Master Thesis

Web Service Interoperability between J2EE and .NET

Author: Thomas Wermter
Student ID: 0310843

First supervisor: Prof. Dr. K. Reinhardt (FH Mannheim)
Second supervisor: Dr. G. Zörntlein (IBM)

WS 2003/04
Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst habe und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Mannheim, den 27.02.2004

Thomas Wermter

Q 3, 5
68161 Mannheim
E-Mail: wermter@gmx.net
Matrikel-Nr.: 0310843
Abstract

Interoperability between different platforms is the key promise of Web services. Yet, interoperability comprises many different aspects and therefore, a classification approach is presented to help organizing and structuring the concerning problem domains. Six views in the revised model (five in the first approach) show important areas regarding Web services: Protocol Stack, Programming Languages, Programming Concepts, Devices, Business Processes and Operational Views. Each view addresses defined regions which can be problematic in case different platforms need to work effectively together.

To show the correctness of this approach, proof-of-concept interoperability test cases have been performed for some of the views. The interoperability tests were executed in the two most common platform environments, J2EE and .NET, with the main stress put on the automated interoperability support of two chosen IDEs (WebSphere Application Developer and Visual Studio .NET). Notably, the word 'automated' is of special interest here because the need for manual effort should be minimized. Thus, achieving this automated interoperability with minimal effort requires a certain degree of usability in the IDEs which is addressed in a short excursus in this thesis.
Table of Contents

Erklärung ................................................................................................................................. II

Abstract ................................................................................................................................. III

Abbreviations ......................................................................................................................... VI

1 Introduction .......................................................................................................................... 1
   1.1 Motivation ....................................................................................................................... 1
   1.2 Problem Statement ......................................................................................................... 2
   1.3 Goal of the Thesis ........................................................................................................... 2
   1.4 Structure of the Thesis .................................................................................................. 4

2 Technical Fundamentals ....................................................................................................... 5
   2.1 Service Oriented Architecture (SOA) ........................................................................... 5
   2.2 RPC Technology ........................................................................................................... 7
   2.3 Web Services ................................................................................................................ 8
   2.4 The Evolutionary Way to Web Services ....................................................................... 12
   2.5 Interoperability ............................................................................................................ 14
   2.6 Web Service Standards ................................................................................................. 15
      2.6.1 SOAP ....................................................................................................................... 16
      2.6.2 WSDL .................................................................................................................... 19
      2.6.3 UDDI ...................................................................................................................... 22
      2.6.4 Collaboration of Web Service Standards ................................................................. 23
   2.7 Interoperability Profiles ................................................................................................. 23
      2.7.1 WS-I Basic Profile 1.0 ............................................................................................. 24
      2.7.2 WS-I Basic Profile 1.0 Testing Tools ....................................................................... 24
      2.7.3 Other Profiles ......................................................................................................... 24
   2.8 Security .......................................................................................................................... 25
      2.8.1 Declarative against Programmatic Security ............................................................ 26
      2.8.2 Certificates ............................................................................................................. 27
   2.9 Development Platforms and Environments .................................................................... 27
      2.9.1 J2EE ....................................................................................................................... 28
      2.9.2 .NET ..................................................................................................................... 29
      2.9.3 IBM WebSphere Application Developer .................................................................. 30
      2.9.4 Comparison Eclipse and WSAD .............................................................................. 31
      2.9.5 Microsoft Visual Studio .NET ................................................................................. 31
   2.10 Interoperability from a Developer's Point of View ....................................................... 31

3 Web Service Classification Approach .................................................................................. 33
   3.1 Classification Model ....................................................................................................... 33
   3.2 Addressed Views ........................................................................................................... 36
   3.3 Classification Model Revised ......................................................................................... 37

4 Interoperability Tests ......................................................................................................... 38
   4.1 Test Architecture ........................................................................................................... 38
   4.2 SOAP Communication and Encoding Styles ................................................................. 39
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3</td>
<td>Web Service Examples</td>
<td>40</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Simple Web Service</td>
<td>40</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Object Model Transformations</td>
<td>44</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Security Example</td>
<td>47</td>
</tr>
<tr>
<td>4.4</td>
<td>WS-I Testing Tools Applied</td>
<td>49</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Monitor</td>
<td>50</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Analyzer</td>
<td>51</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Conclusion</td>
<td>51</td>
</tr>
<tr>
<td>4.5</td>
<td>Web Service Semantics</td>
<td>52</td>
</tr>
<tr>
<td>5</td>
<td>Usability</td>
<td>53</td>
</tr>
<tr>
<td>5.1</td>
<td>Usability and the Classification Model</td>
<td>53</td>
</tr>
<tr>
<td>5.2</td>
<td>Concept Support Comparison</td>
<td>54</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Asynchronous and Synchronous Concepts</td>
<td>54</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Transactional Concepts</td>
<td>55</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Business Process Concepts</td>
<td>57</td>
</tr>
<tr>
<td>5.3</td>
<td>Visualization</td>
<td>57</td>
</tr>
<tr>
<td>5.4</td>
<td>Integrated Help</td>
<td>57</td>
</tr>
<tr>
<td>6</td>
<td>Summary</td>
<td>59</td>
</tr>
<tr>
<td>Appendix A</td>
<td>.NET Framework Data Types</td>
<td>61</td>
</tr>
<tr>
<td>Appendix B</td>
<td>CD Content</td>
<td>63</td>
</tr>
<tr>
<td>Bibliography</td>
<td></td>
<td>64</td>
</tr>
<tr>
<td>Alphabetical Index</td>
<td></td>
<td>67</td>
</tr>
</tbody>
</table>
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARPA</td>
<td>Advanced Research Project Agency</td>
</tr>
<tr>
<td>ATIS</td>
<td>Alliance for Telecommunications Industry Solutions</td>
</tr>
<tr>
<td>CICS</td>
<td>Customer Information Control System</td>
</tr>
<tr>
<td>CLR</td>
<td>Common Language Runtime (.NET)</td>
</tr>
<tr>
<td>COM</td>
<td>Component Object Model</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>DII</td>
<td>Dynamic Invocation Interface</td>
</tr>
<tr>
<td>EAI</td>
<td>Enterprise Application Integration</td>
</tr>
<tr>
<td>FCL</td>
<td>Framework Class Library (.NET)</td>
</tr>
<tr>
<td>GIF</td>
<td>Graphics Interchange Format</td>
</tr>
<tr>
<td>IDL</td>
<td>Interface Description Language</td>
</tr>
<tr>
<td>J2EE</td>
<td>Java 2 Enterprise Edition</td>
</tr>
<tr>
<td>JAX-RPC</td>
<td>Java API for XML-Based RPC</td>
</tr>
<tr>
<td>JNDI</td>
<td>Java Naming and Directory Interface</td>
</tr>
<tr>
<td>JSR</td>
<td>Java Specification Request</td>
</tr>
<tr>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>MS</td>
<td>Microsoft</td>
</tr>
<tr>
<td>OASIS</td>
<td>Organization for the Advancement of Structured Information Standards</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>SEI</td>
<td>Service Endpoint Interface</td>
</tr>
<tr>
<td>SO</td>
<td>Service Orientation</td>
</tr>
<tr>
<td>SOA</td>
<td>Service Oriented Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>SOAP! (formerly Simple Object Access Protocol)</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description, Discovery and Integration</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>WS-I</td>
<td>Web Services Interoperability Organization</td>
</tr>
<tr>
<td>WSIF</td>
<td>Web Service Invocation Framework</td>
</tr>
<tr>
<td>WSIL</td>
<td>Web Services Inspection Language</td>
</tr>
<tr>
<td>WSXL</td>
<td>Web Services Experience Language</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
</tbody>
</table>
1 Introduction

The introduction focuses on motivation of the thesis and provides a description of the problem domain which is addressed. Furthermore, the goal of the thesis and a plan how to achieve this goal are presented.

1.1 Motivation

With the increasing importance of integrated applications in enterprises during the last decade the need for exchanging information between computer systems also steadily intensified. First, this exchange focused on in-house integration, then dedicated communication lines were leveraged to connect branches and companies. The advent of the public Internet provided an infrastructure for cheap and comprising company-spanned information exchange.

Over time a lot of technologies for distributed computing have been introduced, yet, most of them have been invented and established by single companies. Early solutions were designed or conceived to connect the same type of systems and thus lacked the support for heterogeneous environments. The introduction of TCP/IP was a breakthrough for communication between heterogeneous systems. A first approach for realizing interoperability at a higher communication level was CORBA. In my opinion, one reason why CORBA did not become the standard computing model for distributed systems was that it addressed interoperability at a very late stage. Consequently, other higher-level Internet protocols, like HTTP, in combination with HTML and XML have been explored for their potentials as general distributed computing model. This finally led to the standards package commonly known as Web services.

According to a Forrester Research report in September 2003 [WWW21], eighty-five percent of the surveyed companies planned to use Web services predominantly for internal projects. However, approximately fifty percent of those companies also intend to use Web services to interact with their customers or partners. Internet companies like Amazon, Google and Ebay already provide access to their businesses over Web services. Companies believe in Web services' ability to be a cost-cutting technology for the integration of IT systems and to ease the task of exposing functionality to business partners for automating processing-tasks. This indicates that Web services will gain importance in the next years to come. Yet, for being successful one property is essential: interoperability between Web service adopters.
Chapter 1  Introduction

1.2 Problem Statement

Web services promise to be a platform independent and easy-to-use technology which is based on open standards and therefore Web services get industry-wide support. However, the technology is still in an early stage and quite a few specifications are still drafts or under revision. The interoperability of systems based on those standards is key to the success of Web services. However, ensuring interoperability at the communications protocol level is not enough. It is also important that the development of interoperable Web service based systems is also supported. This means that a developer should have tool support for interoperable Web service development minimizing the development tasks that are not business related. Generally, it is the task of Integrated Development Environments (IDEs) to support developers in their programming tasks by automating of recurring jobs, as well as hiding dispensable tasks that are not immediately related to the business logic. Therefore, interoperability has to be addressed already in IDEs by enforcing programming styles or rules that ensure interoperability at run-time environments. This interoperability aspect (support via IDE) will be addressed throughout this thesis.

1.3 Goal of the Thesis

The main topics of this thesis are:

- Develop a Web service classification approach supporting the identification of interoperability issues.
- Collect evidence that the classification approach is comprising.
- Define and perform test examples and consider their automated development.
- Determine the user-friendliness of IDEs with the respect to various concepts derived from the classification.

The reason for the first activity is to identify possible Web service interoperability problems at a generic abstract level. The classification approach arranges interoperability problems around Web services in a way independent from created specifications. Nevertheless, specifications can still be linked to the different layers of the classification but the specifications can also concern more than one of the layers.
Activity two aims at collecting evidence that the classification approach identifies the main interoperability issues. With a limited time frame only certain areas are picked out exemplary, which should satisfyingly prove the correctness of the model.

Web service interoperability is achievable in most cases if business partners agree on basic concepts. Yet, when this is not the case by manufacturers of development tools, agreeing on such concepts and best practices can be a tedious and expensive task for developers especially when the needed collaboration of components is crossing the company's boundaries. Perfectly, specifications would be unambiguous and would not leave much choices on how to implement a particular standard, but as this scenario is unlikely, mostly choices exist and they are taken by companies which are developing tools and IDEs. Therefore, to profit from an interoperability which is easy to achieve two solutions can be brought out:

- Clarify already available specifications
- Leave decisions to the producers of IDEs which options should be taken to achieve interoperability

The clarification process is already happening and it is a continuous process, as the example of the work done by the WS-Interoperability organization shows: their results influence the new versions of specifications. However, this problem is not part of this work.

Whereas the first option is obviously the preferable one, IDE manufacturers have to provide support for standards as soon as possible to stand out against competitors. This implies that they cannot wait until specifications get mature and that allows them to choose the second option for achieving interoperability: take decisions on ambiguous formulations and make them part of their integrated development environments. They are the developer's tools for an automated production of interoperable software. Whether the use of up-to-date IDEs allow a seamless\(^1\) development process when they have to provide or use services of the other technology is part of this research work. This question becomes more and more important because interoperability with Web services is the basic requirement and is increasingly used in real-world applications. Therefore, interoperability problems are mentioned when they occur but the main question for the tests is:

---

\(^1\) and therefore satisfyingly development which is fundamental demand as the definition of interoperability will show
how comfortable is the automated interoperability support when developing services or clients for a different platform technology, and in which ways does an IDE provide help for this?'.

This means automation and usability of IDEs are important parts of this thesis, as otherwise interoperability could only be accomplished by knowing all details from the Web service standards and development of clients and services would performed in a text editor. And that is probably not the support developers wish to experience.

1.4 Structure of the Thesis

After explaining the goals this section concentrates on the way how to achieve them. Therefore, it provides an overview of the chapters and allocates the aims to them. The central theme throughout the work is automation connected to the overall target of interoperability, which leads to tasks usually handled by IDEs or additional tools.

First, chapter 2 provides the essential knowledge about Web services, in particular important specifications but also their greater context with Service Orientation. The definition of interoperability clarifies what to expect from an automated support from IDEs. As the interoperability tests are limited to the two major platforms, J2EE and.NET, only those are briefly explained.

Chapter 3 treats the topic of the mentioned classification approach, declares the model started with and also announces a revised model which integrates changes that have been proven as useful during the work with the model. Additionally, explanations which parts of the model get treated in this thesis and which not are given.

Chapter 4 Interoperability tests, and 5 Usability collect on one hand the evidence for the correctness of the classification but on the other hand also deal with the last two remaining goals as their names indicate. Three selected proof-of-concept interoperability tests are presented following the pattern 'objective, method and result'. Finally, sections about the interoperability testing tools and the semantic behavior of Web services between different platforms close chapter 4. Chapter 5 picks out another part of the model and discusses the IDE’s automated support for different concepts.
2 Technical Fundamentals

This chapter discusses the core definitions and technologies essential for this work in terms of comprehension and agreement on non-ambiguous descriptions. Hence, keywords like Service Oriented Architecture, Web services and interoperability have to be specified as well as specifications in the Web service environment need to be explained.

2.1 Service Oriented Architecture (SOA)

When writing about Web services, sooner or later the terms Service Orientation (SO) and Service Oriented Architecture (SOA) are mentioned. Service Orientation means to promote the efforts of loosely coupled software units which provide registered services other units can dynamically bind to. The goal is to enhance the reuse of already available source code instead of developing new functionality and therefore increase, essential for every management decision, the Return Of Investment (ROI) because an application's lifetime can be extended and complexity reduced.

A Service Oriented Architecture is a way to build up and organize complete IT infrastructures. It is a paradigm shift from object-oriented systems to message-oriented systems which influences the style, development and integration of applications. Although messages were already passed around between objects in object-oriented systems, SOA messages are a technology neutral way of communication and they are explicitly detached from local or distributed calls. The W3C Glossary [WWW13] defines a Service Orientated Architecture as

“a set of components which can be invoked, and whose interface descriptions can be published and discovered.”
This definition includes the main requirements for SOA:

- existence of independent services or modules (the definition above uses the overloaded term components)
- services are invokable through platform independent interface descriptions
- interface descriptions can be dynamically located and invoked (entered and found in registries)

These definitions build the basic concept of a SOA and are depicted in Figure 1. After the creation of a Service, a description of it can be published (in the figure labeled as step 1) into a Directory where it can be found (step 2) by a Requestor. This Requestor now reads the Service's description and this way it gets the information about the location and how to bind (step 3) to the Service. As a next step the Requestor can start to use the Service. Note that there are sometimes different names used for the three mentioned entities:

- Requestor: (Service) Consumer or Client
- Service: (Service) Provider
- Directory: (Service) Registry

But SOA is more than the collection of the three components Service, Requestor and Directory, it is also about a proper design and architecture of collaborating services and it states that the granularity of a service should be more on a business process level than wrapping low-level methods only. A report from CBDi Forum “SOA – Save Our Assets”[WWW20] defines SOA as:

“...The policies, practices and frameworks that enable application functionality to be provided and consumed as a set of services published at a granularity relevant to the service consumer, which are abstracted away from the implementation using a single, standards based form of interface.”

This definition names essential artifacts of Service Orientation. Not only a technical framework is involved but also best practices and policies which show that Service Orientation is more like an engineering process than a pure, technical way of providing services. This is also the reason why a delimitation of the words Service Orientation and Service Oriented Architecture should be taken. The word 'architecture' is usually used for the design of a technical system in the computer science lingo and is therefore too weak to name a development process. Hence, the word Service
Orientation should be used in a process oriented context, whereas Service Oriented Architecture should be applied when referring to the actual design of a software system which implements the service oriented approach.

The CBDi definition integrates the mentioned SO concepts into a single sentence and is therefore more specific than the W3C definition. It also explicitly states the abstraction of any implementation related information as this could affect the interoperability between Service and Requestor. The reference to an interface which is based on standards and why this is important is discussed in the interoperability chapter (2.5, page 14).

### 2.2 RPC Technology

A Remote Procedure Call is a way of distributed computing and is commonly used to realize Client-Server infrastructures. Basically, RPC allows a program running on one host (client) to cause the execution of code on another host (server). Figure 2 below presents the general functionality provided by RPC technologies. A so-called stub is a layer which knows how to package a request and how to transport the call (RPC protocol) to the server. The server-side stub is sometimes also called skeleton and responsible for accepting requests, execute the appropriate code and to return the result to the client. In the optimal case, for developers this communication happens completely transparent, which means a local method call does not differ from a Remote Procedure Call at all. To be able to create the stubs automatically a description of the Remote Procedure Call Interface is necessary which can be formulated in an Interface Description Language (IDL).

![Figure 2 General functioning of RPC](image)
The general order of events when creating RPC is like this:

2. Usage of an special compiler for creating the Server and Client-stubs.
3. Usage of the methods/procedures during development like local calls.
4. Compiling of the Server and Client applications against their stubs.

Some examples for RPC technologies are:

- The Open Group's Distributed Computing Environment (DCE)
- SUN's RPC
- Microsoft's Distributed Component Object Model (DCOM)
- Object Management Group's Common Object Request Broker Architecture (CORBA)
- Java Remote Method Invocation (RMI)
- Distributed Smalltalk

### 2.3 Web Services

Web services can be understood in two totally different ways. In a broader sense, every service available in the Web or even the Internet\(^2\) can be referred to as a Web service. Examples for this are (web-based) email, video on demand, Voice over IP (VoIP), search engines, online shops and auction markets. The unique feature of this kind of Web services is the interaction with human persons as the services are created for them, for example HTML – a machine would not care about colors, font size or animated GIFs. A person, on the other hand, does not need a semantic description of an email service in a way a machine does because he or she either just knows the meaning of it or needs a textual description about significance and handling. Nevertheless, there have been approaches to to make HTML also understandable to computers: applications parse the HTML source and try to extract the relevant information. An example for this are the techniques of meta search engines that take the input once from the user and forward the request to

---

\(^2\) Today the terms Web and Internet are generally used to describe the same thing. But to be precise the Web (originally World Wide Web) is based on the protocol http whereas the Internet also comprises other protocols, like SMTP, FTP or whatever applications (and therefore protocols) are used after the connection to an Internet Service Provider (ISP) is established.
several regular search engines. This meta search engines are heavily dependent on a stable HTML interface to the other search engines because as soon as a minor change occurs in that interface, perhaps invisible to a human user, the meta search engine needs to be manually adapted to that change. This problem lead to another kind of Web services that are automatically processable by computers and is explained next.

In a more tighter sense, a Web service is a technical term with a completely different target group. It is about the automated communication between computers, or to be precise between applications, and as a machine needs to get semantic information about a Web service which is normally obvious to a person, communication follows another style based on different languages – SOAP\(^3\) for computers instead of HTML for humans. Nevertheless, both rely on the same underlying transfer protocol HTTP\(^4\). In the following, Web services are solely used in this second, tighter sense.

Web services received and still receive a lot of public attention. Unfortunately, this implies some misunderstanding and superficial knowledge about what Web services are, what they can do and most importantly, where they are inappropriate. A very good in-depth discussion about this can be found in [ZIM03]. Here, a look at the definitions of Web services by other people helps to define the context in which Web services are used in this work. The definitions start with the easiest (informal) to more complex (formal) ones (additional definitions, partly in German, can be found in [WWW12]):

1. One of the easiest understandable explanations of Web services is mentioned in [WWW05]:

   “Probably the easiest way to think of Web services are as pieces of code on a remote host that you can access.”

2. Definition of Web services by [KRE01]:

   “A Web service is an interface that describes a collection of operations that are network-accessible through standardized XML messaging.”

3. Definition by [CER02]:

   “A web service is any service that is available over the Internet, uses a standardized XML messaging system, and is not tied to any one operating system or programming language.”

\(^3\) SOAP is explained in chapter 2.6.1 on page 16

\(^4\) this is not completely true as SOAP can use other transfer protocols but mostly HTTP is in action
4. The organization W3C claimed to have a “shared essence of the term” on November 14th, 2002 [WWW11]:

“A Web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols.”

5. At the time of writing the W3C’s current definition from August 8th, 2003 [WWW13]:

“There are many things that might be called 'Web services' in the world at large. However, for the purpose of this Working Group and this architecture, and without prejudice toward other definitions, we will use the following definition:

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.”

The first definition is a very general one and is mainly mentioned in order to have a starting point for the delimitations of areas, because this description would also be appropriate for Web services' predecessors Java RMI, CORBA, Distributed Smalltalk or MS DCOM. The problem of these RPC-based technologies is their orientation on proprietary 'standards' (DCOM), respectively their limitation in platform (also DCOM) or language dependency (Java, Smalltalk). The CORBA architecture fulfills most of the former requests but is rather complicated and tunneling the IIOP calls through a firewall is even more complicated, in case the calls are not blocked by default. Each of those technologies had and still has its right to be used in a certain area, for example when it is clear that a system and its environment will not make a technology shift. And even in such a case Web services can be leveraged as a 'Wrapper-technology' to gain later needed interoperability. Web services can be seen as the latest approach of an RPC technology with the aim of replacing the former mentioned ones with the help of open and proven Internet technologies.

5 The CORBA 3.0 specification comprises over 1100 pages which can serve as a complexity-indicator.
Therefore, Web services are based on open standards specified by organizations like W3C and OASIS and because of this openness and the way they are specified, Web services are not bound to a specific platform or language. One could also say they help to further the goal of the object-oriented programming paradigm to build loosely coupled objects or components. Web services push this aim a step further to build (or integrate) loosely coupled systems – the SOA principle. Also, there are rarely problems with firewalls as Web services mostly communicate over the HTTP standard, but this could change with new firewall products when security standards get impaired.

Figure 3 depicts the formerly introduced Service Oriented Architecture figure in the terminology of Web services. The Web service's description is expressed in the XML-based Web Service Description Language (WSDL) – compare definition 4 and 5. This WSDL file gets published in the Universal Description, Discovery and Integration (UDDI) registry (step 1) where it can be found by a Web service consumer (step 2). From the WSDL file the consumer understands the description of the Web service and is therefore able to use the service provided. The communication between Web service and its consumer usually leverage the RPC protocol SOAP (definition 5) but other protocols are also possible, for example XML-RPC. Notably, the existence of an UDDI-Registry is optional if the Web service consumer already has the WSDL file or knows where to get it.

---

6 There are already products available which are able to understand SOAP.
With their independence Web services force themselves on business processes which surpass company boundaries or integration of (legacy) IT systems into an existing infrastructures\(^7\). Although it is also possible to build Web service systems for a Human-To-Application (H2A) communication, biggest benefits are gained for Application-To-Application (A2A) communication. For a detailed view on those terms see 'Chapter 1 – The Business Perspective' in [ZIM03].

The Web service context which is used in this work follows the fifth definition (the latest from W3C) as it is based on best practices, like the mentioning of SOAP illustrates. Although SOAP is not the only RPC protocol that can be used (XML-RPC would be a simple alternative), it is the one most commonly used – the same applies to the transport protocol HTTP. Especially important is the reference to “machine-to-machine interaction” which is equivalent to A2A just learned before. This kind of communication implies an automatism where machines can handle certain processes by themselves without a person's intervention.

The question how Web services and the Service Oriented Architecture are related can be compared to a class – object/instance relationship. The Web services technology is an implementation of the Service Oriented Architecture which realizes the concepts of SOA. Next, the details of the Web service and SOA relationship are explained in detail.

### 2.4 The Evolutionary Way to Web Services

This chapter builds the bridge between SOA and Web services, and it also presents the origins of the SOA approach. In the age of the collaboration of machines two different ways of communication developed.

The first and older one arose out of economic reasons in large corporations or banks because of the need for sharing information about accounts, products or production plans. They had (or still have) those different systems for various tasks with the necessity for communication and, as a result, such two systems have been welded together by the IT department. This tight-coupling process had to be repeated for every system pair which needed to share data, thus, the process required enormous amounts of efforts. As soon as the data exchange was not only limited to the company internal software systems but also the communication between enterprises got a requirement, the tight-coupled approach proved to be an obstacle. Note, that the usage of the terms 'tight-coupled' and 'loosely-coupled' are only used in the meaning of coupled software units in applications and are not related to hardware related couplings in this context.

---

\(^7\) Also called Enterprise Application Integration (EAI).
The second way of collaborating computers occurred with the advent of the Internet after it had arisen from its military-based predecessor ARPAnet. The Internet technology follows the principle of autonomous computer networks which are enabled to communicate and forward data to the destination network. These distributed networks use standardized protocols, for example TCP/IP as one of the oldest, which makes it possible to connect further networks to it in an easy and inexpensive way. Especially the redundant layout of connections between the networks guarantees the proper functioning in case of a single network breakdown or its link. With the growth of this loosely-coupled Internet the concept of openly discussed and agreed standards has proven to be successful in terms of interoperability.

Both technologies, tight-coupled Intranet and loosely-coupled Internet technologies, induced a concept of Service Orientation which is explained in chapter 2.1. One implementation of a Service Oriented Architecture is the Web service technology as depicted in Figure 4. If Web services is one possible implementation it is a justifiable question to ask for other implementations of the Service Oriented Architecture. However, currently there is only the Web service implementation available and to understand why, it is important to recall the origins of SOA which are the loosely-coupled Internet and the tight-coupled Intranet technologies. Those roots did not only induce the SOA principle but also influenced the development of its implementation, Web services. To answer the question for an alternative implementation it must be thought of possible
technologies which could take influence on it. In other words, when answering the following questionnaire and after the fourth question Web services are still not an option, one should think about proposing a new SOA implementation:

1. Are there IT problems that have not yet been addressed by the current Web services implementation?

2. Could SOA be a better solution to that kind of problems than already available ones?

3. Has the problem area already been addressed by Web services?

4. Could an additional Web service standard solve the problem?

5. Take the pre-SOA solutions and let them influence the new SOA implementation.

### 2.5 Interoperability

The following definitions by ATIS [WWW01] are supposed to clarify the term *interoperability*.

1. The ability of systems, units or forces to provide services to and accept services from other systems, units, or forces and to use the services so exchanged to enable them to operate effectively together.

2. The condition achieved among communications-electronics systems or items of communications-electronics equipment when information or services can be exchanged directly and satisfactorily between them and/or their users. The degree of interoperability should be defined when referring to specific cases.

3. Allows applications executing on separate hardware platforms, or in multi-processing environments on the same platform, to share data and cooperate in processing it through communications mechanisms such as remote procedure calls, transparent file access, *etc.*

4. The ability of a set of modeling and simulation to provide services to and accept services from other modeling and simulation, and to use the services for exchange enabling them to operate effectively together.

5. The capability to provide useful and cost-effective interchange of electronic data among, *e.g.*, different signal formats, transmission media, applications, industries, or performance levels.
The definitions show that interoperability can have different interpretations as the definitions are directed towards varying environments, like hardware and software but also with respect to telecoms industry, et cetera. As definition 5 commits only the wire format for cost-effective data interchange as interoperability, definition 3 talks about cooperation and sharing data. This means interoperability is not limited to a specified wire format, it also implies cooperation on higher level tiers\(^8\).

The common ground of all definitions is the phrase “effectively or satisfactorily cooperation and data exchange,” which is either hard or software independent. However, the words effectively and satisfactorily can be a bit fuzzy when trying to get defined and reproducible results. Therefore, it is important to clarify what “satisfying data exchange” means in the context of this work. When nowadays talking about the word “satisfying” it urges to regard the effect of usability. If an automatic creation of software artifacts reduces the amount of time and effort to gain interoperable software this would be satisfying enough for the purposes of this work. An IDE is for example such a tool that is able to support developers with an automatic creation of interface descriptions or source code templates. However, if IDEs rely on proprietary standards, no matter how good they behave in the supposed environment, without telling the developer that the usage will result in conversation problems would not be satisfying enough.

In the context of this work interoperability is defined as:

- ability of software systems to exchange and process (understand) data
- connection of platforms over open protocols is achievable satisfactorily and cost-effective, which means that communication is possible without the need for programmatic data conversations by developers

### 2.6 Web Service Standards

The mass of standards concerning Web services does not make it easy to keep an overview. When talking about interoperability it is essential to talk about standards, otherwise there would be no interoperability at all. Why there are so many standards and why some even address the same problems is explained by a real-world example in the next paragraph.

---

\(^8\) which is an essential point for the model described in chapter 3.1, page 33
At first, the competition between the players in the Web services game and the standards development over time has to be considered. As always when competitors work in parallel on similar issues, they try to sell their solution as the best, even if it is a proprietary one. That is one of the reasons why Web services have awakened so much hope: they promise the interoperability customer's always wanted. To keep that promise, IBM and Microsoft initiated the founding of the Web Services Interoperability Organization (WS-I). So far the theory, but when the WS-I was founded in February 2002, SUN Microsystems was invited only a few days before as a regular member and not as a contributing one. Other companies had got their invitation weeks before that. As a reason for the late invitation by the founders it was mentioned that they thought SUN Microsystems would not be interested in a participation at all. The background: SUN worked on their own set of standards about some issues with Web services but also supported work on open standards. Finally SUN was allowed to join WS-I as a contributing member.

This story makes it plausible why sometimes several standards (no matter if proprietary or open) exist until one wins over another. In the following sections there will only be a short summary of the issues the particular standards address. Nonetheless, there are further references mentioned in the actual paragraphs.

### 2.6.1 SOAP

SOAP is a XML-based RPC wire protocol for exchanging information in a distributed environment. It is a language-independent method of depicting messages which are transported over an internet connection to a remote service. Therefore, SOAP leverages existing transport protocols, this is usually HTTP/S but also SMTP and others are possible. The language independency is achieved through SOAP-defined primitive data types which are mapped to language dependent data types. Restricting the usage to those primitive types ensures interoperability, whereas using proprietary data types, which are sometimes offered by SOAP tools, breaks it.
Chapter 2  Technical Fundamentals

An exemplary SOAP Request can be seen in the box SourceCode 1. The first five lines (01-05) do not belong to the SOAP message directly but they show how the message is embedded in the most commonly used binding, the HyperText Transfer Protocol (HTTP). The first two lines (01-02) identify the URI http://www.example.net/Shop/ to which the SOAP message gets delivered. The Content-Type field (line 04) must be set to application/soap+xml since SOAP 1.2 whereas the selection of a character set is optional. The last HTTP header line (05) must contain the correct length of the subsequent SOAP message.

The SOAP message from the example shows the order for an item including a description and price information. SOAP messages can be divided into two parts: an optional SOAP Header (08) and an obligatory SOAP Body (14) which are enclosed by the SOAP Envelope tag (07). Although said that SOAP Headers are optional, the attribute mustUnderstand (09) makes the processing of the header mandatory for the receiver or an SOAP intermediary lying in between. Headers are meant for carrying contextual information relevant for the processing of a message which have the property that such header blocks can be altered, deleted, or added by a header's processor. For the example above, one could think about three SOAP nodes (sender, intermediary, and receiver) whereas the intermediary reads the header block customer (09), performs a credit-worthy check for the customer recognized by the unique id (10). In case the
check is successful the message is forwarded to the receiver immediately because the customer priority (11) indicates the importance of the customer. If the credit-worthy check fails on the other hand, the intermediary could add an additional header block with an according information for the receiver.

The SOAP Body (14) contains only one block called PlaceOrder (15) which in turn includes all information (16-22) concerning the order the application on receiver's side needs for the order placement. The scenario used above is called 'Conversational Message Exchange' as a dialog could start between sender and receiver until additional information is exchanged, like shipping address or method of payment. It is the applications area of responsibility to use the information and call the correct methods. It is not possible to infer the method name from the name of the body block. This is the case for the other kind of possible message exchange – the Remote Procedure Calls. For a discussion about the two styles refer to chapter 4.2 on page 39.

Table 1 lists again the important terms and gives a short explanation of them. This can be used as a quick lookup reference.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Body</td>
<td>The body contains the main message. Mandatory.</td>
</tr>
<tr>
<td>Envelope</td>
<td>Top element of the XML document; contains a header and body and defines used XML namespaces.</td>
</tr>
<tr>
<td>Header</td>
<td>Contains additional information for the receiver, i.e. for authentication purposes, transactions or priority. Optional.</td>
</tr>
<tr>
<td>Intermediary</td>
<td>A node between a SOAP sender and receiver which is able to process SOAP headers.</td>
</tr>
<tr>
<td>Message Exchange</td>
<td>Describes the way how messages are being transferred between a SOAP sender and a SOAP receiver:</td>
</tr>
<tr>
<td></td>
<td>1. Conversational Message Exchange (document/literal)</td>
</tr>
<tr>
<td></td>
<td>2. Remote Procedure Calls (RPC/literal)</td>
</tr>
<tr>
<td>Protocol Binding</td>
<td>Specifies the underlying transport protocol which is used for the message transport. Examples are HTTP and SMTP.</td>
</tr>
<tr>
<td>SOAP Fault</td>
<td>A fault message is sent from the server to the client in case of an occurred exception during the processing of a request</td>
</tr>
<tr>
<td>SOAP Receiver</td>
<td>The node performing a SOAP request.</td>
</tr>
<tr>
<td>SOAP Request</td>
<td>A request message is sent from the client to the server.</td>
</tr>
</tbody>
</table>

9 dialog means here the exchange of several messages between the nodes
<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAP Response</td>
<td>A response message is sent from the server to the client in case of a successful processing of a request.</td>
</tr>
<tr>
<td>SOAP Sender</td>
<td>The node answering with a SOAP response to a SOAP request.</td>
</tr>
</tbody>
</table>

Table 1 SOAP keywords

### 2.6.2 WSDL

The Web Service Description Language (WSDL) is used to describe interfaces of Web services. It contains information about available operations, the network endpoint and a protocol binding for the transport. A WSDL description consists of a service interface and a service implementation. Such a description can be published in registries like UDDI for enabling clients to find the service.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definition name="shopOrderService" targetNamespace="http://example.net/ns/shop/wsl"
xmlns:tns="http://example.net/ns/shop/wsl"
xmlns:SOAP-EXT="http://schemas.xmlsoap.org/wsdl/soap/
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <schema targetNamespace="http://example.net/ns/shop/wsl"
      xmlns="http://www.w3.org/2001/XMLSchema">
      <complexType name="PlaceOrder">
        <sequence>
          <element name="ManufacturerID" type="xsd:int">
          <element name="ItemNumber" type="xsd:int">
          <element name="ItemName" type="xsd:string">
          <element name="ItemDesc" type="xsd:string">
          <element name="OrderQuantity" type="xsd:int">
          <element name="WholeSalePrice" type="xsd:float">
          <element name="OrderDateTime" type="xsd:dateTime">
        </sequence>
      </complexType>
    </schema>
  </types>
  <message name="PlaceClientOrderIn">
    <part name="request" element="tns:PlaceOrder"/>
  </message>
  <message name="PlaceClientOrderOut">
    <part name="response" element="xsd:boolean"/>
  </message>
  <portType name="ShopOrderInterface">
    <operation name="getOrder">
      <input message="tns:PlaceClientOrderIn"/>
      <output message="tns:PlaceClientOrderOut"/>
    </operation>
  </portType>
<!-- End of WSDL interface description -->
```
The WSDL file depicted in SourceCode 2 is the corresponding service description to the SOAP message presented in SourceCode 1. Note, that elements, which are referencing each other, are marked in the same emphasized style: bold, italic, underlined and combinations of them. A WSDL file’s root tag is called <definition> (line 02) and can contain the child elements

- <types> (03-17)
- <message> (18-20 and 21-23)
- <portType> (24-29)
- <binding> (32-43)
- <service> (44-49).

The <types> element comprises the abstract data types that are used in SOAP messages. For an element attribute type description, here, the XML schema definition (xsd) is leveraged. The elements (07-13) are wrapped by a <ComplexType> tag to express that the elements belong to an unit, comparable to an object.

A <message> element defines data passed during one interaction between the Web service and its client. The first message (18-20) specifies the expected data the client has to deliver to the Web service. Thus, the <part> attribute element references for the request the formerly described complexType PlaceOrder. The other message included in the WSDL file (21-23) is the response from the service to the client with the simple schema type boolean (22).
The names of the messages, \texttt{PlaceClientOrderIn} and \texttt{PlaceClientOrderOut}, are referenced (26+27) inside the \texttt{<portType>}. The \texttt{portType} element (24-29) defines the actual service methods which are available to clients. Such methods are called \texttt{operations} in WSDL and bind a request (\texttt{input} – 26) and response (\texttt{output} – 27) message together. It is also possible to add a \texttt{<fault>} element to an \texttt{operation} but this is not depicted in the example. The task of such an element is the managing of the error handling.

So far, the explained elements are known as the WSDL interface description because the service's interface is fully specified. The following two elements belong the part called WSDL interface implementation because the client still needs information about the network access and the protocol.

The \texttt{<binding>} is responsible for mapping a concrete protocol implementation to the formerly explained \texttt{<portType>}. Therefore, the \texttt{port type} (32), as well as the \texttt{operation} (34) name must be referenced in the \texttt{binding} element. The structure of the binding operation (34-42) and the port type operation (25-28) resemble each other whereas the binding contains additional extensibility elements (33, 35, 37, 40). These elements belong to a different namespace, indicated by \texttt{SOAP-EXT} in the example, and show a binding to the \texttt{SOAP} protocol. Furthermore, the SOAP communication (document – 33) and encoding style\textsuperscript{10} (literal – 37, 40) are fixed, as well as the transport protocol (HTTP – 33) is specified. According to the WS-I Basic Profile HTTP/S is the only allowed protocol for the SOAP binding.

The last element, the \texttt{<service>} tag, adds the last piece of information needed for a complete description of the service: the \texttt{address location} where the service can be found. Thus, an element named \texttt{<port>} alludes to the binding (46) and the SOAP extensibility element \texttt{address} (47) defines the location which is an URI because the chosen transport protocol was HTTP (33).

Further information about WSDL can be found in [ZIM03] or [WWW02]. The following list enumerates again the most important terms and gives a short explanation of them. The third column states the number of how often the element is allowed to occur in a WSDL document.

\textsuperscript{10} see chapter 4.2 SOAP Communication and Encoding Styles on page 39 for more information.
<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bindings</td>
<td>The binding element specifies how each operation is sent; usually SOAP.</td>
<td>0, 1, *</td>
</tr>
<tr>
<td>Message</td>
<td>A message is either a request or a response and consists of none, one or more <code>&lt;port&gt;</code> elements.</td>
<td>0, 1, *</td>
</tr>
<tr>
<td>Operations</td>
<td>Included in a port type, operations define the syntax for calling their methods and contain input/output messages.</td>
<td>0, 1, *</td>
</tr>
<tr>
<td>Part</td>
<td>A part corresponds to a parameter.</td>
<td>0, 1, *</td>
</tr>
<tr>
<td>Port</td>
<td>A port specifies a service and must have a unique <code>&lt;binding&gt;</code> element. The WSDL 2.0 specification renamed port to endpoint.</td>
<td>0, 1, *</td>
</tr>
<tr>
<td>Port Type</td>
<td>A port type defines operations. This information is used when using a WSDL compiler for generating Stubs and Service Interfaces. One client stub will be generated for each port type defined. In the WSDL specification 2.0 the name port type was changed to interface.</td>
<td>0, 1, *</td>
</tr>
<tr>
<td>Service</td>
<td>A service is a collection of <code>&lt;port&gt;</code> elements.</td>
<td>0, 1, *</td>
</tr>
<tr>
<td>Types</td>
<td>Types are used to define own (complex) data types.</td>
<td>0, 1</td>
</tr>
</tbody>
</table>

### 2.6.3 UDDI

UDDI (Universal Description, Discovery and Integration) is the centralized *discovery agency* implementation for the following tasks:

- **Publish**: *description* of a business and its Web services (includes publishing information to the registry) – white pages
- **Find**: *discovery* of Web services – yellow pages
- **Bind**: *integration* of Web Services (based on technical interfaces published in a UDDI registry how to connect and interact with that Web service) – green pages

In contrast to the centralized approach like UDDI, there is also a distributed (file-based) approach where service providers make only their own Web service description available at their Web server. The Web Service Inspection Language (WSIL) tells how customers can *inspect* this website for a certain service. WSIL superseded Microsoft's own DISCO (Discovery) solution. As
UDDI is important but not used in this work this general overview is not further deepened. For more information about UDDI, a good start could be IBM's developerWorks article “Understanding UDDI” [WWW22].

### 2.6.4 Collaboration of Web Service Standards

As Web service standards are still striding ahead in a fast paced manner and the advent of new standards is certain, this paragraph tries to make out the current position and what to expect in the future. The evolvement of standards can be classified into different phases:

1. XML schema, SOAP, WSDL, UDDI, WS-I Basic Profile
3. BPEL, Systems Management, Orchestration, WS-Transaction, Provisioning
4. Development tools and enterprise applications which leverage the phases' results.

The different phases do not evolve sequentially but iteratively as well as parallel. This means, currently the security phase 2 and the enterprise phase 3 are maturing. At the same time new versions of the basic connection phase 1 are published. Applications can be based on different versions of the standards. The current position is therefore fuzzy but the first iteration is located around phase 3 whereas the second iteration has already started to reach phase 2.

Seen from the interoperability view, WS-I completed phase 1 and has to address phase 2 and 3 next. As tool support is currently missing for the second iteration11 there are rarely applications available for getting information about possible communication problems between services implementing different specification versions.

### 2.7 Interoperability Profiles

The Web Service-Interoperability (WS-I) organization produces so-called profiles as their deliverables. The goal of a profile is to name “groups of specifications at given versions levels with conventions about how they work together” [WWW24]. Furthermore, WS-I also provides usage scenarios, sample applications and test suites. In the following, their first products, the Basic Profile and according testing tools, are presented and other current profile drafts are mentioned.

---

11 In plain language SOAP 1.2, WSDL 1.2 and UDDI 3.0
2.7.1 WS-I Basic Profile 1.0

WS-I takes care of interoperability between a Web service provider and a requestor. This panel of company experts does not change existing standards but works to find some common ground on which to base. This recommendations are published in profiles. It seems that the already published Basic Profile\textsuperscript{12} gets acknowledged by companies and standards organizations and will be included in following standards.

The Basic Profile addresses platform and protocol problems, as well as problems between the Web service standards version. The main parts are about messaging (SOAP) and service description (WSDL). A small part is about the service publication and discovery (UDDI) but all they say about security is the permission for a service to require HTTPS. According to WS-I the broad field of Web service security will be addressed next.

2.7.2 WS-I Basic Profile 1.0 Testing Tools

The WS-I provides tools for testing the conformity to the Basic Profile. Those tools are available for Java and C#-Web services implementations. At the time of writing the current release is version 1.0 approval draft. It is important to know that performed tests do not completely ensure the interoperability but the WS-I says itself that “the tools are then an indicator of conformance of a Web service to the Basic Profile, based on the artifacts produced”, [WSI03] rather than a certification tool because of their non-intrusiveness. Chapter 4.4 on page 49 provides an example of how to apply the testing tools.

2.7.3 Other Profiles

The WS-I Basic Profile is just the start for interoperability profiles. Currently, WS-I works on the following profiles:

- Basic Profile 1.1
- Simple SOAP Binding Profile 1.0
- Attachments Profile 1.0
- Basic Security Profile

\textsuperscript{12} http://www.ws-i.org/
2.8 Security

Security standards for Web services are emerging. Standards are already published in final releases but wearing the interoperability hat, this is a complicated topic. It is the next big focus the WS-I members will work on but it will take time to agree on common parts of the standards which are worth the interoperability recommendation.

Security can be implemented either in the transport or application layer, whereas implementing security in the transport layer means using HTTPS for an end-to-end encryption. This approach can only be used until intermediaries come into operation. Intermediaries are hops between the service-provider and requestor which can alter SOAP messages, for example process certain SOAP headers. In such a scenario encryption on the transport layer is not sufficient. To secure an end-to-end communication it is possible (and recommended by the WS-I Basic Profile 1.0) to use HTTPS/SSL transport security. In case more sophisticated features are needed the WS-Security standard describes how to implement security on the application layer. Though, WS-Security does not define its own algorithms but makes use of established methods for encrypting\(^{13}\) SOAP messages and using digital signatures or certificates.

There are many other features addressed in the Web service security stack which are briefly explained in the following to be able to keep an overview [WWW18]:

- **WS-Security**
  describes how to use XML encryption and XML signatures in SOAP messages, as well as information about the security tokens which can be X.509 certificates or Kerberos tickets.

- **WS-Privacy**
  describes a model for how Web services and requestors state subject privacy preferences and organizational privacy practice statements.

- **WS-Trust**
  describes a framework for trust models that enables Web services to securely interoperate.

- **WS-Policy**
  describes the capabilities and constraints of the security (and other business) policies on intermediaries and endpoints (e.g. required security tokens, supported encryption algorithms, privacy rules).

\(^{13}\) for example XML encryption
• WS-SecureConversation
describes how to manage and authenticate message exchanges between parties including security context exchange and establishing and deriving session keys.

• WS-Federation
describes how to manage and broker the trust relationships in a heterogeneous federated environment including support for federated identities.

• WS-Authorization
describes how to manage authorization data and authorization policies.

2.8.1 Declarative against Programmatic Security

There are two ways of implementing the security standards. First, the declarative method which uses configuration files only for defining the security constraints. IBM chose this way, thus, means there is almost\(^\text{14}\) no need for taking care of the wished security features during the development. For the deployment of the application the security configuration is specified in one or more XML files. The application server reads this file and implements the requested features. The simplicity of changing the security configuration without the need of recompiling the whole project is an advantage of this solution. After a change of the XML configuration file the application server automatically recognizes the alteration or must be restarted, and then it applies the new security constraints. Furthermore, this solution makes it easier to provide a wizard which leads the user through the configuration process because it is easier to have an automatic build of an standardized XML file instead of the generation of source code.

The second way, programmatic security, provides an API for the developer who wants to implement security. After the enumeration of advantages for the declarative way, these are the drawbacks for programmatic security. Therefore, also Visual Studio does not only use this method but combines both ways even though it relies more on the use of an API than WebSphere. On the WebSphere side, on the other hand, are no WS-Security APIs available for handling security, it consequently implements the declarative solution.

\(^{14}\) Almost because it is always a good idea to think about security early.
2.8.2 Certificates

Certificates are the essential core when talking about asymmetric encryption in enterprises. Certificates prove the identity of persons or companies and can be used for encryption or signing documents. With certificates and Certificate Authorities (CA), trustworthy certificate issuers who confirm identities, it is possible to achieve the security goals integrity, authentication, confidentiality and non-repudiation.

The most commonly used certificates follow the X.509 standard. Such a certificate contains information about the owner, issuer (CA), validity and certificate version. Furthermore, the owner's public key, the issuer's signature and details about the used algorithms are included. It is also possible to create own certificates, WSAD ships with a tool called IKeyman (IBM Key Manager) which is not only able to create but also to request (from a CA), import and export certificates. The creation of certificates with Visual Studio on the other hand is not possible. It depends on certificates from the Windows operating system which only can be created when a Windows 2000 Enterprise certification authority is located in the domain the computer belongs to. For first tests it is also possible to use the included administrator certificate which is normally used for managing the Encrypting File System (EFS) or to create a self-signed certificate with IKeyman which then can be imported by Windows.

It is also possible to export the Windows certificates for the current user\(^{15}\). In case the private key is also selected for the export the option PKCS#12\(^{16}\) is available which produces a file with the ending \(*.pfx\). Changing it to \(*.p12\) enables the IBM's Key Manger tool to import the certificate and store it in a Java Keystore (JKS). This way, it is attainable to share the same certificate for the tests and prevent problems in this area. However, certificates are not the only ways to secure communication, it is also possible to use a symmetric encryption or Kerberos tickets, yet, those topics are not covered here as they are not used in the interoperability tests.

2.9 Development Platforms and Environments

The question if one development platform is better than another, namely J2EE and .NET, is almost impossible to answer today. It heavily depends on the developers existing knowledge and preferences. The question which platform to use for a certain project is also out of scope for this

---

\(^{15}\) Control Panel -> Users and Passwords -> Advanced -> Certificates

\(^{16}\) PKCS#12 specifies the “Personal Information Exchange Syntax Standard”
thesis. Asking one of those questions will generally end in verbal fights between the supporters of the actual party as many forums on the Internet illustrate. The choice is often a company-wide strategic decision and sometimes developers even do not have the right to a say.

However, the statement that there is a need for data exchange between different platforms and legacy systems is unchallenged. Therefore, the following chapters give a short technical description of the used technology platforms and IDEs without valuing one of them.

2.9.1 J2EE

The Java 2 Enterprise Edition (J2EE) is a platform that defines the infrastructure for Java-based distributed enterprise applications and leverages the capabilities of the Java 2 Standard Edition (J2SE). The Standard Edition comes with the Java Virtual Machine (JVM) which is provided for several operating systems and is able to execute a platform independent language, the Java Byte Code. Developers can now use the available object-oriented Java API and compiler to produce valid Java Byte Code. This approach makes it possible to follow the pattern “program once, run it everywhere” which aims at the operating system independence of Java programs. The J2EE specification builds up on J2SE and defines an application programming model which consists of four types of containers for application components:

- **Applet Container:**
  Applets are lightweight client components that are typically executed in a Web browser environment. The binary code is downloaded for every request but has restricted access to the local system resources. Communication to the server-side is only possible over HTTP/S to a Web Container.

- **Application Client Container:**
  Application clients are also called rich or fat client programs because of their functionality in contrast to an applet solution. Beside accessing the server-side Web Container over HTTP/S, rich clients are also able to communicate with the EJB Container or a database directly.

- **Web Container:**
  The Web Container includes the runtime environment for the Java Server Pages and Servlet technology and is one half of a J2EE application server. In large J2EE applications this component typically handles the presentation and control logic. The Web Container can access the EJB Container and databases.
- **Enterprise JavaBeans Container:**

  Enterprise JavaBeans are used to implement business logic and model business data and are located in the other half of a J2EE application server, the EJB Container. Like the Web Container they can also access possible databases.

  The applet and Application Client Container are generally located on the client side whereas the Web and EJB Container as well as the database are on the server-side.

2.9.2 **.NET**

The .NET framework basically consists of two different modules. First, the Common Language Runtime (CLR) and second, the Framework Class Library (FCL). They are explained briefly in the following paragraph.

The CLR is the execution unit for the Common Intermediate Language (CIL). Compared to the Java world the CLR would be the Java Virtual Machine (JVM) and CIL is the equivalent to Java Byte Code. The .NET compiler creates CIL instructions during the compilation process which in turn will be transformed into native machine code by a Just-In-Time (JIT) compiler during runtime. The CLR is also able to separate one single process into several application domains.
which means more than one executable application can run in one process context. The benefits are less memory consumption and less overhead for switching processes. Security issues should not arise because the managed modules cannot violate their application domain boundaries. Managed modules are all those based on the .NET framework. Unmanaged modules, on the other hand, are bypassing the .NET framework because they use outdated technologies, like MFC or COM, and are therefore not executed by the CLR (see Figure 6).

<table>
<thead>
<tr>
<th>Managed Code</th>
<th>Unmanaged Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIL</td>
<td></td>
</tr>
<tr>
<td>.Net Framework (CLR+FCL)</td>
<td></td>
</tr>
<tr>
<td>Operating System</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 6 .NET high level overview*

The FCL is the API to the .NET framework and replaces various older, programming language dependent APIs that gathered since the advent of the Windows operating system, for example Win32 API or the Microsoft Foundation Classes (MFC). Solely the fact of having only one, truly object-oriented API simplifies the programming process extremely. The FCL consists of different namespaces with the most important ones Microsoft and System. The Microsoft namespace contains all those services which are specific to the Microsoft platform (for example registry access), whereas the Service namespace includes the .NET data types and more general services (for example XML, threading or IO services).

**2.9.3 IBM WebSphere Application Developer**

The IBM tool WebSphere Application Developer was used for the interoperability tests on the Java side. It expands the open source tool Eclipse and provides a lot of additional plugins for enterprise development. The software configuration for the development was:

- WebSphere Studio Application Developer (Windows) Version 5.1.0

or

- Eclipse (Windows) 2.1.1

- IBM Web Service Development Kit V5.1 (WSDK 5.1)
2.9.4 Comparison Eclipse and WSAD

Most of the examples in this work can also be done with the open source IDE Eclipse. To get comfortable wizards for the Web services development as well IBM's Web Service Development Kit (WSDK) 5.1\textsuperscript{17} is needed.

New plugins are available in Eclipse after the installation. Additional to the wizards, an embedded version of IBM's WebSphere Application Server – Express V5.0.2 is provided which can be used for a test execution environment for the developed Web services. The installation of the WSDK 5.1 means that almost all features for Web service development are also available to the Eclipse community.

2.9.5 Microsoft Visual Studio .NET

The tool for the development of Web services for the .NET environment was Microsoft's Visual Studio .NET. The software configuration for .NET development was:

- Microsoft Visual Studio .NET 2003 Version 7.1.3088
- Microsoft .NET Framework 1.1 Version 1.1.4322
- Microsoft Web Service Enhancements (WSE) 1.0 SP1
- Microsoft Web Service Enhancements (WSE) 2.0 Beta

2.10 Interoperability from a Developer's Point of View

A developer who does not want to be bound to a specific development environment has the choice of creating a Web service in one IDE and importing the automatically created WSDL-files into another IDE to develop a Web services client. An article on how this works for WSAD 5 and Visual Studio .NET Professional 2002 can be found on IBM's Developer Works site, referenced in [WWW07].

If communication between a .NET client and a WSAD service (or the other way around) fails, the low-level SOAP calls have to be investigated to find the problem's origin. The Apache SOAP implementation comes with the proxy-tool TunnelGUI for intercepting those calls. Also

\textsuperscript{17} https://www6.software.ibm.com/dl/devworks/dw-wsdk-p?S_TACT=103AMW17&s_CMP=LPWSDK
the WS-I Basic Profile Testing Tools can be used\(^{18}\) as they act as an proxy for the SOAP calls, analyze them and can give indications of the problem's origin. Furthermore, WSAD and the MS SOAP Toolkit also provide ways to capture SOAP messages.

With the JAX-RPC API it is possible to develop J2SE clients which are also called *unmanaged clients*, whereas Web service clients developed with JSR 109 are called *J2EE container-managed clients*. JAX-RPC and JSR 109 are both part of the J2EE 1.4 specification. Those two kinds of clients differ in the way the code is written, packaged and invoked. For example a service lookup using JAX-RPC is performed by creating an API class `ServiceFactory` whereas JSR 109 conforming clients use a JNDI lookup.

Generally, there are two different methods of how to develop a Web services. First, the bottom-up approach where the functional code exists first and then gets exposed as a Web service through the automated creation of a WSDL file. The second method starts with the manual development of a WSDL file and then have tools generating stub code for service and clients. This is called top-down development and can especially be used if no existing systems have to be exposed.

---

\(^{18}\) see Chapter 4.4, page 49 for an basic example how to apply the WS-I testing tools
3 Web Service Classification Approach

There are different basic approaches to classify Web services but most of them are based on specifications level which means that the specifications are somehow grouped together. This aspect is included here as well as those documents can be assigned to (at least) one so-called View but the classification model also tries to enlarge its scope and comprises more general topics. With the help of this model the next chapters document interoperability tests which are based on certain Views of the model, but the tests should also prove the correctness of the model.

3.1 Classification Model

As the Web services world develops standards and recommendations in a fast-paced way, a classification model can help organizing the view on this world. Although this model was mainly created for the “Interoperability Question” its aim is more general. In this section the initial model will be presented, which had been developed before the actual work on interoperability began. Chapter 3.3 again discusses the revised model where improvements and experience from this work have made changes useful or necessary. Note that the following views were not developed on a specification level. That means, the content of certain specifications can overlap several views. Again, this model tries to classify Web services on a more abstract level than a categorization of specifications.

![Figure 7 Web service classification model with five views](image)

Figure 7 Web service classification model with five views
This approach consists of five *views* which are arranged in layers. This is depicted in Figure 7, and each view is explained in general, available specifications are mentioned and a section with regard to interoperability is professed. It starts with the lowest view in ascending order:

1. **Protocol Stack View**

   **General:**
   This low-level view contains specifications about the communication between service and requestor as well as solutions to find a specific service. This is one of two very essential views for interoperability.

   **Specifications:**
   WSDL, SOAP, UDDI, WS-Inspection, WSIF, WS-Attachments, WS-Addressing

   **Interoperability:**
   As this layer is the point where systems exchange data and communicate to find, call and respond interoperability is crucial. WS-I published their interoperability advices in their Basic Profile 1.0.

2. **Programming Languages View**

   **General:**
   Programming languages have different object models which must be mapped to a common wire format and back. Typically, this is the work for a wizard in an IDE. The developer selects the methods which should be exposed as services and the IDE automatically handles the binding from a programming language type to a XML SOAP type.

   **Specifications:**
   XML Schema, language object models

   **Interoperability:**
   Here, two solutions are possible. First, for the used programming language exists a defined mapping table to a language independent standard format which encompasses all object and value types. This means that the mapping table gets huge and languages, which have a lack of certain types, have to work around this problem. Second, a mapping table is defined for the most common types and guidelines for the different languages exist which the programmer can use to be aware of possible interoperability problems. Solution two is the preferable one as the first idea is good for developers taking care about their language only but regarding to interoperability such an approach can become a nightmare.
3. **Programming Concepts View**

   **General:**
   This view contains all concepts which can be available in any (object-oriented) programming language. The realization differs depending on the language but the concept behind the implementation stays the same. A simple example is the concept of an asynchronous communication. More sophisticated examples are component models which obviously differ in J2EE, .NET or CORBA but their aim to have loosely coupled software blocks with a determined interface stays the same. An incomplete list of concepts:
   Object-Oriented concepts like inheritance and polymorphism, or synchronous and asynchronous communication, stateful and stateless connections, transactions, micro- and macroflows (duration of workflows), component models, business processes, etc.

   **Specifications:**
   WS-Transaction, WSXL, BPEL4WS, JSR 109 Implementing Enterprise Web services (J2EE), partly WSDL and SOAP

   **Interoperability:**
   Most important is the fact that the concepts of this view are supported by the platform or application itself which lies behind the Web services technology. Example: an existing transaction processing system which uses proprietary protocols can interoperate when wrapped by a WS-Transaction implementation. Interoperability should therefore be guaranteed when both communication partners agreed on implementing the same specification. But as already seen in the WS-I Basic Profile it can be necessary to clarify ambiguous passages or choose the most common option from several possibilities.

4. **Device View**

   **General:**
   Depending on the devices which are used by end-users or involved in automatic communication, not only the development can differ also the usage of the underlying concepts, language or protocols may change. Possible clients are regular applications, web browser or the broad field of mobile and embedded devices.

   **Specifications:**
   JSR 172 – J2ME Web services

   **Interoperability:**
   Ideally, this view is not involved in interoperability problems at all because the preceding views make this one independent but especially the limited memory equipment or processor
power can make it necessary not to use certain features. A Web services toolkit for Mobile Devices (WSTK MD) can be found on IBM's Alphaworks site\textsuperscript{19}. There are also special IDEs for mobile devices like WebSphere Studio Device Developer (WSDD) or Windows Embedded Studio.

5. **Operational View**

   **General:**
   
The Operational View lies in parallel to the other four views as it influences all of them. This view contains the typical *Non Functional Requirements* (NFR). Examples are: Performance, security, availability, reliability, scalability.

   **Specifications:**
   

   **Interoperability:**
   
   In this view interoperability is very important especially security needs to interoperate between platforms. Help for the security issues comes from WS-I. Other specifications from this view are still new and have to be integrated in development environments.

The previously mentioned specifications are meant as examples of the actual view. They are not intended as a complete listing of available specifications.

### 3.2 Addressed Views

As the above mentioned model tries to comprise all Web service aspects it is not possible to practically prove all views. Thus, it is necessary to unambiguously explain which views the reader can expect to be discussed and which are out of scope, at least for this thesis.

- **Protocol Stack View** (not addressed)

  As already mentioned, this view ensures interoperability already through the Basic Profile from WS-I. Therefore, no tests are carried out.

- **Programming Language View** (addressed)

  For this view the object models and their Web service interoperability are examined. See chapter 4.3.2 on page 44.

\textsuperscript{19} [http://www.alphaworks.ibm.com/tech/wstkmd/]
• Programming Concepts View (addressed)
  This view gets addressed in chapter 5.2, page 54 where IDEs are compared in regard to their support of concepts. This support must be available by an integrated tool or wizard.

• Device View (not addressed)
  The Device View is not addressed as this would include tests on different mobile devices which is out of scope for this thesis.

• Operational View (addressed)
  From the Operational View the aspect of security is picked out and an interoperability test performed. See chapter 4.3.3 on page 47.

3.3 Classification Model Revised

During the work with the model described in chapter 3.1 the idea that business processes are located in the Programming Concepts View turned out to be inconvenient. First of all, the concept of business processes does not fit completely into the chosen definition of a programming concept because they are more abstract than the aforementioned concepts. But more importantly, because of that abstraction a Business Process View has to be located on top of the model's views. Business processes are described in their own language (BPEL4WS) and make use of already available services. Therefore, it is sensible to extract this view and change the model.

For the reason that the work started with the model from chapter 3.1 it is important to show the development of the model instead of confronting the reader with an already changed classification.

![Figure 8 Revised Web service classification model](image-url)
4 Interoperability Tests

These interoperability tests try to prove the correctness of the introduced model by showing different aspects of it, namely of the Programming Languages and Operational View. Tests are performed and results are discussed so that the snapshot shows what is possible today and in which areas manual effort is still needed.

4.1 Test Architecture

First approaches to build a complete test harness which covers the different areas showed to be too inflexible and hard to adapt to modifications. By large, this was caused by leveraging the provided wizards and automatisms. Therefore, the choice was made to create small individual test suites that are specialized on the critical regions. In the following, thoughts and results are presented starting with a general discussion about the used communication and encoding style and why there is not really a choice when building interoperable Web services. Then, Web service examples are presented, describing how to achieve interoperability. First, a very basic example starts to present the necessary steps to generate a Web service and a consumer belonging to it, as it is not as intuitive as it could be. Especially WSAD makes different ways available which can be used to create Web services or consumers. Unfortunately, not all ways induce the same results which can be very confusing until the correct way is found. The second test stresses the IDEs qualities for creating secure Web services with a more sophisticated feature than HTTPS transport layer security. Thereafter, a WS-I testing tools example shows the usage and benefits when running into interoperability problems on a WSDL, SOAP or UDDI basis. At last, the semantics of Web services are discussed. The structure of the test cases is divided into four parts:

- Objective:
  The objective explains the reason why the actual test is performed, as well as the addressed problem area and the expected result is presented. After reading the objective the reader should have learned whether the test case can provide interesting information or not.

- Method:
  The method illustrates the way how the test case is implemented. This helps to understand how the goals from the objective are realized.
• Test Case:
  This section about the actual test case includes the description of the Web services and consumers and helps to reconstruct the test case if necessary.

• Result:
  The result shows the analysis and evaluation of the performed tests, the expected results are compared to the achieved ones and assessment about the quality of the test is made.

### 4.2 SOAP Communication and Encoding Styles

There are two different communication styles and two encodings which can be combined to four ways how to serialize SOAP 1.1 messages:

1. RPC / Section 5 encoding
2. RPC / literal encoding
3. Document / Section 5 encoding
4. Document / literal encoding

The two encoding styles take care of how to represent data types in SOAP messages. The SOAP Section 5 encoding has its name from the fifth section in the SOAP 1.1 specification and defines a data model which is not completely based on XML schema and has certain ambiguities. Therefore, the Basic Profile 1.0 does not allow the Section 5 encoding any more. The literal encoding on the other hand depends on a XML schema description which is responsible for the data type mapping between applications. This means there are only two combinations left to choose from:

1. RPC / literal encoding
2. Document / literal encoding

The choice between the RPC and document communication style is a fundamental decision between the easy-to-use RPC-APIs and the document style which has better interoperability properties because it can carry any XML instance in the SOAP message. Because of the reason that a complete XML message can be included in the SOAP body the document style is also called *message-oriented style.*
The RPC style has a formal definition of how to exchange information in the SOAP message. To be precise, there is a Remote Procedure Call and a Remote Procedure Call Response which are able to transport parameters and return values, respectively. This makes RPC style messages easier to use because the message has to conform to a specific structure but the drawback not being able to transport a whole XML message disqualifies this communication style for truly interoperable Web services. Thus, this leaves, at least for larger projects, only the choice to rely on using the message-oriented style document/literal.

### 4.3 Web Service Examples

This chapter presents the used Web service examples of this work. All examples follow a pattern in which the service is explained, first for the Java/WSAD side then for the similar .NET/Visual Studio service. Afterwards, the same is done for the Web service clients, whereas the Java client has to interoperate with the .NET service and vice versa. Problems, which occurred during the tests, are mentioned in the test case part and are discussed in the result section.

#### 4.3.1 Simple Web Service

**Objective**

A simple 'Hello World' Web service shows the basic tasks which have to be fulfilled for realizing Web services. The reason for such a simple test is to see how Web services are implemented and how consumers for the other technology platform are developed. This is especially for WSAD not as easy as it sounds because different ways of using wizards to produce a Web service consumer can result in a not fully working application. Yet, as the exchanged messages carry only strings this test case is not expected to result in real interoperability problems.

**Method**

Precondition for the test case is an already existing project, in this case only one simple class with one method which returns the string 'HelloWorld'. For this introducing example, the clients are static requestors, that means the information of the Web service location (URL) is hard-coded. Therefore, a location change of the service implicates a new program compilation.
Test Case

Java Hello World Web Service

The precondition for a Web service with WSAD or Eclipse is a Dynamic Web Project. In case the existing project is not of that type it is necessary to import your source code into a new Dynamic Web Project. Then a new Web Service Project can be created. The wizard asks for the Web service type which in this case is a 'Java bean Web Service'. Optionally, a Client proxy can be generated which provides a remote procedure call interface to the Web service. When the checkbox for testing the generated proxy is selected, a little web application is created which tests the service and proxy.

In the Service Deployment Configuration pane an embedded application server can be chosen or created. Also, the web project which should be turned into a Web service, must be specified here. The next pane is called Web Service Java Bean Selection and as the name indicates the Java bean has to be selected.

After a click on the finish button the wizard uses the collected information to create a bunch of files. Table 2 shows the existing files after the Web service wizard has run. The ones created by the tool are emphasized in bold face.

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HelloWorld.java</td>
<td>Original Java bean with the getWelcome() method</td>
</tr>
<tr>
<td>HelloWorld_SEI.java</td>
<td>A Java interface which defines all exposed methods and extends java.rmi.Remote</td>
</tr>
<tr>
<td>HelloWorld.wsdl</td>
<td>WSDL interface and implementation description</td>
</tr>
<tr>
<td>HelloWorld_mapping.xml</td>
<td>JAX-RPC file for the mapping from Java packages to XML namespaces</td>
</tr>
<tr>
<td>ibm-webservices-bnd.xmi</td>
<td>IBM WebSphere -specific deployment information (webservices)</td>
</tr>
<tr>
<td>ibm-webservices-ext.xmi</td>
<td></td>
</tr>
<tr>
<td>webservices.xml</td>
<td>JSR109 deployment descriptor</td>
</tr>
<tr>
<td>web.xml</td>
<td>J2EE Web deployment descriptor</td>
</tr>
<tr>
<td>ibm-web-bnd.xmi</td>
<td>IBM WebSphere -specific deployment information (web)</td>
</tr>
<tr>
<td>ibm-web-ext.xmi</td>
<td></td>
</tr>
</tbody>
</table>

Table 2 Existing files after WSAD Web service wizard run

20 A Java bean is a regular Java class which follows some designated rules.
Chapter 4  Interoperability Tests

.NET Hello World Web Service

For creating Web services with Visual Studio .NET a project of the type “ASP.NET Web service” has to be created. The class of the already existing ‘HelloWorld’ project can be imported to the ASP.NET project by drag-and-drop. This class has to be derived from System.Web.Services.WebService and all methods, which should be exposed as Web service, need to have the WebMethod attribute [WebMethod]. With such attributes in front of methods or classes it is also possible to set the namespace and description for the service.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service1.asmx</td>
<td>The automatically created file for Web service methods.</td>
</tr>
<tr>
<td>Web.config</td>
<td>Application specific configuration settings for the CLR and the application.</td>
</tr>
<tr>
<td>Global.asax</td>
<td>Contains code for responding to application-level events raised from ASP.NET.</td>
</tr>
<tr>
<td>AssemblyInfo.cs</td>
<td>Contains information, which are used for packaging the application into a .NET assembly.</td>
</tr>
</tbody>
</table>

Java Hello World Web Service Client

For creating a client with WSAD for a .NET service a Dynamic Web Project must be created (File -> New -> Project -> Web -> Dynamic Web Project).

1. Create Web service client

A wizard exists for creating a Web service client. It can be opened by selecting File -> New -> Other -> Web services -> Web service client.

2. Link client to Web Project

The wizard creates a Java proxy client (mark the checkbox: Test generated proxy) which has to be linked to the previously generated Dynamic Web Project.

3. Select WSDL file

The Web service description can be imported from a WSDL-, WSIL- or HTML-document.

The four files from Table 3 are automatically created by WebSphere Application Developer which uses the IBM WSDL2Java tool.
<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service.java</td>
<td>Interface which contains proxy related methods.</td>
</tr>
<tr>
<td>ServiceLocater.java</td>
<td>Contains the Web service location (URI) and creates the proxy object; implements the Service interface.</td>
</tr>
<tr>
<td>ServiceSoap.java</td>
<td>Interface which contains all methods of the Web service.</td>
</tr>
<tr>
<td>ServiceSoapStub.java</td>
<td>Proxy object; implements the ServiceSoap interface</td>
</tr>
</tbody>
</table>

Table 3 Existing files after importing the Web services' WSDL file

.NET Hello World Web Service Client

A few steps have be done to access the Java Web service. At the end a console application (a simple executable) is produced which consumes the `getWelcome()` method of the Web service. To get started, a new project must be created: File -> New -> Project. The project type is 'Visual C# project' and for the template 'Console Application' has to be selected.

1. **Search for the Web service and add a web reference**

   To be able to use a Web service in a project, a web reference must be added to the current project:
   
   Project -> Add web reference

   The new window needs the URL of the WSDL file from the Web service as the only information. A description with available service methods is shown and with the button 'Add reference' it is added to the current project.

2. **Include the name space of the web reference**

   To be able to access the proxy object it is necessary to include the created namespace:

   ```
   using ConsoleApplication.localhost;
   ```

3. **Create an instance of the proxy object**

   In the `main()` method the proxy class can now be instantiated:

   ```
   HelloWorldService service = new HelloWorldService();
   ```

   The name of the proxy class `HelloWorldService` originates from the WSDL `<service>` tag attribute `name`. 
4. **Invoke a method on the proxy object**

The call of the `getWelcome()` method handles the remote calls completely transparent for the developer:

```csharp
Console.WriteLine(service.getWelcome());
```

The method name comes from the `<portType>` child tag `<operation>`.

The files created after the steps mentioned above are shown in Table 4. The file emphasized with the bold face is automatically generated after adding the web reference.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class1.cs</td>
<td>Contains the <code>main()</code> method which creates the local proxy object and makes the Web service call.</td>
</tr>
<tr>
<td>Reference.cs</td>
<td>The local proxy object with the class <code>HelloWebService</code>.</td>
</tr>
<tr>
<td>AssemblyInfo.cs</td>
<td>Contains information, which are used for packaging the application into a .NET assembly.</td>
</tr>
<tr>
<td>App.ico</td>
<td>The Icon used for this application.</td>
</tr>
</tbody>
</table>

*Table 4 Existing files after adding the web reference*

**Result**

This test case showed the realization of Web service and their consumers from the other technology platform. As expected, interoperability did not occur but the solid base for following creation of Web services is build as they will be developed in the exact same way as described above.

### 4.3.2 Object Model Transformations

**Objective**

As already mentioned in the description of the Programming Languages View object types from languages have to be mapped to an independent format with the help of XML Schema. Primitive types, like `int`, `float` or `string`, are the essential ones when exchanging information. Those are supported by every programming language and must be correctly mapped to XML schema types. For complex types, on the other hand, it cannot be expected that every language supports the same kind of type. As such complex types already exist in a wide range in the platform APIs, here are only two problematic examples discussed.
**Method**

For this test case the common primitive types were chosen and implemented as simple read methods on each type in both platforms. All those methods had then been exposed as Web service operations which were accessed by consumers from the other platform. Especially, the number ranges (maximum and minimum) have been observed to see if the platforms behave correctly in that area. In case that the numbers were equal on the service, as well as on the consumer side, that type was considered as interoperable.

It is important to note, that for the test case only the automatic mapping support was leveraged and no manual effort was made to enhance interoperability by defining own schema types what is certainly possible for the complex type examples.

**Test Case**

Table 5 depicts the objects which have been tested for interoperability. The first three columns show the particular object types in Java, XML Schema and C#. The note column refers to peculiarities that possibly appeared during the test. Those are discussed in the result section.

<table>
<thead>
<tr>
<th>Java type</th>
<th>XML Schema type</th>
<th>.NET CLR type</th>
<th>Note</th>
<th>interoperable</th>
</tr>
</thead>
<tbody>
<tr>
<td>String</td>
<td>string</td>
<td>String</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>boolean</td>
<td>boolean</td>
<td>Boolean</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>float</td>
<td>float</td>
<td>Single</td>
<td>see text, page 46</td>
<td>yes</td>
</tr>
<tr>
<td>double</td>
<td>double</td>
<td>Double</td>
<td>see text, page 46</td>
<td>yes</td>
</tr>
<tr>
<td>short</td>
<td>short</td>
<td>Int16</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>int</td>
<td>int</td>
<td>Int32</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>long</td>
<td>long</td>
<td>Int64</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>Date</td>
<td>dateTime</td>
<td>DateTime</td>
<td></td>
<td>yes</td>
</tr>
<tr>
<td>Vector</td>
<td>---</td>
<td>---</td>
<td>see text, page 47</td>
<td>no</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>Collection</td>
<td></td>
<td>no</td>
</tr>
</tbody>
</table>

*Table 5 Object type mapping between Java and C#*

Note: column number 3 contains the corresponding .NET framework type which is available in the System namespace for all .NET languages, not only C#.
The interoperability column shows if problems occurred when transferring a certain type from WebSphere to Visual Studio and the other way round. Only when both directions worked without problems the column contains a yes. Yet, even when the interoperable column contains a no, interoperability is still possible but not automatically achieved by the tools. It is always possible to create own XML schema types which can be used for exchanging information.

**Result**

Table 5 from the test case section already anticipated the interoperability results for the mappings. The primitive types showed to be interoperable without exception. A minor pitfall for developers is presented in the next paragraph. After that, a few comments are made about the complex type problems.

**Float and Double Objects**

During the tests it became obvious that a misunderstanding can occur when using the data types `Float`\(^\text{21}\) and `Double`. Java as well as C# provide special members for the numerical data types:

- MinValue, MaxValue, NaN, NegativeInfinity and PositiveInfinity in C#\(^\text{21}\)
- MIN_VALUE, MAX_VALUE, NaN, NEGATIVE_INFINITY and POSITIVE_INFINITY in Java

The member NaN stands for *Not a Number* and is returned when using invalid floating point operations like \(0/0\) or parsing a non-numerical string to a floating point data type. But NaN and PositiveInfinity\(^\text{22}\), NegativeInfinity and MaxValue show no problems. At first glance, also MinValue seems equal but the understanding differs in Java and C#. The semantic of MinValue in Java means a “constant holding the smallest positive nonzero value of type float” [WWW15]. In .NET, on the other hand, the MSDN library says about the MinValue field: “Represents the smallest possible value of Single,” which is in fact the smallest negative number. This applies for both, Float and Double objects. Yet, it does not effect the wire interoperability because the number boundaries are the same on both sides, but an incorrect use of the member MinValue could still lead to a false number boundary check.

---

\(^\text{21}\) the corresponding .NET CLR type is called `Single` but when programming in C#, `float` is equivalent.

\(^\text{22}\) For a easement of reading only this kind of spelling is used in the following but means both types J2EE and .NET
**Java Vector Objects**

The Java language has the concept of an object independent container which automatically enlarges space for objects when necessary. This type does not have a similar XML schema type for Web services which means that it is not interoperable. However, WSAD knows another possibility of describing the Vector type in WSDL. It tries to describe the `Vector` as an proprietary *Apache Soap* type which, if Web service interoperability is wanted, should completely be avoided.

This shows that it is important to take care of types not available in default XML schema types. A possible solution for the `Vector` type, for example, would be to move elements from the `Vector` into an array which only contains standard types. For complex types this can get a very complicated task and, if the developer has access to both the Web service and Web service consumer the creation of an own XML schema type is the better solution.

The Java `Vector` object is only exemplary for other types where the same problems can occur. Another example is the Java type `Map`. There are more types like this in the Java and .NET frameworks. If interoperability is needed and it is planned to publish the return values as a Web service without knowing the accessing platform, foresight during the design is advisable.

### 4.3.3 Security Example

**Objective**

Secure Web services have not yet been addressed by the WS-I organization, therefore, it is interesting to see how good the IDE's automatic support is and how interoperable the results are. Thus, this test case only uses the provided features of the IDEs to secure the exchange of information. For the WS-Security specification, the word *security* describes following security options:

- Data integrity with XML signatures
- Data confidence with XML encryption
- User authorization and authentication with security tokens

Those are the security features in the WS-Security specifications in the Web service security stack. In this stack are quite more features addressed\(^2\), but for this example only WS-Security is addressed as it is the foundation for all other security specifications.

\(^2\) See chapter 2.8 on page 25
**Method**

For this test case a consumer accesses a simple Web service which is secured by the tools provided by the IDE. The body of the exchanged SOAP messages should either be signed or encrypted because in both cases it is necessary to use certificates\(^\text{24}\). The way how security is implemented in the particular IDE, programmatically or declaratively, is described in chapter 2.8.1 (page 26).

**Test Case**

**Java Web Service and .NET Consumer**

It is important to mention that the documentation of WSAD explicitly states that, if security features are used, interoperability with .NET services is not given with the current version (WSAD 5.1) of the product\(^\text{25}\). A .NET Web service consumer is trying to access a Java Web service that is configured to encrypt the sent data. The first call to the service throws a .NET exception from the type SoapException with the message: “SOAP body is not encrypted.” This raised exception comes from WebSphere and complains that the request was not encrypted at all.

The reason for that is the implemented security model in WebSphere. In front of the client/server Web service modules, which can be JavaBeans or EJBs, a Web service security handler component is located. This handler can be configured declaratively, which means no API calls are needed for configuring the security features. Therefore, all security constraints have to be independently edited in XML description files\(^\text{26}\) for requestor and sender. If those adjustments do not match the created ones, SOAP messages do not contain the expected content, which is exactly what happened when the .NET consumer tried to access the Java Web service without knowing the necessary security constraints.

But even programmatically implemented security for the .NET client which signs the SOAP message body produces the same exception. This indicates that the WebSphere Application Server, which runs the security handler and that component is responsible for checking the signature, does not understand the included security information.

\(^{24}\) Refer to chapter 2.8.1 for information about security certificates

\(^{25}\) Again, WS-I has not yet addressed the security interoperability.

\(^{26}\) ibm-webservicesclient-ext.xmi and ibm-webservicesclient-bnd.xmi for consumers and ibm-webservices-ext.xmi and ibm-webservices-bnd.xmi for servers
Chapter 4  Interoperability Tests

.NET Web Service and Java Consumer
Similar to the scenario above, a seamless interoperability without manual changes is not possible. The .NET Web service provides one method which returns the sum of two integers after checking for a signed SOAP message with a X.509 certificate. In order to avoid an error because of different certificates the check for a correct and known certificate has been disabled. This way, the only requirement has been a signed SOAP body with any certificate.

As a result an exception was raised by the .NET Web service with reference to an unsigned SOAP body. Further investigation and logging showed that the .NET Web service was not able to find the provided security token in the WebSphere SOAP message because the WSE 1.0 did not understand WebSphere's security part in the SOAP header.

Result
No doubt, secure and interoperable Web services and clients are possible. Microsoft as well as IBM provide working examples of one platform communicating successful with the other. But a comfortable way for developers to achieve this is still not visible. For implementing WS-Security there is more information needed than the WSDL description contains. The WSDL specification is not responsible for providing information about certificates and algorithms what also shows through the different model views, WSDL in the Protocol Stack View and security in the Operational View. Therefore, it is required to have additional knowledge about a Web service and its usage. But this additional knowledge can find its way into WSDL files over an extension mechanism which is leveraged by the WS-Policy specification stack. WS-SecurityPolicy is the appropriate document where security extensions are described that can be included in WSDL files to require a Web service requestor to attach necessary information, for example a security token. This neither happens in the two IDEs automatically nor do they provide comfortable ways to add those information to the Web service description.

4.4 WS-I Testing Tools Applied
This chapter describes how to use the WS-I Testing Tools which can be applied to find interoperability problems. The following sections present the two essential command-line tools, a monitor for capturing the exchanged messages and an analyzer for testing the compliance with the Basic Profile.
4.4.1 Monitor

To monitor the HTTP traffic between service and requestor the Monitor intercepts the traffic, logs it and then forwards the request to the communication partner. Therefore, it is necessary to make some changes to the service and/or requestor. When the service is running on port 6080 and the Monitor is configured to accept requests on port 4040 the requestor has to address its service call to port 4040. There exists a pitfall when the client requests the WSDL file from the service and that file still contains a port binding to the original port 6080. In that case, the WSDL file delivering is logged by the Monitor but the actual request bypasses the Monitor because it is addressed directly to the service. Dependent on the time when the WSDL file is accessed by the client it is necessary to change the service description on the requestor (static client) or service-side (dynamic client).

The Monitor reads its configuration from a XML file like the one shown in Source-Code 3. The log file and the formerly mentioned ports are highlighted. A description about other XML elements can be found in the WS-I Tools documentation.

When the Monitor is started it logs all received HTTP traffic and makes no assumptions whether the messages contain a SOAP message or not. That is done because SOAP is only one of many possible transport protocols.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<wsi-monConfig:configuration
 xmlns:wsi-monConfig="http://www.ws-i.org/testing/2003/03/monitorConfig/"
 xmlns:wsi-common="http://www.ws-i.org/testing/2003/03/common/">
 <wsi-monConfig:comment>
  Test for HelloWorld service.
 </wsi-monConfig:comment>
 <wsi-monConfig:logFile replace="true" location="log-helloworld.xml">
  <wsi-common:addStyleSheet href="C:/programme/wsi-test-tools/common/xsl/log.xsl"
     type="text/xsl"/>
 </wsi-monConfig:logFile>
 <wsi-monConfig:logDuration>600</wsi-monConfig:logDuration>
 <wsi-monConfig:cleanupTimeoutSeconds>3</wsi-monConfig:cleanupTimeoutSeconds>
 <wsi-monConfig:manInTheMiddle>
  <wsi-monConfig:redirect>
   <wsi-monConfig:comment>
    Redirect to HelloWorld WS on WSAD Test Server on 6080 and listen on Port 4040 for a connection.
   </wsi-monConfig:comment>
   <wsi-monConfig:listenPort>4040</wsi-monConfig:listenPort>
   <wsi-monConfig:schemeAndHostPort>http://localhost:6080</wsi-monConfig:schemeAndHostPort>
   <wsi-monConfig:maxConnections>1000</wsi-monConfig:maxConnections>
   <wsi-monConfig:readTimeoutSeconds>15</wsi-monConfig:readTimeoutSeconds>
  </wsi-monConfig:redirect>
 </wsi-monConfig:manInTheMiddle>
 </wsi-monConfig:configuration>
```

SourceCode 3 Minimal configuration file for WS-I Monitor
4.4.2 Analyzer

The captured communication can be checked for conformity to the WS-I Basic Profile with help of a tool called Analyzer. This file also needs a configuration file for running the test suite. A sample configuration file can be found in the listing SourceCode 4. This file needs a few more changes, which are highlighted in the presented source code. The Analyzer must know the input (“log-helloworld.xml”) and output file (“helloWorld_report.xml”). If the Analyzer also checks the WSDL file, a <wsdlReference> element has to be included. Besides WSDL (description) and the captured XML log files (messages) the tool can also control UDDI entries (discovery). WS-I summarizes these three input types under the term artifacts.

The result of an Analyzer run is a XML file which contains a summary and detailed information about the TestAssertions. The summary just informs if the analyzed data complies to the Basic Profile and the detailed view shows informations about the particular TestAssertions.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsi-analyzerConfig:configuration name="Sample Basic Profile Analyzer Configuration" xmlns:wsi-analyzerConfig="http://www.ws-i.org/testing/2003/03/analyzerConfig/"
 xmlns:wsi-common="http://www.ws-i.org/testing/2003/03/common/">
 <wsi-common:description xml:lang="en">
  This file contains the configuration file for the Basic Profile Analyzer. HelloWorld example.
 </wsi-common:description>
 <wsi-analyzerConfig:verbose>false</wsi-analyzerConfig:verbose>
 <wsi-analyzerConfig:assertionResults type="all" messageEntry="true" failureMessage="true"/>
 <wsi-common:addStyleSheet href="C:/programme/wsi-test-tools/common/xsl/report.xsl" type="text/xsl"/>
 <wsi-analyzerConfig:reportFile replace="true" location="helloWorld_report.xml">
  <wsi-common:addStyleSheet href="C:/programme/wsi-test-tools/common/xsl/report.xsl" type="text/xsl"/>
 </wsi-analyzerConfig:reportFile>
 <wsi-analyzerConfig:testAssertionsFile/>
 <wsi-analyzerConfig:logFile correlationType="endpoint">
  log-helloworld.xml
 </wsi-analyzerConfig:logFile>
 <wsi-analyzerConfig:wsdlReference>
  <wsi-analyzerConfig:wsdlElement type="port" parentElementName="HelloWorldService"
  <wsi-analyzerConfig:wsdlURI>C:/programme/wsi-test-tools/java/bin/HelloWorld.wsdl</wsi-analyzerConfig:wsdlURI>
 </wsi-analyzerConfig:wsdlReference>
</wsi-analyzerConfig:configuration>
```

SourceCode 4 Minimal configuration file for WS-I Analyzer tool

4.4.3 Conclusion

The WS-I testing tools provide a great help when looking for the cause of an interoperability problem. They are able to check for SOAP, WSDL and UDDI incompatibilities and although said that they cannot find all kind of problems they certainly can give hints for the original cause. The handling of the command-line tools and the documentation is good but the need for manual
configuring the Monitor and Analyzer XML files is tedious. The XML format predetermines the integration into WSAD and Visual Studio. WSAD has already the option to check imported WSDL files not only for validity but also for WS-I conformance. Following version of the IDEs will contain the complete functionality of the WS-I tools with automatic creation of the XML configuration files. As a long-term objective, the unnecessity of the testing tools would be the best solution because standards are unambiguous and implemented without proprietary changes. But to be realistic, the amount of specifications will make it needful to enlarge the endeavor for the WS-I organization, as well as for the scope of the WS-I testing tools.

4.5 Web Service Semantics

The semantic of a Web service depends on the underlying platform and their implementation. This topic has already been touched upon in chapter 4.3.2 Object Model Transformations on page 44, which examines the size of numeric data types. The exchange of the same data types between different platforms on which they can have different meanings may result in hard to find errors. Another real-world misunderstanding was the meaning of the NULL value and its representation as an empty string. One IDE mapped NULL to an empty string which was mapped back as an actual empty string by the receiving application server. Some tests were performed on this topic especially stressing the values

- NULL,
- empty strings, and
- blanks (whitespaces).

The results showed that nowadays these values do not cause problems any more. Empty strings, strings containing blanks only, and strings with leading blanks are correctly mapped and recognized from both platforms.

But even after the mentioned issues are resolved, it is important for developers to keep an eye on the semantics when creating or using Web services, especially if the used technology is newer than a mature one.
5 Usability

As usability of an IDE is a broad area this chapter focuses only on certain topics. Usability and user-friendliness always depend on previous knowledge of the particular user, too. For that reason, it is difficult to get objective results. Thus, some metrics for the measurement of the usability are necessary.

In-depth knowledge of a technology or specification is always an advantage during development and is especially necessary when looking for errors. But wherever possible, this knowledge should also be available in the IDE for supporting the developer during work. This way the protection from errors will be a benefit regardless of the cause of the error, either due to carelessness or lack of knowledge. The amount of new or revised specifications makes it hard to keep track especially because often a profound knowledge is only available for one technology platform. In this case, the help of an IDE to secure interoperability is an appreciated feature.

To be as objective as possible, at the end of the chapter there will neither be a winner-nor a loser-product. Goal of this chapter is to oppose certain features of the IDEs, which should allow the reader to compare, to form an opinion and to choose the right product, in case the decision is depending on this aspect. Yet, more important for the decision-making should be the aspect of the need for multi-operating system (J2EE) or multi-programming language (.NET) enabled software.

5.1 Usability and the Classification Model

The usability question spans itself over all five views of the proposed model in chapter 3.1. In the following the views are listed again and a short paragraph explains the link between the usability and the model.

- Protocol Stack View
  Usability for the Protocol Stack View means abstraction from the wire protocols. The developer should not be involved in the creation of a WSDL description or a SOAP message. Nevertheless, access to an interface description must still be possible in case the user wants to make manual changes.
Chapter 5 Usability

- **Programming Language View**
  This view is realized in every IDE. Mapping the language specific object model to the language independent object model is a must when supporting Web services. Again, developer's freedom of action for the realization of own mappings is desirable.

- **Programming Concepts View**
  The Programming Concepts View is the place for tool and wizard support in IDEs. An intelligent integration helps users during the development process. This is also the objective which is examined in the following chapters.

- **Device View**
  Special versions of an IDE or plugins support developing for special devices like mobile phones or PDAs.

- **Operational View**
  Very similar to the Programming Concepts View, specification support can be implemented in an IDE through wizards and support developers in realizing the NFRs\(^\text{27}\).

### 5.2 Concept Support Comparison

The usability and user-friendliness question is answered for a very limited area in this work, namely the Programming Concepts View. It is of interest 'how easy' it is to create, deploy and test a Web service with the help of the IDEs wizards. To be able to compare WebSphere and Visual Studio the support for elements from the Programming Concepts View are examined. The support should be automated by a wizard or the IDE should support the developer with an integrated tool to simplify the tasks. Therefore, the following chapters provide information about different concepts and their availability in the examined IDEs.

#### 5.2.1 Asynchronous and Synchronous Concepts

Synchronous Web service calls are the usual way to invoke services or local methods and is obviously supported by both IDEs. The disadvantage of the synchronous concept is that the client who makes the call has to wait until it gets a response or an internal timer expires. During this time the whole thread cannot do anything beside waiting for the service to return. This is especially a problem for long-running business processes.

\(^{27}\) Non Functional Requests
The asynchronous concept, on the other hand, solves the waiting problem and encourages the realization of loosely coupled systems and is therefore one of the key concepts of the Service Oriented Architecture. The client fires a request to a service provider but instead of waiting for a response the client thread continues its work and gets informed when the server returns a result. One way of realizing this is to have a local proxy which handles the communication with the service and informs the client thread over a callback mechanism when the result is available.

On .NET side this solution is integrated in the .NET framework which provides a design pattern for making asynchronous Web service calls. The standard proxy generation tool Wsdl.exe for clients automatically creates methods for making asynchronous calls to Web services. This way, the developer always has the choice to decide which type of call the wants to use without performing additional steps, which is a user-friendly solution.

The corresponding technology on the Java side is the Web Service Invocation Framework (WSIF) which is included in WebSphere and also supports asynchronous calls based on the Java Message Service (JMS). WebSphere Application Developer and WebSphere Application Server are shipped with an embedded JMS provider.

The JMS specification provides more features than support for asynchronous calls only, it is a complete messaging framework for reliable messaging. The .NET framework also provides comparable functionality in the namespace System.Messaging and the WS-Reliable Messaging specification is about to bring interoperability to this topic.

### 5.2.2 Transactional Concepts

Support for transactional concepts is included in both WebSphere and Visual Studio. In Visual Studio, for example, Web service methods can be marked as participating in an automatic transaction by adding a certain attribute in front of the method. The .NET framework takes care of possible exceptions which result in a rollback. WSAD support is slightly different but is also included. Transaction J2EE mandatory support is included in WebSphere Application Server for Enterprise JavaBeans (container-managed transaction) and it also can be used programmatically by using the Java package javax.transaction.

---

28 WSIF providers which support asynchronous operations are the Native JMS and the SOAP over JMS provider.
This leads to the more interesting question which tends to the server-side and the implementation of the current Web service specifications. Those can only be useful if the underlying platform also supports the transactions which is true for J2EE/WSAD and .NET/Visual Studio. In September 2003 two transaction-related specifications were updated, namely WS-Atomic-Transaction and WS-Coordination. A third, WS-BusinessActivity, is not yet available as final version. The following list gives a short overview from [WWW17]:

- **WS-AtomicTransaction**
  
  “defines the Atomic Transaction coordination type and is appropriate to use when building applications that require a consistent agreement on the outcome of a short-lived distributed activity, where strong isolation is required until the transaction completes.”

- **WS-Coordination**
  
  “defines the protocols for creating activities, registering in activities, and transmitting information to disseminate an activity. WS-Coordination provides an extensible framework in which participants can join in activities enabling the coordination of distributed applications.”

- **WS-BusinessActivity**
  
  “defines the Business Activity coordination type. It is appropriate to use when building applications that require a consistent agreement on the coordination of a distributed activity, where strong isolation is not feasible, and application-specific compensating actions are used to coordinate the activity.”

Implementations of those three specifications are not yet included in the examined products. However, for WebSphere Application Server there is a technology preview available on alphaworks\(^\text{29}\) which comprises WS-AtomicTransaction and WS-Coordination.

Transaction-related interoperability cannot be expected until the the specifications are integrated into the application servers and development environments.

\(^{29}\) http://www.alphaworks.ibm.com/
5.2.3 Business Process Concepts

The modeling of business processes is not available in the inspected releases of the IDEs. Supplementary tools are necessary for designing workflows and business processes which in turn need special versions of application servers to be executed. IBM provides an extended IDE edition with the name 'WSAD Integration Edition' for these tasks and the corresponding server-part would be the 'WebSphere Application Server Enterprise'. The equivalent programs from Microsoft are 'MS Office Developer XP' and the 'BizTalk Server'.

5.3 Visualization

Visualization for drag-and-drop GUI (Graphical User Interfaces) is included in both IDEs. They support development for .NET/MFC and AWT/Swing, respectively. Visual Studio integrates a GUI builder for quite a while in contrast to the relatively new support from Java IDEs. As a result, the drag-and-drop of graphical elements feels\(^{30}\) slower in WSAD than in Visual Studio. Beside the visualization of a GUI it is also important to depict the structure of an enterprise application. The Unified Modeling Language (UML) prevailed for such tasks but it only slowly finds its way into the standard versions of IDEs. Visual Studio Professional does not provide UML support at all and refers to the VS .NET Enterprise Architects edition which is shipped with a tool called Microsoft Visio for Enterprise Architects. WSAD on the other hand includes at least support for UML class diagrams and refers to the tool Rational XDE for further UML support. Class diagrams can be created in a top-down, as well as in a bottom-up approach which means it is possible to create source code from a diagram or to build the class diagram from already available source code.

5.4 Integrated Help

Both IDEs come with a large amount of documentation. WSAD has an integrated help which opens in a new WSAD window, whereas the Visual Studio includes an integrated approach where the help is viewed inside a region of the window. IBM's IDE implies a feature called 'cribs' that opens a new perspective and displays succinct, hyperlinked tasks a developer has to perform to achieve a certain task. WSAD, as well as VS.NET obviously include class library documentation but also examples, tutorials and web references. Furthermore, it is possible to install the

\(^{30}\) The word 'feels' has been chosen intentionally because no actual test has been performed on the time it took for a drag-and-drop component to show up on the whiteboard.
MSDN Library which is shipped with Visual Studio .NET and contains additional documentation, technical reports, magazines and even complete books. During the usage of wizards Visual Studio provides a context-sensitive help button which opens another window with further information to the options. WSAD lacks such an context-sensitive help, the reader needs to browse or search the documentation for information.

Personally, I prefer to have the documentation in a separate window and therefore, a maximized source code view, but that is a matter of taste. Nonetheless, the quality of the documentations is equally good and search functions are simple but sufficient. The reader is able to navigate through the documentation's tree structure and can add pages to a Bookmarks/Favorites view.
6 Summary

The desired goals of this thesis have been split into two areas. First, a classification of the Web services world regarding interoperability was developed. In fact, the created model even tries to be independent from interoperability. It divides the Web services area into five Views which throw light on Web services from different perspectives. As a result, the classification proved helpful on organizing and structuring the different test cases, though it was not possible to take all aspects of the model into account. But also the attention given to the regarded aspects showed the usefulness and therefore the correctness of the approach, at least for the selected area of responsibility.

The second goal was to examine the quality of an automatic support for the creation of interoperable Web services between the .NET and J2EE platform with regard to the developed classification approach. Therefore, the capabilities and usabilities of up to date Integrated Development Environments, namely IBM WebSphere 5.1 and Visual Studio.NET 2003, were examined. Concerning the practical test cases, interoperability in general is possible between both platforms. The honor goes to the work from the Web Service – Interoperability organization as they chose best practices and clarified the ambiguous specification passages for the core Web service protocols. However, for areas where no work has been done yet by WS-I the situation is different, the IDEs are mainly focused on their own platform interoperability. On one hand, this is understandable but when taking Web service's fundamental aim of interoperability into account, on the other hand, the automatic support for gaining interoperability between different platforms is still a major problem. This became especially obvious when the security test case was performed. Different concepts of providing X.509 certificates have been a problem as well as the inability to understand the competitor's SOAP header. Therefore, developers should keep an eye on the WS-I deliverables and prepare themselves for manual effort in case that no profile is available for the area of interest.

For the aspect of usability, the IDEs include support for basically the same concepts. Thus, minor differences regarding help system, Web service wizards or extra tools are negligible. The extremely plug-in based WSAD makes it possible to integrate all functionality into the IDE whereas separate programs are needed for Visual Studio. If that is advantage or drawback highly depends on the developer's favour. Personally, I prefer working with WSAD but the easy concept of integrating a Web service consumer into every kind of program in Visual Studio
showed the disadvantage of the WSAD approach which needs a special kind of project (Dynamic Web project). However, in most cases the question “WSAD or Visual Studio” will not occur as another question must be answered first: “.NET or J2EE” which has not been addressed in this thesis.

**Outlook**

As this thesis could not address all Views of the classification approach this could be a task for further work on this topic. Especially, the Device View with different end user devices could not be considered but also the Operational View with far more aspects than security could be an interesting starting point for further investigation. Additionally, the extraction of the Business View from the model, which resulted in a revised model, must be proven to be a benefit for the classification approach.

Furthermore, applying the model to a concrete project where interoperability is an issue would show its accuracy or the need for refining the Views. Hereby, the model could be used for structuring different problem domains and arranging the the actual test cases.
# Appendix A - .NET Framework Data Types

The following table gives an overview of the value types in the .NET framework. Instead of writing the .NET framework type (column two: Class name) the correspondent language keyword can be used (for example, column five: C# data type in C# programs). To avoid possible misunderstanding this table from the MSDN library [WWW19] is included here.

<table>
<thead>
<tr>
<th>Category</th>
<th>Class name</th>
<th>Description</th>
<th>Visual Basic data type</th>
<th>C# data type</th>
<th>Managed Extension for C++ data type</th>
<th>JScript data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>Byte</td>
<td>An 8-bit unsigned integer</td>
<td>Byte</td>
<td>byte</td>
<td>char</td>
<td>Byte</td>
</tr>
<tr>
<td></td>
<td>SByte</td>
<td>An 8-bit signed integer. Not CLS compliant</td>
<td>SByte</td>
<td>sbyte</td>
<td>signed char</td>
<td>SByte</td>
</tr>
<tr>
<td></td>
<td>Int16</td>
<td>A 16-bit signed integer.</td>
<td>Short</td>
<td>short</td>
<td>short</td>
<td>short</td>
</tr>
<tr>
<td></td>
<td>Int32</td>
<td>A 32-bit signed integer.</td>
<td>Integer</td>
<td>int</td>
<td>int</td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>Int64</td>
<td>A 64-bit signed integer.</td>
<td>Long</td>
<td>long</td>
<td>_int64</td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>UInt16</td>
<td>A 16-bit unsigned integer. Not CLS compliant.</td>
<td>UInt16</td>
<td>ushort</td>
<td>unsigned short</td>
<td>UInt16</td>
</tr>
<tr>
<td></td>
<td>UInt32</td>
<td>A 32-bit unsigned integer. Not CLS compliant.</td>
<td>UInt32</td>
<td>uint</td>
<td>unsigned int</td>
<td>UInt32</td>
</tr>
<tr>
<td></td>
<td>UInt64</td>
<td>A 64-bit unsigned integer. Not CLS compliant.</td>
<td>UInt64</td>
<td>ulong</td>
<td>unsigned _int64</td>
<td>UInt64</td>
</tr>
<tr>
<td>Floating point</td>
<td>Single</td>
<td>A single-precision (32-bit) floating-point number.</td>
<td>Single</td>
<td>float</td>
<td>float</td>
<td>float</td>
</tr>
<tr>
<td></td>
<td>Double</td>
<td>A double-precision (64-bit) floating-point number.</td>
<td>Double</td>
<td>double</td>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>Logical</td>
<td>Boolean</td>
<td>A boolean value (true or false)</td>
<td>Boolean</td>
<td>bool</td>
<td>bool</td>
<td>bool</td>
</tr>
<tr>
<td>Other</td>
<td>Char</td>
<td>A unicode (16-bit) character</td>
<td>Char</td>
<td>char</td>
<td>wchar_t</td>
<td>char</td>
</tr>
<tr>
<td>Category</td>
<td>Class name</td>
<td>Description</td>
<td>Visual Basic data type</td>
<td>C# data type</td>
<td>Managed Extension for C++ data type</td>
<td>JScript data type</td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------------------</td>
<td>--------------</td>
<td>-------------------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Decimal</td>
<td></td>
<td>A 96-bit decimal value.</td>
<td>Decimal</td>
<td>decimal</td>
<td>Decimal</td>
<td>Decimal</td>
</tr>
<tr>
<td>IntPtr</td>
<td></td>
<td>A signed integer whose size depends on the underlying platform (a 32-bit value on a 32-bit platform and a 64-bit value on a 64-bit platform).</td>
<td>IntPtr</td>
<td>IntPtr</td>
<td>IntPtr</td>
<td>IntPtr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No built-in type.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UIntPtr</td>
<td></td>
<td>An unsigned integer whose size depends on the underlying platform (a 32-bit value on a 32-bit platform and a 64-bit value on a 64-bit platform). Not CLS compliant.</td>
<td>UIntPtr</td>
<td>UIntPtr</td>
<td>UIntPtr</td>
<td>UIntPtr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No built-in type.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class objects</td>
<td>Object</td>
<td>The root of the object hierarchy.</td>
<td>Object</td>
<td>object</td>
<td>Object*</td>
<td>Object</td>
</tr>
<tr>
<td>String</td>
<td></td>
<td>An immutable fixed-length string of Unicode characters.</td>
<td>String</td>
<td>string</td>
<td>String*</td>
<td>String</td>
</tr>
</tbody>
</table>

Table 6 Value types in the .NET framework and some .NET languages
Appendix B - CD Content

The printed version of this thesis comprises a CD which contains additional material:

- The mentioned source code of this thesis.
- Programs which were helpful during the developing and testing phase.
- This thesis as pdf and zipped XML format (Open Office) and used pictures for further usage.
The bibliography is divided into three parts:

1. Referenced books

2. Referenced World Wide Web (WWW) - pages

3. Referenced technical reports

Whereas books and technical reports are not problematic because their authors are always specified, WWW-pages sometimes lack not only an author but they also tend to be fast moving and not available again on the other day. Nevertheless, working with Web services is an Internet-driven topic and most information is available online. Therefore, the author tried to collect the information about WWW references as complete as possible but in case no author or year was stated this is indicated by the term 'not specified'.

**Books:**

CER02: Cerami, Ethan, Web Services Essentials, 2002

ZIM03: Zimmermann, Olaf; Tomlinson, Mark; Peuser, Stefan, Perspectives on Web Services, 2003

**World Wide Web-pages:**

WWW01: no author specified, Glossary of Telecommunications Terms, no year specified, http://www.atis.org/tg2k/


WWW12: Jeckle, Mario, Definition of the term Web services, 2003, http://www.jeckle.de/webServices/index.html#def


**Technical Reports:**

KRE01: Kreger, Heather, Web Services Conceptual Architecture, 2001


## Alphabetical Index

<table>
<thead>
<tr>
<th>Term</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon</td>
<td>1</td>
</tr>
<tr>
<td>Apache Soap</td>
<td>47</td>
</tr>
<tr>
<td>Apache SOAP</td>
<td>31</td>
</tr>
<tr>
<td>Applet Container</td>
<td>28</td>
</tr>
<tr>
<td>Application Client Container</td>
<td>28</td>
</tr>
<tr>
<td>Application-To-Application architecture</td>
<td>6</td>
</tr>
<tr>
<td>ARPAnet</td>
<td>13</td>
</tr>
<tr>
<td>Attachments Profile</td>
<td>24</td>
</tr>
<tr>
<td>automated communication</td>
<td>9</td>
</tr>
<tr>
<td>automation</td>
<td>4</td>
</tr>
<tr>
<td>autonomous computer networks</td>
<td>13</td>
</tr>
<tr>
<td>Basic Profile</td>
<td>24</td>
</tr>
<tr>
<td>Basic Profile Testing Tools</td>
<td>24</td>
</tr>
<tr>
<td>non-intrusiveness</td>
<td>24</td>
</tr>
<tr>
<td>Basic Security Profile</td>
<td>24</td>
</tr>
<tr>
<td>BPEL4WS</td>
<td>35, 37</td>
</tr>
<tr>
<td>business logic</td>
<td>29</td>
</tr>
<tr>
<td>business processes</td>
<td>12</td>
</tr>
<tr>
<td>Certificate Authorities</td>
<td>27</td>
</tr>
<tr>
<td>Certificates</td>
<td>27</td>
</tr>
<tr>
<td>CIL</td>
<td>29</td>
</tr>
<tr>
<td>Common Intermediate Language</td>
<td>29</td>
</tr>
<tr>
<td>Classification Model</td>
<td>33</td>
</tr>
<tr>
<td>CLR</td>
<td>29</td>
</tr>
<tr>
<td>Common Language Runtime</td>
<td>29</td>
</tr>
<tr>
<td>COM</td>
<td>30</td>
</tr>
<tr>
<td>communication styles</td>
<td>30</td>
</tr>
<tr>
<td>container-managed clients</td>
<td>32</td>
</tr>
<tr>
<td>CORBA</td>
<td>1, 10</td>
</tr>
<tr>
<td>DCOM</td>
<td>10</td>
</tr>
<tr>
<td>definition</td>
<td>10</td>
</tr>
<tr>
<td>interoperability</td>
<td>14</td>
</tr>
<tr>
<td>Device View</td>
<td>35</td>
</tr>
<tr>
<td>Directory</td>
<td>6</td>
</tr>
<tr>
<td>DISCO</td>
<td>22</td>
</tr>
<tr>
<td>Distributed Component Object Model</td>
<td>8</td>
</tr>
<tr>
<td>distributed computing</td>
<td>1</td>
</tr>
<tr>
<td>Distributed Computing Environment</td>
<td>8</td>
</tr>
<tr>
<td>Ebay</td>
<td>1</td>
</tr>
<tr>
<td>Eclipse</td>
<td>30</td>
</tr>
<tr>
<td>encodings</td>
<td>39</td>
</tr>
<tr>
<td>engineering process</td>
<td>6</td>
</tr>
<tr>
<td>Enterprise JavaBeans Container</td>
<td>29</td>
</tr>
<tr>
<td>FCI</td>
<td>29</td>
</tr>
<tr>
<td>Framework Class Library</td>
<td>29</td>
</tr>
<tr>
<td>firewall</td>
<td>11</td>
</tr>
<tr>
<td>Forrester Research report</td>
<td>1</td>
</tr>
<tr>
<td>Google</td>
<td>1</td>
</tr>
<tr>
<td>green pages</td>
<td>22</td>
</tr>
<tr>
<td>HTML interface</td>
<td>9</td>
</tr>
<tr>
<td>HTTP</td>
<td>1, 16</td>
</tr>
<tr>
<td>HTTPS</td>
<td>24p.</td>
</tr>
<tr>
<td>Human-To-Application</td>
<td>12</td>
</tr>
<tr>
<td>Interface Description Language</td>
<td>7</td>
</tr>
<tr>
<td>IT infrastructures</td>
<td>5</td>
</tr>
<tr>
<td>J2EE</td>
<td>28</td>
</tr>
<tr>
<td>J2EE 1.4</td>
<td>32</td>
</tr>
<tr>
<td>JSR 109</td>
<td>32, 35</td>
</tr>
<tr>
<td>JSR 172</td>
<td>35</td>
</tr>
<tr>
<td>J2ME Web services</td>
<td>35</td>
</tr>
<tr>
<td>JVM</td>
<td>29</td>
</tr>
<tr>
<td>Java Remote Method Invocation</td>
<td>8</td>
</tr>
<tr>
<td>Java RMI</td>
<td>10</td>
</tr>
<tr>
<td>JAX-RPC</td>
<td>32</td>
</tr>
<tr>
<td>JSR 109</td>
<td>32, 35</td>
</tr>
<tr>
<td>managed modules</td>
<td>30</td>
</tr>
<tr>
<td>message-oriented style</td>
<td>39</td>
</tr>
<tr>
<td>message-oriented systems</td>
<td>5</td>
</tr>
<tr>
<td>MFC model</td>
<td>33</td>
</tr>
<tr>
<td>NFR</td>
<td>36</td>
</tr>
<tr>
<td>Non Functional Requirements</td>
<td>36</td>
</tr>
<tr>
<td>OASIS</td>
<td>11</td>
</tr>
<tr>
<td>Object Model Transformations</td>
<td>44</td>
</tr>
<tr>
<td>PKCS#12</td>
<td>27</td>
</tr>
<tr>
<td>Programming Concepts View</td>
<td>35</td>
</tr>
<tr>
<td>Programming Languages View</td>
<td>34</td>
</tr>
<tr>
<td>Protocol Stack View</td>
<td>34</td>
</tr>
<tr>
<td>Remote Procedure Call</td>
<td>7</td>
</tr>
<tr>
<td>Requestor</td>
<td>6</td>
</tr>
<tr>
<td>Return Of Investment</td>
<td>5</td>
</tr>
<tr>
<td>RPC</td>
<td>10</td>
</tr>
<tr>
<td>Section 5 encoding</td>
<td>39</td>
</tr>
<tr>
<td>semantic description</td>
<td>8</td>
</tr>
<tr>
<td>Service</td>
<td>6</td>
</tr>
<tr>
<td>Service Orientation</td>
<td>4p.</td>
</tr>
<tr>
<td>Service Oriented Architecture</td>
<td>5, 11</td>
</tr>
<tr>
<td>Simple SOAP Binding Profile</td>
<td>24</td>
</tr>
<tr>
<td>skeleton</td>
<td>7</td>
</tr>
<tr>
<td>Smalltalk</td>
<td>10</td>
</tr>
<tr>
<td>SMTP</td>
<td>16</td>
</tr>
</tbody>
</table>
2.1 SOA .......................................................... 5
requirements .................................................. 6
SOAP ......................................................... 9, 16, 24, 34
SOAP .......................................................... 18
Body .......................................................... 18
Envelope ..................................................... 18
Fault .......................................................... 18
Header ....................................................... 18
Intermediary ................................................ 18
Message Exchange ....................................... 18
Protocol Binding .......................................... 18
Request ...................................................... 18
Response .................................................... 19
stub ............................................................. 7
TCP/IP ......................................................... 1, 13
tight-coupled ............................................... 12
UDDI ......................................................... 11, 22, 24, 34
unmanaged clients ......................................... 32
usability ...................................................... 4
Virtual Machine ........................................... 28
Visual Studio .NET ......................................... 31
W3C .......................................................... 11
Web Container ............................................ 28
Web service definition ................................... 10
W3C .......................................................... 10
Web Service Standards .................................. 15
WebSphere Application Developer .................. 30
white pages ............................................... 22
WS-Addressing .......................................... 34
WS-Attachments ......................................... 34
WS-Authorization ........................................ 26
WS-Coordination ........................................ 36
WS-Federation ........................................... 26, 36
WS-I .......................................................... 24
Basic Profile .............................................. 24
WS-I Basic Profile 1.0 ................................... 24
WS-I Testing Tools ....................................... 49
WS-Inspection .......................................... 34
WS-Manageability ....................................... 36
WS-Policy ............................................... 25, 36
WS-Privacy ............................................... 25
WS-ReliableMessaging .................................. 36
WS-SecureConversation ................................. 26, 36
WS-Security ............................................... 25, 36
WS-Transaction ......................................... 35
WS-Trust .................................................. 25, 36
WSDK ........................................................
  Web Service Development Kit ...................... 31
WSDL ......................................................... 11, 19, 24, 34
WSDL ........................................................
  Bindings ............................................... 22
  endpoint ............................................... 22
  interface ............................................ 22
  Message .............................................. 22
  Operations .......................................... 22
  Part .................................................. 22
  Port .................................................. 22
  Port Type ............................................ 22
  Service .............................................. 22
  Types .................................................. 22
WSIF ......................................................... 34
WSIL ......................................................... 22
WSXL ....................................................... 35
X.509 ......................................................... 27, 49
XML encryption ......................................... 47
XML schema ............................................ 47
XML Schema ........................................... 34
XML signature .......................................... 47
XML-RPC .................................................. 11
yellow pages ............................................. 22