Sopera ESB) but in this research the focus is on .NET based solutions.

Windows Azure AppFabric Service Bus is an interesting solution for a company that is intending to host their products in the cloud. Azure AppFabric Service Bus allows applications to scale well because they can use more resources when necessary, but does not provide as much out-of-the-box capabilities as Microsoft BizTalk Server and Neudesic Neuron ESB.

We have selected the three following solutions for our evaluation:

1. BizTalk Server 2010
2. Neuron ESB 2.6
3. WCF with WF4

We selected BizTalk Server 2010 and Neuron ESB 2.6 because they are the two state-of-the-art commercially available .NET-based ESBs. In addition, we selected WCF with WF4 as two building blocks for the development of a custom ESB to stress the benefits of a commercially available solution over a custom ESB implementation.
In this chapter we evaluate Neuron ESB 2.6, BizTalk Server 2010 and WCF with WF4 by comparing and analyzing the capabilities specified in the capability model. We compared the solutions using the documentation that was provided online by the various vendors and our knowledge acquired by performing a literature study and the case study.

Section 5.1 explains the Evaluation Criteria which we used.
Section 5.2 gives the results of our capability evaluation for Communication in tabular form.
Section 5.3 shows the results when the Integration capabilities are considered.
Section 5.4 treats the aspect of Service Interaction in the capability evaluation model.
Section 5.5 evaluates Message Processing.
Section 5.6 is a table showing the evaluation results with respect to Business Processes.
Section 5.7 handles the Quality of Service category.
Section 5.8 exposes the capability aspect with respect to Security.
Section 5.9 treats last but not least the capability evaluation of the Management category.

5.1 Evaluation Criteria

Each supported capability gets one of the ratings that are explained in table 5.1.
We give two examples to explain and clarify how we carried out the evaluation. Let us look, for example, at how we applied the Capability Evaluation on the publish-subscribe engine considering the Communication category. Both BizTalk Server and Neuron ESB provide such an engine out-of-the-box and WCF with WF4 does not. So, in this case we rated BizTalk Server and Neuron ESB with the ++ rating and WCF with WF4 was rated with the – rating. This does not mean that using the .NET framework this mechanism can not be implemented, but just that WCF with WF4 does not provide this capability out-of-the-box. When a capability can be implemented with little effort then the - rating was given.

The second example is the Web Service Generation capabilities from the Business Process Category. The description of this capability is: “Ability to generate and publish Web services from orchestrations”. Both BizTalk Server and Neuron ESB fully support this. WCF with WF4 also supports the creation of orchestrations via workflows and it is also possible to automatically generate a Web service that can invoke this workflow, but, in this case, the generated Web service has to be published manually using other tools, while BizTalk Server and Neuron ESB can do this automatically. So, in this example we rated BizTalk Server and Neuron ESB with the ++ rating and WCF with WF4 with the + rating.
### 5.2 Communication

<table>
<thead>
<tr>
<th>Category</th>
<th>Capability</th>
<th>BizTalk Server</th>
<th>Neuron ESB</th>
<th>WCF &amp; WF4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication Protocols</td>
<td>Protocol Transformation</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>FTP</td>
<td>++</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>HTTP</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>HTTPS</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>POP3</td>
<td>++</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>SMTP</td>
<td>++</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>TCP/IP</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Routing</td>
<td>Content Based</td>
<td>++</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Itinerary Based</td>
<td>++</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td>Message Exchange Patterns</td>
<td>Asynchronous Invocations</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>One Way</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Publish Subscribe Engine</td>
<td>++</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>Request Response</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
</tbody>
</table>

- **Communication protocols.** WCF does not provide direct support for the FTP, POP3 and SMTP protocols. Neuron ESB only does not support the POP3 protocol.

- **Content-based routing.** WCF provides a routing service that can be used for content-based routing. However, the routing service has to be configured using a XML configuration file whereas the other two solutions provide a user interface for the configuration of their content-based routing mechanism.

- **Itinerary-based routing.** WCF does not provide built-in support for itinerary-based routing.

- **Publish Subscribe Engine.** WCF does not provide a publish-subscribe engine.
5.3 Integration

<table>
<thead>
<tr>
<th>Integration</th>
<th>Category</th>
<th>Capability</th>
<th>BizTalk Server</th>
<th>Neuron ESB</th>
<th>WCF &amp; WF4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integration</td>
<td>Framework for Custom Adapters</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td></td>
</tr>
<tr>
<td>Application Adapters</td>
<td>Excel</td>
<td>--</td>
<td>++</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>Language Adapters</td>
<td>SharePoint</td>
<td>++</td>
<td>++</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>Technology Adapters</td>
<td>C#</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Web Services</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WCF Adapters</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SQL Server</td>
<td>++</td>
<td>++</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MSMQ</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Silverlight</td>
<td>--</td>
<td>--</td>
<td>++</td>
<td></td>
</tr>
</tbody>
</table>

- **Excel.** Neuron ESB is the only solution that provides support to convert an Excel worksheet into an XML document with configuration only.

- **Sharepoint.** WCF does not provide a built-in adapter to communicate with Sharepoint.

- **SQL Server.** WCF does not provide a built-in adapter to communicate with SQL Server.

- **Silverlight.** BizTalk Server and Neuron ESB do not provide an API for Silverlight.
5.4 Service Interaction

<table>
<thead>
<tr>
<th>Service Interaction</th>
<th>Capability</th>
<th>BizTalk Server</th>
<th>Neuron ESB</th>
<th>WCF &amp; WF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Registry</td>
<td>UDDI</td>
<td>++</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>ebXML</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Interface</td>
<td>WSDL</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Service Protocols</td>
<td>SOAP</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>REST</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>WS-* Specifications</td>
<td>WS-Addressing</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>WS-MetadataExchange</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>WS-Discovery</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>WS-Policy</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
</tbody>
</table>

- **UDDI.** WCF does not provide built-in support for a UDDI service registry.

- **ebXML.** None of the solutions provide support for the ebXML service registry.

5.5 Message Processing

<table>
<thead>
<tr>
<th>Message Processing</th>
<th>Capability</th>
<th>BizTalk Server</th>
<th>Neuron ESB</th>
<th>WCF &amp; WF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation</td>
<td>XSD Schema</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Data Enrichment</td>
<td>Database</td>
<td>++</td>
<td>++</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Web Service</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Transformation</td>
<td>XSLT</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Schema Mapping</td>
<td>++</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

- **Data enrichment.** WCF with Workflow does not provide functionality to enrich a message with content from a database with configuration only. However, this functionality can be implemented with a little effort by implementing an activity that can be used by WF4.

- **Schema Mapping.** BizTalk Server is the only solution that provides a extensive schema mapping GUI.
5.6 Business Processes

<table>
<thead>
<tr>
<th>Business Processes</th>
<th>Capability</th>
<th>BizTalk Server</th>
<th>Neuron ESB</th>
<th>WCF &amp; WF4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service Orchestration</td>
<td>Long Running Orchestration</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Web Service Generation</td>
<td>++</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Tracking and Debugging Flows</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Orchestration GUI</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>WS-BPEL</td>
<td>+</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Business Events</td>
<td>Publish-Subscribe Mechanism</td>
<td>++</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>Event Management GUI</td>
<td>++</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td>Business Rules</td>
<td>Rule Authoring/Definitions</td>
<td>++</td>
<td>--</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Versioning</td>
<td>++</td>
<td>--</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>API</td>
<td>++</td>
<td>--</td>
<td>+</td>
</tr>
</tbody>
</table>

- **Web Service Generation.** WCF with WF4 support the creation of orchestrations via workflows and it is also possible to automatically generate a Web service that can invoke this workflow, but, in this case, the generated Web service has to be published manually using other tools, while BizTalk Server and Neuron ESB can do this automatically.

- **WS-BPEL.** WS-BPEL is not used by the BizTalk Server orchestrations, but BizTalk Server can import and export WS-BPEL 1.1 processes.

- **Business Events.** WF4 does not provide a business event framework.

- **Business Rules.** Neuron ESB does not provide a framework to manage business rules, but decisions are always part of the orchestrations. Windows Workflow Foundation 3 (WF3) provides a rules engine that can be used by WF4, but this is not an ideal solution because the applications then use multiple versions of WF.

WF4 provides functionality for business rules, but unlike the rules created with the rules engine of WF3, there is no out-of-the-box functionality to manage these rules from outside of the workflows that use them.
5.7 Quality Of Service

<table>
<thead>
<tr>
<th>Category</th>
<th>Capability</th>
<th>BizTalk Server</th>
<th>Neuron ESB</th>
<th>WCF &amp; WF4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transactions</td>
<td>WS-Atomic Transaction</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Availability</td>
<td>High Availability</td>
<td>++</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td>Reliability</td>
<td>WS-ReliableMessaging</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Failed Message Routing</td>
<td>++</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td>Performance</td>
<td>Load Balancing</td>
<td>++</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Message Throttling</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>Scalability</td>
<td>Vertical Scaling</td>
<td>++</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>Horizontal Scaling</td>
<td>++</td>
<td>++</td>
<td>--</td>
</tr>
</tbody>
</table>

- **WCF with WF4.** WCF depends on other tools to host the services, so that is why they do not support the High Availability, Failed Message Routing, Vertical Scaling and the Horizontal Scaling capabilities. These capabilities can be supported by the hosting environment.

- **Load Balancing.** Neuron ESB and WCF provide detailed information on how to configure the product with a load balancer, but they require an additional load balancer that performs the actual load balancing.

5.8 Security

<table>
<thead>
<tr>
<th>Category</th>
<th>Capability</th>
<th>BizTalk Server</th>
<th>Neuron ESB</th>
<th>WCF &amp; WF4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security</td>
<td>WS-Security</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Content Encryption and Decryption</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Content Based Authentication and Authorization</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Digital Signatures</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Password Synchronization</td>
<td>++</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>Non-Repudiation</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Single SignOn</td>
<td>++</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>
• **Neuron ESB.** The Neuron ESB documentation does not mention anything about the Password Synchronization, and Single SignOn capabilities.

• **WCF with WF4.** The Password Synchronization, and Single SignOn capabilities are not supported by WCF with WF4 because there is no central management application.

### 5.9 Management

<table>
<thead>
<tr>
<th>Category</th>
<th>Capability</th>
<th>BizTalk Server</th>
<th>Neuron ESB</th>
<th>WCF &amp; WF4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Management</strong></td>
<td>Logging</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Alerts</td>
<td>++</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>Message Tracking</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>Exception Reporting</td>
<td>++</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td><strong>Monitoring</strong></td>
<td>End Point</td>
<td>++</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>Process</td>
<td>++</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>Performance</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
</tbody>
</table>

• **Neuron ESB.** The Neuron ESB documentation does not mention anything about the Password Synchronization, and Single SignOn capabilities.

• **Process Monitoring.** Neuron ESB and WF4 do not support process tracking through GUI wizards, portals / dashboards. However, they support the textual tracking of the processes. So, customized tools can be built to visualize and analyze data. The WF4 SDK (Software Development Kit) provides a sample administrative tool to visualize tracked data. This sample tool can be enhanced to provide richer capabilities that match the standards of BAM.

### 5.10 Conclusion

We performed a capability evaluation based on the information which was provided in the documentation from the various vendors.

Table 5.2 provides an overview of how many times the different ratings are assigned to the evaluated solutions.
Based on this overview we can see that:

1. BizTalk Server has the most extensive capability support (64).
2. Neuron ESB supports fewer capabilities (55) than BizTalk Server.
3. WCF with WF4 has the least out-of-the-box capability support (36). This result was to be expected.

Although WCF with WF4 supports less capabilities than the others, however it still supports many capabilities but these do not form a complete solution with a central management application, like the others do. In other words the capabilities supported by WCF with WF4 are just an indication of the capabilities which are available and which can be used to build a custom ESB. A lot of work has to be done to develop a custom ESB that provides the capabilities and can be monitored and managed with a single management application based on WCF with WF4.

BizTalk Server and WCF with WF4 are the best documented solutions and Microsoft provides extensive documentation and examples on them. Moreover, a lot of additional information can be found online in third-party websites.

Neuron ESB is also well-documented, however almost no additional information can be found in third-party websites.

BizTalk Server and Neuron ESB are closely matched in terms of capabilities, but overall BizTalk Server provides the most extensive capability support.
CASE STUDY

In this chapter We describe the case study which we performed to evaluate Neuron ESB 2.6, BizTalk Server 2010 and WCF with WF4. We did this by providing an implementation for an integration scenario and analyzing the results.

Section 6.1 describes the case study.
Section 6.2 describes the evaluation criteria.
Section 6.3 describes the implementation using BizTalk Server 2010.
Section 6.4 describes the implementation using Neuron ESB.
Section 6.5 describes the implementation using WCF with WF4.
Section 6.6 summarises the evaluation of the three solutions.
Section 6.7 gives the conclusion of this case study.

6.1 Case Description

The problem which we have to solve in this case study is an integration scenario for a Travel Intermediary (TI), which is a platform that promotes tour operators. Operators use a TI to advertise their offers whilst the TI promotes its platform directly to the public.

We have limited the focus of this case study to the TI's reservation process in order to keep its size and complexity manageable. This process allows customers to make reservations with one of the tour operators and consists of the following steps:

1. The TI receives a reservation request from a customer.
2. The request needs to be routed to a suitable tour operator based on the content of the reservation request.
3. The customer needs to receive the tour operator’s response.

Figure 6.1 gives an overview of these steps.
6.1.1 Requirements

The TI has the following requirements.

- The TI wants to support a broad range of customers to enable them to communicate. So, customers should be able to communicate with the TI via a SOAP Web service over HTTP. However, it should be easy to add, in future, additional communication mechanisms, such as XML files via FTP, MSMQ or SMTP.

- Customers should be able to communicate with the TI using a predefined XML data format, but it should be possible, in future, to flexibly support additional data formats.

- The TI should be able to communicate with two tour operators who are accessible via SOAP Web services over TCP, but it should be easy to add additional tour operators that use other communication mechanisms in the future.

6.1.2 Scope

Our first challenge is to make it possible for customers and tour operators to use different transport protocols and data formats. However, using specific point-to-point transformations between each service causes the number of transformation instances to increase exponentially with the number of participants. This is commonly referred to as the N-squared problem.

Our second challenge is to ensure that, in future, the implementation should be flexible enough so that it can be easily changed and extended without affecting the already connected customers.

6.1.3 ESB Advantages

Using an ESB as an intermediary for message traffic brings a number of important advantages, including:
• **Location transparency:** the physical addresses of the services are known only to the ESB, eliminating the impact on applications that use these services, in case the service address changes.

• **Protocol mediation:** endpoints are free to use any protocol, if the customer can connect to the ESB, messages are routed and converted as required, without any impact on the customers.

• **Message processing:** messages can be transformed, enhanced, and processed as they pass through the ESB.

The ESB can solve the N-squared problem by using a canonical XML data exchange format so that the number of transformations increase linearly as new services are brought into the integration. When a new service is added, only one new type of transformation to and from the canonical format is needed and they do not require any transformation because they can use the canonical XML format directly.

Besides the N-squared problem, the ESB could also increase the flexibility of the solution by making it possible to change the business processes at runtime without the need to recompile any code.

In addition, the ESB could support all the capabilities described in the capability model and this would significantly simplify the implementation and management of the integration scenario.

### 6.1.4 Technical Specification

We have created a client application that simulates multiple customers sending reservation requests to the TI. Listing 6.1 shows an example of a contract for a reservation request. The TI should use the tour operator identifier to determine which tour operator should be selected.

Listing 6.1: Example customer reservation request contract

```xml
<Reservation>
  <Touroperator>1</Touroperator>
  <FirstName>Dennis</FirstName>
  <LastName>Dam</LastName>
</Reservation>
```
We have also created two SOAP tour operator Web services, each with a slightly different contract for making a reservation. Both services accept a simple MakeReservation element that accepts the customer’s name. The only difference is that they have a different number of sub-elements. The first tour operator requires a single element with the customer’s full name (see Figure 6.2). While the second tour operator requires two separate sub-elements for the name (see Listing 6.3).

Listing 6.2: Reservation request contract of tour operator 1

```xml
<MakeReservation>
  <FullName>Dennis Dam</FullName>
</MakeReservation>
```

Listing 6.3: Reservation request contract of tour operator 2

```xml
<MakeReservation>
  <FirstName>Dennis</FirstName>
  <LastName>Dam</LastName>
</MakeReservation>
```

Both tour operators provide metadata for their endpoint, which consists of a WSDL document describing the services.

Figure 6.2 gives a more detailed representation of the case study.

![Figure 6.2: Case study detailed overview](image)

### 6.2 Evaluation Criteria

Three implementations have been made and evaluated according to the following criteria:

- How many lines of developer written code are necessary to program the integration scenario?
• How much time does it take to add an additional customer who uses TCP instead of HTTP to communicate with the TI?

• How much time does it take for a customer to send 1000 sequential messages and receive the replies?

• How long does it take to install and configure the product used to develop the solution?

We executed the implementations in a virtual machine installed with Windows 7, 3.2 GB RAM and a 2.83GHz QuadCore processor. The performance indications in the case study are just to give a quick comparison between the three implementations and not to serve as a benchmark for the three solutions. We have omitted the time it took to install Visual Studio from the installation and configuration time of a solution, because Visual Studio is part of a standard .NET development environment and it was used to develop the client application that is used to communicate with all three solutions.

6.3 BizTalk Server

We realized the integration scenario by using BizTalk Server, Visual Studio and the BizTalk Server Administration tool. We performed the following steps to develop the implementation:


2. Create a new orchestration in the BizTalk Server application.

3. Import both tour operators using the Consume WCF Service Wizard by providing the link to their WSDL document.

4. Add both tour operator’s ports to the orchestration. A port defines a link to an external service but does not contain the concrete endpoint definitions.

5. Define the XML Schema for the incoming messages from the customers using a GUI provided by Biztalk Server.

6. Detail all the messages which have been used in the orchestration.

7. Define the mappings (see Figure 6.4) using the BizTalk Mapper to visually create the XML transformations for the defined messages.
8. Add shapes to the orchestration to call the correct tour operators, transform the messages and send a reply to the customer. See Figure 6.3 for the resulting orchestration.


10. Create an endpoint for the customers using the BizTalk WCF Service Publishing Wizard.

11. Import the generated tour operator services in BizTalk Server using the BizTalk Server Administration tool.

12. Configure the BizTalk Server application using the BizTalk Server Administration tool by mapping the three ports defined in the orchestration to the imported concrete endpoints.

13. Start the BizTalk Server application.
Figure 6.3: BizTalk Server Orchestration in Visual Studio
Evaluation

- How many lines of developer written code are necessary to program the integration scenario?
  - 1 for the selection of the operator.

- How much time does it take to add an additional customer who uses TCP instead of HTTP to communicate with the TI?
  - Less than ten minutes. The same orchestration can be used, but a new endpoint has to be created and configured using the BizTalk WCF Service Publishing Wizard and the BizTalk Server Administration tool.

- How much time does it take for a customer to send 1000 sequential messages and receive the replies?
  - Average time over 10 runs: 20.51 minutes.

- How long does it take to install and configure the product used to develop the solution?
  - More than an hour. The BizTalk Server installation is well documented, but it takes a lot of time because it has a lot of prerequisites including SQL Server 2008, Internet Information Services and many more that have to be installed in order for BizTalk Server to be installed.
6.4 Neuron ESB

We realized the integration scenario by using the Neuron ESB and the Neuron IDE without the use of any additional tools.

We performed the following steps to develop the implementation:

1. Create a Topic called Reservation for the Neuron ESBs publish/subscribe mechanism.

2. Create a Publisher (the customers) so that the customer can publish messages to the ESB.

3. Create a client SOAP endpoint for the customers by using a wizard in the IDE.

4. Import the Web services of the two tour operators using the link to their WSDL document.

5. Create two XSTL transformations to convert the input message format to the destination message format.

6. Create a process orchestration (see Figure 6.5) that checks for which tour operator the message is meant, calls that tour operator and returns the result to the customer.
Evaluation

- How many lines of developer written code are necessary to program the integration scenario?
  - 8 for the first XSTL transformation.
  - 6 for the second XSTL transformation.
  - 1 for the selection of the operator.

- How much time does it take to add an additional customer who uses TCP instead of HTTP to communicate with the TI?
  - Less than five minutes. The same orchestration can be used but only a new endpoint has to be created using the IDE.

- How much time does it take for a customer to send 1000 sequential messages and receive the replies?
  - Average time over 10 runs: 8.32 minutes.
• How long does it take to install and configure the product used to develop the solution?
  – Less than 15 minutes. Neuron ESB comes with a simple installer, and no additional configuration was necessary.

6.5 WCF with WF4

We realized the integration scenario by using WCF with WF4 in conjunction with Visual Studio, and the Internet Information Services 7 Manager to host the applications.

We performed the following steps to develop the implementation:

• Create a WCF Workflow Service Application using Visual Studio.

• Make a reference to the Web services of the two tour operators using the link to their WSDL file. This generated two activities which we used in the workflow to call the tour operators.

• Specify the input parameter, which is a simple C# class, from which the customer contract is automatically generated.

• Create a Workflow (see Figure 6.6) that checks for which tour operator the message is meant, calls that tour operator with the generated activities and returns the result to the customer.

• Assign all parameters in the Workflow. No XML transformations are necessary like with the other solutions, just simple strong typed code statements such as the condition "ServiceInput.TourOperator = 1" which can be seen in Figure 6.6. ServiceInput is the name of the simple C# class that was selected as the input parameter, and TourOperator is an integer property of that class.

• Host the Workflow Service using Internet Information Services.
Figure 6.6: Case study global overview

Evaluation

• How many lines of developer written code are necessary to program the integration scenario?
  – 1 for the selection of the operator.
  – 1 for the concatenation of the first name and the last name so that the message can be sent to tour operator 2.

• How much time does it take to add an additional customer who uses TCP instead of HTTP to communicate with the TI?
– Less than five minutes. The same orchestration can be used but only a new endpoint has to be specified in the configuration file of the service.

• How much time does it take for a customer to send 1000 sequential messages and receive the replies?

  – Average time over 10 runs: 4.28 minutes.

• How long does it take to install and configure the product used to develop the solution?

  – Less than 5 minutes. For the WCF and WF4 implementation we only used Internet Information Services 7 (IIS7) in addition to Visual Studio. IIS7 is a built-in windows feature that we enabled.

6.6 Comparison

This section provides an overview of the evaluation results and compares the three solutions by describing the advantages and disadvantages of each solution.

We have used the following ratings for the comparison:

• The + rating was given to the solution that provided the best result.

• The □ rating was given to the solution that provided second best result or in case multiple solutions had the same result.

• The - rating was given to the solution that provided the weakest result.

Table 6.1 provides the overview of the evaluation results.

Based on this overview we can see that the WCF with WF4 implementation provided the best results, Neuron ESB provided the second best results and BizTalk Server provided the weakest results.

The remainder of this section discusses the advantages and disadvantages of the evaluated solutions.
### Table 6.1: Case study evaluation results

<table>
<thead>
<tr>
<th>Evaluation Criteria</th>
<th>BizTalk Server</th>
<th>Neuron ESB</th>
<th>WCF &amp; WF4</th>
</tr>
</thead>
<tbody>
<tr>
<td>How many lines of developer written code are necessary to program the integration scenario?</td>
<td>+ (1 line)</td>
<td>− (15 lines)</td>
<td>□ (2 lines)</td>
</tr>
<tr>
<td>How much time does it take to add an additional customer who uses TCP instead of HTTP to communicate with the TI?</td>
<td>□ (&lt;5min)</td>
<td>□ (&lt;5min)</td>
<td>□ (&lt;5min)</td>
</tr>
<tr>
<td>How much time does it take for a customer to send 1000 sequential messages and receive the replies?</td>
<td>(20,51 min)</td>
<td>□ (8,32 min)</td>
<td>+ (4,28 min)</td>
</tr>
<tr>
<td>How long does it take to install and configure the product used to develop the solution?</td>
<td>− (&gt; 1 hour)</td>
<td>□ (&lt; 15 min)</td>
<td>+ (&lt; 5 min)</td>
</tr>
</tbody>
</table>

#### 6.6.1 BizTalk Server

**Advantages**

- **Strong abstraction.** When creating an orchestration it is not necessary to provide concrete endpoint definitions. Instead, the orchestration uses ports which can later be mapped to concrete endpoints without having to modify the application itself.

- **Well documented.** Documentation is found online. [27]

- **Familiar development environment.** For .NET developers it feels natural to develop applications in Visual Studio. BizTalk Server makes good use of this by employing Visual Studio as the development environment and integrating it perfectly with BizTalk Server.

- **Biztalk Mapper.** BizTalk Server was the only solution that allows XML transformations to be created using an user-friendly GUI.

- **Strong debugging capabilities.** BizTalk Server provides debugging capabilities which helped to debug the implementation which we developed.
Disadvantages

- **Steep learning curve.** Implementing the integration scenario with BizTalk Server took a lot more time than implementing it with the two other solutions. BizTalk Server was the last solution we used to implement the integration scenario. So, the experience gained from implementing the integration scenario with the other two solutions did not adversely affect the evaluation of the BizTalk Server implementation.

- **Resource intensive.** BizTalk Server needs the most system resources when compared with the other solutions. This also justifies the relatively poor performance of BizTalk Server to process the messages. During the test all required software like a full SQL Server, IIS etc. ran in a single virtual machine with few resources.

6.6.2 Neuron ESB

Advantages

- **Single Tool.** The complete implementation could be created using the Neuron IDE, which was used to define the orchestration, write and test the XSLT transformations, configure, monitor, test and host the entire integration scenario.

- **Medium learning curve.** The interface is quite intuitive which, allowed many actions to be performed without having to read many pages of documentation.

- **Well-documented.** Documentation can be found online. [28]

Disadvantages

- **XSLT transformations.** There is no GUI to visually define the transformations.

- **Smaller user base.** We found few results when searching online for examples or error messages when compared to the other solutions.
6.6.3 WCF with WF4

Advantages

- **Low learning curve.** It is likely that .NET developers are already familiar with WCF and WF4, which allows them to be productive quite quickly.

- **Well-documented.** Documentation can be found online. [29] [30]

Disadvantages

- **Not designed to be an ESB.** WCF with WF4 can be used to develop an ESB, but they are not designed to be an ESB, like BizTalk Server or Neuron ESB. The result is that many of its capabilities need to be implemented from scratch, or are achieved by using other tools. For example, when a WCF application is developed, the developer has to take care of the hosting and monitoring of the service himself, while the other solutions take care of this.

6.7 Discussion

This case study shows that the three solutions imported different approaches for solving the same problem. One of the most interesting observations is the difference in designing the contract for the communication with the client application that simulates the customers.

When designing a service contract, you can choose to use either the code-first or contract-first approach. It is interesting that contract-first is generally considered the right model to follow when building clients. Virtually all frameworks start by generating code from a WSDL document on the client side. This makes sense because you need to conform to a contract, so you start with the contract and generate the code. That is exactly what we observed in our case study, namely that all thee solutions imported the tour operator contracts based on the previously defined WSDL documents.

However, not all solutions use the same model for implementing services. In our case study each of the three evaluated solutions forced us to use a different approach to design the contract to communicate with the client application.

- BizTalk Server is the only solution that allowed a contract-first approach.
• Neuron ESB’s approach is neither pure code-based nor contract-based because it uses untyped Web services that do not care about the contents of a message.

• The WCF with WF4 example forced the use of the code-first approach, although there is also support for untyped Web services.

When there is a strong emphasis on the messages being exchanged or when there is a need to ensure conformance to an existing message format, using the contract-first approach is, without doubt, the most natural approach.

An advantage of Neuron ESB’s approach is that a single Web service can provide a single URL access point for all traffic, which simplifies the required configuration of client applications. However, a major disadvantage, which is a direct result of the untyped Web services, is the lack of a WSDL document that describes the structure of the expected messages, which prevents clients from automatically generating a proxy.

The WCF with WF4 implementation showed that the code-first approach drastically improves productivity because we only had to write a normal C# class with getters and setters to define the contract for the client application. The framework automatically generated the endpoint that exposes the WSDL document. Using this approach the developer is not aware of the XML in any way, and this is great for the productivity but not for the flexibility, because in case the structure of the messages changes the code has to be modified and recompiled.

So, what we can conclude from is that ultimately the decision hinges on the trade-off between ensuring interoperability and flexibility, or improving productivity.
Final Remarks

This chapter presents the contributions of our work and draws our main conclusions. Furthermore, it discusses which points require further investigations.

Section 7.1 presents the conclusions and the contributions of our work. Section 7.2 discusses future work.

7.1 Conclusions

The first goal of our research was to acquire general knowledge about SOAs and ESBs. We acquired this knowledge by perusing existing literature and used this knowledge to develop a capability model to categorize ESB capabilities and to evaluate ESB systems. The second goal was to provide an overview of the state-of-the-art commercially available .NET-based ESB solutions and to perform an evaluation to understand the benefits and drawbacks of these solutions.

We provided this overview by researching existing solutions and assessing their maturity. Based on this overview we identified Microsoft BizTalk Server and Neuron ESB as the two state-of-the-art commercially available .NET-based ESBs, and selected these two solutions for the evaluation.

In addition we identified WCF with WF4 as suitable building blocks for the development of a custom ESB and we also selected these techniques for the evaluation to stress the benefits of the commercially available solutions over that of a custom ESB implementation. Finally we evaluated the three selected solutions using the capability model and with a case study.

The evaluation of the capability showed that:

- BizTalk Server provides the most extensive capabilities support and was thus rated the highest.
- Neuron ESB provided the second best result because it supports fewer capabilities than BizTalk Server.
- WCF with WF4 provided, as expected, the least extensive capabilities and was thus rated the lowest.
The evaluation of the case study drew the opposite conclusion i.e., BizTalk Server was rated the lowest and WCF with WF4 was rated the highest.

The difference in rating between the case study and the capability evaluation is because the performance, installation and configuration efforts were only important for the case evaluation and not for the capability evaluation.

In a platform agnostic enterprise, Java-based ESBs might be a suitable choice, but in a .NET centric organization with a large budget that needs to solve many complex integration scenarios BizTalk Server or Neuron ESB might be a better choice.

BizTalk Server supports the most capabilities and is the only solution that provides an extensive user-friendly GUI to visually create XSTL transformation to transform messages. It is however the most expensive of the three evaluated solutions.

Neuron ESB supports less capabilities, but it is less expensive, requires less resources and has a lower learning curve.

The two solutions could even work together to provide the best of both solutions.

From this research we can conclude that commercially available ESBs like BizTalk Server and Neuron ESB provide support for many capabilities, which allows them to solve a wide range of complex, integration scenarios. However, the case study demonstrated that it is not always necessary to have a commercially available ESB to come up with a solution for an integration scenario. This is because their extensive capabilities support make them more complex which introduces more overhead for simple integration scenarios like the one in the case study.

So, in case a smaller budget is available, or just a subset of capabilities is needed, developing a custom ESB could be a suitable choice. We identified WCF and WF4 as two building blocks for the development of a custom ESB. However, although this concludes our research it can also be seen as the starting point for further research on "Building Custom ESB systems".

7.2 Future work

The capability evaluation showed that although WCF with WF4 support less capabilities than the commercially available ESBs nevertheless it still supports many capabilities. The problem is that these capabilities do not form a complete solution with a central management application, but they give a very good indication of the
capabilities which are available and which can be used to build a custom ESB.

A lot of work still has to be done to develop a custom ESB that provides the capabilities and that can be managed and monitored with a single management application.

In order to build a custom ESB the architectural principles for developing ESB systems should be researched, and based on this research a flexible architecture should be designed. In this research we have already identified two major components that should be part of this architecture.

The first component is a publish-subscribe engine.

In the WCF with WF4 implementation of the case study the message recipients were hard-coded in the workflow, but this is not very extensible and flexible. Based on our research of the existing .NET-based ESBs we noticed that most of the ESBs provide a publish-subscribe engine to increase the flexibility of sending messaging to multiple recipients. Publish-subscribe messaging is a far more scalable architecture than point-to-point messaging since message publishers only need to concern themselves with creating the original message, and can leave the task of servicing subscribers to the messaging infrastructure. Implementing a publish-subscribe engine is not a trivial task and there are many variations that should be researched when implementing a custom ESB.

The second component is an adapter framework.

An ESB should connect and coordinate the interaction of a significant number of diverse applications, but it is not known beforehand which applications should communicate with the ESB. So, the ESB may encounter systems with which it does not know how to communicate. To cope with this problem the ESB should provide a flexible adapter framework that allows new adapters to be added at runtime. An adapter should know how to communicate natively with the target system and also how to communicate with the ESB.

Further research should identify the techniques that could be used for this framework and define an architecture that allows the adapters to flexibly communicate with the publish-subscribe engine.

This should result in an architecture for a custom ESB system that will be able to flexibly facilitate the coordination and interaction of a significant number of diverse applications.
BIBLIOGRAPHY


