BAMBERGER BEITRÄGE

ZUR WIRTSCHAFTSINFORMATIK UND ANGEWANDTEN INFORMATIK

ISSN 0937-3349

Nr. 87

Analyzing the Interoperability of WS-Security and WS-ReliableMessaging Implementations

Johannes Schwalb and Andreas Schönberger

September 2010

FAKULTÄT WIRTSCHAFTSINFORMATIK UND ANGEWANDTE INFORMATIK

OTTO-FRIEDRICH-UNIVERSITÄT BAMBERG

Distributed Systems Group

Otto-Friedrich Universität Bamberg

Feldkirchenstr. 21, 96052 Bamberg, GERMANY

Prof. Dr. rer. nat. Guido Wirtz

`http://www.uni-bamberg.de/pi/`

Due to hardware developments, strong application needs and the overwhelming influence of the net in almost all areas, distributed systems have become one of the most important topics for nowadays software industry. Owing to their ever increasing importance for everyday business, distributed systems have high requirements with respect to dependability, robustness and performance. Unfortunately, distribution adds its share to the problems of developing complex software systems. Heterogeneity in both, hardware and software, permanent changes, concurrency, distribution of components and the need for inter-operability between different systems complicate matters. Moreover, new technical aspects like resource management, load balancing and guaranteeing consistent operation in the presence of partial failures and deadlocks put an additional burden onto the developer.
The long-term common goal of our research efforts is the development, implementation and evaluation of methods helpful for the realization of robust and easy-to-use software for complex systems in general while putting a focus on the problems and issues regarding distributed systems on all levels. Our current research activities are focussed on different aspects centered around that theme:

- *Reliable and inter-operable Service-oriented Architectures*: Development of design methods, languages, tools and middle-ware to ease the development of SOAs with an emphasis on provable correct systems that allow for early design-evaluation due to rigorous development methods. Additionally, we work on approaches and standards to provide truly inter-operable platforms for SOAs.
- *Implementation of Business Processes and Business-to-Business-Integration (B2Bi)*: Starting from requirements for successful B2Bi development processes, languages and systems, we investigate the practicability and inter-operability of different approaches and platforms for the design and implementation of business processes with a focus on combining processes from different business partners.
- *Quality-of-Service (QoS) Aspects for SOA and B2Bi*: QoS aspects, especially reliability and security, are indispensable when putting distributed systems into practical use. We work on methods that allow for a seamless observance of QoS aspects during the entire development process from high-level business processes down to implementation platforms.
- *Agent and Multi-Agent (MAS) Technology:* Development of new approaches to use Multi-Agent-Systems for designing, organizing and optimizing complex systems ranging from service management and SOA to electronic markets and virtual enterprises.
- *Visual Programming- and Design-Languages*: The goal of this long-term effort is the utilization of visual metaphors and languages as well as visualization techniques to make design- and programming languages more understandable and, hence, more easy-to-use.

More information about our work, i.e., projects, papers and software, is available at our homepage (see above). If you have any questions or suggestions regarding this report or our work in general, don't hesitate to contact me at `guido.wirtz@uni-bamberg.de`

Guido Wirtz

Bamberg, January 2010

Analyzing the Interoperability of WS-Security and WS-ReliableMessaging Implementations

Johannes Schwalb and Andreas Schönberger

Lehrstuhl für Praktische Informatik, Fakultät WIAI

Abstract Since their invention as lightweight integration technology about a decade ago, Web Services have matured significantly. Today, major middleware solution vendors as well as industry communities like RosettaNet are propagating Web services even for exchanging business-critical data and implementing inter-organizational business processes. Core enablers for using Web services in this domain are stateful interactions using the Web Services Business Process Execution Language (WS-BPEL) as well as advanced communication features like security and reliability using the WS-Security and WS-ReliableMessaging standard specifications. However, advanced communication features come at the price of complexity which challenges interoperability across different Web services stack implementations. Interoperability, in turn, is a predominant requirement for an integration technology such as Web services, in particular if inter-organizational business processes are supposed to be implemented on top of that technology.

This paper approaches the problem of testing the interoperability of the so-called WS-* standards, advanced Web services communication features that are typically defined as SOAP extensions and configured using WS-Policy. Being essential to business process integration, WS-Security and WS-ReliableMessaging are selected as representatives of this group and the two major Java-based Web services stack implementations Metro and Axis2 are tested for interoperability. We operationalize the notion of interoperability for testing WS-* standards, suppose an approach for deriving test cases from WS-* specifications as well as a method for performing the test cases, and we provide a comprehensive interoperability review of the two selected Web services stack implementations.

Keywords WS-Security, WS-ReliableMessaging, Quality-of-Service, Interoperability, Web Services, Testing

Contents

List of Figures

List of Tables

List of Listings

Notational Conventions

This work uses the following syntax within listings that define the structure of XML elements:

- Characters are appended to elements and attributes to indicate their cardinality. An element or attribute without one of the following characters occurs exactly once.
  - “`?`” denotes zero or one occurrence
  - “`+`” denotes one or more occurrences
  - “`*`” denotes zero or more occurrences
- “`|`” indicates a choice between alternatives
- Items in brackets (“`(`” and “`)`”) are treated as group with respect to cardinality or choice.

Listings using this syntax, are marked as “Structure of ...”. Other XML code, such as XML message examples, does not make use of this syntax.

The syntax defined above follows to the notational conventions of OASIS and W3C documents.

Namespaces

The Table below lists the XML namespaces that are used in this work. The choice of any namespace prefix is arbitrary and not semantically significant, however the namespace prefixes defined below are used within the scope of this work.

Prefix	**Namespace**	**Specification**
`bpel`	`http://docs.oasis-open.org/wsbpel/2.0/process/executable`	WS-BPEL
`ds`	`http://www.w3.org/2000/09/xmldsig#`	XML Signature
`env`	`http://www.w3.org/2003/05/soap-envelope`	SOAP
`sp`	`http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702`	WS-SecurityPolicy
`wsa`	`http://www.w3.org/2005/08/addressing`	WS-Addressing
`wsam`	`http://www.w3.org/2007/05/addressing/metadata`	WS-Addressing
`wsc`	`http://docs.oasis-open.org/ws-sx/ws-secureconversation/200512`	WS-Secure-Conversation
`wsdl`	`http://schemas.xmlsoap.org/wsdl`	WSDL
`wsrm`	`http://docs.oasis-open.org/ws-rx/wsrm/200702`	WS-Reliable-Messaging
`wsrmp`	`http://docs.oasis-open.org/ws-rx/wsrmp/200702`	WS-RM Policy
`wsp`	`http://www.w3.org/ns/ws-policy`	WS-Policy
`wss`	`http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd`	WS-Security
`wst`	`http://docs.oasis-open.org/ws-sx/ws-trust/200512`	WS-Trust
`wst14`	`http://docs.oasis-open.org/ws-sx/ws-trust/200802`	WS-Trust
`wsu`	`http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd`	WS-Security
`xenc`	`http://www.w3.org/2001/04/xmlenc#`	XML Encryption

List of Acronyms

API	Application Programming Interface
B2B	business-to-business
B2Bi	business-to-business integration
BC	Binding Component (JBI)
BPEL	Business Process Exceution Language, short for WS-BPEL
BPEL4WS	Business Process Execution Language for Web services, now WS-BPEL
C14N	Canonicalization
CA	certification authority
EAI	enterprise application integration
ebBP	ebXML Business Process
ebXML	Electronic Business using XML
EJB	Enterprise JavaBeans
ESB	Enterprise Service Bus
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure (HTTP over SSL)
IDE	Integrated Development Environment
IUT	implementation under test
JAXB	Java Architecture for XML Binding
JAX-RPC	Java API for XML-Based RPC
JAX-WS	Java API for XML-Based Web Services
JBI	Java Business Integration
J2EE	Java Platform 2, Enterprise Edition, now JEE
JDK	Java Development Kit
JEE	Java Platform, Enterprise Edition, also Java EE, previously J2EE
JMS	Java Message Service

JRE	Java Runtime Environment
JSP	Java Server Pages
J2SE	Java Platform 2, Standard Edition, now JSE
JSE	Java Platform, Standard Edition
JVM	Java Virtual Machine
LTPA	Lightweight Third-Party Authentication
MCC	Message Control and Choreography (RosettaNet)
MEP	message exchange pattern
MIME	Multipurpose Internet Mail Extensions
MTOM	SOAP Message Transmission Optimization Mechanism
OASIS	Organization for the Advancement of Structured Information Standards
OSGi	Open Services Gateway initiative (Organization), meaning dropped
OSOA	Open Service Oriented Architecture (Organization)
PC	personal computer
QName	qualified name
QoS	quality of service
RAD	IBM Rational Application Developer for WebSphere Software
REL	Rights Expression Language
RMI	Remote Method Invocation
RPC	remote procedure call
RST	`wst:RequestSecurityToken` element (WS-Trust)
RSTC	`wst:RequestSecurityTokenCollection` element (WS-Trust)
RSTR	`wst:RequestSecurityTokenResponse` element (WS-Trust)
RSTRC	`wst:RequestSecurityTokenResponseCollection` element (WS-Trust)
SAML	Security Assertion Markup Language
SCA	Service Component Architecture
SCT	Security Context Token (WS-SecureConversation)
SDO	Service Data Objects

SE	Service Engine (JBI)
SHA	Secure Hash Algorithm
SMTP	Simple Mail Transfer Protocol
SOA	service-oriented architecture
SOAP	Simple Object Access Protocol, meaning dropped since Version 1.2
SSL	Secure Sockets Layer
StAX	Streaming API for XML
STR	Security Token reference
STS	Security Token Service (WS-Trust)
SwA	SOAP with Attachment
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium (Organization)
WAS	IBM WebSphere Application Server
WESB	IBM WebSphere Enterprise Service Bus
WID	IBM WebSphere Integration Developer
WPS	IBM WebSphere Process Server
WSDL	Web Services Description Language
WSIT	Web Services Interoperability Technologies
WS-BPEL	Web Services Business Process Execution Language, previously BPEL4WS
WS-I	Web Services Interoperability Organization
WS-Stack	Web services stack
XML	Extensible Markup Language
XMLNS	XML namespace
XPath	XML Path Language
XSD	XML Schema Definition
XSLT	Extensible Stylesheet Language Transformations

1 Introduction

Nowadays, an increasing amount of software is designed in a distributed way. Service-oriented architecture (SOA) is one of the most important architecture paradigm in distributed systems. Today, there are various definitions of the term "SOA". However, nearly every definition refers to widely accepted central points of service-orientation. These are:

- distributed services as units of logic
- loose coupling of services
- composability of services

On the basis of these central points every author extends his own definition of SOA. A widely accepted definition of the term SOA is given by the Organization for the Advancement of Structured Information Standards (OASIS)[1] in the *Reference Model for Service Oriented Architecture 1.0*: SOA is "[..] a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains." [OAS06b, page 8, lines 128/129]

Other definitions include additional points, but all definitions identify "distributed capabilities" (better known as *services*) as essential part of a SOA. The central idea of a SOA is to decompose a large and complex problem into a collection of smaller and well-understood pieces, the so-called services. The solutions of these subproblems are then assembled to the solution of the complex problem.

A service in terms of service-orientation is an independent software program which offers a certain functionality, requires a specified input and returns a result in a defined format as output. This information is published in a so-called *service contract*, a published meta information document about the service. This contract can also contain additional information about quality of service (QoS) features, behaviors and limitations of the service.[2] The service contract is a formal contract and is the only part of the service that is visible to potential service requesters, thus, the contract abstracts from the underlying logic.[3] Due to the fact that services may vary in their complexity from very simple functions up to complex actions, a service needs to be composable from other services in order to sustain the idea of simplicity.

However, the SOA concept is still a theoretical concept. Following only the SOA principles listed above, every company and every software architect would implement their own SOA, but these SOAs would not tend to be interoperable. Therefore, standards are needed to implement a SOA across the boundaries of companies. The World Wide Web Consortium (W3C)[4] provides *Web services* [W3C04a] as one possible realization of SOA. The interface of a Web service is defined by the Web Services Description Language (WSDL) [W3C01], an Extensible Markup Language

[1]`http://www.oasis-open.org`

[2]Thomas Erl, *Web Service Contract Design and Versioning for SOA*, page 35, Prentice Hall, Upper Saddle River, New Jersey, USA, 2008.

[3]Thomas Erl, *Service-Oriented Architecture - Concepts, Technology, and Design*, pages 298 - 301, Prentice Hall, Upper Saddle River, New Jersey, USA, 2005.

[4]`http://www.w3c.org`

(XML)-based solution to write service contracts for Web services. The WSDL interface of a Web service contains all the information needed to invoke the service [FF03]. Most programming languages, development tools, and middleware solutions already support the WSDL as interface language for Web services [Kre03]. By decoupling implementation and interface, standardizing the interface description, and providing the interface over the Internet or an intranet, the Web services technology is ready to provide services in a SOA.

In order to build a SOA, some aspects are still missing. Due to the fact that Web services are often seen "as the foundation of a new generation of business-to-business (B2B) and enterprise application integration (EAI) architectures" [Kre03], the Web services technology needs to be extended. Web services are not able to solely guarantee all features of a SOA. Therefore, e.g., the Web Services Business Process Execution Language (WS-BPEL) [OAS07] provides functionality to create automated business processes (so-called *workflows*) that orchestrate Web services and again offer this workflow as a Web service.[5] Another important standard extending the Web services technology is the *Simple Object Access Protocol* (SOAP)[6] [W3C07b], a transport protocol used to transmit XML messages between Web services. Besides the transmitted information, SOAP may contain extensions to the message format decoupled from the application data, e.g., functionality for security, reliability, or object references.[7] Due to its independence of operating systems and programming environments, SOAP is the de facto standard messaging protocol for Web services.[8] Thomas Erl states that "Contemporary SOA is intrinsically reliant on Web services - so much so that Web services concepts and technology used to actualize service-orientation have influenced and contributed to a number of common SOA characteristics [...]. An understanding of SOA begins with a close look at the overall framework that has been established by the [...] Web services extensions."[9]

WSDL, SOAP and WS-BPEL extend Web services to a framework of software, supporting interoperable and fully automatic B2B interactions across the boundaries of companies. However, this is insufficient for meeting the business requirements for the transmission of critical business documents. QoS properties like integrity, reliability, interoperability, and confidentiality of the transmitted messages [OMB07] are needed. For example, a legal contract sent from company A to company B has to be protected against disclosure of information to unauthorized individuals or systems, the reliability of message delivery must be fulfilled, and the non-repudiation of the exchanged documents may be required [Ros10, pages 20/21] [OAS06a, pages 71-73].

These requirements can be partially met with the help of secured transport protocols (e.g., Transport Layer Security (TLS)/Secure Sockets Layer (SSL)), but these security mechanisms only provide point-to-point protection. Since intermediaries are designated to be used in Web services communication, a secure and reliable point-to-point connection is insufficient to avoid message disclosure or message loss. Therefore, mechanisms guaranteeing QoS features on the

[5]Michael P. Papazoglu, *Web Services: Principles and Technology*, pages 307 - 369, Pearson Education, Harlow, England, 2008

[6]Meaning of the acronym dropped since Version 1.2, since then just SOAP

[7]Michael P. Papazoglu, *Web Services: Principles and Technology*, pages 128/129, Pearson Education, Harlow, England, 2008

[8]Michael P. Papazoglu, *Web Services: Principles and Technology*, page 120, Pearson Education, Harlow, England, 2008

[9]Thomas Erl, *Service-Oriented Architecture - Concepts, Technology, and Design*, page 110, Prentice Hall, Upper Saddle River, New Jersey, USA, 2005.

messaging level (end-to-end) are required [Bro03].

For this reason, a set of open standards, the so-called *WS-**[10] frameworks, has been developed. These standards bring QoS aspects into the Web services world. The inclusion of QoS features in Web services shows a certain kind of maturity transformation: Web services have grown from an experimental technology to an operable business-to-business integration (B2Bi) solution [Lud03].

The requirement for the use of WS-* standards can be included in the WSDL document using the WS-Policy framework [W3C07i]. Typically, most Web services platforms already include an implementation of the WS-* standards, the so-called *Web services stack (WS-Stack).* Thus, a Web services developer can rely on these implementations in order to provide QoS aspects for Web services.

[Lea04] states that "Web services have no value if they're not interoperable, and interoperability is based on standards compliance." The interoperability of Web services using different platforms, such as the Java Platform, Enterprise Edition (JEE) or the Microsoft .NET Framework, has already been discussed in scientific publications, e.g., [SV09], but the interoperability of complete WS-Stack implementations is still a problem. There are different sources for interoperability problems, e.g., every implementor of a standard has his own interpretation of the standard, the implementations are based on different versions of the standard, or the standards itself are ambiguous. Nonetheless, the interoperability of WS-* implementations has to be guaranteed in order to meet B2Bi requirements.

This problem has already been recognized by science and industry. For example, the Web Services Interoperability Organization (WS-I)[11] tries to solve this problem by publishing best practices for the "Web of services"[12]. The WS-I is "an open industry organization chartered to establish Best Practices for Web services interoperability, for selected groups of Web services standards, across platforms, operating systems and programming languages"[13]. However, the WS-I only provides best practices for Web services interoperability and does not provide interoperability tests of different WS-* framework implementations. Hence, an interoperability analysis of different WS-* frameworks is still necessary in spite of accordance with one of the WS-I profiles.

The WS-I defines, among other things, a "Reliable Secure Profile"[14] which covers one of the most important Web services QoS scenarios, the so-called *Secure WS-ReliableMessaging Scenario* [BMPV06] [GHB05] [WS-08]. This scenario combines features of WS-ReliableMessaging [OAS09b] and WS-Security [OAS06f] (including the WS-Trust [OAS09e] and WS-SecureConversation [OAS09c] sub-standards), which provide functionalities to guarantee a reliable and secure transmission of SOAP messages in a Web services environment. This scenario covers typical

[10]the term *WS-** is a commonly used abbreviation for the set of Web services standards beginning with the prefix "WS-"

[11]`http://www.ws-i.org`

[12]"Web of Services refers to message-based design frequently found on the Web and in enterprise software. The Web of Services is based on technologies such as HTTP, XML, SOAP, WSDL, SPARQL, and others." Taken from `http://www.w3.org/standards/webofservices`

[13]Taken from the "WS-I Overview Data Sheet"

[14]Documents available at `http://ws-i.org/deliverables/workinggroup.aspx?wg=reliablesecure`

QoS requirements in B2Bi for the transmission of business documents, as defined in the RosettaNet[15] Message Control and Choreography (MCC) - Profile-Web Services [Ros10, pages 20/21] and the OASIS ebXML Business Process (ebBP) [OAS06a, pages 71-73] specifications.

Therefore, this work provides an interoperability analysis of WS-Security and WS-ReliableMes saging implementations in different WS-BPEL engines to answer the question: "Is it possible to establish a secure and reliable connection in a Web services based SOA solution using WS-Security and WS-ReliableMessaging based on different Web services stack (WS-Stack) implementations?"

A scientific approach to answer these questions requires the definition of the terms *interoperability* and *compatibility* in the context of Web services protocols or, more general, in the context of communication protocols. The main target of this work is to provide a compatibility analysis which "refers to assessing if two services, characterized by specific protocols, can interoperate, and which conversations are and are not possible." [BCT06] This definition already includes the term *interoperate* that expresses the ability of software systems to exchange information through compliance with technical (interoperability) specifications[16]. Thus, this work determines whether two Web services frameworks are interoperable and which conversations are possible between those frameworks. For this reason, the following approach has been chosen.

First of all, the fundamental Web services and XML technologies in the context this work are briefly introduced in section 2. Section 3 gives a short overview of the test environment, consisting of the hardware and the selected test platforms (IBM WebSphere and Sun GlassFish). The main contribution of this work is located in sections 4 to 6. At first, a test method for WS-* interoperability testing is developed in section 4. This method defines the approach to the tests, i.e., the test setup, the testing sequence, the general setting, etc. Besides simple isolated function tests, the test method also provides test cases including multiple WS-Security/WS-ReliableMessaging features and a practical use case, covering the practicability of WS-* frameworks in a typical B2Bi scenario. Section 5 provides an overview of the necessary preparatory measures for the test execution, whereas section 6 presents the design and execution of the test cases. Publications related to this work are presented and discussed in section 7. Section 8 concludes this work by giving a short summary of the results obtained and an outlook to future work.

[15]`http://www.rosettanet.org`

[16]Following the definition of `http://en.wiktionary.org/wiki/interoperability`

2 Fundamentals

This section provides an overview of the fundamental SOA technologies that are employed in this work. This is necessary, since the Web services technology has evolved from a very simple technology to a complex stack of technologies (see figure 1 for a simplified illustration of the WS-Stack, the figure does not claim to be complete with regard to concrete protocols).

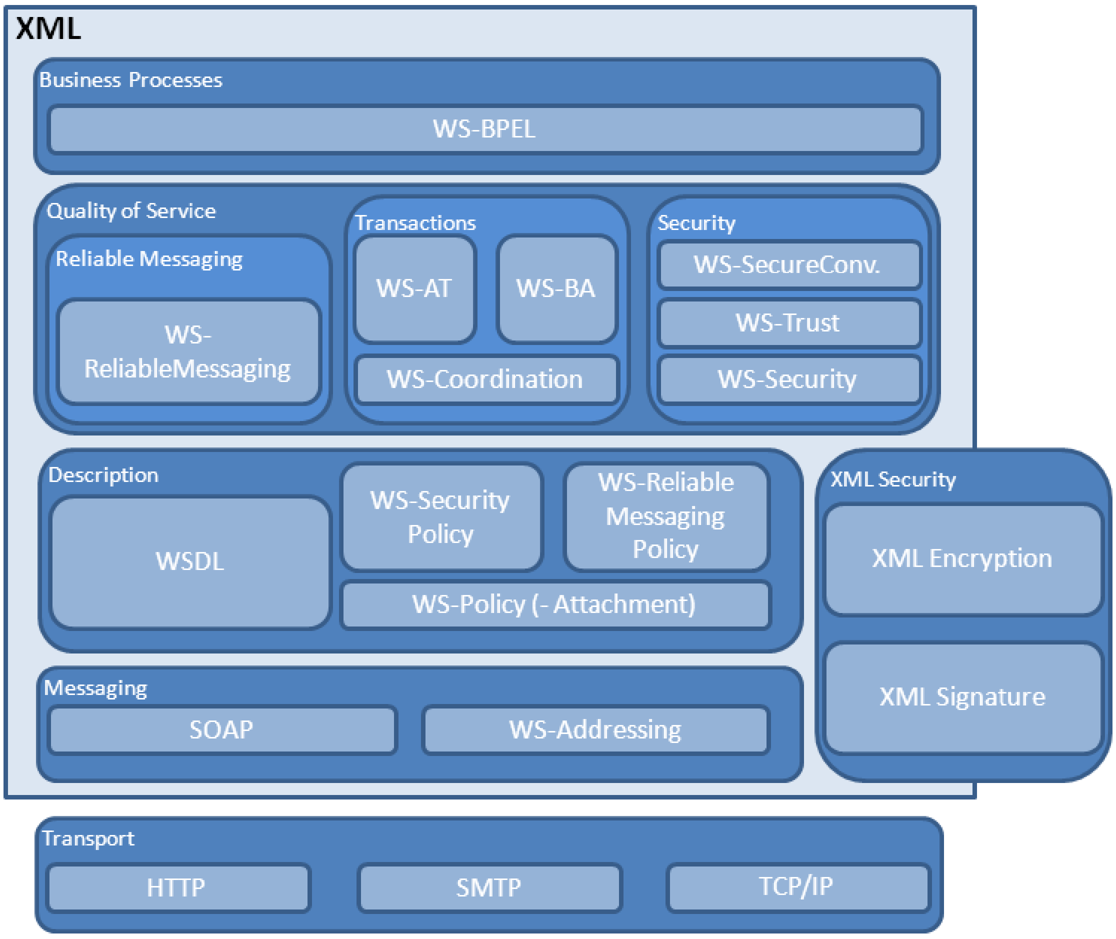

Figure 1: The Web services stack (WS-Stack), simplified illustration

The Web services technology stack (see Figure 1) is based on two enabling technologies: Transport protocols, on the one hand, and Extensible Markup Language (XML), on the other hand. Although SOAP is *the* standard messaging protocol for Web services, it requires an underlying transport protocol for message delivery, such as Simple Mail Transfer Protocol (SMTP), Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), or Transmission Control Protocol (TCP). SOAP and any other Web services technology are based on XML. WSDL and WS-Policy describe the functional and non-functional capabilities of a Web service. By defining a WS-Policy assertion in a WSDL document QoS aspects (e.g., WS-ReliableMessaging for reliable message delivery, WS-Security for message security, WS-Coordination for distributed transaction support) can be guaranteed for Web services invocations. Web services with and without QoS support can be orchestrated using the Web Services Business Process Execution

Language (WS-BPEL), which provides this orchestration as a Web service (including a WSDL interface) for invocation in turn.

The further structure of this section is as follows: At first, a collection of references provides an introduction to XML as basic technology of all SOA technologies in section 2.1. Afterwards, the *core service standards* for message transportation (SOAP), service description (WSDL), and service composition (WS-BPEL) are considered in section 2.2. Section 2.3 presents a short introduction to XML Signature and XML Encryption as XML Security technologies. Finally, section 2.4 presents the most important WS-* standards in the context of this work.

2.1 Extensible Markup Language

As already mentioned, the Extensible Markup Language (XML) is a basic technology for the Web services technology stack. XML is a W3C Recommendation of a formal, human-readable language for straightforward use over the Internet structuring data in a tree format. This tree structure consists of nodes which are, i.a., elements or attributes of these elements (attributes and elements form the so-called *markup vocabulary*) [W3C08a].

The XML standard restricts the structure of an XML document by defining simple well-formedness rules. However, more expressive means to define syntactic validity are needed in many situations. The XML Schema Definition (XSD) [W3C04b] [W3C04c] [W3C04d] has been developed for this purpose: it defines a language for XML data type definitions, XML structure descriptions, and constraints to the contents of an XML document. Thus, XSD adapts the markup vocabulary of XML to define generally intelligible XML documents and enables the B2Bi use of XML with XSD-defined schemas.

The markup vocabulary of an XML document may be defined and used by multiple software modules utilizing the concept of XML namespace (XMLNS). A namespace is identified by an Uniform Resource Identifier (URI). Qualified Names (QNames) are URI references used to identify elements and attributes within XML documents instead of namespace definitions [W3C09].

Based on these core specifications of XML, many other specifications have been developed. One of these specifications is the XML Path Language (XPath) specification. The concept of this language is analogous to that of the QNames, as both are employed to identify message parts. In contrast to QNames, the XML Path Language utilizes the tree structure for selecting nodes from an XML document [W3C07j].

For a detailed introduction to the technologies mentioned in this section consult the corresponding standards.

2.2 Basic Web Services Technologies

Based on XML, many languages, standards, and recommendations in the SOA environment have been developed. For the majority of SOA authors, the primary set of Web services standards consists of WSDL for service description, SOAP as a transport protocol for Web services XML messages, and UDDI for service publication and discovery. In addition to these three standards, WS-BPEL, as Web services composition language, is a fundamental prerequisite for this work. This section successively introduces WSDL, SOAP, and WS-BPEL as basic Web services technologies[17].

2.2.1 Web Services Description Language (WSDL)

As already discussed, a service in a SOA needs a service contract, which describes the (non-)functional requirements and capabilities of a service. The service contract must be written in a generally accepted and machine-readable description language. The XML meets most of these requirements, employing the XSD language to define the structure of the exchanged messages. However, this is not sufficient to describe all aspects for service invocation. Web services as a service realization employ the Web Services Description Language (WSDL) [W3C01] for this purpose. Although WSDL version 2.0 [W3C07f] [W3C07e] [W3C07g] is already available, most platforms primarily support WSDL version 1.1, which is therefore introduced below.

WSDL is an XML-based description language to define the structure of the messages the service sends and accepts, to describe *what* the service does, to specify *how* the service can be invoked, and to publish *where* the service can be accessed.

WSDL does not define a distinct vocabulary for the message structure, but uses the XSD standard. The `wsdl:types` tag can contain several XSD-conform definitions of data types as well as a reference to an external XSD document. Using these type definitions, the concrete messages of a service are described in the `wsdl:message` tag. A `wsdl:message` consists of several message parts, each `wsdl:part` element corresponds to an XSD type or an XSD element. A WSDL document can contain several `wsdl:message` tags, since more than one operation may be defined within a document. These message types are used to describe the Web service interface, the so-called `wsdl:portType` element, which envelops a set of operations, each identified by an `wsdl:operation` tag. An `wsdl:operation` is defined by its `wsdl:input`, `wsdl:output`, and `wsdl:fault` messages, which employ the `wsdl:message` definitions. WSDL intends four message exchange patterns (MEPs) for an operation. They are: one-way, request-response, solicit-response, and notification. Whereas the service acts as receiver in the first two cases, the latter two send a message to a receiver.

Up to this point the functional capabilities of a Web service are described, a Web service consumer now has information about the messages and operations provided by the service. Thus,

[17]Although Universal Description, Discovery and Integration (UDDI) definitely belongs to the set of core service standards it is not used in the scope of this work. Therefore, I desist from introducing UDDI. Consider the UDDI standard [OAS02] or Michael P. Papazoglu, *Web Services: Principles and Technology*, Pearson Education, Harlow, England, 2008, for a detailed introduction to UDDI.

the interface description of the service is complete, but information about which concrete service implementation is bound to the interface is still missing. Therefore, WSDL binds the abstract information in the interface to a concrete implementation using the `wsdl:binding` and the `wsdl:service` element. A `wsdl:binding` specifies concrete transport protocols for the `wsdl:portType`, the `wsdl:operation`, and the transmission of the `wsdl:input`, `wsdl:output`, and `wsdl:fault` messages. Each possible Web services transportation protocol, therefore, defines its own binding with its own namespace, e.g., the `wsdl:binding` for SOAP has the XMLNS `http://schemas.xmlsoap.org/wsdl/soap/` for the WSDL 1.1. The `wsdl:service` element binds the Web service to an accessible location in a network. To realize this, the `wsdl:service` element contains one or more `wsdl:port` elements, which represents an endpoint realizing an interface. A `wsdl:port`, therefore, references a `wsdl:binding`.

Thus, WSDL proposes an incremental approach to describe a Web service. At first, the interface (`wsdl:portType`) with its operations is defined. Afterwards, concrete protocols are bound to the operations (`wsdl:binding`), and a concrete network address for the endpoint is specified (`wsdl:port`).

The most common transport protocol in Web services environments is SOAP, which is presented in section 2.2.2.

2.2.2 SOAP

SOAP [W3C07a] [W3C07b] [W3C07c] originally stood for *Simple Object Access Protocol*. However, the meaning of this acronym has been canceled with SOAP version 1.2 so that there is currently no meaning behind this acronym. In any case, the SOAP standard defines an XML based message format that is not bound to a concrete network transport protocol. Therefore, a SOAP message must be embedded in a transport protocol, such as the well-known Internet protocols HTTP, SMTP, or TCP, but also transport protocols like Remote Method Invocation (RMI) or Java Message Service (JMS).

SOAP is designed as a simple, stateless, one-way message exchange protocol between a SOAP sender and a SOAP receiver, whereas the SOAP W3C recommendation differentiates between intermediaries and ultimate recipients (sender, recipients, and intermediaries are refered to as *SOAP nodes*). For each message transmission between a SOAP sender and a SOAP receiver (the so-called *hop*) the SOAP message may be bound to an other transport protocol. It is hence possible that an initial sender sends a SOAP message to intermediary A via HTTP which forwards this SOAP message to intermediary B using the TCP. In this case, the ultimate recipient may receive the SOAP message bound to a JMS message.

But this simple one-way SOAP MEP is only the basic scenario. It is also possible to establish more complex interaction patterns using application logic behind the SOAP message. In this way, SOAP is able to realize MEPs from simple request/response transmissions up to multiple, conversational exchanges. The SOAP W3C recommendation does not provide any specific communication protocols, so that the concrete realization of those message exchanges can be developed depending on the type of use.

In contrast to that, the structure of a SOAP message is independent of concrete usage. Each SOAP message is an XML document with a mandatory root element, the so-called SOAP `env:Envelope`. This envelope contains two elements defined by SOAP: the optional SOAP `env:Header` and the obligatory SOAP `env:Body`.

The SOAP `env:Header` element can contain several namespace qualified header blocks, each encapsulating meta-information about the message, such as routing information, security extensions, references, etc. The W3C recommendation for the SOAP does not define any limitations about the content of a header element, except that header blocks must not carry payload information. Each header block may include several attributes defined in the SOAP namespace, i.a., the `env:mustUnderstand` attribute, which is a boolean attribute to indicate whether the processing of the corresponding header block is optional or not. If this attribute has the value `true` (the value "1" is also possible, but not recommended), each SOAP receiver, even intermediaries, must process the associated header block. In case the header block cannot be processed by the recipient, a fault message is returned to the SOAP sender.

Whereas the optional SOAP `env:Header` contains meta-information about the payload, the mandatory SOAP `env:Body` is intended to carry the payload. The payload consists of an arbitrary number of body entries, each namespace qualified. One possible body entry is a SOAP `env:Fault`. If a SOAP message encapsulates a fault, it must not contain any other payload. Listing 1 shows an exemplary SOAP message.

```
<env:Envelope>
   <env:Header>
      <!-- SOAP message header -->
   </env:Header>
   <env:Body>
      <!-- SOAP message body -->
   </env:Body>
</env:Envelope>
```

Listing 1: Example of a SOAP message

In practical use, new requirements concerning the message delivery arise. In order to meet these requirements, extensions to the SOAP are employed. One extension is the SOAP with Attachment (SwA) [W3C00] which allows the attachment of different kinds of data to a SOAP message. In this way, SOAP messages are no more limited to XML representations of data, but can include, e.g., engineering drawing, legal documents, or images.

The versatility of SOAP predestines it for the use in strongly heterogeneous environments. The modular, simple, and extensible structure of SOAP enable the flexible use in B2Bi. Due to the wide spread of the HTTP as standard Internet protocol, "SOAP over HTTP" is established as *de facto* standard transportation protocol in SOAs.

2.2.3 Business Process Execution Language (WS-BPEL)

SOAP and WSDL provide the infrastructure for a Web services call. However, these standards do not define how services can be composed in order to execute business processes or, to be more precise, workflows automatically. At this point, the Web Services Business Process Execution Language (WS-BPEL), often just called BPEL, comes into play. WS-BPEL is an XML-based [OAS07] flow language for specifying interaction protocols (the so-called *WS-BPEL processes*) by orchestrating Web services. Instead of focusing on functionality for application logic, WS-BPEL concentrates on reusing existing application logic available as services. In order to deal with the heterogeneity of distributed services, WS-BPEL employs WSDL for abstracting from service implementation details. Since complex workflows may include hundreds of service calls and control constructs, WS-BPEL provides a simple strategy to reduce the complexity of a WS-BPEL process: each WS-BPEL process has its own WSDL interface and is, therefore, also accessible by another WS-BPEL process or a process consumer as a service.

A WS-BPEL process defines a so-called `bpel:partnerLink` which is a WSDL interface for process consumers, via which a SOAP message arrives in the process. The incoming message is processed in a `bpel:receive` block. `bpel:assign` blocks are used to assign incoming values to process variables (`bpel:variables` with one or more `bpel:variable` sub-elements) and for variable transformations. The `bpel:invoke` element is employed to call external Web services (also called `bpel:partnerLink`) using the `bpel:variables`. Another `bpel:assign` block assigns the process variables to the result variable of the whole process that is sent back to the process initiator using the `bpel:reply` block. This short introduction to the very basic constructs of the WS-BPEL represents a basic WS-BPEL process. However, the WS-BPEL provides constructs for more sophisticated processes, e.g., handlers for events and faults, while loops, switches, or sequences [OAS07].

These constructs allow for the design of more complex business processes as WS-BPEL processes. For a detailed introduction to WS-BPEL refer to the specification [OAS07] or the relevant literature[18].

2.3 Basic XML Security Technologies

Due to the fact that many business processes are based on written, signed documents, any electronical business (e-business) enhancement, dealing with electronical documents, has to guarantee the same qualities that are commonly seen in written documents. Traditional business correspondence employs signatures and the secrecy of the post to ensure integrity, confidentiality, and authenticity of the corresponding parties. Because of falsification of documents and forging of signatures, these qualities cannot be guaranteed by written documents. However, these qualities also have to be guaranteed in e-business [Sur10].

XML as the basis for e-business technologies, provides specifications for these use cases. Two of these specifications belong to the group of XML security technologies: XML Signature [W3C08b] and XML Encryption [W3C02]. Both are employed by WS-Security (see section 2.4) to guarantee integrity resp. confidentiality of XML documents.

This section only gives a short overview of the structure and the processing of XML Signature

[18]For example, Michael P. Papazoglu, *Web Services: Principles and Technology*, Pearson Education, Harlow, England, 2008.

and XML Encryption and does not discuss security considerations or the use of certain algorithms. Refer to the corresponding standards or the relevant literature[19] for a more detailed discussion of the XML security standards.

2.3.1 XML Signature

XML Signature defines an XML syntax for the integrity protection of data. Like digital signatures, XML Signatures ensure message integrity, the identity of the signer, and the non-repudiation of the message. The XML Signature technology can be used to sign almost any type of resource, XML structures as well as non-XML data[20].

An XML Signature may be constructed as a detached, enveloping, or enveloped signature. A detached signature refers to an XML element or a non-XML file outside the signatures hierarchy. An enveloping signature encloses the item to be signed in the `ds:Object` element inside the signature. Last but not least, an enveloped signature points to another element under the parent XML element.

Listing 2 exemplarily shows the structure of an XML `ds:Signature` element (observing the notational conventions). The root element of an XML Signature is the `ds:Signature` element, which must contain at least the `ds:SignedInfo` and the `ds:SignedValue` element and can optionally have a `ds:KeyInfo` and one or more `ds:Object` elements.

```
<ds:Signature ID?>

   <ds:SignedInfo>
      <ds:CanonicalizationMethod/>
      <ds:SignatureMethod/>
      (<ds:Reference URI? >
         (<ds:Transforms>)?
         <ds:DigestMethod>
         <ds:DigestValue>
      </ds:Reference>)+
   </ds:SignedInfo>
   <ds:SignatureValue>
   (<ds:KeyInfo>)?
   (<ds:Object ID?>)*

</ds:Signature>
```

Listing 2: Structure of an XML Signature element

The `ds:SignedInfo` element contains a `ds:CanonicalizationMethod` (Canonicalization (C14N) is the way an XML document is normalized to guarantee equivalence on bit-level, even over operating system boundaries), a `ds:SignatureMethod` (which contains the algorithm to be used

[19]For example, Jothy Rosenberg and David Remy, *Securing Web Services with WS-Security - Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption*, Sams Publishing, Indianapolis, Indiana, USA, 2004.

[20]Jothy Rosenberg and David Remy, *Securing Web Services with WS-Security - Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption*, page 106, Sams Publishing, Indianapolis, Indiana, USA, 2004.

for signing documents and verifying signatures), a `ds:SignatureValue` (the base64-encoded value of the signature), and several `ds:Reference` elements.

A `ds:Reference` refers to resources to be signed by an URI attribute and contains a `ds:Digest Method` and a `ds:DigestValue`. Optionally, the `ds:Reference` has a `ds:Transforms` element, which contains a list of `ds:Transform` elements. Each of these `ds:Transform` elements defines an algorithm (and parameters for this algorithm, if necessary) that alters the object to be signed. The output of a `ds:Transform` is the input for the successive `ds:Transform`, the output from the last `ds:Transform` is the input for the `ds:DigestMethod`. The `ds:DigestMethod` defines an algorithm to calculate the digest of the `ds:Reference` URI (the only required algorithm is the Secure Hash Algorithm (SHA1)). The `ds:DigestValue` element contains the base64-encoded value of the digest.

The optional `ds:KeyInfo` element contains information on how the signature validation key can be obtained. Therefore, this element may include the `ds:KeyName`, the `ds:KeyValue`, an `ds:RetrievalMethod`, or key data, such as a `ds:X509Data` element that provides either an identifier of an X.509 certificate or the X.509 certificate itself. However, as the `ds:KeyInfo` element itself, all these child elements are optional, so that a signing party can also assume that the receiver is in possession of a signature validation key and does not include a `ds:KeyInfo` element.

Finally, an `ds:Object` element can have different functions. This element can contain the XML code to be signed (in the case of an enveloping signature), a reference to an object to be signed (such as an image or the like), a `ds:Manifest` element (a list of references to objects to be signed), or a `ds:SignatureProperties` element (contains additional information about the signature generation).

At this point, the structure of an XML Signature is clear, but not how different elements of the signature work together and how a signature is created to provide integrity protection of an XML document. The procedure for the creation of an XML Signature is as follows:

1. Create the `ds:Reference` elements for each data to be signed
   (a) Apply the `ds:Transforms` to the data to be signed
   (b) Calculate a `ds:DigestValue` over the data to be signed
   (c) Create a `ds:Reference` element, i.a., containing the `ds:DigestValue`
2. Create the `ds:SignedInfo` element, including the created references (at least one)
3. Apply the `ds:CanonicalizationMethod` to the `ds:SignedInfo`
4. Calculate the `ds:SignatureValue` over the `ds:SignedInfo` element using the `ds:SignatureMethod`
5. Create the `ds:Signature` element as described above

The signature validation is a similar process: the referenced data is collected, transformed, and a digest value is calculated. If the digest value calculated by the validator is different from the `ds:DigestValue` element in the signature, the validation fails. Otherwise, the validator obtains the validation key according to the `ds:KeyInfo` element (if present), creates a hash of the canonicalized `ds:SignedInfo` element, and verifies the `ds:SignatureValue` over the `ds:SignedInfo` element using the previously obtained key.

2.3.2 XML Encryption

Whereas XML Signature is used to confirm the content of an XML document by a known entity and to protect the content of the document from unauthorized modification, XML Encryption has a different approach: It ensures the confidentiality of XML documents or single parts of these documents using encryption technologies. It is even possible to encrypt different parts of an XML document with different keys, so that different recipients (or intermediaries) are only able to decrypt the message part which is intended for them[21].

The XML Encryption specification is used for two confidentiality protection scenarios: the encryption with a shared symmetric key, on the one hand, and the encryption using an `xenc:EncryptedKey`, on the other hand. The former requires that both parties are in possession of the same key, whereas the latter embeds the asymmetrically encrypted symmetric encryption key (`xenc:EncryptedKey`), i.e., the symmetric encryption key is encrypted usually using the public key of the recipient, so that only the recipient is able to decrypt the `xenc:EncryptedKey` with his private key.

Due to the fact that XML Encryption, as well as XML Signature, is a W3C standard, the structure of both specifications is similar. The root element of an XML Encryption is the `xenc:EncryptedData` element. This root element can optionally contain an `xenc:EncryptionMethod` (specifies the encryption algorithm to be used), a `ds:KeyInfo` element (which has the same namespace and the same function as in the XML Signature), and an `xenc:EncryptionProperties` element. The only mandatory element of an XML Encryption is the `xenc:CipherData` element. The structure of an `xenc:EncryptedData` is presented in Listing 3.

```
<xenc:EncryptedData Id? Type? MimeType? Encoding?>

   <xenc:EncryptionMethod/>?
   <ds:KeyInfo>
      <xenc:EncryptedKey>?
      <xenc:AgreementMethod>?
      <ds:KeyName>?
      <ds:RetrievalMethod>?
      <ds:*>?
   </ds:KeyInfo>?
   <xenc:CipherData>
      <xenc:CipherValue>?
      <xenc:CipherReference URI?>?
   </xenc:CipherData>
   <xenc:EncryptionProperties>?

```

[21]Jothy Rosenberg and David Remy, *Securing Web Services with WS-Security - Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption*, page 148, Sams Publishing, Indianapolis, Indiana, USA, 2004.

```
</xenc:EncryptedData>
```

Listing 3: Structure of an XML Encryption element

The algorithm specified in the optional `xenc:EncryptionMethod` element is used to encrypt the data of the `xenc:CipherData` element. If no algorithm is defined in this element, the used algorithm must be known to the recipient of the encrypted data.

As with XML Signature, the `ds:KeyInfo` element has the function to provide the encryption key directly or at least to give a hint on how to retrieve the key. This hint is given via the `ds:RetrievalMethod` element. Another possibility to get the encryption key is to make use of the `xenc:AgreementMethod` element, which employs an asynchronous key agreement protocol to generate an encryption key.

The encryption key is often not enclosed in the `ds:KeyInfo` element directly, but as `xenc:EncryptedKey`. The `xenc:EncryptedKey` element is another `xenc:EncryptedData` element, which is a child of `ds:KeyInfo`, and holds an encrypted encryption key. This encryption of the encryption key can be used arbitrarily deeply. The intent of this key encryption is to have either a shared symmetric key, which is not present in the XML file, or a combined symmetric and asymmetric key strategy, whereby the shared symmetric key is encrypted by the public asymmetric key of the recipient.

The `xenc:CipherData` contains either the encrypted data or an URI reference to this data. The former is located in a `xenc:CipherValue` element, the latter employs a `xenc:CipherReference` with an URI as attribute. The content of the `xenc:CipherValue` is base64-encoded, whereas the `xenc:CipherReference` is structured as the `ds:Reference` element of an XML Signature, including `ds:Transforms` and URI.

The `xenc:EncryptionProperties` element is similar to the `ds:SignatureProperties` element of the XML Signature: its purpose is to save additional information about the encryption, e.g., the timestamp or the serial number of cryptographic hardware used during encryption.

As with the XML Signature, the XML encryption requires a certain procedure:

1. Obtain a key
2. Serialize the data to be encrypted (UTF-8 for XML data, else octets)
3. Encrypt the data to be encrypted using the `xenc:EncryptionMethod` and the encryption key
4. Specify the encrypted data type: as `xenc:CipherValue` for XML-data, otherwise as `xenc:CipherReference`
5. Create a corresponding `xenc:EncryptedData` structure. If an `xenc:EncryptedKey` is used, execute the process with an asymmetric key recursively.

The decryption of an `xenc:EncryptedData` element requires the encryption algorithm (and related parameters) and the encryption key (either a secret symmetric key or a private asymmetric key to decrypt the `xenc:EncryptedKey` element). The key and the algorithm are used to decrypt the `xenc:CipherData`. Since the decryption produces UTF-8 encoded bytes, the result of this process must be converted to XML or passed to an application.

2.4 WS-* Standards

The technologies presented in sections 2.1 to 2.3 are sufficient to describe Web services (using WSDL), establish a Web services connection and exchange messages (using SOAP), implement workflows (using BPEL), and even to protect the exchanged XML messages using the XML security specifications. Additional features, such as message routing, transport security, or reliability of message delivery could be left to the network protocols. This raises the question why the WS-* standards are declared to the “second-generation Web services specifications”[22] in contrast to the first generation Web services specifications, such as WSDL or SOAP.

The character of a SOA requires higher-level QoS mechanisms. Since a message exchange may include multiple request/response messages, involve intermediaries, or be part of a long-term business transaction extending over several hours or days, message protection on transport layer is insufficient. In addition to this, Web services communication is not bound to a specific transport protocol, so not all QoS requirements may be met on transport level. On that account, the QoS features for Web services communication must be located on (SOAP) message level.

Since “SOAP poses no constraints on the potential scope of such features, example features may include reliability, security, correlation, routing, and message exchange patterns (MEPs) [...]” [W3C07b], the WS-* frameworks employ SOAP messages to bring QoS features on message level in Web services communication. In this way SOAP may be enriched with QoS features like integrity, confidentiality, reliability, or transaction support without losing its transport protocol independency.

2.4.1 WS-Addressing

The WS-Addressing specification enables a transport protocol-neutral transmission of message addressing information in the SOAP message header. For this purpose, WS-Addressing defines two concepts: the endpoint reference and the message addressing properties. The former defines a standardized XML structure to communicate a reference to a Web service endpoint, the latter provides message addressing information relating to the message delivery, e.g., the sender (`wsa:From`), the recipient (`wsa:To`), or the relationship of the current message to previous message (`wsa:RelatesTo`). For the use of WS-Addressing in composition with other WS-* specifications, the `wsa:Action` message property is most interesting. This property explains the semantics of a message by an URI [W3C06a].

Although the WS-Addressing framework is not limited to the use in SOAP messages, the most common use of WS-Addressing is in conjunction with SOAP. In this case, the WS-Addressing properties are located in the SOAP message header [W3C06b].

[22]Thomas Erl, *Service-Oriented Architecture - Concepts, Technology, and Design*, page 156, Prentice Hall, Upper Saddle River, New Jersey, USA, 2005.

2.4.2 WS-ReliableMessaging

The reliability of message transmission is one of the the most important requirements of any communication. Although most transport protocols meet this requirement, this is not sufficient to guarantee the delivery of a series of business documents. For example, if a business document is intercepted before it reaches the transport channel the transport protocol is not aware of this document.

The WS-ReliableMessaging protocol [OAS09b] defines a mechanism to guarantee the reliable delivery of a SOAP message between a WS-ReliableMessaging source (RM Source) and a WS-ReliableMessaging destination (RM Destination). For this purpose, a WS-ReliableMessaging sequence (hereafter just *sequence*) is created by the RM source (message type `wsrm:CreateSequence`), the RM Destination confirms the sequence creation (`wsrm:CreateSequenceResponse`). Each sequence has an unique identifier for its whole lifetime and may optionally include an expiration date, a behavior for incomplete sequences (`wsrm:DiscardEntireSequence`, `wsrm:DiscardFollowingFirstGap`, or `wsrm:NoDiscard`), and a delivery assurance. Whereas the other properties are referenced in the WS-ReliableMessaging block of the SOAP message header, the delivery assurances are defined in the policy of a service (see section 2.4.4).

After RM Source and RM Destination have agreed upon the sequence creation, each SOAP payload message of the sequence contains a WS-ReliableMessaging header including the sequence identifier and the number of the message in the sequence. The RM Source may always request an acknowledgment of the transmitted messages (`wsrm:AckRequested`), the RM Destination responds to this request with a `wsrm:SequenceAcknowledgement`. If the RM Source detects unacknowledged messages when matching the acknowledged messages with the messages sent, it retransmits the unacknowledged message(s) according to the incomplete sequence behavior. However, a `wsrm:SequenceAcknowledgement` does not terminate a sequence. Instead, an arbitrary number of messages may follow.

In order to close a sequence, the WS-ReliableMessaging standard provides two methods: to close a sequence and to terminate a sequence. A sequence may be closed by both parties, RM Source and RM Destination, at any time using a `wsrm:CloseSequence` and a `wsrm:CloseSequenceResponse` message, respectively. In contrast to that, a sequence is terminated by the RM Source when it has completed the message sequence. It then sends a `wsrm:TerminateSequence` message that is confirmed by the RM Destination with a `wsrm:TerminateSequenceResponse`. Figure 2 illustrates a WS-ReliableMessaging sequence with three payload messages, in which the second message is lost and retransmitted according to the `wsrm:IncompleteSequenceBehavior` `wsrm:NoDiscard`.

The sequence management elements, i.e., the sequence creation, termination, and closing messages, are located in the SOAP message `env:Body` whereas the other messages are part of the SOAP message header. However, a sequence is only used for communication in one direction. So, a second sequence for bilateral reliable communication is needed.

The WS-ReliableMessaging specification is aware of security threats of a sequence, i.a., integrity threats [OAS09b, lines 1074 - 1088] or sequence hijacking [OAS09b, lines 1115 - 1142], and, therefore, proposes countermeasures to address these problems. Besides transport layer

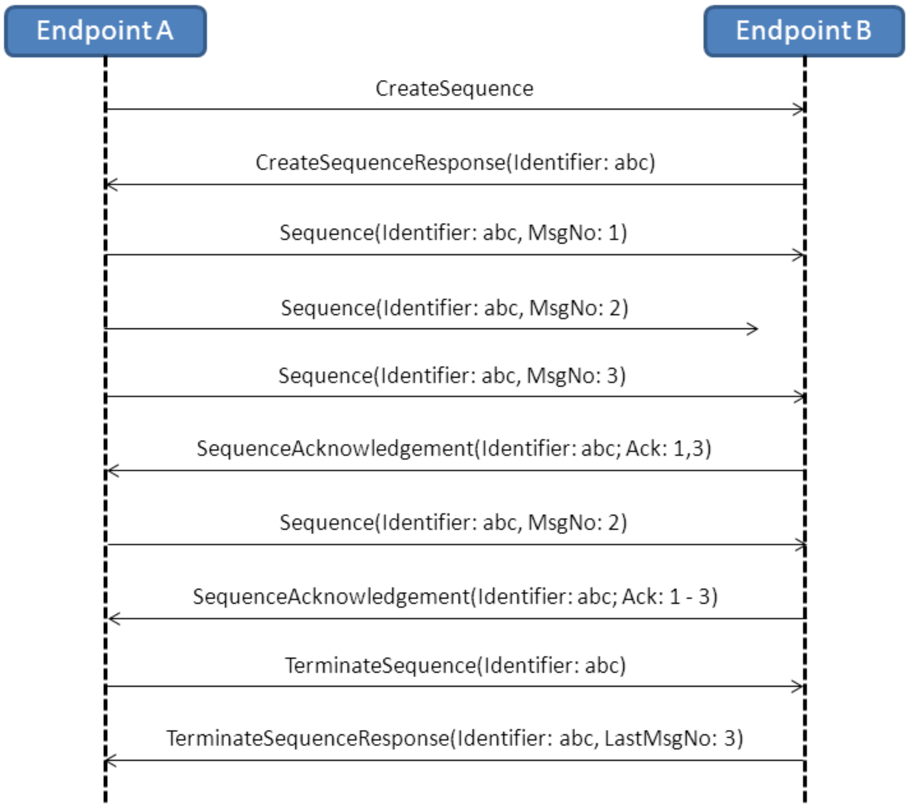
Endpoint A
Endpoint B
CreateSequence
CreateSequenceResponse(Identifier: abc)
Sequence(Identifier: abc, MsgNo: 1)
Sequence(Identifier: abc, MsgNo: 2)
Sequence(Identifier: abc, MsgNo: 3)
SequenceAcknowledgement(Identifier: abc; Ack: 1,3)
Sequence(Identifier: abc, MsgNo: 2)
SequenceAcknowledgement(Identifier: abc; Ack: 1 - 3)
TerminateSequence(Identifier: abc)
TerminateSequenceResponse(Identifier: abc, LastMsgNo: 3)

Figure 2: The WS-ReliableMessaging Protocol, following the Figure at [OAS09b, page 14]

security mechanisms, the WS-ReliableMessaging specification also includes considerations for a compound WS-Security/WS-Trust/WS-SecureConversation scenario.

2.4.3 WS-Security

As already discussed in the previous sections, security features, such as integrity or confidentiality protection are the most important requirements for B2Bi and even other WS-* standards recommend the use of security features. This raises the question how security features are implemented in the Web services world.

In "first generation Web services" it was common to use transport-layer security, such as Hypertext Transfer Protocol Secure (HTTPS), in order to bring security features to Web services. Although transport-level security is a well-known and established concept, this approach has many disadvantages, e.g., the security features cannot easily be extended to persisted messages and they are bound to point-to-point communication. In contrast to that, WS-Security is a transport-protocol independent SOAP message security specification. This has many benefits, i.a., security features are valid from the sender to the ultimate receiver (end-to-end security), messages are independent of transport protocols, and different security measures may be combined. Figure 3 illustrates the advantages of message-level security of WS-Security in comparison to transport-level security of HTTPS.

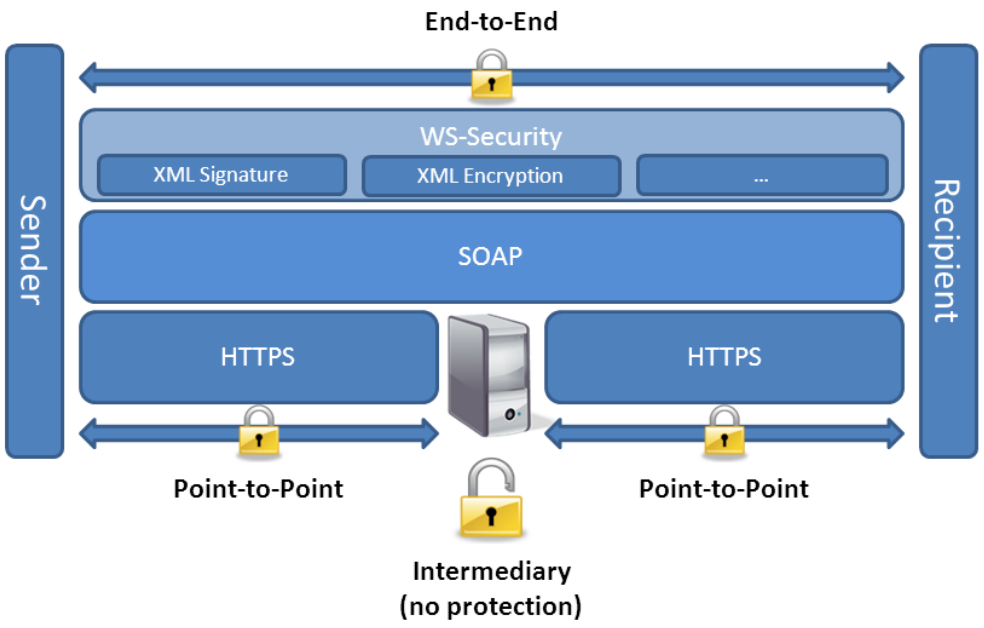

Figure 3: Comparison of the scopes of security features of WS-Security and HTTPS, adapted from [Bro03, page 2]

However, WS-Security does not introduce new security features, but defines a framework on how existing specifications may be combined to fulfill security requirements. Section 2.3 already introduced the XML security technologies that are employed by WS-Security: XML Signature (integrity) and XML Encryption (confidentiality) are two of the three pillars of WS-Security, the third is the Security Assertion Markup Language (SAML) (as security tokens), an XML-based language for exchanging authentication and authorization information between different domains [OAS05].

WS-Security determines how these specifications are used in conjunction with SOAP messages. For that purpose, the WS-Security specification defines the so-called *security header*, an element located in the SOAP message header. The security header may contain security token definitions, XML Signatures, encrypted keys, or a `ReferenceList`. Security tokens contain information for authentication and authorization, e.g., X509 certificates. The WS-Security specification defines three types of security tokens:

- Username Token
- XML Token
- Binary Token

In this way, WS-Security enables the authorization and authentication with a username/password combination, XML-based credentials, or binary key information, such as X509 certificates. Often, a X509 certificate is used as signature validation key. Since WS-Security supports both encryption types, shared key and encrypted key, the security header may contain an `xenc:EncryptedKey`. WS-Security enables signing and encrypting arbitrary message parts, except the security header or parts of it. However, the message time stamp is an exception. This element inside the SOAP message header determines the *freshness* and the validity of a SOAP message. Besides the creation date of a message it also contains its expiration date (By definition, expired messages should not be accepted). If a `wsu:Timestamp` element is present in the security header, it must be signed [OAS09d, lines 1718 - 1722]. Figure 4 shows schematically the structure of the security header inside a SOAP message [OAS06f].

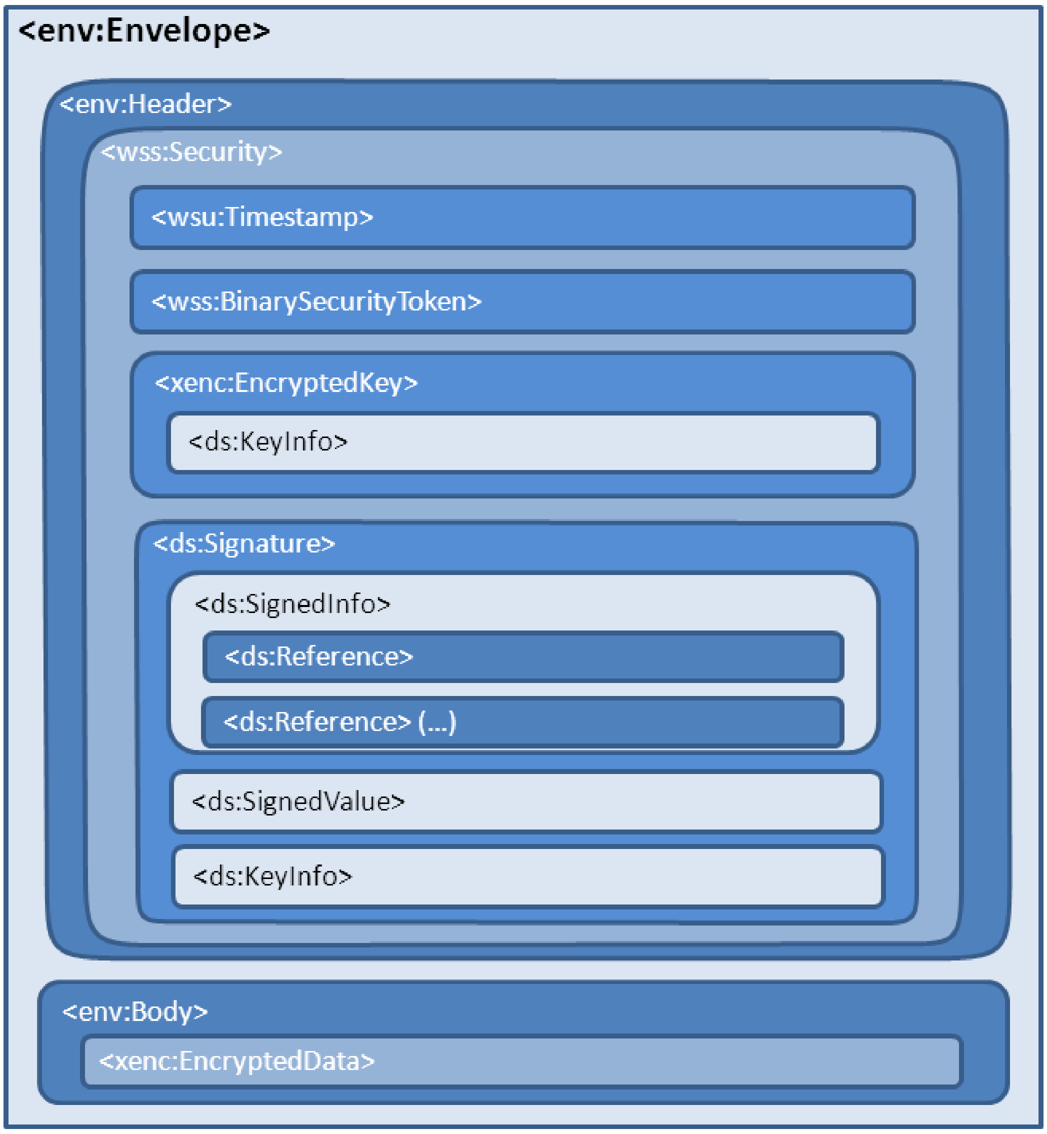

Figure 4: WS-Security elements and structure inside a SOAP message

But for all that functionality, the basic WS-Security framework still does not manage to cover all required security aspects, such as, to build up trust to an unknown party, or to minimize the key-exchange overhead for long-term business transactions with frequent message exchange. These requirements are met by extensions to the WS-Security specification, namely WS-Trust [OAS09e] and WS-SecureConversation [OAS09c]. Therefore, WS-Trust and WS-SecureConversation are introduced in sections 2.4.3 and 2.4.3.

WS-Trust

As already mentioned, WS-Trust supports the establishment of trust in Web services communication. The WS-Trust framework focuses on methods for issuing and exchanging security tokens, as well as establishing and accessing trust relationships. For this reason, WS-Trust introduces a new participant to the Web services communication model: The Security Token Service (STS). In addition to this, the message elements `wst:RequestSecurityToken` element (RST) and `wst:RequestSecurityTokenResponse` element (RSTR) are defined for the exchange of security information.

Figure 5 shows the WS-Trust model for providing a STS. Assuming that a Web service requires that a service requester is in possession of a certain claim, e.g., a certain X509 certificate. Each message without a proof of possession of this X509 certificate is rejected by the service, and thus each service requester needs this certificate. It is assumed that the service requester is not in possession of this key, and, therefore, the requester has to acquire this claim. The claim may be requested from the token issuer, i.e, either from the Web service itself or from a third party, the STS. In both cases, the service requester sends a message including a `wst:RequestSecurityTokenCollection` element (RSTC), containing one or more RSTs, to the token issuer, one RST for each requested claim. Of course, this message is subject to the policy of the token issuer and therefore the message must satisfy the policy of the token issuer. The RST is answered with a `wst:RequestSecurityTokenResponseCollection` element (RSTRC) in the response message, containing at least one RSTR. The RSTR contains the requested claim, thus the service requester now may use this claim to communicate with the service [OAS09e].

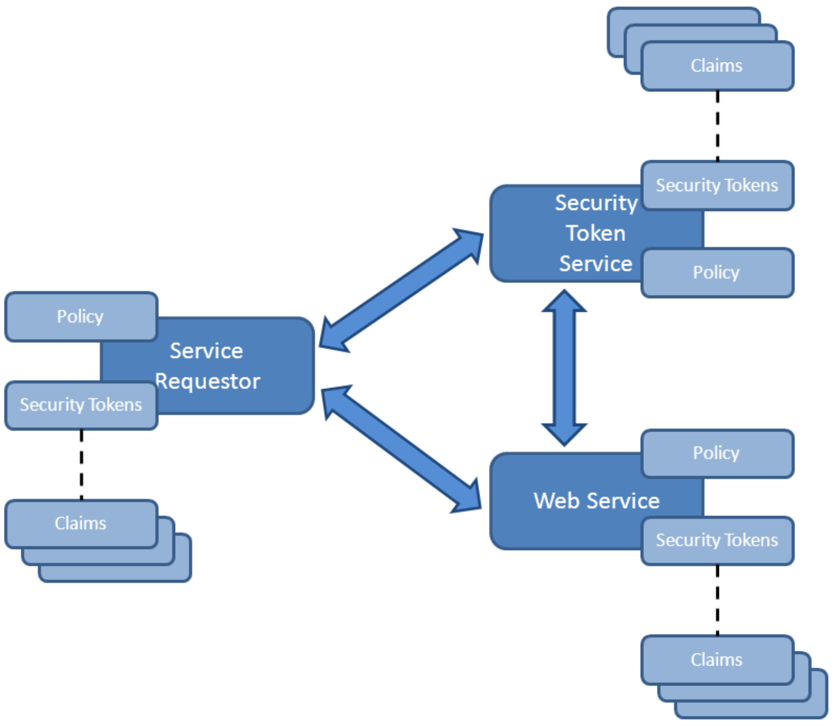

Figure 5: The WS-Trust model, following [OAS09e, page 12]

The RSTC and the RSTRC are located in the `env:Body` of the SOAP message. The request-response-token framework is extended by WS-SecureConversation (see section 2.4.3). However, a simple request-response message exchange is not sufficient to build up trust between unknown parties, so WS-Trust extends the simple RST/RSTR framework to enable exchanges for negotiation and challenges. Both parties, the initiator and the recipient, may challenge the other party to authenticate. Negotiations/challenges are arbitrarily nestable, however, the message exchange model is fixed:

1. Initial request, containing a RST, may contain an initial negotiation/challenge information
2. Response to the initial request, contains a RSTR with additional negotiation/challenge information. This message may also return security tokens in a RSTRC
3. If the exchange is not complete, the requester sends a RSTR with additional negotiation/challenge information to the responder
4. For this message there are two possible cases:
   - If the negotiation/challenge process is completed, return a RSTR
   - Otherwise, go back to step 2 until the challenge/negotiation is completed

The elements within a RST specify the kind of challenge, e.g., a signature challenge (`wst:Sign Challenge`) or an user interaction challenge (`wst14:InteractiveChallenge`).

WS-SecureConversation

The WS-SecureConversation framework employs the RST and RSTR messages of WS-Trust to provide a mechanism for a secure messaging semantics to be used in multiple messages, the so-called *security context*. The two participating parties agree upon a security context, which contains a shared secret for key derivation. The security context is established using asymmetric encryption of the SOAP messages (specified in the `sp:BootstrapPolicy`), whereas the further communication is performed using symmetric keys. Once the context is established, both parties derive their signature and encryption keys from the security context in order to avoid permanent key exchange in long-term business transactions. In contrast to classical WS-Security message protection using the public key method, the computationally expensive message authentication and verification are no longer necessary.

The security context may be created in a variety of ways: negotiation of the participating parties, propagation of the context by one of the parties, or creation of the context by a third party. In any case, a `wsc:SecurityContextToken` identifies the security context. This security token must be referenced in all `wsc:DerivedKeyTokens` as well as for all operations concerning the security context, such as amend, renew, or cancel. Due to these context-administration messages, the WS-SecureConversation specification is not beneficial for scenarios with only a few messages.

In a way, the security context concept may be compared to the SSL protection, but on message-level instead of transport-level: both protocols use public-key encryption to agree upon a shared secret. The shared secret is then used to derive (symmetric) session keys for message protection.

2.4.4 WS-Policy

Up to this point, the most important WS-* frameworks for a secure and reliable message exchange have been presented. However, a Web service must explicate the requirement for a certain WS-* feature in its service contract. The WS-Policy framework provides an extensible framework "for expressing policies that refer to domain-specific capabilities, requirements, and general characteristics of entities in a Web services-based system" [W3C07i, page 3]. A so-called *policy assertion* represents one of these "capabilities, requirements, or general characteristics". One or more policy assertions are gathered in a policy alternative. A policy in terms of WS-Policy consists of one or more policy alternatives [W3C07i].

The XML representation of a policy is a policy expression (see Listing 4 for the structure of a WS-Policy). The root element which identifies the policy is the `wsp:Policy` element. It has one child-element, the `wsp:ExactlyOne` element, which embodies one or more policy alternatives. Each `wsp:All` element contains a policy alternative with multiple policy assertions, whereby each policy assertion may contain a policy expression in turn, i.e., the WS-Policy framework is arbitrary nestable [W3C07i]. Since each party involved in a Web services communication may define policies, the WS-Policy specification provides the concept *policy intersection* to reduce the number of possible and compatible policies alternatives. For a detailed introduction to policy intersection, refer to [W3C07i, pages 29 - 33].

```
<wsp:Policy>
   <wsp:ExactlyOne>
      (<wsp:All>
         (<Assertion (wsp:Optional="true")?>
            <!-- Assertion -->
          </Assertion> )*
       </wsp:All> )*
   </wsp:ExactlyOne>
</wsp:Policy>
```

Listing 4: Structure of a WS-Policy

A WS-Policy is independent from other Web services specifications and may be bound to service descriptions, such as WSDL documents, but also to service directories, such as UDDI [W3C07h]. However, WS-Policy is primarily used in conjunction with WSDL.

```
<wsp:Policy>
   <wsp:ExactlyOne>
      <wsp:All>
         <wsam:Addressing>
            <wsp:Policy/>
         </wsam:Addressing>
      </wsp:All>
   </wsp:ExactlyOne>
```

```
</wsp:Policy>
```

Listing 5: Example of a WS-Policy expressing the mandatory use of WS-Addressing

Since the WS-Policy standard defines an abstract model for expressing policies, it does not provide concrete syntax for the use of WS-Addressing, WS-Security, WS-ReliableMessaging, or other specifications - each WS-* standard must define its own policy assertions. For example, [W3C07d] specifies WS-Policy assertions for WS-Addressing. Listing 5 shows an example of a WS-Policy assertion that expresses the mandatory use of WS-Addressing.

WS-Policy defines an additional argument for most assertions, the boolean attribute `wsp:Optional`. It indicates whether the use of the corresponding assertion is possible (if the value is `true`) or mandatory (if the value is `false`). Listing 6 shows almost the same policy as Listing 5, however, this time the use of WS-Addressing is not mandatory but optional.

```
<wsp:Policy>
   <wsp:ExactlyOne>
      <wsp:All>
         <wsam:Addressing wsp:Optional="true">
            <wsp:Policy/>
         </wsam:Addressing>
      </wsp:All>
   </wsp:ExactlyOne>
</wsp:Policy>
```

Listing 6: Example of a WS-Policy expressing the optional use of WS-Addressing

Since WS-Addressing is one of the most basic WS-* frameworks, there are not many other possible settings for WS-Addressing policy expressions. However, the more complex frameworks, such as WS-ReliableMessaging and WS-Security define a much more sophisticated syntax for policy expressions. The syntax of these domain-specific policy expression languages is presented in the subsequent sections.

WS-ReliableMessaging Policy

The WS-ReliableMessaging Policy (WS-RM Policy) defines a policy assertion for the use of WS-ReliableMessaging, the `wsrmp:RMAssertion`. This assertion expresses a requirement for the mandatory use of WS-ReliableMessaging, except if the attribute `wsp:Optional` has the value `true` [OAS09a].

```
<wsrmp:RMAssertion (wsp:Optional="true")? ... >
   <wsp:Policy>

      (<wsrmp:SequenceSTR /> |
       <wsrmp:SequenceTransportSecurity /> ) ?

      <wsrmp:DeliveryAssurance>
         <wsp:Policy>
            (<wsrmp:ExactlyOnce /> |
             <wsrmp:AtLeastOnce /> |
              <wsrmp:AtMostOnce />  )
            <wsrmp:InOrder /> ?
         </wsp:Policy>
      </wsrmp:DeliveryAssurance> ?

   </wsp:Policy>
   ...
</wsrmp:RMAssertion>
```

Listing 7: Structure of a `wsrmp:RMAssertion` (WS-RM Policy) element

As Listing 7 shows, the `wsrmp:RMAssertion` may be used to express security requirements for the WS-ReliableMessaging sequence. For this purpose, the specification defines the mutually exclusive assertions `wsrmp:SequenceSTR` and `wsrmp:SequenceTransportSecurity`. The former requires to bind a security token to the sequence, whereas the latter makes use of transport-layer security (e.g., SSL) mandatory for the sequence. Lines 7 - 14 of Listing 7 show how delivery assurances can be defined in a WS-RM Policy. The supported delivery assurances are:

- **ExactlyOnce:** Each message is delivered exactly once. The RM Source must ensure that each message is delivered exactly once, i.e., the RM Source retries to send a message until it receives an acknowledgment, while the RM Destination refuses to accept duplicates of already received messages.
- **AtLeastOnce:** Each message is delivered at least once. The RM Source repeats the transmission of unacknowledged messages until it receives an acknowledgment.
- **AtMostOnce:** Each message is delivered at most once. The RM Destination must ignore duplicate messages, whereas the RM Source may retransmit unacknowledged messages.
- **InOrder:** The messages of the sequence must be delivered in the correct order according to the message number.

The `wsrmp:InOrder` assertion is freely combinable with any other delivery assurance.

WS-Security Policy

Since the WS-Security specification allows much more settings than the WS-ReliableMessaging or WS-Addressing specifications, the WS-SecurityPolicy standard is more complex than the already presented domain-specific policy standards. WS-SecurityPolicy formulates requirements for security features by determining *what* message parts are to be protected and *how* the protection of the message parts is realized. For that purpose, WS-SecurityPolicy defines a set of pattern that are combined to a valid security policy. These WS-SecurityPolicy patterns are [OAS09d]:

- **Protection Assertions:** these assertions identify what is being protected and which kind of protection is applied to the protected object(s). The objects are either integrity or confidentiality protected, the latter concerns either a whole XML element or only its content. In addition to this, it is also possible to define required message parts using a protection assertion. These protection requirements may either be expressed using name and namespace (`sp:SignedParts`, `sp:EncryptedParts`, etc.) of the object to be protected or by an XPath expression (`sp:SignedElements`, `sp:EncryptedElements`, etc.).

- **Token Assertions:** token assertions are used to describe requirements for a security token. A security token is a collection of claims, such as a name, an identity, or a key. Token assertions are used to specify the types of tokens required in a message. The WS-SecurityPolicy specification defines eleven token types (see Table 1).

- **Security Binding Assertions:** these assertions describe requirements for different security mechanisms of a service, i.e., message protection on transport-level (`sp:TransportBinding`) or on message-level. The protection on message-level is realized using either symmetric (`sp:SymmetricBinding`) or asymmetric keys (`sp:AsymmetricBinding`). These message-level security bindings employ security tokens to protect message parts, e.g., a `sp:X509Token` containing a X509 certificate to encrypt the `env:Body` of a message. In addition, the security binding determines the layout (`sp:Layout`) of the security header and the `sp:AlgorithmSuite`, i.e., algorithms used, key lengths, etc., for the message exchange with the protected service.

- **Supporting Tokens Assertions:** security tokens are not only used in conjunction with security bindings to protect message parts but also to transmit additional claims, such as a username/password combination, additional certificates, or Digital Rights Management (DRM) licenses, independent of any binding. For this purpose, the `sp:SupportingTokens` assertion is used. Besides an arbitrary number of various security tokens, this assertion may define additional protection assertions (signature and encryption) and an additional `sp:AlgorithmSuite` for the cryptographic operations performed with the supporting tokens. Listing 8 shows the structure of a `sp:SupportingTokens` assertion. Furthermore, the WS-SecurityPolicy standard allows to express requirements for integrity and confidentiality protection of the supporting tokens, as well as an integrity protection of the message signature as supporting tokens. All in all, seven specializations of the `sp:SupportingTokens` are defined by WS-SecurityPolicy, e.g., `sp:Endorsing-SupportingTokens`, `sp:SignedEndorsingSupportingTokens`, or `sp:SignedEndorsing-EncryptedSupportingTokens`.

```
<sp:SupportingTokens>
   <wsp:Policy>
      (<!-- Token Assertions -->) +

      (<sp:AlgorithmSuite>
         ...
      </sp:AlgorithmSuite>)?

      (<sp:SignedParts> ... </sp:SignedParts> |
       <sp:SignedElements> ... </sp:SignedElements> |
       <sp:EncryptedParts> ... </sp:EncryptedParts> |
       <sp:EncryptedElements> ... </sp:EncryptedElements> )*

   </wsp:Policy>
   ...
```

Token	**Token description**
`sp:UsernameToken`	Describes how a Web service consumer may authenticate his identity using a *username* and optionally a password, a shared secret, or a password equivalent [OAS06g] [OAS09d, pages 28 - 30]
`sp:CreateIssuedToken`	Security Token issued by a 3rd party, i.e., a STS [OAS09d, pages 30 - 32]
`sp:X509Token`	Subtype of a `sp:BinarySecurityToken` containing a X509 certificate [ITU08] [OAS06h] [OAS09d, pages 32 - 34]
`sp:KerberosToken`	Requirement for a Kerberos ticket [IET93] [OAS06c] [OAS09d, pages 34 - 36]
`sp:SpnegoContextToken`	Represents a requirement for the execution of a n-leg SPNEGO (Simple and Protected GSSAPI (Generic Security Services Application Program Interface) Negotiation Mechanism) with the Web service (see WS-Trust) [OAS09d, pages 36/37]
`sp:SecureConversationToken`	A security context that must be retrieved from a predefined issuer (see WS-SecureConversation) [OAS09d, pages 37/38]
`sp:SecurityContextToken`	Identifies a security context (see WS-SecureConversation) [OAS09d, pages 38 - 42]
`sp:SamlToken`	A Security Assertion Markup Language (SAML) assertion as a security token [OAS05] [OAS06e] [OAS09d, pages 42 - 44]
`sp:RelToken`	A Rights Expression Language (REL) assertion as a security token [OAS06d] [OAS09d, pages 44/45]
`sp:HttpsToken`	Requirement for the use of HTTPS in a `sp:TransportBinding` [OAS09d, pages 45/46]
`sp:KeyValueToken`	Represents a requirement for a key value in a security token, e.g., a `sp:RsaKeyValue` [OAS09d, pages 46 - 48]

Table 1: The various types of Security Tokens defined by WS-Security Policy

```
</sp:SupportingTokens>
```

Listing 8: Structure of a `sp:SupportingTokens` element

- **WS-Security and WS-Trust Options:** WS-Security and WS-Trust also define additional aspects that are independent of the assertions defined above. For that purpose, the WS-SecurityPolicy specification provides optional assertions for WS-Security and WS-Trust features, e.g., `sp:RequireSignatureConfirmation` or `sp:RequireRequestSecurityTokenCollection`.

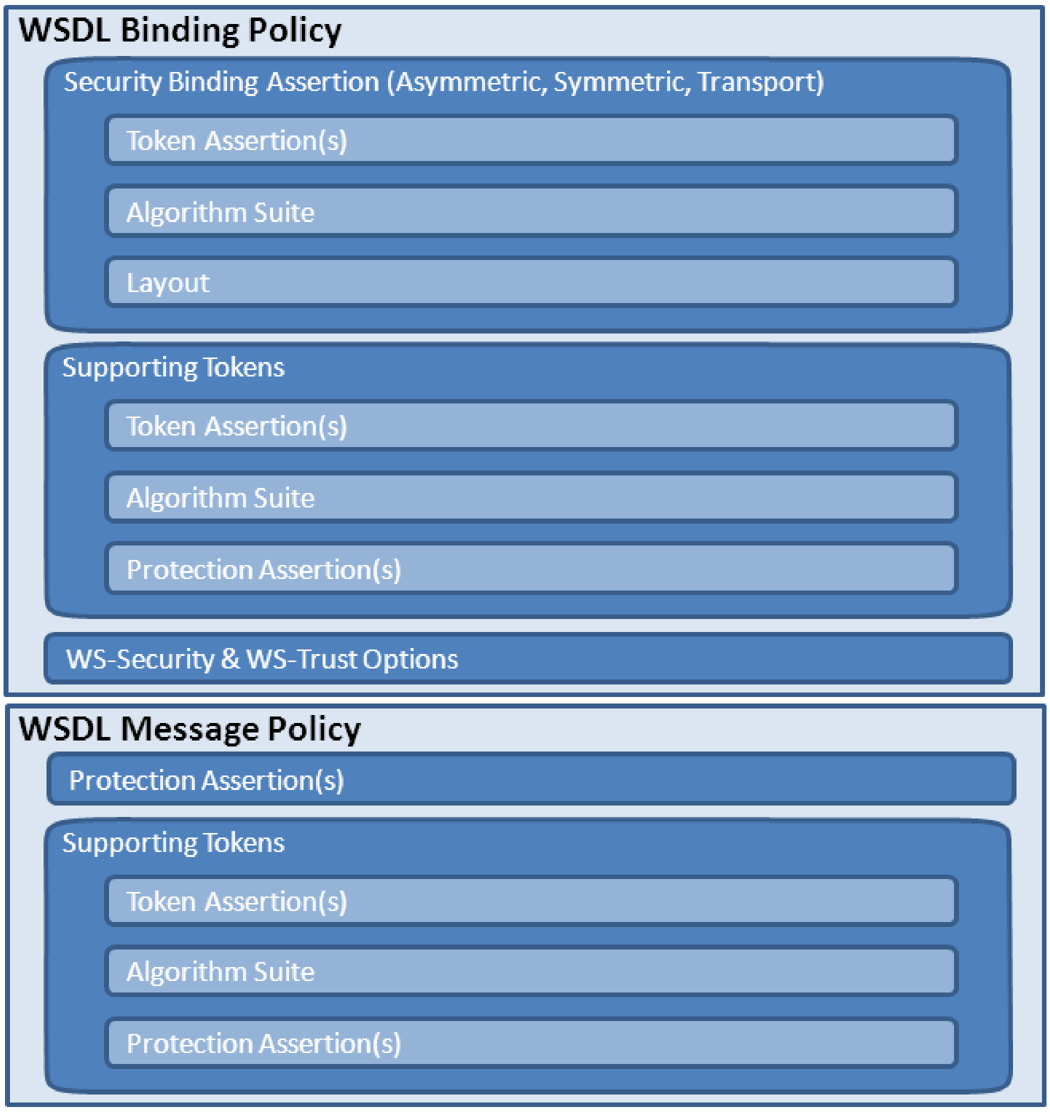

Figure 6: WS-SecurityPolicy elements and structure inside a WSDL document

A WS-SecurityPolicy configuration is composed according to the WS-SecurityPolicy scheme above. Figure 6 shows the structure of a composed WS-SecurityPolicy: on the one hand there is a policy that is bound to the `wsdl:binding` definition, on the other hand one or more message protection policy for each message definition in a WSDL document (`wsdl:input`, `wsdl:output`, or `wsdl:fault`). Each *Binding Policy* must have a Security Binding Assertion with one or more Token Assertion (according to the type and the options of the binding), an `sp:AlgorithmSuite` and optionally a security header layout determination. In addition to this, the binding policy may contain a `sp:SupportingTokens` assertion (with one or more Token Assertions, an optional `sp:AlgorithmSuite`, and optional Protection Assertions) or/and WS-Security and WS-Trust options. The WS-SecurityPolicy for a message contains usually one or more Protection Assertions and may also contain `sp:SupportingTokens` assertions (same structure as with the `sp:SupportingTokens` of the binding policy).

3 Platforms

The WS-* frameworks, which have been discussed in the previous section, are just specifications by the OASIS and the W3C. In the context of this work, a platform, i.a., provides an implementation of those frameworks and an opportunity to develop, deploy, configure, and orchestrate applications using those frameworks.

Each platform consists of various parts: an Integrated Development Environment (IDE), for test application development and configuration, and an application server to deploy the test applications. The application server either bundles an implementation of (parts of) the WS-Stack or allows the installation of an additional WS-* implementation. In order to create SOAs for more complex and practical test cases, Enterprise Service Buses (ESBs) and BPEL-Engines are employed. Both are either integrated in the platform or available as platform extensions. Frequently, an ESB includes a BPEL-Engine. In general, ESBs build the backbone of a SOA by combining the approaches of EAI and Message-Oriented Middleware (MOM). An ESB expands the classical hub-and-spoke architecture paradigm of EAI by detaching it from determined message formats and protocols to overcome heterogeneity. Services and other applications can communicate via the ESB using standardized interfaces, such as WSDL interfaces, and standardized message protocols, whereby the ESB manages heterogeneity issues. Typical ESB features are, i.a, message routing, message transformation (mediation), providing application adapters and security features, or enable process orchestration, e.g., using WS-BPEL [Men07] [dLPW07].

In this section, the test environment (hardware and software environment in which tests will be run) will be presented. All tests have been conducted on personal computers (PCs) with Intel Core 2 Duo or Core 2 Quad processors, running Windows XP Professional 32-bit, Service Pack 3. Each platform includes a BPEL engine to execute workflows as well as a WS-* implementation (at least including WS-Addressing, WS-ReliableMessaging, WS-Security, WS-Trust, and WS-SecureConversation) to satisfy the requirements for B2Bi as defined in [Ros10, pages 20/21] [OAS06a, pages 71-73]. The tested platforms including IDEs, application servers, and additional software (e.g., Feature Packs, ESBs, or BPEL-Engines) are defined in the subsequent sections. First, the IBM WebSphere Platform is presented in section 3.1, then the components of the Sun Microsystems GlassFish Platform are pointed out in section 3.2.

3.1 IBM WebSphere Platform

WebSphere is a software product line for business and application integration, developed and published by the IBM corporation. The core product of WebSphere is the IBM WebSphere Application Server (WAS), a proprietary Java Platform, Enterprise Edition (JEE) 5 application server.

The WAS is based on the Eclipse[23] plug-in environment, the Equinox[24] framework, which is an implementation of the OSGi Core Framework R4 specification[25]. So the WAS runtime is designed as a collection of plug-in components. Additional plug-ins can be developed as Eclipse Plug-In Projects and plugged in the WAS Equinox environment [Ren09, pages 35-43].

As Java Virtual Machine (JVM) for the execution of the WAS components IBM does not make use of the standard Sun Microsystems JVM (short: Sun JVM), but has developed its own JVM, called IBM J9, including some features of the open source Apache Harmony JVM[26]. In contrast to the standard Sun JVM, the J9 is optimized for use on IBM hardware (e.g., for the use on the IBM POWER or z/Architecture platforms) as well as for the support of enterprise features of an application server [Ren09, pages 15-34].

Since the WAS is a JEE application server, it supports the Application Programming Interface (API) specifications of JEE, which includes, i.a., JMS, Enterprise JavaBeans (EJB), and Web services. The latter is realized using Java API for XML-Based RPC (JAX-RPC) and Java API for XML-Based Web Services (JAX-WS), whereby the WAS employs a plug-in for JAX-WS support [Ren09, pages 199 and 217-222]. This plug-in includes the open source SOAP-engine Apache Axis2[27] (the Apache Foundation describes it as "a Web Services / SOAP / WSDL engine"), which supports a wide variety of specifications, transport protocols, and data bindings for Web services. The most important Axis2 features in the context of this work are:

- JAX-WS support for services and clients
- WSDL, version 1.1 (including SOAP and HTTP binding) and 2.0
- SOAP, version 1.1 and 1.2
- WS-Addressing
- WS-Policy

As can be seen from the list above, Axis2 only supports two WS-* specifications, which does not fulfill the requirements for the WS-* implementation. However, Axis2 provides a plug-in environment for additional Web services features. The Apache Foundation currently provides

[23]Eclipse is an open source community, developing, i.a., an IDE and a plug-in environment. Projects and further information are available at `http://www.eclipse.org`

[24]available at `http://www.eclipse.org/equinox`

[25]Further information is available at `http://www.osgi.org/Release4`

[26]available at `http://harmony.apache.org`

[27]available at `http://ws.apache.org/axis2`

three plug-ins for Axis2, these are Apache Sandesha2[28] for WS-ReliableMessaging, Apache Rampart[29] for WS-Security, and Apache Kandula2[30] for WS-Coordination. The IBM WAS 7 employs the Sandesha2 plugin for WS-ReliableMessaging support and IBM self-developed implementations of WS-Security and WS-Coordination [Ren09, pages 217-222].

The IBM WS-Security implementation extends the security token types defined in the WS-Security and WS-SecurityPolicy standards by additional, customized or IBM self-developed token types, e.g., the Lightweight Third-Party Authentication (LTPA) Token[31] and customized Kerberos and SAML Tokens. The latter token types are defined in the XMLNS `http://www.ibm.com/xmlns/prod/websphere/200605/ws-securitypolicy-ext` as specialization of the `WssCustomToken` type. IBM introduced the SAML security token type in the WAS 7 with Fix Pack 7 (WAS 7.0.0.7). Fix Pack 9 (WAS 7.0.0.9 improved and extended the SAML functionality of the WAS. However, the scope of this work only includes Fix Pack 7 since Fix Pack 9 was released during the test execution phase.

The WAS realizes the policy configuration of a Web service via the so-called *policy sets*; WS-Policy declarations in a WSDL document that is directly packaged in a deployable application are refused by the WAS and thus are not deployable. A policy set defines a set of WS-Policy-based policy assertions that can be assigned to a service. A policy set consists of a collection of specific policies, such as a policy (IBM calls it policy type) for WS-ReliableMessaging, WS-Security, and WS-Addressing, but also non-WS-* QoS definitions like SSL transport. The information of the policy set is either published in the WSDL document of a service or transmitted using WS-MetadataExchange [W3C10]. The collection of policies is defined in a `policySet.xml` file, which is an index of the policy set defining its name an the policy types used in this set. Each definition of a policy type is defined in a separate file [CLM+09, pages 264-268].

Although the whole policy set definition is based on XML and WS-Policy, the single policy type definitions are not necessarily built on the corresponding policy specifications for a WS-* standard. For example, the policy type for WS-Security and standards built on top of WS-Security is defined using WS-SecurityPolicy, whereas the policy types for WS-ReliableMessaging are not defined using the corresponding RM-Policy standard. However, the policy set definition of the WS-ReliableMessaging policy type are converted to WS-ReliableMessaging Policy assertions in the WSDL document of the service. This suggests that for each possible setting of the WS-ReliableMessaging policy type assertions a predefined WS-ReliableMessaging Policy assertion is deposited in the implementation. Interestingly, a Web services client deployed on a WAS is not able to process a WS-ReliableMessaging Policy other than the WAS-defined policies. This allows the conclusion that the WAS does not provide full WS-ReliableMessaging Policy support.

A WAS Web services client may also define an own policy set, the so-called *client policy*, which is primarily used to specify transport protocol requirements, since these are not expressible as WAS-conform policies using a transport binding. However, client policies are also used to

[28]available at `http://ws.apache.org/sandesha/sandesha2`
[29]available at `http://ws.apache.org/axis2/modules/rampart/1_2/security-module.html`
[30]not available yet, see project page at `http://ws.apache.org/kandula/2`
[31]Further information available at `http://www.ibm.com/developerworks/websphere/techjournal/0607_desprets/0607_desprets.html`

calculate *effective policies* (see policy intersection). In this case, the client policy can be further restricted by the server policy, but the effective policy is definitely acceptable to both partners, service and client - incompatible policies are refused and thus are not processed.

Due to the fact that policy sets do not include environment-specific information, such as keystore and truststore definitions, this must be added in another way. The WAS employs bindings for this purpose. A binding defines the concrete details of a policy, e.g., the signature and encryption key, properties of UsernameTokens, trusted certificates, or settings concerning the transport protocols. The WAS employs different bindings for service and client, since both need an own private key and the requirements on each side can differ. The relationship between bindings, policy sets, and policy types is depicted in Figure 7 [CLM$^+$09, pages 265-267].

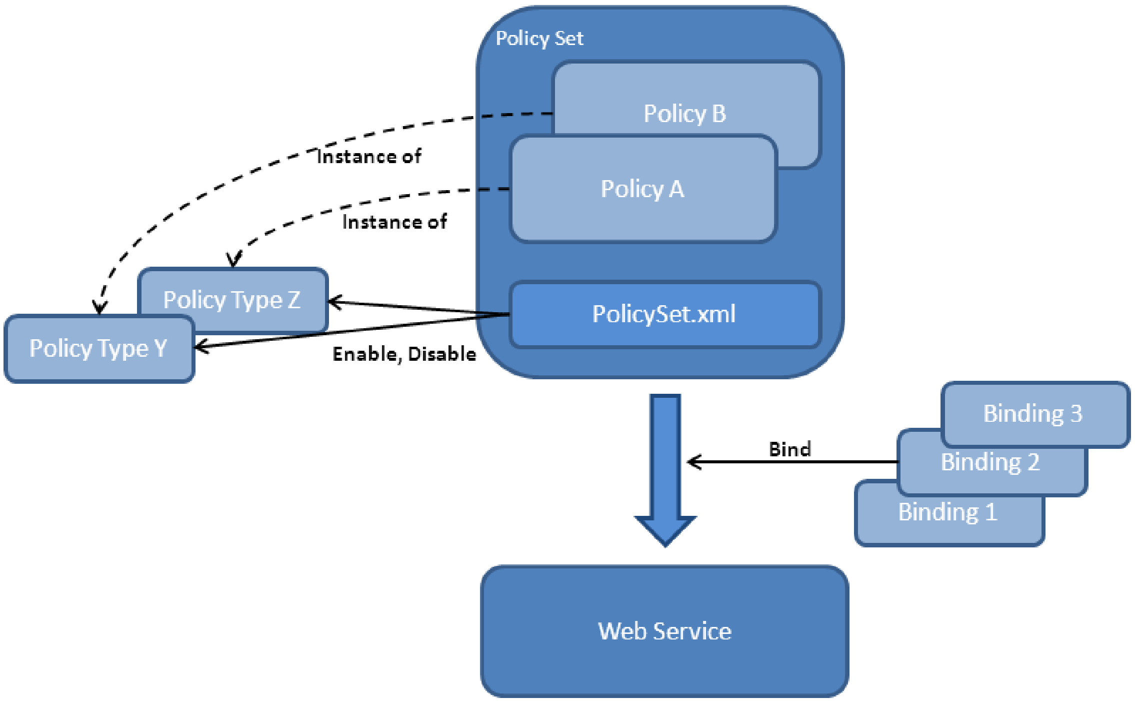

Figure 7: Relationship between bindings, policy sets, and policy types in the WAS environment (Web service provider)

The assignment of policy sets and bindings to services is performed either in the Web interface or in an IDE for the WebSphere product line. The most common IDEs for the use with WebSphere products are the IBM Rational Application Developer for WebSphere Software (RAD) and the IBM WebSphere Integration Developer (WID). Both Eclipse-based tools are employed in the context of this work for developing Web services and Web services clients. The latter one also provides a "BPEL designer" for WS-BPEL processes, which may be deployed on the IBM WebSphere Process Server (WPS), which is the WS-BPEL-engine for the WAS. The WPS is not only a WS-BPEL-engine, but also provides additional features for business process management, such as the integration of human tasks in WS-BPEL processes or the Business State Machines, which supports event-driven business process scenarios. Thus, the WPS makes the WAS ready for B2Bi use [IJC08, pages 75-79 and 150-170].

The WPS operates on top of the IBM WebSphere Enterprise Service Bus (WESB), which enables the EAI. The WESB provides an ESB for the integration of service-oriented, event-driven, and message-oriented technologies with support of the Service Component Architecture (SCA) [OSO07] and Service Data Objects (SDO) [OSO06]. Hence, the WESB is a middleware solution, which abstracts form concrete transport protocols, taking over tasks, such as message routing, message enrichment, and message transformation [IJC08, pages 321/322 and 335] [Ren09, pages 319/320].

In the context of this work, the *IBM WebSphere Platform* consists of the following components:

- IBM WebSphere Integration Developer (WID) 7.0.0.101
- IBM Rational Application Developer for WebSphere Software (RAD) 7.5.5.1
- IBM WebSphere Application Server (WAS) 7.0.0.7, including
  - Feature Pack for Service Component Architecture 1.0.1.1
  - Feature Pack for XML 1.0.0.3
- IBM WebSphere Enterprise Service Bus (WESB) 7.0
- IBM WebSphere Process Server (WPS) 7.0.0.1

From here the bundle of software listed above will be referred to as *IBM WebSphere Platform*. References to particular parts of the IBM WebSphere Platform will use the full name or the acronym of the according component.

3.2 Sun Microsystems GlassFish Platform

The other JEE 5 application server in the context of this work is the Sun Microsystems GlassFish v2[32] (current version 2.1.1). In contrast to the WAS, GlassFish is an open source project initiated and supported by Sun Microsystems (short: Sun). Since Sun is also the developer of Java, the GlassFish application server is frequently designated as reference implementation of JEE. GlassFish provides support for EJBs, Java Architecture for XML Binding (JAXB), Streaming API for XML (StAX), Web services, and many more[Sun10].

To enable Web services and related technology support, the GlassFish application server bundles the Metro Web services stack[33], which is part of the GlassFish project, but can also be used outside of the GlassFish application server. Metro is built on top of JAXB and therefore necessarily needs a preinstalled Java Development Kit (JDK), version 1.5.0_14 or higher. The Metro core implements the JAX-WS API and allows to plug in WS-* implementations. These implementations are delivered by the Web Services Interoperability Technologies (WSIT)[34], an implementation of Web services features, which has been developed in close collaboration with Microsoft to guarantee interoperability of Java and .NET application servers. The WSIT provides implementations for core XML features (XML, XSD, XML Namespaces), Web services messaging protocols (SOAP, WS-Addressing, SOAP Message Transmission Optimization Mechanism (MTOM)), Web services description specifications (WSDL, WS-Policy, WS-MetadataExchange), and, of course, an implementation of Web services QoS features [Sun].[35]

The WSIT includes the following implementations of [Sun]:

- WS-Security including WS-Trust and WS-SecureConversation
- WS-SecurityPolicy
- WS-ReliableMessaging
- WS-Coordination including WS-AtomicTransactions

WS-ReliableMessaging Policy is not listed in the official product descriptions of Metro and WSIT but has been tested successfully and is therefore used in the context of this work.

The deployment of JAX-WS Web services on the GlassFish application server implies the packaging of the application logic in a WAR or EAR file, including a WSDL document with the corresponding interface and binding information. Web services QoS extensions are directly placed in the WSDL file in WS-Policy format. If a policy definition is not valid or not supported by the Metro Web services stack, the application server refuses to deploy the service. The Netbeans IDE[36] provides a graphical editor for the creation of Web services, including the

[32]available at `http://glassfish.dev.java.net`
[33]available at `http://metro.dev.java.net`
[34]available at `http://wsit.dev.java.net`
[35]Thilo Frotscher, Die Qual der Wahl, Javamagazin (2008), no. 2, pp. 108-117.
[36]available at `http://www.netbeans.org`

policy configuration of a service. Netbeans also allows to integrate the GlassFish application server into the IDE for direct deployment and administration of developed applications.

However, neither GlassFish nor the Metro WS-Stack include a WS-BPEL-engine or an ESB. Thus, the GlassFish basic version is not ready for B2Bi use. However, due to the open and extensible predisposition of the GlassFish application server, this functionality can be added easily. The OpenESB[37] project enables the implementation of a SOA in the GlassFish environment, implementing the Java Business Integration (JBI) specification [Sun05]. JBI is a plug-in environment based on WSDL for the integration of external services and the realization of business logic. The heterogeneity of external services is resolved by using Binding Components (BCs) for protocol independent communication, e.g., the HTTP BC or the JMS BC. BCs provide connectivity to JBI-external services. The JBI specification includes the concept of Service Engines (SEs) to bring business logic to the ESB. A SE can consume and provide business logic as well as provide transformation services. The OpenESB implements, i.a., BCs for HTTP and JMS and SEs for WS-BPEL and Extensible Stylesheet Language Transformations (XSLT).

OpenESB with a bundled GlassFish application server, including the Metro WS-Stack and a customized version of the Netbeans IDE is available under the name GlassFish ESB[38]. All components of that package are aligned to each other, for example, the Netbeans IDE already includes the GlassFish server plug-in and has a built-in WS-BPEL editor.

For this work the following components represent the *Sun GlassFish Platform*, which is equivalent to the GlassFish ESB 2.2:

- Netbeans IDE 6.7.1
- Sun GlassFish Application Server 2.1.1
- OpenESB 2
- Metro 1.4

Although there are newer versions of the GlassFish ApplicationServer and the Netbeans IDE available, the versions above have been preferred due to support of the OpenESB BPEL engine. OpenESB 2.2 bundles the Sun GlassFish Application Server 2.1.1 and the Netbeans IDE 6.7.1.

The *Sun GlassFish Platform* and its components will be referenced analogously to the IBM WebSphere Platform.

[37]available at `http://open-esb.dev.java.net`
[38]also available at `http://open-esb.dev.java.net`

4 Test Method

The two previous chapters presented the standards to be tested (section 2) and the platforms under test, including the bundled WS-* standard implementations (section 3). Now, the approach to the execution of the compatibility test has to be determined.

As already mentioned, compatibility is the ability of two protocols resp. protocol implementations, such as the WS-* frameworks, to interoperate, i.e., there is at least one possible successful conversation between two communication partners. This kind of compatibility is sometimes reffered to as *partial compatibility* in contrast to *full compatibility*, which means that any possible conversation generated by a sender A is understood by a recipient B, and vice versa. For the sake of simplicity, the *partial compatibility* is just denoted as compatibility in the context of this work [BCT06].

In order to determine whether two WS-* framework implementations are compatible, policies (in particular, policies according to WS-Policy) are used to specify concrete test scenarios. Therefore, the WSDL of a sample Web service is configured with a policy. This policy should contain exactly one policy alternative to force a communication partner (service consumer) to use exactly this policy and not any other policy. However, configuring a WS-Policy with exactly one alternative is not always possible due to platform constraints. In such a case, the amount of policy alternatives is kept as low as possible. Furthermore, it is possible that the Web services platform rejects a policy assertion, although it is syntactically and semantically correct. If that is the case, the feature under test is considered to be malfunctioning.

If a sample Web service, including the policy assertions, is deployed correctly, a service consumer has to recognize, to understand, and to process the policy. Afterwards, the Web service client[39] has to generate one or more SOAP messages taking the policy into account. These SOAP messages can contain several elements in the message header with meta information of the WS-* frameworks. Some frameworks use additional messages without payload containing meta-information. However, the message headers and the additional messages are transparent to the end users and the application logic, since the SOAP engine filters these metadata. Clearly, these messages and message parts provide information whether the client handles the policy correctly, and therefore the SOAP message headers and the SOAP messages, only containing metadata, have to be analyzed, too. The SOAP messages transmitted from the client to the server are not the only messages of interest; the response from the server to the client is important to determine the interoperability of WS-Stack implementations, too. The server can also reject client messages and, e.g., ask for more information or abort the communication. Due to the fact that the WS-* standards under examination define whole message exchange protocols, including multiple messages, the reason for malfunctioning can be located anywhere in this communication. Consequently, the whole message exchange between a Web service provider and a Web service consumer has to be logged and analyzed.

Due to the fact that the SOAP message traffic provides information on how a WS-* framework

[39]The Web service client is either a standalone Java Platform, Standard Edition (JSE) application including a runtime from one of the platforms or a Java Platform, Enterprise Edition (JEE) application deployed on one of the platforms.

is actually working (especially the fault messages), the policy of malfunctioning test cases can be altered in such a way that a QoS-supported communication becomes possible. To this end, i.a., SOAP test tools (soapUI 3.0.1[40] and SOAP Sonar Personal Edition 5.0.8[41]) are used to inject manually adjusted SOAP messages. This approach allows a more precise and quicker configuration of the QoS-aspects in SOAP messages than policies do. However, the test tools are only used to figure out which changes must be applied to a policy to get a test case to work and not to test the robustness of the implementations.

Figure 8 shows the test setup described above.

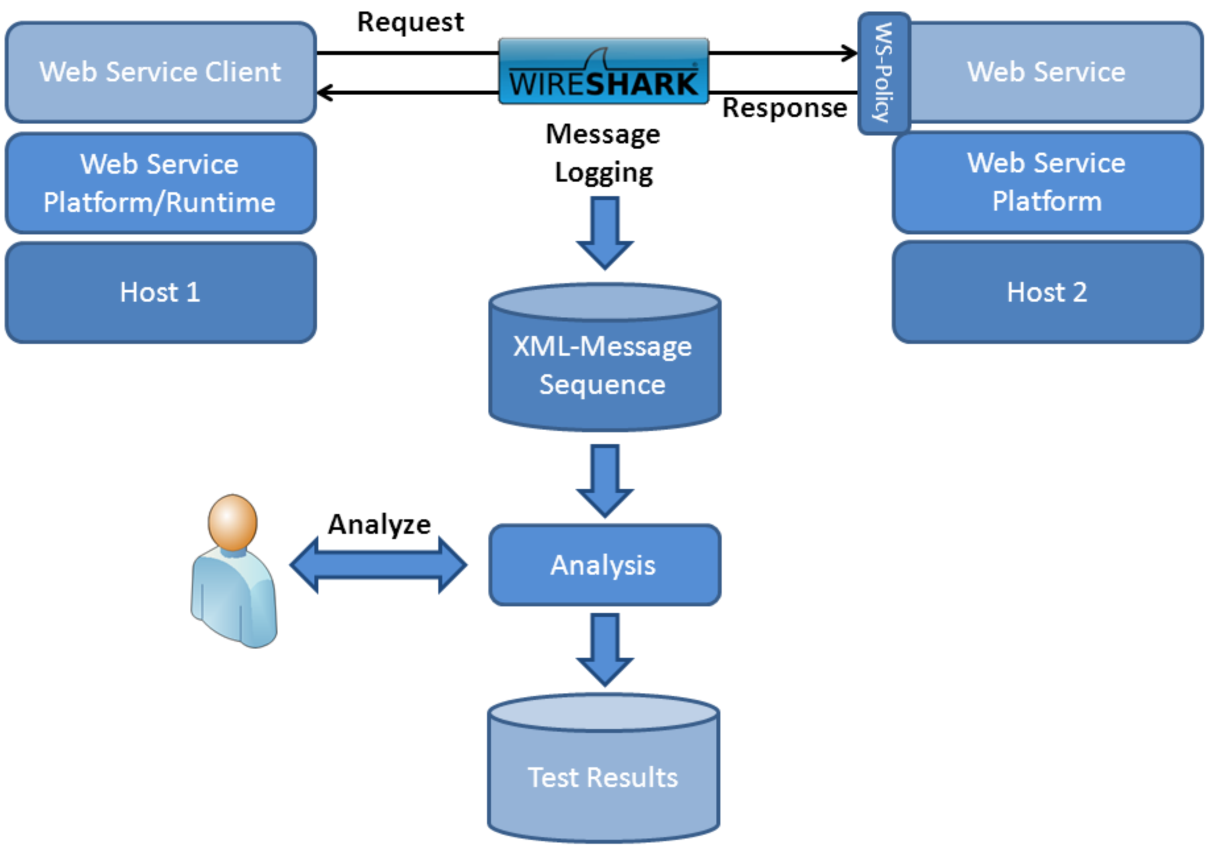

Figure 8: The test setup

Some studies, e.g., [PF04], show where Web services are susceptible to incompatibilities. Based on the results of the relevant publications (see section 7) and the test setup defined above, the following possible results for a single test run can be expected:

[40]available at `http://www.soapui.org`
[41]available at `http://www.crosschecknetwork.com`

1. **Server refuses WS-Policy:** The server does not accept the policy of the service, i.e., the service cannot be deployed on the server, or the server states to ignore the feature under test.

2. **Server cannot process WS-Policy correctly:** It is possible to deploy the sample service on the server, but the WSDL file containing the policy cannot be retrieved.

3. **Client refuses WS-Policy:** The client can retrieve the WSDL file but cannot process the policy and therefore no request is sent to the server, or the client states to ignore the feature under test.

4. **Client cannot process WS-Policy correctly:** The client can retrieve the WSDL file but cannot process the policy, and therefore a SOAP message without WS-* extensions is sent to the server. The server returns an error code.

5. **Server and client ignore WS-Policy:** Both, server and client ignore the published policy. The client sends a SOAP message without WS-* extensions, the server responds to this message with a regular SOAP message, i.e., neither an error code nor a message including WS-* extensions are returned from the server. This case also includes that parts of the policy are ignored by server and client.

6. **Server cannot process the initiating client message (initial request) correctly:** The client sends a SOAP message to the server observing the policy requirements, but the server is not able to process this message correctly. This means that the server responds with an error code to the client request.

7. **Client cannot process the initiating server message (response to initial request) correctly:** The client sends a SOAP message to the server observing the policy requirements, the server responds to this message observing the policy requirements, but the client is not able to process the response from the server.

8. **Client terminates communication:** The protocol initiation (client request, server response) has been performed without problems, but the client terminates the message exchange with a terminate message, specified in the according WS-* standard, before the whole communication protocol has been completed.

9. **Client aborts communication:** The protocol initiation has been performed without problems, but the client aborts the communication with a fault before protocol completion.

10. **Server terminates communication:** The protocol initiation has been performed without problems, but the server terminates the message exchange with a terminate message, specified in the corresponding WS-* standard, before the whole communication protocol has been completed.

11. **Server aborts communication:** The protocol initiation has been performed without problems, but the server aborts the communication with a fault before protocol completion.

12. **Protocol success:** The communication protocol has been performed successfully.

As already mentioned, a WS-* aware communication has at least two roles: the initiator (Web service client) and the responder (Web service provider). The initiator establishes the communication whereas the other party is responsive. Both roles differ significantly, so that, e.g., a platform can take the role of the responder without problems, but is incapable of being the initiator. This is the reason why every tested WS-* combination has to be executed with alternating initiator and responder. To determine the functional integrity of a certain WS-* feature on a platform at all, it is essential to test the features for cross-platform testing on a single platform before. Hence, there are four test runs for each WS-* feature and feature combination under test as depicted in Figure 9 (A and B denote the different platform types). Test run 1 (A-A) and 2 (B-B) are platform-internal and determine the functional integrity of the platform, test run 3 (A-B) and 4(B-A) are cross-platform tests and determine the compatibility of the two platforms.

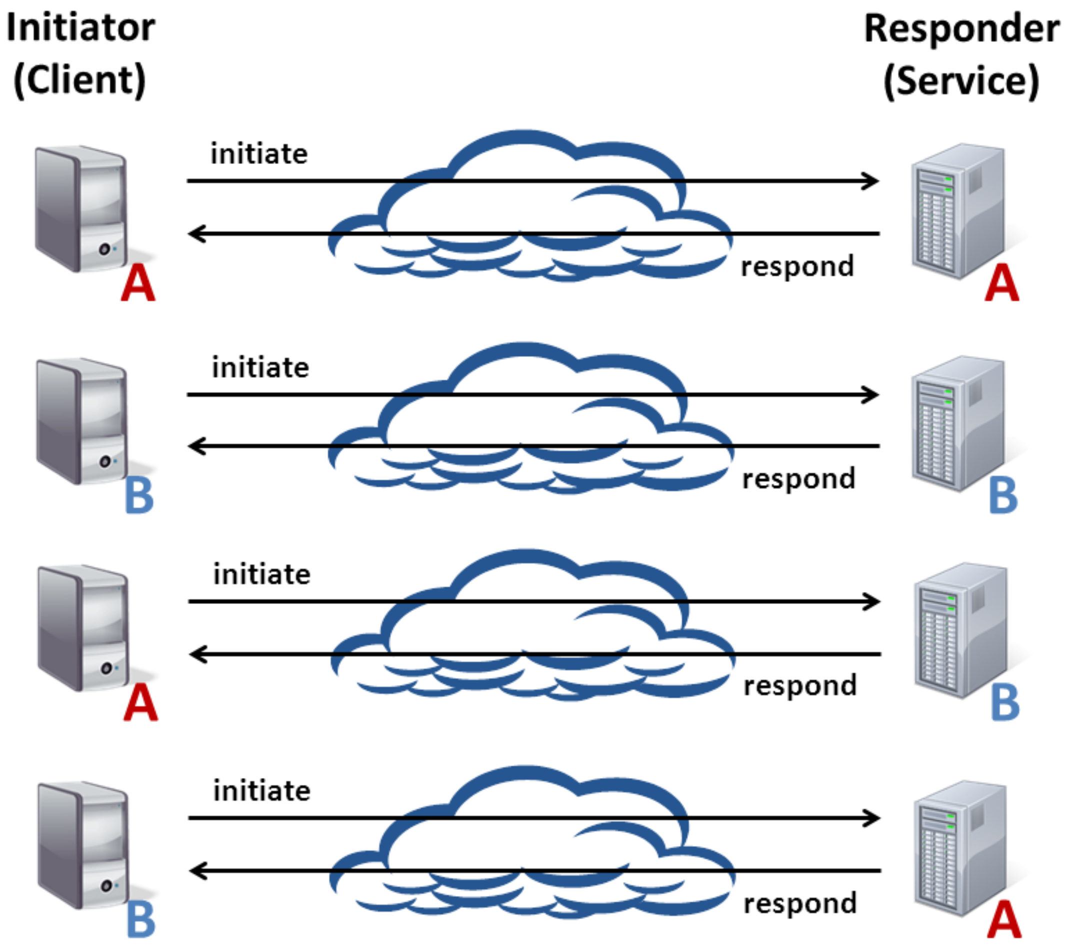

Figure 9: The four test runs of each test case

In general, SOA-Testing is a relatively new field in software testing and thus a *hot topic*. Although several authors have examined robustness or performance issues of Web services and even WS-* frameworks in homogeneous and heterogeneous environments (see the related work in section 7), there have only been few publications about compatibility issues of different Web services platforms. For this reason, the test method of this work employs concepts from protocol and software testing.

Since the tests focus on QoS features, which are characterized as non-functional properties of a system, the test approach is to perform functional testing of the non-functional properties. The tests are executed on the basis of the specification of the corresponding WS-* standards (*specification-based*) and are designed to cover any defined assertion/element of the WS-* standards (*structural testing*). Therefore, the test cases are defined as specification-based structural testing [ZHM97].

However, these test cases are not able to achieve the same results as a formal analysis of the standards and implementations. Yet, they are an adequate way to determine which functions are compatible and which are not. In addition, the results of the tests does not only provide a compatibility analysis of two WS-* implementations, but also an analysis of to what extent the WS-* frameworks under test implement the features intended by the corresponding standards (*coverage analysis*).

Considering the test setup and the determination of four test runs for every feature, the *How to test* is already clear but not the *What to test*. Obviously, it makes no sense to start with complex test cases including a series of policy assertions without assuring that each of these assertions is processable on both platforms. Therefore, the following test procedure has been developed: First of all, the functionality of the WS-* frameworks is tested in isolated test cases (see 4.1). In these cases each feature is tested in isolation (at least as far as possible). Second, a selection of the most common and most important features of WS-Security and WS-ReliableMessaging is tested in combined test cases, which is presented in section 4.2. Finally, the practical suitability of the tested WS-* implementations is verified in section 4.3. Each step in this test approach is based on the results of the previous step: if an isolated test case detects an inability to interoperate, the feature will be excluded from the combined test cases. As with the result of the isolated test cases, the results of the combined feature test cases are employed for the practical test: malfunctioning features are not included in these practical scenarios.

4.1 Isolated Function Tests

The first step in compatibility testing is to determine whether certain functionalities are available and well-working in each implementation under test. For this reason, each particular function of a WS-* framework is tested in isolation. This, of course, is not possible for every feature of the frameworks under test, since some features are only usable in a broader context. For example, a protection assertion such as signing or encrypting SOAP message parts cannot be declared in a policy without a valid security binding. In such cases, the related assertions are tested together.

The first step in compatibility testing needs a basis: the functionalities provided by the WS-* standards must be analyzed in order to identify every feature to be tested. Based on this analysis, a test plan containing test cases is developed. Afterwards, these tests are implemented as executable test cases. The tests are then executed accordingly to the test setup defined above. The following step-by-step plan describes the steps to be taken for each particular feature in detail:

1. Analysis of the WS-* frameworks under test to determine the functions to be tested
2. Development of a test plan containing test cases to test all the functions determined in step 1
3. Development of concrete implementations of the test cases in accordance with the test plan, consisting of
   - a Web service,
   - a WS-Policy assigned to this service,
   - and a Web service client to invoke the service.
4. Execution of the test cases:
   (a) Deployment of the Web service on both platforms on host 1, preparation of the Web service client on both platforms or using both runtimes on host 2
   (b) Invocation of the sample Web service by the Web service client in the following order (on the left-hand side, host 1 with the Web service client, on the right-hand side, host 2 with the Web service):
      - platform A (as client) - platform A (as server)
      - platform B (as client) - platform B (as server)
      - platform A (as client) - platform B (as server)
      - platform B (as client) - platform A (as server)

   (c) Documentation of the test execution and the test output
5. Analysis of the test documentation

The result of step 1 and step 2 is a list of functions to be tested and concrete test cases (the test plan) to validate the availability of these functions. To execute the tests, the test cases must be implemented in the test environments. A Web service with variable policy features is developed for this purpose. The test cases are executed in accordance with the test setup depicted in Figure 9. The preparatory measures and realization strategies for the test cases are presented in section 5, the implementation of the test cases and the results of the test execution are discussed in section 6.1.

4.2 Combined Function Tests

After the determination of the compatible features in each implementation under test (IUT), the next step is to test the functional features in combination. The combination of the features to be tested can point out new incompatibilities between the IUTs, due to the fact that additional features or alternating settings may change the behavior of an implementation. However, the combination of all available and compatible features may lead to a *test case explosion*. In order to prevent this problem the total amount of test cases must be kept as small as possible or lowered to a reasonable level. Therefore, restrictions to the test cases must be defined.

First, it is necessary to define a test goal for the execution of the combined test cases. Since the target of this work is to provide a compatibility analysis of WS-Security and WS-ReliableMessag ing implementations, this goal is already clear: testing combinations of WS-Security and WS-ReliableMessaging features. In any case, the WS-Security Policy standard provides much more possible settings than the WS-ReliableMessaging Policy standard does. Therefore, the WS-SecurityPolicy settings must be restricted to a reasonable level.

Considering the WS-SecurityPolicy standard, five WS-Security test classes are defined:

- Protection Assertions
- Token Assertions
- Security Bindings
- Supporting Tokens
- WS-Security and WS-Trust Options

For each of these patterns reasonable restrictions may be determined. The combination of signed and encrypted message elements is important in practical use, and thus the protection assertions should be tested in combination, at least as far as they are available and compatible. Besides the protection on message level, it is also possible to guarantee the integrity and confidentiality of a message on transport level. On this account, the SSL/TLS interoperability is tested, too. Considering the token assertions, the token types used in the combined cases are restricted, since many of the defined tokens are unusual in practical use. The selected token types are all used in one version, which will be determined after the isolated test cases have been performed successfully. The token types to be used in the combined test cases are:

- X509 Token
- Username Token
- SecureConversation Token

The security binding options are limited to one functional `sp:AlgorithmSuite`, a strict SOAP security header layout without `wsu:Timestamp` and an `sp:AsymmetricBinding` with separate tokens for the initiator and the recipient. For test cases with symmetric cryptography, a symmetric binding will be employed instead of the standard asymmetric binding. Since the support of `sp:SupportingTokens` in an IUT has already been proven in the isolated test cases, the `sp:SupportingTokens` are not covered in detail. The WS-Security and WS-Trust options primarily define *must support* characteristics, which are less interesting, and no compatibility issues concerning these settings can be expected. Thus, the only relevant feature of this pattern is the `sp:RequireSignatureConfirmation` assertion that will receive our attention in the combined test cases.

By analogy with the isolated test cases, the step-by-step plan for the combined test cases looks as follows:

1. Determination of the compatible features of the WS-* frameworks
2. Development of a test plan in accordance with the results of step 1 and the restrictions defined above
3. Development of concrete implementations of the test cases in accordance with the test plan (particular realization as defined in step 3 of the step-by-step plan in section 4.1)
4. Execution of the test cases (particular realization as defined in step 4 of the step-by-step plan in section 4.1)
5. Analysis of the test documentation

Like in the case of the isolated function tests, section 5 describes the test preparation and section 6.2 discusses the test cases and test results.

4.3 Practicability Test: Secure WS-ReliableMessaging Scenario

After having done the analysis of the features of the WS-* implementations in isolation and combination, it is necessary to figure out if the two IUTs are able to work together in a more complex, practical use case, which employs multiple WS-* features. As mentioned in the introduction, the *Secure WS-ReliableMessaging Scenario* [BMPV06] [GHB05] [WS-08, pages 28-36] defines a combination of QoS features that are frequently required in B2Bi use. This scenario defines a protocol which is far more complex than the isolated communication protocol run of a single WS-* framework. It combines the use of WS-Security (in particular WS-SecureConversation) and WS-ReliableMessaging. Both frameworks are combined in such a way that the whole WS-ReliableMessaging sequence is completely integrity and confidentiality protected.

QoS features like document security and reliability are core requirements for B2Bi, which are also defined by open e-business process standards [Ros10, pages 20/21] [OAS06a, pages 71-73]. The *Secure WS-ReliableMessaging Scenario* covers the core requirements of these e-business standards employing a combination of WS-Security, WS-SecureConversation, and WS-ReliableMessaging features. However, the combination of security elements is a critical issue, since many combinations do not maintain the assured QoS properties. On this account, [BMPV06] gives a symbolic and cryptographic analysis of the *Secure WS-ReliableMessaging Scenario* and proves the formal correctness of this communication protocol. [BMPV06] proves the correctness on the basis of the specifications and thus a successful execution of the scenario also depends on standard compliance of the WS-Stack implementations. Consequently, the *Secure WS-ReliableMessaging Scenario* is a practical use case for compatibility testing of WS-* implementations of different platforms.

Although this test case is much more complex then the previous cases, the approach to the test is almost the same. The step-by-step plan is defined analogously to the isolated test cases and the combined ones:

1. Analysis of the scenario definitions:
   - Determination of the WS-* features required for the scenario
   - Determination of the message sequence
2. Development of a policy in accordance with the results of step 1
3. Platform-specific adaptation of the policy and development of a concrete implementation of the test case (particular realization as defined in step 3 of the step-by-step plan in section 4.1)
4. Execution of the test case (particular realization as defined in step 4 of the step-by-step plan in section 4.1)
5. Analysis of the test documentation

Section 5 lists the preparatory measures for the scenario performance, whereas section 6.3 defines the policies for the *Secure WS-ReliableMessaging Scenario* and presents the test results of the scenario compatibility analysis with particular reference to standard/scenario compliance.

5 Test Preparation

Section 4 has presented the general method of the tests. The section “Test Preparation” establishes ties between the test method and the execution of the test cases by determining the features under test in section 5.1 and realizing the preparatory measures on the two platforms under test in section 5.2.

5.1 Determination of the Features under Test

After the introduction of the standards under test in section 2 and the test method in section 4, the standards are then analyzed to determine what features must be tested to achieve a good coverage of the isolated test cases. Since the test method stipulates a policy-driven approach to the compatibility analysis, WS-Policy, WS-ReliableMessaging Policy (WS-RM Policy), and WS-SecurityPolicy are the base for the test cases. Each of these specifications is used to develop the test plan for the isolated function tests.

Since the WS-Policy specification does not define any functional restrictions to WS-Security or WS-ReliableMessaging, it is only a framework for the policy configuration. The WS-RM Policy and WS-SecurityPolicy are the WS-Policy based specifications to define the WS-Security and WS-ReliableMessaging features in a Web services communication. Both standards are analyzed to determine the available features and the possible combinations of features. Each feature may have multiple settings, e.g., the `wsrmp:DeliveryAssurance` defined by the WS-RM Policy has all in all six possible settings. In this case, it must be also determined which combinations of the features are possible, i.e., whether the settings `OR` (none, one, or multiple alternatives are valid), `XOR` (exactly one alternative is valid), and `AND` (all alternatives are valid with regard to the multiplicity) are connected or the features are independent of each other? Besides, some of the features are mandatory, while others are optional. The mandatory features must not be omitted, and therefore test cases of other features must include these mandatory features in order to be functional. For example, if a WS-SecurityPolicy protection assertion is defined in a WS-Policy, it is mandatory to define a binding assertion in the same policy. In such cases, the mandatory part of the policy is an already tested standard configuration of the feature so that the focus of the test case is on the actual feature under test. Additionally, a certain setting may be present multiple times and therefore the multiplicity of a certain setting has to be considered, e.g., a WS-SecurityPolicy protection assertion may include none, one, or multiply `sp:SignedParts` assertions. For this reason, the following multiplicities for assertions are defined: ’0/1’ (none or exactly one occurrence), ’1’ (exactly one occurrence), ’0*’ (no occurrence or more), and ’1*’ (at least one occurrence).

In order to keep track of things the test plan is structured following the patterns of the WS-SecurityPolicy standard plus the WS-RM Policy assertions. They are:

- WS-RM Policy Assertions
- WS-SecurityPolicy Protection Assertions

- WS-SecurityPolicy Token Assertions
- WS-SecurityPolicy Security Bindings
- WS-SecurityPolicy Supporting Tokens
- WS-SecurityPolicy WS-Security and WS-Trust Options

An example of the approach for generating the test cases is presented for the WS-RM Policy assertions. The complete list of test cases is located in Appendix B. Based on the assertion defined by [OAS09a, page 8] (see Listing 9), the functions offered and their combination possibilities are determined.

```
1  <wsrmp:RMAssertion (wsp:Optional="true")? ... >
2    <wsp:Policy>
3
4       (<wsrmp:SequenceSTR /> |
5        <wsrmp:SequenceTransportSecurity /> ) ?
6
7       <wsrmp:DeliveryAssurance>
8          <wsp:Policy>
9             (<wsrmp:ExactlyOnce /> |
10             <wsrmp:AtLeastOnce /> |
11             <wsrmp:AtMostOnce />  )
12            <wsrmp:InOrder /> ?
13         </wsp:Policy>
14      </wsrmp:DeliveryAssurance> ?
15
16   </wsp:Policy>
17   ...
18 </wsrmp:RMAssertion>
```

Listing 9: Structure of a `wsrmp:RMAssertion` (WS-RM Policy) element

The `wsrmp:RMAssertion` defines an optional *sequence security policy* (feature test # 2) in lines 4 and 5 of the Listing 9. According to the structure of the `wsrmp:RMAssertion`, a WS-ReliableMessaging sequence may be bound either to a security token or to a session of a transport-level security protocol. Since these assertions are mutually exclusive the logical operator is an 'exclusive or' (`XOR`). Due to the fact that both settings of the feature *sequence security* are optional, the multiplicity of the assertions is '0/1'. The lines 7 - 14 of the Listing 9 define the feature *delivery assurance* (feature test # 3), which is also optional. It is necessary to set one of the delivery assurances `wsrmp:ExactlyOnce`, `wsrmp:AtLeastOnce`, or `wsrmp:AtMostOnce` when using this feature. Therefore, these settings are connected using the `XOR` operator with the multiplicity '1', i.e., exactly one of these settings must be selected. Additionally, the *delivery assurance* may require ordered delivery. The `wsrmp:InOrder` delivery setting is usable in free combination with other delivery assurance settings, thus the logical operator to connect the `wsrmp:InOrder` setting to other `wsrmp:DeliveryAssurance` settings is an `AND`. Since the `wsrmp:InOrder` assertion may be present or not, the multiplicity of this setting is '0/1'. However, the test cases do not test the functionality of the *delivery assurance* features, but only if the participating parties are able to process the assertion. Since it is outside the scope of this work to inject errors to the communication, it cannot be determined whether a message is delivered, e.g., exactly once. Therefore, it is assumed that a *delivery assurance* test case passes if the assertion is not refused by the service platform and the service client. This rule applies to any policy assertion that has no effect on the SOAP message.

Table 2 summarizes the results of the analysis of the `wsrmp:RMAssertion`, where '# 1' denotes the basic WS-ReliableMessaging case without optional features, '# 2' denotes the *sequence security* feature, and '# 3' denotes the *delivery assurance* feature. The column "definition" refers to the definition of the setting in the specification.

#	Op1	Op2	Mul	Setting	Definition
1	-	-	1	Basic `RMAssertion`	[OAS09a, lines 132-134]
2	`XOR`	-	0/1	`SequenceSTR`	[OAS09a, lines 141-144] [OAS09a, lines 263-277]
			0/1	`SequenceTransportSecurity`	[OAS09a, lines 145-149] [OAS09a, lines 278-307]
3	`AND`	`XOR`	1	`ExactlyOnce`	[OAS09a, lines 166/167]
			1	`AtLeastOnce`	[OAS09a, lines 168/169]
			1	`AtMosttOnce`	[OAS09a, lines 170/171]
			0/1	`InOrder`	[OAS09a, lines 172/173]

Table 2: Features and settings of the `wsrmp:RMAssertion` (WS-RM Policy) element to be tested

The results summarized in Table 2 are the base for the development of the test cases. The logical operations give information on how many test cases are necessary to cover all possible settings of a feature. Besides the basic case (an empty `wsrmp:RMAssertion`), there are the *sequence security* properties with two additional cases and the *delivery assurance* properties with another six test cases. Table 3 enlists the test cases derived from the standard analysis where each case is assigned to a feature. The sequence identifier of a test case depends on the sequence identifier of the feature, e.g., test case 2.2 is a case to test a setting of feature number 2, *sequence security.*

#	Feature	Case	Settings
1	Basic `RMAssertion`	1.0	-
2	Sequence Security	2.1	`SequenceSTR`
		2.2	`SequenceTransportSecurity`
3	Delivery Assurance	3.1	`ExactlyOnce`
		3.2	`AtLeastOnce`
		3.3	`AtMostOnce`
		3.4	`ExactlyOnce + InOrder`
		3.5	`AtLeastOnce + InOrder`
		3.6	`AtMostOnce + InOrder`

Table 3: Test plan for the features of the `wsrmp:RMAssertion` (WS-RM Policy) element

This approach has been applied to the entire WS-RM Policy and WS-SecurityPolicy standards in order to develop the test plan for the isolated function tests. The results of the analysis are located in Appendix B.

The determination of the test cases for the combined function tests and the practicability test is discussed in the sections 6.2 and 6.3, respectively, since these test cases are based on the results of the isolated function tests, which are presented in section 6.1.

5.2 Setup of the Test Environment

After the test cases for the compatibility analysis have been developed, they are realized on the platforms to execute the test runs. However, the test platforms have to be prepared before the tests can be run. These preparatory measures include, i.a., the key exchange, the policy adaptation to platform-specific conditions, and the service configuration.

As already mentioned in section 2, keys play a special role in WS-Security. For this reason, most WS-Security scenario descriptions propose the use of keys that are certified by an independent certification authority (CA). In this way, it is not necessary to confirm the key exchange between a client and a service explicitly, but if a client C trusts the CA X and a service S has an key which is certified by CA X, C trusts the key of S, too. In practical use cases of B2Bi such an approach makes sense, but in a test environment it only causes administration effort. On this account, it is assumed that both platforms trust the signature and encryption keys of each other. To realize this, X509 certificates containing the public key(s) are exported from the keystore of each platform and imported to the truststore of the other platform.

Another platform-specific point is policy adaptation. Whereas this point is almost no problem for the Sun GlassFish Platform, the IBM WebSphere Platform requires IBM WAS policy set compliant policies. Therefore, a policy set is developed for each test case including the required assertions (at least as far as it is possible due to the restrictions of the IBM policy sets). In addition, the bindings on client and server side of the IBM WebSphere Platform must fit the requirements of the policy set, e.g., the correct signature and encryption keys must be chosen or the correct configuration of the incoming and outgoing authentication tokens has to be guaranteed.

The policy is not the only point that must be adapted to platform-specific conditions, the functional and non-functional properties of the service must also be given on both platforms. To get a Web services client working with a QoS-aware service, even within platform boundaries, is not easy at all. The IBM WebSphere Platform as well as the Sun GlassFish Platform provide client runtimes to enable the QoS-features of the WS-* frameworks. Whereas a preliminary study verified the availability of the GlassFish runtime for standalone JSE clients for Web services calls, the WebSphere runtime environment shows weaknesses in WS-Security applications. On this account, the WebSphere client is a JEE Web application deployed on the WAS, calling the service via Java Server Pages (JSP). The GlassFish JSE standalone client has been proven successful, except for the test cases using transport layer security, since the JSE client runtime is not able to realize SSL. For the transport level security test cases, a Web application with a JSP-based user interface is deployed on the GlassFish server. Since JAX-RPC does not support the use of WS-* features, all clients employ JAX-WS for invoking the services.

In addition to the test services, another type of service is set up on both servers, the Security Token Service (STS). Whereas the IBM WebSphere Platform includes a predefined STS, the

STS of the Sun GlassFish Platform is self-developed. The development, deployment, and integration of a GlassFish STS is described in the *Metro Users Guide*[42] in detail.

In order to analyze the message traffic between client, service, and additional services, such as the STS, the message traffic is captured, recorded, and visualized. The interception of message sequences can be realized using various tools. For example, both platforms include a TCP/IP monitor for message analysis. However, these tools are relatively simple. Thus, the network package analyzer *Wireshark*[43], version 1.2.6, is employed for logging the SOAP message traffic between Web service provider and Web service consumer. The logged messages are an evidence for the functional integrity of the WS-* implementations. The Wireshark trace is the starting point for the analysis of the SOAP messages. On the basis of the captured message exchange, the test results are determined.

[42]available at `https://metro.dev.java.net/guide/`, section 12.8
[43]available at `http://www.wireshark.org`

6 Test Results

Now that the test cases are determined and the platforms are configured for the test case execution, the compatibility analysis is ready to be performed. As defined in section 4, the analysis is performed in three stages: at first the functions of the WS-* frameworks are tested in isolation, subsequently the functioning and compatible features of the WS-* frameworks are combined to larger test cases (in accordance to the restrictions defined in section 4.2), and last but not least the WS-* implementations are tested for their practical applicability in B2Bi when the compliance to the *Secure WS-ReliableMessaging Scenario* is analyzed.

In accordance with the test method defined in section 4, each test case is performed four times to cover all possible client/service configurations, i.e., WebSphere client - WebSphere service (test run 1), GlassFish client - GlassFish service (test run 2), WebSphere client - GlassFish service (test run 3), and GlassFish client - WebSphere service (test run 4). See Figure 9 on page 39 for a visualization of the four test runs.

However, before a single test case may be carried out, it must be guaranteed that both platforms are able to interact at all. For this reason, the basic case of QoS-unaware Web services communication via SOAP messages is performed to prove the IBM WebSphere/Sun GlassFish Web services interoperability. Since there are protection assertions for SOAP with attachments, a basic SwA case is also considered.

Table 4 gives an overview of the test results of the basic Web services interoperability cases without use of WS-*. The first two rows define the test environment (Which platform hosts the service? Which platform acts as client?) and the first column identifies the test case number according to the complete list of test cases in Appendix B. The results of the test cases are denoted using the numbers for possible test results defined in section 4 on page 38. The results in Table 4 show that the *Basic SOAP* use case be performed successfully for all four cases, whereas the *Basic SwA* use case cannot be performed, since both platforms ignore the corresponding assertions in the WSDL. On implementation level a platform-specific realization of a SwA message exchange is possible, however, this is outside the scope of this work.

Service ▶	**WebSphere**	**GlassFish**	**WebSphere**	**GlassFish**
Client ▶	**WebSphere**	**GlassFish**	**GlassFish**	**WebSphere**
Case ▼				
0.1	12	12	12	12
0.2	5	5	5	5

Table 4: Test results of the basic test cases

6.1 Isolated Function Tests

Before more sophisticated test cases can be performed, every setting of every feature has to be tested. Not only does this test procedure answer the question whether the tested feature is interoperable in the two WS-* implementations, but it answers other questions as well such as whether the tested feature is implemented in both WS-Stacks, whether the server is able to deploy the corresponding policy, and whether the client is able to understand the policy correctly. Thus, apart from the compatibility analysis of the implemented features in both WS-* implementations is also given.

Due to the numerous features of WS-ReliableMessaging and, especially, WS-Security, the test execution and the test results are divided into six groups of functional areas. Since the WS-RM Policy standard only defines *sequence security* and *delivery assurance* assertions, it forms the first test case group (see section 6.1.1). In contrast to this, the WS-SecurityPolicy standard is much more extensive, so its test cases are divided in five groups according to the patterns defined in the WS-SecurityPolicy standard: protection assertions, token assertions, security binding assertions, supporting token assertions, and WS-Security and WS-Trust options. These subdivisions have already been used in section 5.1 to organize the features under test and the test cases.

The subsequent sections present the results of the test execution showing extracts from the Table containing the results of isolated test cases, that is located in Appendix C. The WS-Security relevant test cases are primarily tested with an `sp:AsymmetricBinding` and `sp:X509Token` types, for each exceptions is given a particular explanation.

6.1.1 WS-ReliableMessaging

After introducing the test execution and the notation of the test results of the isolated function test cases, the WS-ReliableMessaging section discusses these topics in more detail than the following sections. Since the approach to the test execution and the notation of the test results is similar for each test case group, only modifications and specialties are discussed in the subsequent sections.

As already defined in section 5.1, the test execution starts with a general, basic test case and is successively specialized to more sophisticated test cases, e.g., the basic WS-ReliableMessaging test case (1.0) only includes an empty `RMAssertion` element and is extended with *sequence security* (2.x) resp. *delivery assurance* (3.x) assertions. These cases are converted to policies: For the Sun GlassFish Platform a WS-Policy with the corresponding WS-ReliableMessaging assertions, for the IBM WebSphere Platform a corresponding WS-ReliableMessaging policy set.

However, the IBM WAS policy set definition of the WS-ReliableMessaging features is not based on the WS-RM Policy and is strongly restrictive. Most of the settings intended by the test cases are not realizable as a policy set and therefore a part of the test cases is not executable on the IBM WebSphere Platform. Thus, it is impossible to deploy either a policy set corresponding to the basic test cases or policy sets with *sequence security* assertions. The IBM WebSphere Platform only allows the setting `wsrmp:ExactlyOnce` with the optional setting `wsrmp:InOrder`, and therefore only the test cases 3.1 and 3.4 are able to be carried out since any other assertion is rejected by the WAS. When the IBM WebSphere Platform acts as service container, these two cases are executed successfully for test run 1 and 3. However, a WebSphere client is not able to interpret a WS-ReliableMessaging policy other than the policy set-generated WS-ReliableMessaging assertions. This means that all cases in test run 4 fail. In contrast to the IBM WebSphere Platform test runs, the Sun GlassFish Platform-internal test runs are executed successfully except for test case 2.1. The Wireshark message trace does not prove the use of a `wsrmp:SequenceSTR` (as intened by the WS-RM Policy standard). An additional test case that defines a Security Token in a WS-SecurityPolicy and a `wsrmp:SequenceSTR` in a `wsrmp:RMAssertion` shows that the Sun GlassFish Platform employs the WS-SecurityPolicy-defined for the `wsrmp:SequenceSTR`. Table 5 summarizes the test results using the same notation as in Table 4.

Service ▶	**WebSphere**	**GlassFish**	**WebSphere**	**GlassFish**
Client ▶	**WebSphere**	**GlassFish**	**GlassFish**	**WebSphere**
Case ▼				
1.1	1	12	1	4
2.1	1	5[b]	1	4
2.2	1	12	1	4
3.1	12[a]	12	12[a]	4
3.2	1	12	1	4
3.3	1	12	1	4
3.4	12[a]	12	12[a]	4
3.5	1	12	1	4
3.6	1	12	1	4

[a] The configuration of the policy set permits to enable this feature.
[b] If a WS-SecurityPolicy including a Security Token is defined in addition to the `wsrmp:SequenceSTR` assertion, the Sun GlassFish Platform references this token.

Table 5: Test results of the WS-ReliableMessaging test group

6.1.2 WS-Security Protection Assertions

The WS-SecurityPolicy standard defines seven protection assertions that determine what parts of a SOAP message are protected or required in a SOAP message. Each protection assertion is tested in multiple test case considering the message parts to be protected, i.e., the whole SOAP message `env:Body`, parts of the SOAP message `env:Body`, parts of the SOAP message `env:Header`, or attachments of a SOAP message. All message parts except the SOAP attachment may be protected either by the QName of the element or using an XPath expression.

Interestingly, the Sun GlassFish Platform does not support the use of XPath expressions in such policy assertions, as the `sp:SignedElements` (test cases 5.1 - 5.3), the `sp:Encrypted-Elements` assertions (test cases 7.1 - 7.3), the `sp:ContentEncryptedElements` (test cases 8.1 -

8.3), and the `sp:RequiredElements` (test cases 10.1 and 10.2). The IBM WebSphere Platform supports these protection assertions, however it is not possible to sign a part of the `env:Body` (test case 5.1) or to encrypt the whole `env:Body`. The other XPath-based test cases have been performed successfully on the IBM WebSphere Platform. Obviously, the XPath-based protection assertions are not suitable to protect SOAP message sequences between the two platforms.

On this account, particular focus is put on the QName-based assertions. Since the SwA assertions are not processable by both servers, the `sp:SignedParts` (test cases 4.1 - 4.3) and `sp:EncryptedParts` (test cases 6.1 - 6.3) assertions are only testable with the whole SOAP message `env:Body` (test cases 4.1 and 6.1) and parts of the SOAP message header (test cases 4.2 and 6.2). As Table 6 indicates, the IBM WebSphere Platform does not accept the signed `env:Body` (test case 4.1) of a SOAP message from the Sun GlassFish Platform. The reason for this is that the Sun GlassFish Platform signs the first element within the `env:Body` element, whereas the IBM WebSphere Platforms signs the `env:Body` element itself. The Sun GlassFish Platform accepts both ways of signing the `env:Body`, the IBM WebSphere Platform on the other hand requires the signature of the `env:Body` element itself. Since the WS-SecurityPolicy standard defines for the `sp:SignedParts/sp:Body` assertion that the "[...] soap:Body element, it's attributes and content [...]"[OAS09d, lines 427-429] are to be integrity protected, the Metro implementation of this feature is not standard-compliant.

Interestingly, the same problem does not occur with the `sp:EncryptedParts/sp:Body` assertion (test case 6.1), since both platforms encrypt the first element in the `env:Body` in this case. However, encrypted SOAP messages exchanged across both platforms result in key-specific problems. The IBM WebSphere Platform is not able to decrypt encryption keys from the Sun GlassFish Platform (test run 4). In addition to this, the IBM WebSphere Platform rejects any encrypted message header in a SOAP message from the Sun GlassFish platform.

The `sp:ContentEncryptedElements`, `sp:RequiredParts`, and `sp:RequiredElements` assertions are not implemented on both platforms, so these assertions cannot be tested.

6.1.3 WS-Security Token Assertions

The WS-SecurityPolicy standard defines eleven tokens which are used to protect or bind tokens and claims to the message. Nevertheless, both platforms do not implement all these token types. The tokens that are not implemented by both servers are: the `sp:SpnegoContextToken`, the `sp:SecurityContextToken` (not to be confused with the `sp:SecureConversationToken`), the `sp:RelToken`, and the `sp:KeyValueToken`. In addition, the operating system of the test environment does not allow to issue a `sp:KerberosToken`.

Each of the working token types is tested with sensible security bindings, protection assertion, and `sp:SupportingTokens` combinations. Since the WS-SecurityPolicy standard also defines various token versions for some token types, each version is tested in isolation to guarantee full coverage of all possible isolated function test cases. Additionally, general token settings can be defined for each token type. The general features such as the token inclusion values or the token issuer are tested with a standard X509 Token. Since the optional element `sp:Claims`

Service ▶	WebSphere	GlassFish	WebSphere	GlassFish
Client ▶	WebSphere	GlassFish	GlassFish	WebSphere
Case ▼				
4.1	12	12	9[a]	6[a]
4.2	12	12	12	12
5.1	6[b]	3	3	6
6.1	12	12[c]	12[c]	6[d]
6.2	12	12[c]	6[e]	6[d]

[a] The IBM WebSphere Platform rejects the message, since the Sun GlassFish Platforms signs the first element in the `env:Body` but not the `env:Body` itself.
[b] Rejected with the error message "An error occurred while parsing native data: The error message is: java.io.IOException: Error Reading XML."
[c] Only works with token inclusion value "Never".
[d] Rejected with the error message "Error occurred while decrypting encrypted Key; Key used to decrypt encrypted key cannot be null."
[e] Rejected

Table 6: Excerpt of the results of the WS-Security Protection Assertions test group

may contain an entire new policy and therefore allows hundreds of possible combinations, due to lack of time and resources, it is only checked whether the platforms accept the `sp:Claims` element or not.

More specific features and settings of token types are tested with the corresponding token type, e.g., `sp:RequireKeyIdentifierReference` for the X509 Token or `sp:RequireExternalUriReference` for the `sp:SecureConversationToken`.

Looking at the general token feature tests (test cases 11.1 - 14.3), it becomes apparent that these settings are primarily supported by the Sun GlassFish Platform. Especially the token inclusion values (except the value `Once`) are supported perfectly. The IBM WebSphere Platform accepts all token inclusion values (test case 11.1 - 11.5), too, but ignores all assertions and always includes the security tokens. This behavior causes an abort in the GlassFish - WebSphere communication in test case 11.1 (token inclusion value `Never`), whereas the GlassFish Platform ignores the included tokens in the cases 11.3 and 11.4. Even the derived keys assertions (test cases 14.1 - 14.3), which have been tested with the `sp:SecureConversationToken` assertion and a `sp:SymmetricBinding`, are due to the IBM WebSphere behavior not fully interoperable. With the WebSphere Platform as server, the test cases 14.2 and 14.3 are refused and case 14.1 is not processable by a GlassFish client. However, when a WebSphere client is used, the GlassFish server rejects the initiating client message, since the `env:Body` is not encrypted.

The test results reveal that both platforms do not support each token version. The `sp:WssX509V3Token10` (test case 17.6) and `sp:WssX509V3Token11` (test case 17.10) are the only two fully interoperable token versions, any other token type or token version is not supported in all four test runs across the platforms (Tested with token inclusion value `Always`). Interestingly, the `sp:UsernameToken` (tested as `sp:SupportingTokens` assertion) is supported in version 1.0 (test case 15.7) by both platforms. But, the IBM WebSphere Platform does not configure the token correctly, so that the fields username and password are left empty. Furthermore, the Sun GlassFish Platform encrypts the `sp:UsernameToken`, except if the `sp:NoPassword` option is selected.

The most interesting token type in the context of the *Secure WS-ReliableMessaging Scenario* is the `sp:SecureConversationToken` (test cases 21.1 - 21.7). Since all test cases of the `sp:SecureConversationToken` must include a `sp:BootstrapPolicy`, the test cases 21.1 to 21.6 focus on the functionality of the token, whereas test case 21.7 focuses on the `sp:BootstrapPolicy` compliance. This token type is supported by both platforms, as both platforms require the use of the `sp:SymmetricBinding` for this token type. Due to the encryption key problems, which already arose in the test cases 14.2 and 14.3, a cross-platform communication using a `sp:SecureConversationToken` is not possible.

Like with the `sp:SecureConversationToken`, the `sp:HttpsToken` (test cases 24.1 - 24.4) requires a special binding, the `sp:TransportBinding`. The `sp:TransportBinding` is not supported by the IBM WebSphere Platform (cf. section 6.1.4), and therefore a `sp:HttpsToken` cannot be used. In contrast to that, the GlassFish platform passes all `sp:HttpsToken` test cases. Like in the previous section, Table 7 shows an excerpt of the test results of the token assertions test group with regard to the discussed results.

Service ▶	**WebSphere**	**GlassFish**	**WebSphere**	**GlassFish**
Client ▶	**WebSphere**	**GlassFish**	**GlassFish**	**WebSphere**
Case ▼				
11.1	5[a]	12	6[a]	4[a]
11.2	5[a]	3[b]	3[b]	6[b]
11.3	12[a]	12	12[a]	12[a]
11.4	12[a]	12	12[a]	12[a]
11.5	12	12	12	12
14.1	12	12	3	12
15.7	12[c]	12	6[d]	6[c]
17.6	12	12	12	12
17.10	12	12	12	12
22.1	12	12	6[e]	6[f]
22.2	12	12	6[e]	6[f]
22.7	12	12	6[e]	6[f]

[a] The IBM WebSphere Platform always includes the `sp:BinarySecurityToken`.

[b] The TokenInclusion value `Once` is not supported by the Sun GlassFish Plaform.

[c] The IBM WebSphere Platform does not fill in any values, such as username or password.

[d] Rejected since the GlassFish client encrypts the `sp:UsernameToken`.

[e] The IBM WebSphere Platform returns the fault "Key object was not obtained."

[f] The Sun GlassFish Platform rejects the message, since the `env:Body` is not encrypted.

Table 7: Excerpt of the results of the WS-Security Token Assertions test group

6.1.4 WS-Security Binding Assertions

The security tokens are not solitary in a WS-SecurityPolicy but often embedded in a security binding. The WS-SecurityPolicy specification defines three types of security bindings: the `sp:AsymmetricBinding`, the `sp:SymmetricBinding`, and the `sp:TransportBinding`. Each of these bindings includes additional properties, such as the `sp:Layout` of the SOAP security header or the `sp:AlgorithmSuite`, to determine algorithms and key lengths for the SOAP message exchange.

The additional properties are tested in cooperation with an `sp:AsymmetricBinding`. These test cases (26.1 - 27.4) show that the IBM WebSphere Platform does not implement the `Sha256` algorithm, and therefore the test cases 26.9 to 26.16 fail with WebSphere clients (test runs 1 and 4). Another interesting point concerning the binding properties is the `wsu:Timestamp` (test cases 28.2, 29.3, and 30.7) support of the IBM WebSphere Platform. Although the WS-SecurityPolicy standard states that the `wsu:Timestamp` of a SOAP message must always be signed when present [OAS09d, lines 1718 - 1722], the IBM WebSphere does not sign the `wsu:Timestamp` automatically. In order to create a SOAP message with an integrity protected `wsu:Timestamp`, the IBM WebSphere employs an XPath expression inside the `sp:SignedElements` assertion to sign the `wsu:Timestamp` (see Listing 10). Interestingly, the `wsu:Timestamp` of the IBM Websphere Platform contains no `wsu:Expires` (used to state the expiration date).

```
<sp:SignedElements>
   <sp:XPath>/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name
       ()='Envelope']/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and
       local-name()='Header']/*[namespace-uri()='http://docs.oasis-open.org/wss/2004/01/
       oasis-200401-wss-wssecurity-secext-1.0.xsd' and local-name()='Security']/*[namespace
       -uri()='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility
       -1.0.xsd' and local-name()='Timestamp']</sp:XPath>
   <sp:XPath>/*[namespace-uri()='http://www.w3.org/2003/05/soap-envelope' and local-name()=
       'Envelope']/*[namespace-uri()='http://www.w3.org/2003/05/soap-envelope' and local-
       name()='Header']/*[namespace-uri()='http://docs.oasis-open.org/wss/2004/01/oasis
       -200401-wss-wssecurity-secext-1.0.xsd' and local-name()='Security']/*[namespace-uri
       ()='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.
       xsd' and local-name()='Timestamp']</sp:XPath>
</sp:SignedElements>
```

Listing 10: Additional `sp:SignedElements` assertion to sign the `wsu:Timestamp`

As already stated in section 6.1.3, the `sp:TransportBinding` (test case 28.1) is not supported by the WebSphere Platform, and thus none of the cross-platform tests works. Since the SSL-protection of Web services is an important requirement in B2Bi, the IBM WebSphere Platform supports the use of SSL-protected Web services connections using a separate policy type. The results of test case 28.* reveal that it is not possible to create an IBM WebSphere client consuming a HTTPS-protected Web service. This problem is also reflected in the disability of the IBM WebSphere IDEs (RAD and WID) to accept a SSL-secured Web service.

The IBM WebSphere Platform only supports the `sp:SymmetricBinding` with one `sp:ProtectionToken` (test case 29.2), but none of the other assertions of the `sp:SymmetricBinding`. However, this assertion does not work in test run 3 due to a key-specific problem and all assertions of test run 4 are refused by the IBM WebSphere client. Therefore, not a single `sp:SymmetricBinding` setting is interoperable over both platforms. The `sp:Asymmetric-Binding` works for the combination of a separate `sp:RecipientToken` and `sp:InitiatorToken` (test cases 30.1 and 30.3) in all four test runs. In addition, only the IBM WebSphere Platform supports the use of separated signature and encryption tokens for initiator and recipient (test cases 30.2 and 30.4). The additional settings of the `sp:AsymmetricBinding` assertion are not supported by the IBM WebSphere Platform, whereby test cases 30.5 to 30.7 in test run 4 fail due to the aforementioned encryption key problem or the client that refuses the policy. An interesting setting of the `sp:AsymmetricBinding` is the `sp:OnlySignEntireHeadersAndBody` assertion (test case 30.8), which provides a workaround to handle the problem of test case 4.1. As the platform-internal test run shows, this assertion forces the GlassFish Platform to sign the `env:Body` element of a SOAP message itself. Actually, the test run 4 of test case 31.8 is executed successfully, but the WebSphere Platform refuses policies including the `sp:OnlySignEntireHeadersAndBody` assertion. Thus, the problem is not solved for the third test run.

Table 8 gives a short overview of the results discussed in this section, the complete test results are located in Appendix C.

Service ▶	**WebSphere**	**GlassFish**	**WebSphere**	**GlassFish**
Client ▶	**WebSphere**	**GlassFish**	**GlassFish**	**WebSphere**
Case ▼				
28.1	1	12	1	3
28.*	3	-	12	-
28.2	1	12	1	3
29.1	1	12	1	3
29.2	12	12	6[a]	3
29.5	12[b]	12	6[a]	3
30.1	12	12	12	12
30.3	12	12	12	12
30.7	12[b]	12	12[b]	12
30.8	1	12	1	12

[a] The IBM WebSphere Platform returns the fault “Key object was not obtained.”

[b] With an XPath-signed Timestamp.

Table 8: Excerpt of the results of the WS-Security Security Bindings test group

6.1.5 WS-Security Supporting Token Assertions

The WS-SecurityPolicy standard defines eight `sp:SupportingTokens` assertions. Besides the basic `sp:SupportingTokens` (test case 31.1) assertion, the standard defines endorsed, signed, and encrypted tokens in seven variations and combinations of these properties (test cases 32.1 - 38.1). All these test cases are performed with an `sp:UsernameToken`, since this is the most common use case for `sp:SupportingTokens` assertions. Although the `sp:SupportingTokens` assertions permit the definition of a separate `sp:AlgorithmSuite` and additional protection assertions, these settings are not tested, since they do not introduce new functionality. It is assumed that the algorithms and the protection assertions work as with the isolated test cases to prevent a *test case explosion*. Therefore, the basic `sp:SupportingTokens` assertions with an `sp:UsernameToken` are tested for deployability on a platform and interoperability with a client.

While the IBM WebSphere Platform rejects all subtypes of the `sp:SupportingTokens` assertions except the basic assertion in case 31.1, all `sp:SupportingTokens` assertions are deployable on the Sun GlassFish Platform. The basic `sp:SupportingToken` assertion is assumend to be functional on the IBM WebSphere Platform. However due to the aforementioned `sp:UsernameToken` problem and the rejection of any other token type, it was not possible to prove the functional integrity of the `sp:SupportingTokens` assertion for test run 1.

The test cases 33.1, 34.1, 37.1, and 38.1 fail on the Sun GlassFish Platform, since the server is unable to find a certain reference in the message signature. All these cases are the *endorsing* `sp:SupportingTokens` and do not contain any `sp:SupportingTokens` in the WS-Security header. Interestingly, with the Sun GlassFish Platform as server, the `sp:SupportingTokens` assertion in test case 32.1 is also encrypted (in contrast to the policy definition as unencrypted token) and the test case 31.1 is not executable without the `sp:UsernameToken` property `sp:NoPassword`. The results of the previous test cases imply that test run 4 of test case 31.1 is not functional, since GlassFish encrypts the `sp:SupportingTokens` (general encryption problem, see section 6.1.2), the `sp:NoPassword` setting is not available on the IBM WebSphere Platform, and the IBM WebSphere Platform rejects any other token type than the `sp:UsernameToken` as `sp:SupportingTokens`.

6.1.6 WS-Security SOAP Message Security and WS-Trust Options

The WS-Security (test cases 39.1 - 40.7) and WS-Trust (test cases 41.1 - 41.9) options define additional requirements to the platforms that are independent of the other WS-SecurityPolicy pattern, e.g., whether the platform must support embedded tokens, or interactive challenges. If these requirements are defined in a policy and the server refuses one of these assertions, the message sequence is terminated with a fault.

The WS-Security options are divided in two groups: the WS-Security 1.0 options and the WS-Security 1.1 options. Although all WS-Security 1.0 options are also included in the WS-Security 1.1 options, both options are tested in isolation (39.1 - 39.4 and 40.1 - 40.4). The results of these tests show that both are supported to the same degree on both platforms. The additional assertions of the WS-Security 1.1 options are available on both platforms, except the requirement that both parties must be able to process references using encrypted key references. This requirement is not met by a WebSphere client, and thus the test runs 1 and 4 fail. The Sun GlassFish Platform as well as the IBM WebSphere Platform do not accept any WS-Trust assertion, no policy containing a WS-Trust option is deployable on both platforms.

6.2 Combined Function Tests

The definition of and the approach to the combined test cases described in section 4.2 is based on the assumption that most of the features are compatible. However, the results of the isolated function test cases show that the amount of available and compatible features for all four test runs is very low. Considering the limitations of the features in section 4.2, there is hardly any feature left to be tested for the combined function tests. The main reasons for this are:

- The IBM WebSphere Platform does not support WS-ReliableMessaging Policy
- Most of the WS-Security Protection Assertion tests fail, due to
  - Missing XPath support in WS-Policy assertions of the Sun GlassFish Platform
  - The cross-platform `wss:EncryptedKey` problem
  - Incorrect integrity protection of the `env:Body` element of the Sun GlassFish Platform
  - Missing support of the `sp:OnlySignEntireHeadersAndBody` assertion of the IBM WebSphere Platform
- Incompatibility of the Security Token types, except the basic `sp:X509Token`
- The IBM WebSphere Platform WS-Stack does not support the `sp:TransportBinding`, the WebSphere Platform IDEs are not able to consume a SSL-secured service.

Taking all relevant factors and isolated function test results into account, there is only one combined function test case left: the combination of SSL and signed (WS-Addressing) message headers. This case is specified as policy for both servers: for the Sun GlassFish platform a `sp:TransportBinding` using the `sp:Basic128Rsa15` algorithm is specified, whereas the policy set of the IBM WebSphere Platform employs an `sp:AsymmetricBinding` with a `sp:X509Token` for initiator and recipient and also a `sp:Basic128Rsa15` algorithm. The SSL-protection is regulated by another policy type in the IBM WebSphere Platform policy set. In both policies the use of WS-Addressing is enabled, and the WS-Addressing header parts `wsa:MessageID` and `wsa:RelatesTo` are intended to be signed.

The execution of the test case shows that a GlassFish client is fully interoperable with both platforms for SSL-secured and message-level integrity protected SOAP messages. Conversely, it is not possible to execute this test case using a WebSphere client.

6.3 Secure WS-ReliableMessaging Scenario

Considering the results from the isolated and combined function test cases, it becomes readily apparent that the *Secure WS-ReliableMessaging Scenario* is not interoperable in a heterogeneous IBM WebSphere/Sun GlassFish environment. However, this scenario is a relevant case for practical B2Bi use, and thus an analysis of the practicability of this scenario is also interesting for homogeneous environments, because the *Secure WS-ReliableMessaging Scenario* defines clear requirements to the message exchange. This raises the question whether it is possible to combine the WS-* frameworks of the platforms under test to meet the requirements of the *Secure WS-ReliableMessaging Scenario*.

The *Secure WS-ReliableMessaging Scenario* is designed to perform business transactions. In order to avoid frequent key exchanges in confidential and integrity-protected business transactions, a secure session has to be established. Therefore, a WS-SecureConversation session (identified by the so-called *security context* in terms of WS-SecureConversation) is established. The bootstrap message exchange for generating the secure conversation must also be integrity and confidentiality protected. For this purpose client and server are expected to be in possession of each other's asymmetric keys. These keys are used to encrypt and sign the WS-SecureConversation bootstrap messages, the recipient decrypts and verifies these bootstrap messages with the matching key. The emerging security context is identified by a Security Context Token (SCT) which is referred to in all subsequent messages.

After the generation of the secure session, the WS-ReliableMessaging sequence is initiated. Once it is open, the exchange of the messages containing the payload is performed. Once the acknowledgment of all payload-messages has arrived, the active WS-* sessions are closed: first the WS-ReliableMessaging sequence is terminated and after that the security context is cancelled.

Due to the fact that the definitions of [BMPV06] and [GHB05] are based on the WS-ReliableMessaging version 1.0 [BEA05], these definitions do not make use of the `wsrm:CloseSequence`, `wsrm:CloseSequenceResponse`, or `wsrm:TerminateSequenceResponse` messages which are included in the WS-ReliableMessaging standard since version 1.1. On that account, message number eight `wsrm:TerminateSequenceResponse` is inserted into the message sequence to achieve standard conformance as intended by [WS-08, pages 28/29]. The steps of the *Secure WS-ReliableMessaging Scenario* communication protocol are listed in the enumeration below, [GHB05, pages 34-60] gives a specification of the complete SOAP message structure for each step.

1. **RequestSecurityToken RST:** The client sends a message containing a RST to the server, asking the server to issue a SCT. This message must be signed and encrypted by the client using the server's public key.

2. **RequestSecurityTokenResponse RSTR:** The server responds with a RSTR message, containing the requested SCT. This message must be signed and encrypted by the server using the client's public key. The SCT must be used for signing and encrypting any message of the subsequent message flow.

3. **CreateSequence:** The client sends a `wsrm:CreateSequence` message to the server. This message includes a `wss:SecurityContextReference` to reference the SCT received in step 2. This SCT is used to sign the `wsrm:CreateSequence` message and encrypt the `wss:Signature`.

4. **CreateSequenceResponse:** The server responds to the CreateSequence request with a `wsrm:CreateSequenceResponse` message. This message is also signed and the `wss:Signature` is encrypted using the SCT.

5. **Payload Message:** The client now sends signed and encrypted messages containing the payload of this communication. Each payload message contains a WS-ReliableMessaging sequence header containing at least the sequence identifier and the according message number of this message in the sequence. In contrast to the previous two and following five messages, the payload messages have an encrypted `env:Body`.

6. **SequenceAcknowledgement:** The server acknowledges the receipt of the payload message(s) with a `wsrm:SequenceAcknowledgement`. The scenario definition proposes a single `wsrm:SequenceAcknowledgement` message with an empty `env:Body`. The acknowledgment headers are also signed and the `wss:Signature` is encrypted.

7. **TerminateSequence:** As soon as the client has received the acknowledgments for each message within the message sequence, it closes this sequence using the `wsrm:TerminateSequence` message defined by WS-ReliableMessaging. This message is signed and the `wss:Signature` is encrypted, too.

8. **TerminateSequenceResponse:** The server confirms the termination of the sequence with a signed `wsrm:TerminateSequenceResponse` message. The `wss:Signature` of this message is encrypted.

9. **CancelSecurityToken:** After termination of the WS-ReliableMessaging sequence, the client asks the server for cancellation of the WS-SecureConversation. The WS-Addressing header elements, the `wss:Signature`, and the `wsu:Timestamp` are integrity protected, additionally the signature is also encrypted, whereas the message `env:Body` is not protected.

10. **CancelSecurityTokenResponse:** The server confirms the `wst:CancelTarget` with a `wst:RequestedTokenCancelled` message. This message is protected in the same way as the `wst:CancelTarget` message.

Once the secure session is started, sender and receiver may create multiple WS-ReliableMessaging sessions for message transmission, as the publications do not impose any restrictions on that. However, in order to fulfill the requirements of the *Secure WS-ReliableMessaging Scenario* each WS-ReliableMessaging session that is started within the WS-SecureConversation session must be closed or terminated within the same session. After the security context is established, all signature and encryption processes are performed using keys derived from the SCT. The last two messages, canceling the security context, are protected using SCT-derived keys, too.

Table 9 enlists the requirements of the scenario definitions for confidentiality and integrity protection. The corresponding message type number is put in the first column. The other columns show the key required for encrypting (enc) or signing (sig) the message elements `wss:Signature` (Sign), `env:Body`, `wsu:Timestamp` (TS), WS-Addressing headers (WS-A), WS-ReliableMessaging headers (WS-R), and `wss:EncryptedKey` (EncKey). The entry SK_X stands for a session key, which is usually encrypted using the receivers public key (PuK_Y). The private key of a party is abbreviated PrK_Y. If a `wsc:SecurityContextToken` is used to derive keys, these keys are labeled as DK_{YX}. The indices X and Y are variables that are substituted in the Table. Instead of the X a number is inserted to identify different instances of the corresponding key type. These instances are independent of the type of party. The index Y determines whether the client (c) or the service (s) is the owner of the key, e.g., PrK_S denotes that the key used for the specified operation is the private key of the server, while DK_{C1} stands for a key derived from a `SecurityContextToken` by the client. Since multiple derived keys may be used in a SOAP message, each derived key has an assigned number, here '1'. The '•' symbol indicates that the corresponding element is present, but not protected, whereas the '○' means that the corresponding element is not present in this message.

Msg	**Sign**	**Body**		**TS**	**WS-A**	**WS-R**	**EncKey**
No	**enc**	**enc**	**sig**	**sig**	**sig**	**sig**	**enc**
1	SK_1		PrK_C			○	PuK_S
2	SK_2		PrK_S			○	PuK_C
3	DK_{C1}	•	DK_{C2}			○	○
4	DK_{S1}	•	DK_{S2}			○	○
5	DK_{C1}		DK_{C2}				○
6	DK_{S1}	•	•	DK_{S2}			○
7	DK_{C1}	•	DK_{C2}				○
8	DK_{S1}	•	DK_{S2}				○
9	DK_{C1}	•	DK_{C2}			○	○
10	DK_{S1}	•	DK_{S2}			○	○

Table 9: Message protection requirements and keys required for protection realization defined by the scenario definitions

The message exchange of the *Secure WS-ReliableMessaging Scenario* requires honest senders and receivers. Each message received has to pass successfully through the decryption, the signature check, and the timestamp check. If one of these steps fails, the communication must be aborted with a failure. Since this work does not provide robustness tests of the WS-* implementations, the case when sender or recipient of a message acts unexpected or dishonestly is out of scope.

Thus, the *Secure WS-ReliableMessaging Scenario* combines features of the following WS-* standards:

- WS-Security
- WS-Trust
- WS-SecureConversation

- WS-ReliableMessaging
- WS-Addressing

In addition to these standards, the scenario also employs WS-Policy, WS-Security Policy, and WS-ReliableMessaging Policy (Sun GlassFish Platform) resp. the according policy types (IBM WebSphere Platform) for specifying the policies of the involved services. Unfortunately, none of the scenario definitions gives a policy configuration. The policy configuration of the *Secure WS-ReliableMessaging Scenario* has to be derived from the message sequence and the security requirements of the scenario descriptions. The policy configuration of this scenario is presented in section 6.3.1, section 6.3.2 analyzes the results of the test runs with regard to scenario compliance.

6.3.1 Policy Configuration

Although the *Secure WS-ReliableMessaging Scenario* is a well-known and practice-relevant application scenario, no policy configuration meeting the requirements of this use case is published. It is for this reason that the policy configuration must be extracted from the specified requirements in [BMPV06], the concrete SOAP messages defined by [GHB05], and the message sequence specified by [BMPV06], [GHB05], and [WS-08, pages 28-36].

The *Secure WS-ReliableMessaging Scenario* is a combined application scenario of the WS-* frameworks WS-ReliableMessaging, WS-Security, WS-Trust, and WS-SecureConversation. First, a secure session using WS-SecureConversation is established, and then a WS-ReliableMes saging sequence is started. The secure session is established in bootstrap messages, which are protected using WS-Security assertions (see Listing 12). The protection of the payload message flow is specified as WS-Security Protection Assertions (see Listing 13 and 14).

Listing 11 shows the policy for a Web service realizing the *Secure WS-ReliableMessaging Scenario*. This policy must define the use of WS-ReliableMessaging (lines 5-7) and WS-Addressing (lines 9-11) as well as the binding for the secure session (lines 13-43). The WS-ReliableMessaging assertion activates the use of WS-ReliableMessaging within the secure session. Neither [BMPV06] nor [GHB05] allow to draw a conclusion about the delivery assurance to be used in the scenario or whether the WS-ReliableMessaging sequence should be bound to a security token using the `wsrmp:SequenceSTR` assertion. In contrast to the imprecise specification for the use of WS-ReliableMessaging, both publications require the use of WS-Addressing. The WS-Addressing assertion enables the use of message addressing properties, e.g., `wsa:To`, `wsa:Action`, `wsa:MessageID`, or `wsa:RelatesTo`. These properties are obligatory for the whole scenario (messages one to ten).

The setup of the secure session is denoted in the `sp:SymmetricBinding` assertion, since the session key is symmetric. A `sp:SecureConversationToken` is established as protection token. This token is the base for signature and encryption key derivation (`sp:RequireDerivedKeys` assertion). The `sp:BootstrapPolicy` is the policy used to obtain the `sp:SecureConversation-Token` from the token issuer (see Listing 12 for a specification of this policy). Within the secure session the `sp:Basic128` algorithm is used for encryption, since [BMPV06] and [GHB05] propose 128-bit cryptography. Considering the examples of [GHB05] message structure, the `sp:Layout` of the SOAP messages is a `sp:Strict` layout (see [OAS09d, pages 52/53]) and a `wsu:Timestamp` must be included. Both aspects are not specified in [BMPV06]. A significant difference between the two scenario specifications is the protection order. Whereas [GHB05] states that “signature occurs before encryption” [GHB05, page 35] (first sign the body and then encrypt body and signature), [BMPV06] proposes to encrypt the message body first, then to sign the corresponding message parts including the `env:Body`, and finally to encrypt the signature. Since [BMPV06] gives a formal cryptographic analysis of this scenario, the `sp:EncryptBeforeSigning` assertion has been chosen. The `sp:EncryptSignature` assertion requires to encrypt the `wss:Signature`.

```xml
<wsp:Policy wsu:Id="SecureRMSessionBinding">
   <wsp:ExactlyOne>
      <wsp:All>

         <wsrmp:RMAssertion>
            <wsp:Policy />
         </wsrmp:RMAssertion>

         <wsam:Addressing>
            <wsp:Policy />
         </wsam:Addressing>

         <sp:SymmetricBinding>
            <wsp:Policy>
               <sp:ProtectionToken>
                  <wsp:Policy>
                     <sp:SecureConversationToken>
                        <wsp:Policy>
                           <sp:RequireDerivedKeys />
                           <sp:BootstrapPolicy>
                              <!--
                              See the XML listing containing the BootstrapPolicy
                              -->
                           </sp:BootstrapPolicy>
                        </wsp:Policy>
                     </sp:SecureConversationToken>
                  </wsp:Policy>
               </sp:ProtectionToken>
               <sp:AlgorithmSuite>
                  <wsp:Policy>
                     <sp:Basic128 />
                  </wsp:Policy>
               </sp:AlgorithmSuite>
               <sp:Layout>
                  <wsp:Policy>
                     <sp:Strict />
                  </wsp:Policy>
               </sp:Layout>
               <sp:IncludeTimestamp />
               <sp:EncryptBeforeSigning />
               <sp:EncryptSignature />
            </wsp:Policy>
         </sp:SymmetricBinding>

      </wsp:All>
   </wsp:ExactlyOne>
```

```
</wsp:Policy>
```

Listing 11: The WS-Policy for the *Secure WS-ReliableMessaging Scenario*

As mentioned earlier, the issuance of a `sp:SecureConversationToken` requires a `sp:BootstrapPolicy`. The `sp:BootstrapPolicy` defines the policy for the `sp:SecureConversationToken` request and the `sp:SecureConversationToken` issuance. [BMPV06] and [GHB05] stipulate the use of X509 certificates in the `sp:BootstrapPolicy`, which are certified by a CA. This approach is realistic in B2Bi, however, for the sake of simplicity, client and service are in possession of a trusted X509 certificate of each other. This change of the scenario does not have any effect on the policy configuration or the SOAP message traffic between client and service, and is therefore not relevant in this work.

The `sp:BootstrapPolicy` in Listing 12 uses an `sp:AsymmetricBinding` (public-key cryptography) with a `sp:X509Token` for the initiator and the recipient. The initiator should send his public key as `wss:BinarySecurityToken` to the recipient, whereas the key of the recipient must not necessarily be transmitted to the client. The `sp:AlgorithmSuite`, the `sp:Layout`, the `wsu:Timestamp` inclusion, the protection order, and the `sp:EncryptSignature` instruction are used as with the `sp:SymmetricBinding` of the main policy.

```
<wsp:Policy>
   <sp:AsymmetricBinding>
      <sp:Policy>
         <sp:InitiatorToken>
            <wsp:Policy>
               <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
                   securitypolicy/200702/IncludeToken/AlwaysToRecipient">
                  <wsp:Policy>
                     <sp:WssX509V3Token10 />
                  </wsp:Policy>
               </sp:X509Token>
            </wsp:Policy>
         </sp:InitiatorToken>
         <sp:RecipientToken>
            <wsp:Policy>
               <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
                   securitypolicy/200702/IncludeToken/Never">
                  <wsp:Policy>
                     <sp:WssX509V3Token10 />
                  </wsp:Policy>
               </sp:X509Token>
             </wsp:Policy>
          </sp:RecipientToken>
          <sp:AlgorithmSuite>
             <wsp:Policy>
                <sp:Basic128 />
             </wsp:Policy>
          </sp:AlgorithmSuite>
          <sp:Layout>
             <wsp:Policy>
                <sp:Strict />
             </wsp:Policy>
          </sp:Layout>
          <sp:IncludeTimestamp />
          <sp:EncryptBeforeSigning />
          <sp:EncryptSignature />
      </wsp:Policy>
```

```
36     </sp:AsymmetricBinding>
37
38     <sp:SignedParts>
39        <sp:Header Namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing" />
40        <sp:Header Namespace="http://www.w3.org/2005/08/addressing" />
41        <sp:Body />
42     </sp:SignedParts>
43
44     <sp:EncryptedParts>
45        <sp:Body />
46     </sp:EncryptedParts>
47  </wsp:Policy>
```

Listing 12: The `sp:BootstrapPolicy` of the `sp:SecureConversationToken` in the *Secure WS-ReliableMessaging Scenario*

In addition to the security binding assertions, the `sp:BootstrapPolicy` specifies the message elements to be protected. The scenario definition stipulates that the WS-Addressing headers, the `wsu:Timestamp`, and the message `env:Body` are to be signed. The assertion in lines 39 and 40 of Listing 12 indicate that all WS-Addressing headers must be signed (for reasons of compatibility the XMLNS of two WS-Addressing versions are specified), the assertion in line 41 denotes that the `env:Body` must be signed. The `wsu:Timestamp` of a SOAP message must always be covered by a signature due to the `sp:IncludeTimestamp` definition in [OAS09d, page 51]. The encryption of the `env:Body` is regulated by lines 44-46, the encryption of the signature is mandatory due to the `sp:EncryptSignature` element in the binding.

The protection assertions of the messages from the client to the service (see Listing 13) and vice versa (see Listing 14) are similar to the assertions defined in the `sp:BootstrapPolicy`: the WS-Addressing headers and the `env:Body` are intended for integrity protection, and the `env:Body` is also encrypted. In addition, the WS-ReliableMessaging header sections are signed.

```
<wsp:Policy wsu:Id="SecureRMSessionInput" >
   <sp:SignedParts>
      <sp:Header Namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"/>
      <sp:Header Namespace="http://www.w3.org/2005/08/addressing"/>
      <sp:Header Namespace="http://schemas.xmlsoap.org/ws/2005/02/rm"/>
      <sp:Header Namespace="http://docs.oasis-open.org/ws-rx/wsrm/200702"/>
      <sp:Body />
   </sp:SignedParts>
   <sp:EncryptedParts>
      <sp:Body />
   </sp:EncryptedParts>
</wsp:Policy>
```

Listing 13: The WS-SecurityPolicy protection assertions for the input messages

```
<wsp:Policy wsu:Id="SecureRMSessionOutput" >
   <sp:SignedParts>
      <sp:Header Namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing"/>
      <sp:Header Namespace="http://www.w3.org/2005/08/addressing"/>
      <sp:Header Namespace="http://schemas.xmlsoap.org/ws/2005/02/rm"/>
      <sp:Header Namespace="http://docs.oasis-open.org/ws-rx/wsrm/200702"/>
      <sp:Body />
   </sp:SignedParts>
   <sp:EncryptedParts>
      <sp:Body />
   </sp:EncryptedParts>
</wsp:Policy>
```

Listing 14: The WS-SecurityPolicy protection assertions for the output messages

The policy defined in the Listings 11 to 14 is a standard compliant WS-Policy definition of the *Secure WS-ReliableMessaging Scenario*. However, as the test results in section 6.1 show, the IBM WebSphere Platform is not always compliant to the standards. Moreover, the policy set concept defines other requirements for the protection assertions. Considering the results of section 6.1.4, the IBM WebSphere Platform is not able to process the assertions `sp:EncryptSignature` and `sp:EncryptBeforeSigning` correctly. In addition, the IBM WebSphere Platform does not automatically sign the `wsu:Timestamp`, which is a requirement defined in [OAS09d, page 51]. IBM provides a workaround for those problems concerning the signing of the `wsu:Timestamp` and the encryption of the `wss:Signature` using XPath expressions to define the message elements to be signed and/or encrypted. The XPath expression to access the corresponding elements is specified in the `sp:EncryptedElements` (lines 10-17) and `sp:SignedElements` (lines 1-8) assertions. In order to guarantee interoperability with different SOAP versions, the assertions are inserted twice with different namespaces. The use of XPath is mandatory in these assertions, since none of the security header elements, such as the `wsu:Timestamp` or the `wss:Signature`, are accessible in the `sp:EncryptedParts` and `sp:SignedParts` assertions. Interestingly, the IBM WebSphere Platform does not provide any possibility to specify the protection order of the SOAP message (the `sp:EncryptBeforeSigning` assertion must be omitted). Listing 15 shows the protection assertions necessary to meet the requirements of an integrity protected `wsu:Timestamp` and an encrypted `wss:Signature` (the `sp:EncryptSignature` assertion is no longer necessary in the policy). The XML code of Listing 15 must be added to the protection assertions in the Listings 12, 13, and 14.

```
 1 <sp:SignedElements>
 2    <sp:XPath>
 3       /*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='
             Envelope']/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and
             local-name()='Header']/*[namespace-uri()='http://docs.oasis-open.org/wss/2004/01/
             oasis-200401-wss-wssecurity-secext-1.0.xsd' and local-name()='Security']/*[
             namespace-uri()='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
             wssecurity-utility-1.0.xsd' and local-name()='Timestamp']
 4    </sp:XPath>
 5    <sp:XPath>
 6       /*[namespace-uri()='http://www.w3.org/2003/05/soap-envelope' and local-name()='
             Envelope']/*[namespace-uri()='http://www.w3.org/2003/05/soap-envelope' and local-
             name()='Header']/*[namespace-uri()='http://docs.oasis-open.org/wss/2004/01/oasis
             -200401-wss-wssecurity-secext-1.0.xsd' and local-name()='Security']/*[namespace-
             uri()='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility
             -1.0.xsd' and local-name()='Timestamp']
 7    </sp:XPath>
 8 </sp:SignedElements>
 9
10 <sp:EncryptedElements>
11    <sp:XPath>
12       /*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and local-name()='
             Envelope']/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/' and
             local-name()='Header']/*[namespace-uri()='http://docs.oasis-open.org/wss/2004/01/
             oasis-200401-wss-wssecurity-secext-1.0.xsd' and local-name()='Security']/*[
             namespace-uri()='http://www.w3.org/2000/09/xmldsig#' and local-name()='Signature'
             ]
```

```
    </sp:XPath>
    <sp:XPath>
       /*[namespace-uri()='http://www.w3.org/2003/05/soap-envelope' and local-name()='
           Envelope']/*[namespace-uri()='http://www.w3.org/2003/05/soap-envelope' and local-
           name()='Header']/*[namespace-uri()='http://docs.oasis-open.org/wss/2004/01/oasis
           -200401-wss-wssecurity-secext-1.0.xsd' and local-name()='Security']/*[namespace-
           uri()='http://www.w3.org/2000/09/xmldsig#' and local-name()='Signature']
    </sp:XPath>
</sp:EncryptedElements>
```

Listing 15: The additional WS-SecurityPolicy protection assertions of the IBM WebSphere Platform

However, the results presented in section 6.1.2 show that the Sun GlassFish Platform does not accept the XPath expression of the `sp:EncryptedElements` and `sp:SignedElements`. On that score, these assertions must not be present in the policy deployed on the Sun GlassFish Platform. Yet, the Sun GlassFish Platform requires an additional change to the policy configuration, i.e., the `sp:RequireKeyIdentifierReference` assertion has to be added in the `sp:X509Token` assertion of the `sp:AsymmetricBinding` of the bootstrap policy. Without this amendment the Sun GlassFish client is unable to resolve the reference to the security token.

6.3.2 Test Execution & Test Result

Considering the platform-specific adaptations needed for WebSphere and GlassFish policies so far, it is clear that it is not possible to create a single *Secure WS-ReliableMessaging Scenario*-policy for both platforms. Moreover, the test results of section 6.1 show that an implementation of the *Secure WS-ReliableMessaging Scenario* across these platforms is not possible because the basic encryption/signature/reliability primitives are not compatible. Therefore, this section analyzes the SOAP message traffic with regard to scenario and standard compliance for each platform individually.

The message sequence and the requirements concerning encryption and signature of message parts, intended by the scenario definitions, have already been discussed in the previous section. Now, these policies are deployed on the corresponding platforms. The subsequent procedure is in accordance with the test method of this work (cf. section 4), i.e., invocation of the service by a QoS-aware client, logging of the SOAP message traffic, and analysis of the captured message trace.

The analysis of the test run on the Sun GlassFish Platform shows a close match to the scenario definitions. Yet, the results obtained slightly deviate from the expected results, since the `wss:Signature` is not encrypted when the `env:Body` is not encrypted. This concerns the message types 3, 4, 7, 8, and 9. Message type 6 contains an encrypted and signed `env:Body`, although the scenario definitions intend an empty and unprotected `env:Body`. Although this is a deviation from the scenario definition, the behavior of the Metro WS-Stack is standard compliant since the "primary signature element is NOT REQUIRED to be encrypted [...] when there is nothing in the message that is covered by this signature that is encrypted." [OAS09d, lines 1730/1731]. Surprisingly, the `env:Body` of message type 10 is encrypted, and therefore the `wss:Signature` is also encrypted.

Interestingly, the Sun GlassFish Platform first sends a `wsrm:CloseSequence` message and then a `wsrm:TerminateSequence` message to terminate the sequence. The opposite party answers both messages with a corresponding response message. Although the Sun GlassFish Platform message structure differs from the scenario description, this is a minor deviation since both message types are protected in exactly the same way and both have the same purpose. Therefore, the four messages are treated as a two-part message type 7 and 8 in the scenario analysis below.

Table 10 summarizes the results of the scenario analysis in the Sun GlassFish environment. The Table shows the protected message parts and the keys used for protection. The notational conventions correspond to the ones defined for Table 9.

Msg	**Sign**	**Body**		**TS**	**WS-A**	**WS-R**	**EncKey**
No	**enc**	**enc**	**sig**	**sig**	**sig**	**sig**	**enc**
1	SK_1		PrK_C			○	PuK_S
2	SK_2		PrK_S			○	PuK_C
3	●	●	DK_{C2}			○	○
4	●	●	DK_{S2}			○	○
5	DK_{C1}		DK_{C2}				○
6	DK_{S1}		DK_{S2}				○
7	●	●	DK_{C2}				○
8	●	●	DK_{S2}				○
9	●	●	DK_{C2}			○	○
10	DK_{S1}		DK_{S2}			○	○

Table 10: Message protection and keys used for protection realization on the Sun GlassFish Platform

Compared to the Sun GlassFish Platform, the results obtained from the test runs on IBM WebSphere Platform deviate significantly from the scenario definitions. Considering the message sequence of the WebSphere test run, the fact that neither the WS-ReliableMessaging sequence nor the WS-SecureConversation session is terminated or canceled is striking. On the IBM WebSphere Platform, sequences or sessions have to be closed explicitly by addressing the WS-ReliableMessaging sequence in the source code of a client application. As aforesaid, the scope of this work does not include specific source code adaptations, but only use QoS features that are transparent to the client application.

However, not only the message sequence is not scenario compliant, but also the structure of each particular message: the `wsu:Timestamp` has no expiration date, the message `env:Body` is always encrypted, even when the scenario does not intend a confidentiality protection, and the IBM WebSphere Platforms only uses a `wss:SecurityTokenReference` instead of an embedded `wsc:SecurityContextToken`. Similar to the results of the test run on the Sun GlassFish Platform, message type 6 has an integrity and confidentiality protected `env:Body` which is not scenario compliant. Table 11 enlists the protected parts and gives an overview of the keys used to realize the protection. The notation corresponds to the Tables 9 and 10.

Msg	**Sign**	**Body**		**TS**	**WS-A**	**WS-R**	**EncKey**
No	**enc**	**enc**	**sig**	**sig**	**sig**	**sig**	**enc**
1	SK_1		PrK_C			○	PuK_S
2	SK_2		PrK_S			○	PuK_C
3	DK_{C1}		DK_{C2}			○	○
4	DK_{S1}		DK_{S2}			○	○
5	DK_{C1}		DK_{C2}				○
6	DK_{S1}		DK_{S2}				○

Table 11: Message protection and keys used for protection realization on the IBM WebSphere Platform

7 Related Work

“Web service interoperability” and “Web services testing” are current topics in both, science and industry.

The industry established the WS-I[44] which is “is an open industry organization chartered to establish Best Practices for Web services interoperability, for selected groups of Web services standards, across platforms, operating systems and programming languages.”[45] The deliverables of the WS-I are the so-called *profiles*, which provide guidelines on how Web services specifications should be combined for best interoperability, e.g., the “Basic Security Profile” or the “Reliable Secure Profile”. However, the WS-I only provides “Best Practices for Web services interoperability” and does not attend to full compatibility of the WS-* framework implementations. In contrast to the approach of the WS-I, this work targets to a full compatibility analysis of the WS-* frameworks, since *Best Practices* are not always sufficient to meet real B2Bi needs.

Nevertheless, “Web services interoperability” is also a topic in scientific publications. [SV09] discusses the interoperability issues seen in Web services, focusing on Web services implemented in different environments (Sun JEE and Microsoft .NET). Even for QoS-unaware Web services communication, [SV09] detects incompatibility issues and therefore shows that compatibility testing in up-to-date SOAs is still a compulsory task. Whereas [SV09] focuses on the static properties of Web services, even the dynamic behavior of Web services may be a reason for incompatibility. On that account, e.g., [BSBM04] proposes a formal, state transition-based, model to determine the compatibility of Web services, whereas [Mar03] proposes a petri net-driven approach for the same purpose. The publications of [SV09], [Mar03], and [BSBM04] show that there are two dimensions of compatibility to be regarded in Web services communication: the structure, on the one hand, and the dynamic behavior, on the other hand. [PF04] summarizes both issues in his examination of Web services substitutability and establishes four reasons for incompatibility: structure, value, encoding, and semantics. These publications show why Web services might be incompatible and what types of incompatibility may occur. Although the related work presented in this paragraph focuses on the functional properties of Web services, the non-functional properties are subject to the same problems. However, the WS-* frameworks

[44]http://www.ws-i.org
[45]Taken from the “WS-I Overview Data Sheet”

in current scientific publications on this subject are not under consideration. Therefore, this work focuses on the determination of the interoperability of the non-functional Web services properties.

"Interoperability *testing*" is not extensively researched in the Web services domain. However, there are publications in the field of communication protocols dealing with interoperability issues. Since most WS-* frameworks specify sophisticated communication protocols to guarantee QoS features, the works in the field of protocols are related to this work. For example, [LLY04] discusses the interoperability of protocols using and minimizing formal graphs. Especially the generation of possible test cases for interoperability testing is a current topic in communication protocols, e.g., [DV05], [DV07], and [DV09]. [Kat08] transforms these approaches to generate test cases for interacting services automatically. These publications show how test cases for interoperability tests are designed and generated. The general method of these papers has been adapted for this work to prove the interoperability of the WS-* frameworks.

Although [Kat08] defines a Web services testing approach, QoS aspects for the services are missing in this consideration. Interestingly, the term *quality of service* with reference to Web services has an ambiguous meaning. There are some QoS definitions that are restricted to quantitatively measurable properties, such as the latency of a service [TGN+03] in contrast to the QoS definition of this work, which includes, i.a., security and reliability (according to the QoS definitions of, e.g., [OMB07], [Men02], etc.).

Very few publications consider SOA-testing with regard to QoS-features. [CP06] and [Ger08] discuss challenges of SOA testing, but the QoS aspects are limited to quantifiable factors, such as latency. However, both publications regard building trust between service and client as a "challenge", but do not refer to WS-Trust or WS-Security. [BAP07] proposes a QoS Test-Bed Generator for Web Services, but only defines latency and reliability as QoS features. Although [SK07] provides an evaluation of the WS-Security implementation of the Apache Axis2 SOAP/WSDL/Web services engine, the paper only considers the processing time and message size when using different WS-Security features. In contrast to these publications, this work discusses QoS with respect to Web services according to the definition of [OMB07] and tests the compatibility of these features using different implementations.

8 Conclusion & Future Work

Considering related work, there is no approach that analyzes the QoS frameworks of Web services, i.e., WS-Security, WS-ReliableMessaging, etc., with regard to interoperability issues. Therefore, this work establishes a test method for WS-* interoperability testing and applies this method to two JEE servers using different WS-* implementations.

Based on the SOA definition by OASIS, the distributed services of a SOA "may be under the control of different ownership domains" [OAS06b, page 8, lines 128/129]. The "different ownership domains" frequently include technical aspects, such as the Web services platforms and thus also the WS-* implementations. This work and related work (section 7) imply that platform heterogeneity is one of the central problems of B2Bi in Web services based SOAs. Therefore, Heather Kreger's statement that "Web services have no value if they're not interoperable, and interoperability is based on standards compliance" [Lea04] retains its validity also for the WS-* frameworks. Despite having generally accepted specifications, the implementations may deviate from these specifications or omit some features. The introduction has raised the research question whether it is possible to establish a secure and reliable connection in a Web services based SOA solution using WS-Security and WS-ReliableMessaging based on different Web services stack (WS-Stack) implementations.

To answer this question, it is necessary to examine the feature coverage and the feature compatibility of the implementations. Therefore, this work proposes a test method that employs the specifications of the WS-* features as the starting point to develop a test plan. First, the very basic features of the frameworks are analyzed to create a structured test plan containing all features and settings of a feature of a standard. With regard to interdependencies, each setting of a feature is mapped to a test case which tests the availability of a setting and a feature in an implementation (using a platform-internal test run) and its compatibility across platforms. Second, the available and compatible features are combined to more extensive test cases regarding sensible restrictions to the test cases to prevent the *test case explosion problem.* The test case combination is an iterative process, that ends with highly complex scenarios. In the context of WS-Security and WS-ReliableMessaging use, the *Secure WS-ReliableMessaging Scenario* is a sophisticated practical use case and is therefore employed as final test case.

The test results of the isolated function tests show that a QoS-unaware simple SOAP use case is realizable in every case. However, the QoS-aware test cases detect several problems, starting with the unavailability of whole standards on a platform (e.g., WS-RM Policy on the WAS or XPath on the GlassFish server) up to bugs and other problems in the implementation. The analysis of the isolated function test results reveals four main reasons for failed tests:

- A feature or a setting of a feature is not implemented, e.g., unimplemented security token types on both platforms
- Enabling technologies for a feature are not available, e.g.:
  - WS-RM Policy on the IBM WebSphere Platform
  - XPath on the Sun GlassFish Platform

- The implementation deviates from the standard definition of a feature, e.g.:
  - Realization of the SSL-transport of SOAP messages or the `sp:KerberosToken` implementation on the IBM WebSphere Platform
  - Signature of the `env:Body` element does not cover the whole body on the Sun GlassFish Platform
- Problems with the processing of features or their SOAP representation, e.g., `sp:UserNameToken` callback handler does not work on the IBM WebSphere Platform

The variety of failures during the compatibility analysis makes most combined test cases redundant, since hardly any reasonable combination of features is realizable. Although the test results of the isolated test cases support the conclusion that the *Secure WS-ReliableMessaging Scenario* is not executable as a cross-platform test case, the scenario is still performed due to its practical relevance for the platform-internal test runs. The results of these test runs show that neither the IBM WebSphere Platform nor the Sun GlassFish Platform is able to implement the scenario exactly as intended by the definitions. Whereas the Sun GlassFish Platform reveals minor deviations from the scenario, e.g., regarding the encryption of the `env:Body` and `wss:Signature` elements, the IBM WebSphere Platform *overprotects* the messages by encrypting any SOAP `env:Body`. Regarding the message protection, the IBM WebSphere Platform overfulfills the scenario requirements, but other scenario requirements are not met: the cancelation of the secure session and the termination of the WS-ReliableMessaging sequence are missing. However, no matter how far the test runs on both servers are scenario compliant, both do not meet the scenario definition exactly and are therefore not 100 % scenario compliant. Thus, none of both scenarios implements the formally verified scenario of [BMPV06].

Summarizing the results of this work, the answer of the research question is as follows: It is possible to leverage selected security or reliability features in a Web services based SOA solution using WS-Security and WS-ReliableMessaging based on different Web services stack (WS-Stack) implementations. However, it must be stated that this positive result is only valid for a few combinations of features and for certain service/client relationships of the platforms. For example, a WebSphere client is not able to consume a GlassFish WS-ReliableMessaging policy correctly. Hence, interoperability of arbitrary WS-Security or WS-ReliableMessaging features or combinations thereof cannot be taken for granted.

The results show, that the intended WS-* features must be pinpointed and tested for practical use in heterogeneous SOA environments (at least for the WebSphere and the Metro implementation). To this end, the maturity level of the WS-* environments is too low - further development and adjustment of the WS-Stack implementations may take several years. Another option is to leverage WS-* features by using homogeneous WS-Stack implementations, e.g., Metro may be plugged into different application servers. For example, there have even been efforts to introduce Metro to the IBM WebSphere Platform[46].

[46]Nikhil Thaker, *Enabling a third-party JAX-WS application in WebSphere Application Server V6.1*, IBM Developer Works, available at `http://public.dhe.ibm.com/software/dw/wes/1001_thaker2/1001_thaker2.pdf`

Certainly, the compatibility analysis should be applied to more WS-Stacks, e.g., the Apache CXF[47] project, in order to get more data for a general examination of the compatibility of WS-* implementations. The further development of the current WS-Stacks might resolve incompatibilities, so that each compatibility analysis should be repeated after the release of a new version. Since a compatibility analysis is an expensive project, automation of this process is desirable. In addition to the standards examined in this work, a future approach should include additional WS-* standards, such as WS-Coordination. Another interesting point in the area of compatibility/interoperability testing is to examine the behavior of the WS-Stack implementations in exception cases, e.g., how an implementation reacts when a message in a WS-ReliableMessaging sequence with the delivery assurance `ExactlyOnce` disappears? All these issues may serve as starting points for future work.

As great interest in academia and industry shows, the WS-* frameworks are seen as an enabling technology for a broader use of Web services in B2Bi. However, interoperability is still one of the core requirements for B2Bi use and the results of this work show that this requirement cannot be met by all WS-* implementations. Therefore, the "second-generation Web services specifications"[48] are not able to provide comprehensive QoS support, due to the shortcomings of the tested implementations.

[47]available at `http://cxf.apache.org`

[48]Thomas Erl, *Service-Oriented Architecture - Concepts, Technology, and Design*, page 156, Prentice Hall, Upper Saddle River, New Jersey, USA, 2005.

References

[BAP07] Antonia Bertolino, Guglielmo De Angelis, and Andrea Polini. A QoS Test-Bed Generator for Web Services. In Luciano Baresi, Piero Fraternali, and Geert-Jan Houben, editors, *Proceedings of the 7th International Conference on Web Engineering (ICWE 2007), Como, Italy, July 2007*, volume 4607 of *Lecture Notes in Computer Science*, pages 17–31. Springer Verlag, Berlin/Heidelberg, Germany, July 2007.

[BCT06] Boualem Benatallah, Fabio Casati, and Farouk Toumani. Representing, analysing and managing Web service protocols. *Data & Knowledge Engineering*, 58(3):Elsevier, Amsterdam, Netherlands, pp. 327–357, September 2006.

[BEA05] BEA Systems, IBM Corporation, Microsoft Corporation Inc., TIBCO Software Inc. *Web Services Reliable Messaging Protocol (WS-ReliableMessaging)*, February 2005. first specification (Version 1.0).

[BMPV06] Michael Backes, Sebastian Mödersheim, Birgit Pfitzmann, and Luca Viganò. Symbolic and Cryptographic Analysis of the Secure WS-ReliableMessaging Scenario. In Luca Aceto and Anna Ingólfsdóttir, editors, *Proceedings of the 9th International Conference on Foundations of Software Science and Computation Structure*, volume 3921 of *Lecture Notes in Computer Science*, pages 428–445. Springer Verlag, Berlin/Heidelberg, Germany, March 2006.

[Bro03] Gerald Brose. Securing Web Services with SOAP Security Proxies. In *Proceedings of the International Conference on Web Services 2003 (ICWS 03), Las Vegas, Nevada, USA*, pages 231–234, June 2003.

[BSBM04] Lucas Bordeaux, Gwen Salaün, Daniela Berardi, and Massimo Mecella. When are Two Web Services Compatible? In Ming-Chien Shan and Umeshwar Dayal, editors, *Proceedings of the 5th International Workshop on Technologies in E-Services (TES 2004), Toronto, Canada, August 2004*, volume 3324 of *Lecture Notes in Computer Science*, pages 15–28. Very Large Data Base Endowment Inc. (VLDB Endowment), Springer Verlag, Berlin/Heidelberg, Germany, September 2004.

[CLM$^+$09] Henry Cui, Raymond Josef Edward A. Lara, Rosaline Makar, Nicky Moelholm, and Felipe Pittella Rodrigues. *IBM WebSphere Application Server V7.0 Web Services Guide*. IBM Corporation, first edition, August 2009. IBM Redbook, available at http://www.redbooks.ibm.com/redbooks/pdfs/sg247758.pdf.

[CP06] Gerardo Canfora and Massimiliano Di Penta. Testing Services and Service-Centric Systems: Challenges and Opportunities. *IEEE IT Pro*, (2):10–17, March/April 2006.

[dLPW07] Pierre de Leusse, Panos Periorellis, and Paul Watson. Enterprise Service Bus: An overview. Technical reports, s.o.c. science, University of Newcastle upon Tyne, Newcastle, England, July 2007. available at http://www.cs.newcastle.ac.uk/publications/trs/papers/1037.pdf.

[DV05] Alexandra Desmoulin and César Viho. Formalizing Interoperability Testing: Quiescence Management and Test Generation. In Farn Wang, editor, *Proceedings of the 25th International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2005), Taipei, Taiwan, October 2005*, volume 3731 of *Lecture Notes in Computer Science*, pages 533–537. Springer Verlag, Berlin/Heidelberg, Germany, October 2005.

[DV07] Alexandra Desmoulin and César Viho. A New Method for Interoperability Test Generation . In Alexandre Petrenko, Margus Veanes, Jan Tretmans, and Wolfgang Grieskamp, editors, *Proceedings of the 19th International Conference on Testing of Software and Communicating Systems (TestCom 2007), Tallinn, Estonia, June 2007*, volume 4581 of *Lecture Notes in Computer Science*, pages 58–73. Springer Verlag, Berlin/Heidelberg, Germany, June 2007.

[DV09] Alexandra Desmoulin and César Viho. Formalizing interoperability for test case generation purpose. *International Journal on Software Tools for Technology Transfer (STTT)*, 11(3):Springer Verlag, Berlin/Heidelberg, Germany, pp. 261–267, July 2009.

[FF03] Christopher Ferris and Joel Farrell. What are Web Services. *Communications of the ACM*, 46(6):p. 31, June 2003.

[Ger08] Gerardo Canfora and Massimiliano Di Penta. Service-Oriented Architecture Testing: A Survey. In Andrea De Lucia and Filomena Ferrucci, editors, *Revised Tutorial Lectures of the International Summer Schools for Software Engineering (ISSSE 2006 - 2008), Salerno, Italy*, volume 5413 of *Lecture Notes in Computer Science*, pages 78–105. Springer Verlag, Berlin/Heidelberg, 2006 - 2008.

[GHB05] Kirill Gavrylyuk, Ondrej Hrebicek, and Stefan Batres. WCF (Indigo) Interoperability Lab: Reliable Messaging. Word Document (.doc), 2005. available at http://131.107.72.15/ilab/RM/WCFInteropPlugFest_RM.doc.

[IET93] IETF. *The Kerberos Network Authentication Service (V5)*, September 1993.

[IJC08] Ashok Iyengar, Vinod Jessani, and Michele Chilanti. *WebSphere Business Integration Primer: Process Server, BPEL, SCA, and SOA*. IBM Press, Upper Saddle River, New Jersey, USA, first edition, 2008.

[ITU08] ITU-T, IETF. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, May 2008.

[Kat08] Kathrin Kaschner and Niels Lohmann. Automatic Test Case Generation for Interacting Services . In George Feuerlicht and Winfried Lamersdorf, editors, *Revised Selected Papers of the International Workshops on Service-Oriented Computing (ICSOC 2008), Sydney, Australia, December 2008*, volume 5472 of *Lecture Notes in Computer Science*, pages 66–78. Springer Verlag, Berlin/Heidelberg, Germany, December 2008.

[Kre03] Heather Kreger. Fulfilling the Web Services Promise. *Communications of the ACM*, 46(6):pp. 29–34, June 2003.

[Lea04] Neal Leavitt. Are Web Services Finally Ready to Deliver? *IEEE Computer*, 37(11):pp. 14–18, November 2004.

[LLY04] David Lee, Christine Liu, and Mihalis Yannakakis. Protocol System Integration, Interface and Interoperability. In Teruo Higashino, editor, *Revised Selected Papers of the 8th International Conference on Principles of Distributed Systems (OPODIS 2004), Grenoble, France, December 2004*, volume 3544 of *Lecture Notes in Computer Science*, pages 1–19. Springer Verlag, Berlin/Heidelberg, Germany, December 2004.

[Lud03] Heiko Ludwig. Web Services QoS: External SLAs and Internal Policies - Or: How do we deliver what we promise? In *Proceedings of the Fourth International Conference on Web Information Systems Engineering Workshops (WISEW'03), Rome, Italy*, pages 115–120. IEEE Computer Society, December 2003.

[Mar03] Axel Martens. On Compatibility of Web Services. *Petri Net Newsletter*, 65:Special Interest Group on Petri Nets and Related System Models ot the Gesellschaft für Informatik (GI) e.V., Bonn, Germany, pp. 12–20, October 2003.

[Men02] Daniel A. Menascé. QoS Issues in Web Services. *IEEE Internet Computing*, 6(6):pp. 72–75, November 2002.

[Men07] Falko Menge. Enterprise Service Bus. In *Free and Open Source Software Conference (FrOSCon 2007), Sankt Augustin, Germany*, August 2007.

[OAS02] OASIS. *UDDI Version 2.04 API Specification*, July 2002.

[OAS05] OASIS. *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*, March 2005.

[OAS06a] OASIS. *ebXML Business Process Specification Schema Technical Specification v2.0.4*, December 2006.

[OAS06b] OASIS. *Reference Model for Service Oriented Architecture 1.0*, October 2006.

[OAS06c] OASIS. *Web Services Security Kerberos Token Profile 1.1*, February 2006.

[OAS06d] OASIS. *Web Services Security Rights Expression Language (REL) Token Profile 1.1*, February 2006.

[OAS06e] OASIS. *Web Services Security: SAML Token Profile 1.1*, February 2006.

[OAS06f] OASIS. *Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)*, February 2006.

[OAS06g] OASIS. *Web Services Security UsernameToken Profile 1.1*, February 2006.

[OAS06h] OASIS. *Web Services Security X.509 Certificate Token Profile 1.1*, February 2006.

[OAS07] OASIS. *Web Services Business Process Execution Language Version 2.0*, January 2007.

[OAS09a] OASIS. *Web Services Reliable Messaging Policy Assertion (WS-RM Policy) Version 1.2*, February 2009.

[OAS09b] OASIS. *Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.2*, February 2009.

[OAS09c] OASIS. *WS-SecureConversation 1.4*, February 2009.

[OAS09d] OASIS. *WS-SecurityPolicy 1.3*, February 2009.

[OAS09e] OASIS. *WS-Trust 1.4*, February 2009.

[OMB07] Liam O'Brien, Paulo Merson, and Len Bass. Quality Attributes for Service-Oriented Architectures. In *Proceedings of the International Workshop on Systems Development in SOA Environments (SDSOA), Minneapolis, Minnesota, USA*, page 3. ACM, May 2007.

[OSO06] OSOA. *Service Data Objects For Java Specification Version 2.1.0*, November 2006.

[OSO07] OSOA. *SCA Assembly Model V1.00*, March 2007.

[PF04] Shankar R. Ponnekanti and Armando Fox. Interoperability Among Independently Evolving Web Services. In Hans-Arno Jacobsen, editor, *Proceedings of the International Middleware Conference (Middleware 2004), Toronto, Canada, October 2004*, volume 3231 of *Lecture Notes in Computer Science*, pages 331–351. ACM/IFIP/USENIX, Springer Verlag, Berlin/Heidelberg, Germany, October 2004.

[Ren09] Colin Renouf. *Pro IBM WebSphere Application Server 7 Internals.* Springer-Verlag, New York, New York, USA, first edition, 2009.

[Ros10] RosettaNet. *Message Control and Choreography (MCC) - Profile-Web Services (WS), Release 11.00.00A*, June 2010.

[SK07] M. Shopov and N. Kakanakov. Evaluation of a single WS-Security implementation. In *Proceedings International Conference on Automatics and Informatics, Sofia, Bulgaria*, pages 39–42, October 2007.

[Sun] Sun Microsystems. *Metro Users Guide.* available at https://metro.dev.java.net/guide/.

[Sun05] Sun Microsystems. *Java Business Integration (JBI) 1.0*, August 2005.

[Sun10] Sun Microsystems. *SunGlassFish Enterprise Server v2.1.1 Release Notes*, April 2010. available at http://dlc.sun.com/pdf/821-0188/821-0188.pdf.

[Sur10] Ursula Sury. Digitale Signatur - quo vadis. *Informatik Spektrum*, 33(1):Springer Verlag, Berlin/Heidelberg, Germany, pp. 82–83, February 2010.

[SV09] Sujala Shetty and Sangil Vadivel. Interoperability issues seen in Web Services. *International Journal of Computer Science and Network Security (IJCSNS)*, 9(8):Seoul, Republic of Korea, pp. 160–169, August 2009.

[TGN$^+$03] M. Tian, A. Gramm, T. Naumowicz, H. Ritter, and J. Schiller. A Concept for QoS Integration in Web Services. In *Proceedings of the 4th International Conference on Web Information Systems Engineering Workshops (WISEW'03), Rome, Italy, December 2003*, pages 149–155. IEEE Computer Society, 2003.

[W3C00] W3C. *SOAP Messages with Attachments*, December 2000.

[W3C01] W3C. *Web Services Description Language (WSDL) 1.1*, March 2001.

[W3C02] W3C. *XML Encryption Syntax and Processing*, December 2002.

[W3C04a] W3C. *Web Services Architecture*, Febuary 2004.

[W3C04b] W3C. *XML Schema Part 0: Primer Second Edition*, October 2004.

[W3C04c] W3C. *XML Schema Part 1: Structures Second Edition*, October 2004.

[W3C04d] W3C. *XML Schema Part 2: Datatypes Second Edition*, October 2004.

[W3C06a] W3C. *Web Services Addressing 1.0 - Core*, May 2006.

[W3C06b] W3C. *Web Services Addressing 1.0 - SOAP Binding*, May 2006.

[W3C07a] W3C. *SOAP Version 1.2 Part 0: Primer (Second Edition)*, April 2007.

[W3C07b] W3C. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*, April 2007.

[W3C07c] W3C. *SOAP Version 1.2 Part 2: Adjuncts (Second Edition)*, April 2007.

[W3C07d] W3C. *Web Services Addressing 1.0 - Metadata*, September 2007.

[W3C07e] W3C. *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*, June 2007.

[W3C07f] W3C. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*, June 2007.

[W3C07g] W3C. *Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts*, June 2007.

[W3C07h] W3C. *Web Services Policy 1.5 - Attachment*, September 2007.

[W3C07i] W3C. *Web Services Policy 1.5 - Framework*, September 2007.

[W3C07j] W3C. *XML Path Language (XPath) 2.0*, January 2007.

[W3C08a] W3C. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, fifth edition, November 2008.

[W3C08b] W3C. *XML Signature Syntax and Processing (Second Edition)*, second edition, June 2008.

[W3C09] W3C. *Namespaces in XML 1.0 (Third Edition)*, third edition, December 2009.

[W3C10] W3C. *Web Services Metadata Exchange (WS-MetadataExchange)*, March 2010.

[WS-08] WS-I. *WS-I RSP WG Usage Scenarios 1.0*, December 2008.

[ZHM97] Hong Zhu, Patrick A. V. Hall, and John H. R. May. Software Unit Test Coverage and Adequacy. *ACM Computing Surveys (CSUR)*, 29(4):pp. 366–427, December 1997.

A Complete List of the Features and Settings under Test

#	Op1	Op2	Mul	Setting	Definition
0	-	-	-	Basic SOAP	-
WS-ReliableMessaging Policy Assertions					
1	-	-	1	Basic `RMAssertion`	[OAS09a, l. 132-134]
2	`XOR`	-	0/1	`SequenceSTR`	[OAS09a, l. 141-144] [OAS09a, l. 263-277]
			0/1	`SequenceTransportSecurity`	[OAS09a, l. 145-149] [OAS09a, l. 278-307]
3	`AND`	`XOR`	1	`DeliveryAssurance/ExactlyOnce`	[OAS09a, l. 166/167]
			1	`DeliveryAssurance/AtLeastOnce`	[OAS09a, l. 168/169]
			1	`DeliveryAssurance/AtMosttOnce`	[OAS09a, l. 170/171]
			0/1	`DeliveryAssurance/InOrder`	[OAS09a, l. 172/173]
WS-SecurityPolicy Protection Assertions					
4	-	-	0*	`SignedParts/Body`	[OAS09d, l. 427-429]
			0*	`SignedParts/Header`	[OAS09d, l. 430-447]
			0*	`SignedParts/Attachments`	[OAS09d, l. 448-460]
5	-	-	0*	`SignedElements` (XPath)	[OAS09d, l. 461-494]
6	-	-	0*	`EncryptedParts/Body`	[OAS09a, l. 523-527]
			0*	`EncryptedParts/Header`	[OAS09a, l. 528-544]
			0*	`EncryptedParts/Attachments`	[OAS09a, l. 545-550]
7	-	-	0*	`EncryptedElements` (XPath)	[OAS09d, l. 551-577]
8	-	-	0*	`ContEncElements` (XPath)	[OAS09d, l. 578-604]
9	-	-	0*	`RequiredParts/Header`	[OAS09d, l. 650-657]
10	-	-	0*	`RequriedElements` (XPath)	[OAS09d, l. 612-635]
WS-SecurityPolicy Token Assertions					
11	`XOR`	-	0/1	`IncludeToken/Never`	[OAS09d, l. 663-694]
			0/1	`IncludeToken/Always`	
			0/1	`IncludeToken/Once`	
			0/1	`IncludeToken/AlwaysToRecipient`	
			0/1	`IncludeToken/AlwaysToInitiator`	
12	`XOR`	-	0/1	`Token Issuer`	[OAS09d, l. 696-700]
			0/1	`Token Issuer Name`	[OAS09d, l. 701-710]
13	-	-	0/1	`Claims`	[OAS09d, l. 711-720]
14	`XOR`	-	0/1	`DerivedKeys`	[OAS09d, l. 742-750]
			0/1	`ExplicitDerivedKeys`	[OAS09d, l. 751-754]
			0/1	`ImpliedDerivedKeys`	[OAS09d, l. 755-758]

Please continue reading on the next page

#	Op1	Op2	Mul	Setting	Definition
WS-SecurityPolicy Token Assertion Types					
15	OR	XOR	0/1	`UNT/NoPassword XOR UNT/HashPassword`	[OAS09d, l. 819-826]
			0/1	`UNT/Created OR UNT/Nonce`	[OAS09d, l. 827-834]
		XOR	0/1	`UNT/WssUsernameToken10`	[OAS09d, l. 846-848]
			0/1	`UNT/WssUsernameToken11`	[OAS09d, l. 849-851]
16	-	-	0/1	`IssuedToken`	[OAS09d, l. 852-929]
17	OR	OR	0/1	`X509/RequireKeyIdentifierReference`	[OAS09d, l. 991-993]
			0/1	`X509/RequireIssuerSerialReference`	[OAS09d, l. 994-996]
			0/1	`X509/RequireEmbeddedTokenReference`	[OAS09d, l. 997-999]
			0/1	`X509/RequireThumbprintReference`	[OAS09d, l. 1000-1002]
		XOR	0/1	`X509/WssX509V3Token10`	[OAS09d, l. 1003-1005]
			0/1	`X509/WssX509Pkcs7Token10`	[OAS09d, l. 1006-1008]
			0/1	`X509/WssX509PkiPathV1Token10`	[OAS09d, l. 1009-1011]
			0/1	`X509/WssX509V1Token11`	[OAS09d, l. 1012-1014]
			0/1	`X509/WssX509V3Token11`	[OAS09d, l. 1015-1017]
			0/1	`X509/WssX509Pkcs7Token11`	[OAS09d, l. 1018-1020]
			0/1	`X509/WssX509PkiPathV1Token11`	[OAS09d, l. 1021-1023]
18	OR	-	0/1	`Kerberos/RequireKeyIdentifierReference`	[OAS09d, l. 1078-1080]
		XOR	0/1	`Kerberos/WssKerberosV5ApReqToken11`	[OAS09d, l. 1081-1083]
			0/1	`Kerberos/WssGssKerberosV5ApReqToken11`	[OAS09d, l. 1084-1086]
19	OR	-	0/1	`Spnego/MustNotSendCancel`	[OAS09d, l. 1139-1142]
			0/1	`Spnego/MustNotSendAmend`	[OAS09d, l. 1143-1146]
			0/1	`Spnego/MustNotSendRenew`	[OAS09d, l. 1147-1150]
20	OR	-	0/1	`SecCtx/RequireExternalUriReference`	[OAS09d, l. 1201-1203]
			0/1	`SecCtx/SC13SecurityContextToken`	[OAS09d, l. 1204-1206]
21	OR	-	0/1	`SecConv/RequireExternalUriReference`	[OAS09d, l. 1327-1329]
			0/1	`SecConv/SC13SecurityContextToken`	[OAS09d, l. 1330-1332]
			0/1	`SecConv/MustNotSendCancel`	[OAS09d, l. 1333-1336]
			0/1	`SecConv/MustNotSendAmend`	[OAS09d, l. 1337-1340]
			0/1	`SecConv/MustNotSendRenew`	[OAS09d, l. 1341-1344]
			0/1	`SecConv/BootstrapPolicy`	[OAS09d, l. 1345-1347]
22	OR	-	0/1	`Saml/RequireKeyIdentifierReference`	[OAS09d, l. 1449-1451]
		XOR	0/1	`Saml/WssSamlV11Token10`	[OAS09d, l. 1452-1454]
			0/1	`Saml/WssSamlV11Token11`	[OAS09d, l. 1455-1457]
			0/1	`Saml/WssSamlV20Token11`	[OAS09d, l. 1458-1460]
23	OR	-	0/1	`Rel/RequireKeyIdentifierReference`	[OAS09d, l. 1520-1522]
		XOR	0/1	`Rel/WssRelV10Token10`	[OAS09d, l. 1523-1525]
			0/1	`Rel/WssRelV20Token10`	[OAS09d, l. 1526-1528]
			0/1	`Rel/WssRelV10token11`	[OAS09d, l. 1529-1531]
			0/1	`Rel/WssRelV20Token11`	[OAS09d, l. 1532-1534]
24	XOR	-	0/1	`Https/HttpBasicAuthentication`	[OAS09d, l. 1576-1578]

Please continue reading on the next page

#	Op1	Op2	Mul	Setting	Definition
			0/1	`Https/HttpDigestAuthentication`	[OAS09d, l. 1579-1581]
			0/1	`Https/RequireClientCertificate`	[OAS09d, l. 1582-1584]
25	-	-	0/1	`RsaKeyValueToken`	[OAS09d, l. 1585-1675]

Please continue reading on the next page

#	Op1	Op2	Mul	Setting	Definition
WS-SecurityPolicy Security Binding Assertions					
26	AND	XOR	1	`Algorithm/Basic256`	[OAS09d, l. 1858-1860]
			1	`Algorithm/Basic192`	[OAS09d, l. 1861-1863]
			1	`Algorithm/Basic128`	[OAS09d, l. 1864-1866]
			1	`Algorithm/TripleDes`	[OAS09d, l. 1867-1869]
			1	`Algorithm/Basic256Rsa15`	[OAS09d, l. 1870-1872]
			1	`Algorithm/Basic192Rsa15`	[OAS09d, l. 1873-1875]
			1	`Algorithm/Basic128Rsa15`	[OAS09d, l. 1876-1878]
			1	`Algorithm/TripleDesRsa15`	[OAS09d, l. 1879-1881]
			1	`Algorithm/Basic256Sha256`	[OAS09d, l. 1882-1884]
			1	`Algorithm/Basic192Sha256`	[OAS09d, l. 1885-1887]
			1	`Algorithm/Basic128Sha256`	[OAS09d, l. 1888-1890]
			1	`Algorithm/TripleDesSha256`	[OAS09d, l. 1891-1893]
			1	`Algorithm/Basic256Sha256Rsa15`	[OAS09d, l. 1894-1896]
			1	`Algorithm/Basic192Sha256Rsa15`	[OAS09d, l. 1897-1899]
			1	`Algorithm/Basic128Sha256Rsa15`	[OAS09d, l. 1900-1902]
			1	`Algorithm/TripleDesSha256Rsa15`	[OAS09d, l. 1903-1905]
		OR	0/1	`Algorithm/InclusiveC14N`	[OAS09d, l. 1906-1909]
			0/1	`Algorithm/InclusiveC14N11`	[OAS09d, l. 1910-1915]
			0/1	`Algorithm/SOAPNormalization`	[OAS09d, l. 1916-1918]
			0/1	`Algorithm/STRTransform10`	[OAS09d, l. 1919-1921]
		XOR	0/1	`Algorithm/XPath10`	[OAS09d, l. 1922-1924]
			0/1	`Algorithm/XPathFilter20`	[OAS09d, l. 1925-1927]
			0/1	`Algorithm/AbsXPath`	[OAS09d, l. 1928-1930]
27	XOR	-	1	`Layout/Strict`	[OAS09d, l. 1954-1956]
			1	`Layout/Lax`	[OAS09d, l. 1957-1959]
			1	`Layout/LaxTsFirst`	[OAS09d, l. 1960-1963]
			1	`Layout/LaxTsLast`	[OAS09d, l. 1964-1967]
28	AND	-	1	`Transport/TransportToken`	[OAS09d, l. 1995-1998]
			1	`Transport/AlgorithmSuite`	[OAS09d, l. 2001-2003]
			0/1	`Transport/Layout`	[OAS09d, l. 2004-2006]
			0/1	`Transport/IncludeTimestamp`	[OAS09d, l. 2007-2009]
29	AND	XOR	1	`Symmetric/ProtectionToken`	[OAS09d, l. 2065-2072]
			1	`Symmetric/EncToken + SigToken`	[OAS09d, l. 2052-2064]
		AND	1	`Symmetric/AlgorithmSuite`	[OAS09d, l. 2073-2075]
			0/1	`Symmetric/Layout`	[OAS09d, l. 2076-2078]
			0/1	`Symmetric/IncludeTimestamp`	[OAS09d, l. 2079-2081]
			0/1	`Symmetric/EncryptBeforeSigning`	[OAS09d, l. 2082-2084]
			0/1	`Symmetric/EncryptSignature`	[OAS09d, l. 2085-2087]
			0/1	`Symmetric/ProtectTokens`	[OAS09d, l. 2088-2090]
			0/1	`Symmetric/OnlySignEntireHeadersAndBody`	[OAS09d, l. 2091-2093]

Please continue reading on the next page

#	Op1	Op2	Mul	Setting	Definition
30	AND	XOR	1	`Asymmetric/InitiatorToken`	[OAS09d, l. 2166-2177]
			1	`Asymmetric/InitEncToken + InitSigToken`	[OAS09d, l. 2171-2184]
		XOR	1	`Asymmetric/RecipientToken`	[OAS09d, l. 2185-2191]
			1	`Asymmetric/RecEncToken + RecSigToken`	[OAS09d, l. 2192-2203]
		AND	1	`Asymmetric/AlgorithmSuite`	[OAS09d, l. 2204-2206]
			0/1	`Asymmetric/Layout`	[OAS09d, l. 2207-2209]
			0/1	`Asymmetric/IncludeTimestamp`	[OAS09d, l. 2210-2212]
			0/1	`Asymmetric/EncryptBeforeSigning`	[OAS09d, l. 2213-2215]
			0/1	`Asymmetric/EncryptSignature`	[OAS09d, l. 2216-2218]
			0/1	`Asymmetric/ProtectTokens`	[OAS09d, l. 2219-2221]
			0/1	`Asymmetric/OnlySignEntireHeadersAndBody`	[OAS09d, l. 2222-2224]
WS-SecurityPolicy Supporting Tokens					
31	AND	-	1*	`SupportingTokens/[TokenAssertions]`	[OAS09d, l. 2304/2305]
		-	0/1	`SupportingTokens/[AlgorithmSuite]`	[OAS09d, l. 2306-2309]
		OR	0*	`SupportingTokens/SignedParts`	[OAS09d, l. 2310-2313]
			0*	`SupportingTokens/SignedElements`	[OAS09d, l. 2314-2317]
			0*	`SupportingTokens/EncryptedParts`	[OAS09d, l. 2318-2321]
			0*	`SupportingTokens/EncryptedElements`	[OAS09d, l. 2322-2325]
32	AND	-	1*	`SignedSupTokens/[TokenAssertions]`	[OAS09d, l. 2357/2358]
		-	0/1	`SignedSupTokens/[AlgorithmSuite]`	[OAS09d, l. 2359-2362]
		OR	0*	`SignedSupTokens/SignedParts`	[OAS09d, l. 2363-2366]
			0*	`SignedSupTokens/SignedElements`	[OAS09d, l. 2367-2370]
			0*	`SignedSupTokens/EncryptedParts`	[OAS09d, l. 2371-2374]
			0*	`SignedSupTokens/EncryptedElements`	[OAS09d, l. 2375-2378]
33	AND	-	1*	`EndorsingSupTokens/[TokenAssertions]`	[OAS09d, l. 2412/2413]
		-	0/1	`EndorsingSupTokens/[AlgorithmSuite]`	[OAS09d, l. 2414-2417]
		OR	0*	`EndorsingSupTokens/SignedParts`	[OAS09d, l. 2418-2421]
			0*	`EndorsingSupTokens/SignedElements`	[OAS09d, l. 2422-2425]
			0*	`EndorsingSupTokens/EncryptedParts`	[OAS09d, l. 2426-2429]
			0*	`EndorsingSupTokens/EncryptedElements`	[OAS09d, l. 2430-2433]
34	AND	-	1*	`EndSigSupTokens/[TokenAssertions]`	[OAS09d, l. 2469/2470]
		-	0/1	`EndSigSupTokens/[AlgorithmSuite]`	[OAS09d, l. 2471-2474]
		OR	0*	`EndSigSupTokens/SignedParts`	[OAS09d, l. 2475-2478]
			0*	`EndSigSupTokens/SignedElements`	[OAS09d, l. 2479-2482]
			0*	`EndSigSupTokens/EncryptedParts`	[OAS09d, l. 2483-2486]
			0*	`EndSigSupTokens/EncryptedElements`	[OAS09d, l. 2487-2490]
35	AND	-	1*	`SigEncSupTokens/[TokenAssertions]`	[OAS09d, l. 2491-2497]
		-	0/1	`SigEncSupTokens/[AlgorithmSuite]`	
		OR	0*	`SigEncSupTokens/SignedParts`	
			0*	`SigEncSupTokens/SignedElements`	
			0*	`SigEncSupTokens/EncryptedParts`	

Please continue reading on the next page

#	Op1	Op2	Mul	Setting	Definition
			0*	`SigEncSupTokens/EncryptedElements`	

Please continue reading on the next page

#	Op1	Op2	Mul	Setting	Definition
36	AND	-	1*	EncSupTokens/[TokenAssertions]	[OAS09d, l. 2498-2511]
		-	0/1	EncSupTokens/[AlgorithmSuite]	
		OR	0*	EncSupTokens/SignedParts	
			0*	EncSupTokens/SignedElements	
			0*	EncSupTokens/EncryptedParts	
			0*	EncSupTokens/EncryptedElements	
37	AND	-	1*	EndEncSupTokens/[TokenAssertions]	[OAS09d, l. 2512-2518]
		-	0/1	EndEncSupTokens/[AlgorithmSuite]	
		OR	0*	EndEncSupTokens/SignedParts	
			0*	EndEncSupTokens/SignedElements	
			0*	EndEncSupTokens/EncryptedParts	
			0*	EndEncSupTokens/EncryptedElements	
38	AND	-	1*	SigEndEncSupTokens/[TokenAssertions]	[OAS09d, l. 2519-2525]
		-	0/1	SigEndEncSupTokens/[AlgorithmSuite]	
		OR	0*	SigEndEncSupTokens/SignedParts	
			0*	SigEndEncSupTokens/SignedElements	
			0*	SigEndEncSupTokens/EncryptedParts	
			0*	SigEndEncSupTokens/EncryptedElements	
WS-SecurityPolicy WS-Security and WS-Trust Options					
39	OR	-	0/1	Wss10/MustSupportRefKeyIdentifier	[OAS09d, l. 2676-2678]
			0/1	Wss10/MustSupportRefIssureSerial	[OAS09d, l. 2679-2681]
			0/1	Wss10/MustSupportRefExternalURI	[OAS09d, l. 2682-2684]
			0/1	Wss10/MustSupportRefEmbeddedToken	[OAS09d, l. 2685-2687]
39	OR	-	0/1	Wss11/MustSupportRefKeyIdentifier	[OAS09d, l. 2710-2712]
			0/1	Wss11/MustSupportRefIssureSerial	[OAS09d, l. 2713-2715]
			0/1	Wss11/MustSupportRefExternalURI	[OAS09d, l. 2716-2718]
			0/1	Wss11/MustSupportRefEmbeddedToken	[OAS09d, l. 2719-2721]
			0/1	Wss11/MustSupportRefThumbprint	[OAS09d, l. 2722-2724]
			0/1	Wss11/MustSupportRefEncryptedKey	[OAS09d, l. 2725-2727]
			0/1	Wss11/RequireSignatureConfirmation	[OAS09d, l. 2728-2730]
39	OR	-	0/1	Wst13/MustSupportClientChallenge	[OAS09d, l. 2821-2823]
			0/1	Wst13/MustSupportServerChallenge	[OAS09d, l. 2824-2826]
			0/1	Wst13/RequireClientEntropy	[OAS09d, l. 2827-2829]
			0/1	Wst13/RequireServerEntropy	[OAS09d, l. 2830-2832]
			0/1	Wst13/MustSupportIssuedTokens	[OAS09d, l. 2833-2835]
			0/1	Wst13/RequireRequestSecurityTokenCol.	[OAS09d, l. 2836-2838]
			0/1	Wst13/RequireAppliesTo	[OAS09d, l. 2839-2841]
			0/1	Wst13/ScopePolicy15	[OAS09d, l. 2842-2844]
			0/1	Wst13/MustSupportInteractiveChallenge	[OAS09d, l. 2845-2847]

B Complete List of the Test Cases

#	Feature	Case	Settings
0	Basic SOAP	0.1	Basic SOAP
		0.2	SOAP with Attachments
1	Basic `RMAssertion`	1.1	-
2	Sequence Security	2.1	`wsrmp:SequenceSTR`
		2.2	`wsrmp:SequenceTransportSecurity`
3	Delivery Assurance	3.1	`wsrmp:ExactlyOnce`
		3.2	`wsrmp:AtLeastOnce`
		3.3	`wsrmp:AtMostOnce`
		3.4	`wsrmp:ExactlyOnce + wsrmp:InOrder`
		3.5	`wsrmp:AtLeastOnce + wsrmp:InOrder`
		3.6	`wsrmp:AtMostOnce + wsrmp:InOrder`
4	SignedParts	4.1	`sp:Body`
		4.2	`sp:Header`
		4.3	`sp:Attachments`
5	SignedElements	5.1	`//Body/?`
		5.2	`//Body`
		5.3	`//Header/?`
6	EncryptedParts	6.1	`sp:Body`
		6.2	`sp:Header`
		6.3	`sp:Attachments`
7	EncryptedElements	7.1	`//Body/?`
		7.2	`//Body`
		7.3	`//Header/?`
8	ContentEncryptedElements	8.1	`//Body/?`
		8.2	`//Body`
		8.3	`//Header/?`
9	RequiredParts	9.1	`sp:Header`
10	RequiredElements	10.1	`//Header/?`
		10.2	`//Body/?`
11	Token Inclusion	11.1	`sp:Never`
		11.2	`sp:Once`
		11.3	`sp:AlwaysToRecipient`
		11.4	`sp:AlwaysToInitiator`
		11.5	`sp:Always`
12	Issuer	12.1	`sp:Issuer` (URL)
		12.2	`sp:IssuerName`
13	Claims	13.1	`sp:Claims`
14	Token Properties	14.1	`sp:DerivedKeys`
		14.2	`sp:ExplicitDerivedKeys`
		14.3	`sp:ImpliedDerivedKeys`

Please continue reading on the next page

#	Feature	Case	Settings
15	Username Token	15.1	Basic UNT
		15.2	`sp:NoPassword`
		15.3	`sp:HashPassword`
		15.4	`sp:Created`
		15.5	`sp:Nonce`
		15.6	`sp:Created + sp:Nonce`
		15.7	`sp:WssUsernameToken10`
		15.8	`sp:WssUsernameToken11`
16	IssuedToken	16.1	Basic IssuedToken
17	X509 Token	17.1	Basic X509Token
		17.2	`sp:RequireKeyIdentifierReference`
		17.3	`sp:RequireIssuerSerialReference`
		17.4	`sp:RequireEmbeddedTokenReference`
		17.5	`sp:RequireThumbprintReference`
		17.6	`sp:WssX509V3Token10`
		17.7	`sp:WssX509Pkcs7Token10`
		17.8	`sp:WssX509PkiPathV1Token10`
		17.9	`sp:WssX509V1Token11`
		17.10	`sp:WssX509V3Token11`
		17.11	`sp:WssX509Pkcs7Token11`
		17.12	`sp:WssX509PkiPathV1Token11`
18	Kerberos Token	18.1	Basic KerberosToken
		18.2	`sp:WssKerberosV5ApReqToken11`
		18.3	`sp:WssGssKerberosV5ApReqToken11`
19	SpnegoContext Token	19.1	Basic SpnegoContextToken
		19.2	`sp:MustNotSendCancel`
		19.3	`sp:MustNotSendAmend`
		19.4	`sp:MustNotSendRenve`
20	SecurityContext Token	20.1	Basic SecurityContextToken
		20.2	`sp:RequireExternalUriReference`
		20.3	`sp:SC13SecurityContextToken`
21	SecureConversation Token	21.1	Basic SecureConversation Token
		21.2	`sp:RequireExternalUriReference`
		21.3	`sp:SC13SecurityContextToken`
		21.4	`sp:MustNotSendCancel`
		21.5	`sp:MustNotSendAmend`
		21.6	`sp:MustNotSendRenew`
		21.7	`sp:BootstrapPolicy`
22	Saml Token	22.1	Basic SamlToken
		22.2	`sp:WssSamlV11Token10`
		22.3	`sp:WssSamlV11Token11`
		22.4	`sp:WssSamlV20Token11`
		23.1	Basic SecureConversation Token

Please continue reading on the next page

23 REL Token

#	Feature	Case	Settings
		23.2	`sp:RequireKeyIdentifierReference`
		23.3	`sp:WssRelV10Token10`
		23.4	`sp:WssRelV20Token10`
		23.5	`sp:WssRelV10Token11`
		23.6	`sp:WssRelV20Token11`
24	Https Token	24.1	Basic Https Token
		24.2	`sp:HttpBasicAuthentication`
		24.3	`sp:HttpDigestAuthentication`
		24.4	`sp:RequireClientCertificate`
25	KeyValue Token	25.1	Basic KeyValue Token
		25.2	`sp:RsaKeyValue`
26	AlgorithmSuite	26.1	`sp:Basic256`
		26.2	`sp:Basic192`
		26.3	`sp:Basic128`
		26.4	`sp:TripleDes`
		26.5	`sp:Basic256Rsa15`
		26.6	`sp:Basic192Rsa15`
		26.7	`sp:Basic128Rsa15`
		26.8	`sp:TripleDesRsa15`
		26.9	`sp:Basic256Sha256`
		26.10	`sp:Basic192Sha256`
		26.11	`sp:Basic128Sha256`
		26.12	`sp:TripleDesSha256`
		26.13	`sp:Basic256Sha256Rsa15`
		26.14	`sp:Basic192Sha256Rsa15`
		26.15	`sp:Basic128Sha256Rsa15`
		26.16	`sp:TripleDesSha256Rsa15`
		26.17	`sp:InclusiveC14N`
		26.18	`sp:InclusiveC14N11`
		26.19	`sp:SOAPNormalization`
		26.20	`sp:STRTransform10`
		26.21	`sp:XPath10`
		26.22	`sp:XPathFilter20`
		26.23	`sp:AbsXPath`
27	Layout	27.1	`sp:Strict`
		27.2	`sp:Lax`
		27.3	`sp:LaxTsFirst`
		27.4	`sp:LaxTsLast`
28	TransportBinding	28.1	Basic TransportBinding
		28.*	WebSphere SSL Policy Type
		28.2	`sp:IncludeTimestamp`
		29.1	`sp:EncryptionToken + sp:SignatureToken`
		29.2	`sp:ProtectionToken`

Please continue reading on the next page

29 SymmetricBinding

#	Feature	Case	Settings
		29.3	`sp:IncludeTimestamp`
		29.4	`sp:EncryptBeforeSigning`
		29.5	`sp:EncryptSignature`
		29.6	`sp:ProtectTokens`
		29.7	`sp:OnlySignEntireHeadersAndBody`

Please continue reading on the next page

#	**Feature**	**Case**	**Settings**
30	AsymmetricBinding	30.1	`sp:InitiatorToken`
		30.2	`sp:InitEncToken + sp:InitSigToken`
		30.3	`sp:RecipientToken`
		30.4	`sp:RecEncToken + sp:RecSigToken`
		30.5	`sp:IncludeTimestamp`
		30.6	`sp:EncryptBeforeSigning`
		30.7	`sp:EncryptSignature`
		30.8	`sp:ProtectTokens`
		30.9	`sp:OnlySignEntireHeadersAndBody`
31	SupportingTokens	31.1	Basic `sp:SupportingTokens`
32		32.1	Basic `sp:SignedSupportingTokens`
33		33.1	Basic `sp:EndorsingSupportingTokens`
34		34.1	Basic `sp:SignedEndorsingSupportingTokens`
35		35.1	Basic `sp:SignedEncryptedSupportingTokens`
36		36.1	Basic `sp:EncryptedSupportingTokens`
37		37.1	Basic `sp:EndorsingEncryptedSupportingTokens`
38		38.1	Basic `sp:SignedEndorsingEncryptedSupportingT.`
39	WSS 1.0 Assertions	39.1	`sp:MustSupportRefKeyIdentifier`
		39.2	`sp:MustSupportRefIssuerSerial`
		39.3	`sp:MustSupportRefExternalURI`
		39.4	`sp:MustSupportRefEmbeddedToken`
40	WSS 1.1 Assertions	40.1	`sp:MustSupportRefKeyIdentifier`
		40.2	`sp:MustSupportRefIssuerSerial`
		40.3	`sp:MustSupportRefExternalURI`
		40.4	`sp:MustSupportRefEmbeddedToken`
		40.5	`sp:MustSupportRefThumbprint`
		40.6	`sp:MustSupportRefEncryptedKey`
		40.7	`sp:RequireSignatureConfirmation`
41	WS-Trust Assertions	41.1	`sp:MustSupportClientChallenge`
		41.2	`sp:MustSupportServerChallenge`
		41.3	`sp:RequireClientEntropy`
		41.4	`sp:RequireServerEntropy`
		41.5	`sp:MustSupportIssuedTokens`
		41.6	`sp:RequireRequestSecurityTokenCollection`
		41.7	`sp:RequireAppliesTo`
		41.8	`sp:ScopePolicy15`
		41.9	`sp:MustSupportInteractiveChallenges`

C Complete List of Test Results

Service ▶	WebSphere	GlassFish	WebSphere	GlassFish
Client ▶	WebSphere	GlassFish	GlassFish	WebSphere
Case ▼				
0.1	12	12	12	12
0.2	5	5	5	5
1.1	1	12	1	4
2.1	1	5	1	4
2.2	1	12	1	4
3.1	12	12	12	4
3.2	1	12	1	4
3.3	1	12	1	4
3.4	12	12	12	4
3.5	1	12	1	4
3.6	1	12	1	4
4.1	12	12	9	6
4.2	12	12	12	12
4.3	-	-	-	-
5.1	6	3	3	6
5.2	12	3	3	6
5.3	12	3	3	6
6.1	12	12	12	6
6.2	12	12	6	6
6.3	-	-	-	-
7.1	12	3	3	6
7.2	3	3	3	3
7.3	12	3	3	4
8.1	1	2	1	2
8.2	1	2	1	2
8.3	1	2	1	2
9.1	1	2	1	2
9.2	1	2	1	2
10.1	1	2	1	2
10.2	1	2	1	2
11.1	5	12	6	4
11.2	5	3	3	6
11.3	12	12	12	12
11.4	12	12	12	12
11.5	12	12	12	12
12.1	1	12	1	5
12.2	1	5	1	5
13.1	1	12	1	12
14.1	12	12	3	12

Please continue reading on the next page

Service ▶	**WebSphere**	**GlassFish**	**WebSphere**	**GlassFish**
Client ▶	**WebSphere**	**GlassFish**	**GlassFish**	**WebSphere**
Case ▼				
14.2	1	12	1	6
14.3	1	12	1	6
15.1	1	3/6	1	6
15.2	1	12	1	3
15.3	1	3/6	1	3
15.4	1	3/6	1	3
15.5	1	3/6	1	3
15.6	1	3/6	1	3
15.7	12*	12	6	6
15.8	1	12	1	3
16.1	1	12	1	3
17.1	1	12	1	3
17.2	1	12	1	4/9
17.3	1	12	1	4/9
17.4	1	12	1	4/9
17.5	1	12	1	4/9
17.6	12	12	12	12
17.7	1	12	1	3
17.8	1	12	1	3
17.9	1	12	1	3
17.10	12	12	12	12
17.11	1	12	1	3
17.12	1	12	1	3
18.1	1	-*	1	-*
18.2	1	-*	1	-*
18.3	1	-*	1	-*
19.1	1	3	1	3
19.2	1	3	1	3
19.3	1	3	1	3
19.4	1	3	1	3
20.1	1	3	1	3
20.2	1	3	1	3
20.3	1	3	1	3
21.1	12	12	6	6
21.2	12	12	3	6
21.3	1	5	1	6
21.4	1	5	1	6
21.5	1	5	1	6
21.6	1	5	1	6
21.7	12	12	6	6
22.1	1	12	1/3	3
22.2	1	12	1/3	3

Please continue reading on the next page

Service ▶	**WebSphere**	**GlassFish**	**WebSphere**	**GlassFish**
Client ▶	**WebSphere**	**GlassFish**	**GlassFish**	**WebSphere**
Case ▼				
22.3	1	12	1/3	3
22.4	1	12	1/3	3
23.1	1	3	1	3
23.2	1	3	1	3
23.3	1	3	1	3
23.4	1	3	1	3
23.5	1	3	1	3
23.6	1	3	1	3
24.1	1	12	12	3
24.2	1	12	1	3
24.3	1	12	1	3
24.4	1	12	1	3
25.1	1	12	1	5
25.2	1	5	1	5
26.1	12	12	12	12
26.2	12	12	12	12
26.3	12	12	12	12
26.4	12	12	12	12
26.5	12	12	12	12
26.6	12	12	12	12
26.7	12	12	12	12
26.8	12	12	12	12
26.9	6	12	12	5
26.10	6	12	12	5
26.11	6	12	12	5
26.12	6	12	12	5
26.13	6	12	12	5
26.14	6	12	12	5
26.15	6	12	12	5
26.16	6	12	12	5
26.17	6	12	12	12
26.18	1	1	1	1
26.19	12	1	3	1
26.20	12	12	12	12
26.21	1	1	1	1
26.22	12	1	3	1
26.23	1	1	1	1
27.1	12	12	12	12
27.2	12	12	12	12
27.3	12	12	6	9
27.4	12	12	9	12
28.1	1	12	1	3

Please continue reading on the next page

Service ▶	WebSphere	GlassFish	WebSphere	GlassFish
Client ▶	WebSphere	GlassFish	GlassFish	WebSphere
Case ▼				
28.*	3	-	12	-
28.2	1	12	1	3
29.1	1	12	1	3
29.2	12	12	6	3
29.3	12	12	12	12
29.4	1	12	1	3
29.5	1	12	1	3
29.6	1	12	1	3
29.7	1	12	1	3
30.1	12	12	12	12
30.2	12	2	3	2
30.3	12	12	12	12
30.4	12	2	3	2
30.5	12	12	12	12
30.6	1	12	1	6
30.7	1	12	1	6
30.8	1	12	1	3
30.9	1	12	1	12
31.1	6*	12	6	3
32.1	1	12	1	3
33.1	1	6	1	3
34.1	1	6	1	3
35.1	1	12	1	3
36.1	1	12	1	3
37.1	1	6	1	3
38.1	1	6	1	3
39.1	12	12	12	12
39.2	12	12	12	12
39.3	1	2	1	2
39.4	12	2	3	2
40.1	12	12	12	12
40.2	12	12	12	12
40.3	1	2	1	2
40.4	12	2	3	2
40.5	12	12	12	12
40.6	1	12	12	3
40.7	12	12	12	12
41.1	1	2	1	2
41.2	1	2	1	2
41.3	1	2	1	2
41.4	1	2	1	2

Please continue reading on the next page

Service ▶	**WebSphere**	**GlassFish**	**WebSphere**	**GlassFish**
Client ▶	**WebSphere**	**GlassFish**	**GlassFish**	**WebSphere**
Case ▼				
41.5	1	2	1	2
41.6	1	2	1	2
41.7	1	2	1	2
41.8	1	2	1	2
41.9	1	2	1	2

D List of previous University of Bamberg reports

Bamberger Beiträge zur Wirtschaftsinformatik

Nr. 1 (1989)	Augsburger W., Bartmann D., Sinz E.J.: Das Bamberger Modell: Der Diplom-Studiengang Wirtschaftsinformatik an der Universität Bamberg (Nachdruck Dez. 1990)
Nr. 2 (1990)	Esswein W.: Definition, Implementierung und Einsatz einer kompatiblen Datenbankschnittstelle für PROLOG
Nr. 3 (1990)	Augsburger W., Rieder H., Schwab J.: Endbenutzerorientierte Informationsgewinnung aus numerischen Daten am Beispiel von Unternehmenskennzahlen
Nr. 4 (1990)	Ferstl O.K., Sinz E.J.: Objektmodellierung betrieblicher Informationsmodelle im Semantischen Objektmodell (SOM) (Nachdruck Nov. 1990)
Nr. 5 (1990)	Ferstl O.K., Sinz E.J.: Ein Vorgehensmodell zur Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM)
Nr. 6 (1991)	Augsburger W., Rieder H., Schwab J.: Systemtheoretische Repräsentation von Strukturen und Bewertungsfunktionen über zeitabhängigen betrieblichen numerischen Daten
Nr. 7 (1991)	Augsburger W., Rieder H., Schwab J.: Wissensbasiertes, inhaltsorientiertes Retrieval statistischer Daten mit EISREVU / Ein Verarbeitungsmodell für eine modulare Bewertung von Kennzahlenwerten für den Endanwender
Nr. 8 (1991)	Schwab J.: Ein computergestütztes Modellierungssystem zur Kennzahlenbewertung
Nr. 9 (1992)	Gross H.-P.: Eine semantiktreue Transformation vom Entity-Relationship-Modell in das Strukturierte Entity-Relationship-Modell
Nr. 10 (1992)	Sinz E.J.: Datenmodellierung im Strukturierten Entity-Relationship-Modell (SERM)
Nr. 11 (1992)	Ferstl O.K., Sinz E. J.: Glossar zum Begriffsystem des Semantischen Objektmodells
Nr. 12 (1992)	Sinz E. J., Popp K.M.: Zur Ableitung der Grobstruktur des konzeptuellen Schemas aus dem Modell der betrieblichen Diskurswelt
Nr. 13 (1992)	Esswein W., Locarek H.: Objektorientierte Programmierung mit dem Objekt-Rollenmodell
Nr. 14 (1992)	Esswein W.: Das Rollenmodell der Organsiation: Die Berücksichtigung aufbauorganisatorische Regelungen in Unternehmensmodellen
Nr. 15 (1992)	Schwab H. J.: EISREVU-Modellierungssystem. Benutzerhandbuch
Nr. 16 (1992)	Schwab K.: Die Implementierung eines relationalen DBMS nach dem Client/Server-Prinzip
Nr. 17 (1993)	Schwab K.: Konzeption, Entwicklung und Implementierung eines computergestützten Bürovorgangssystems zur Modellierung von Vorgangsklassen und Abwicklung und Überwachung von Vorgängen. Dissertation

Nr. 18 (1993)	Ferstl O.K., Sinz E.J.: Der Modellierungsansatz des Semantischen Objektmodells
Nr. 19 (1994)	Ferstl O.K., Sinz E.J., Amberg M., Hagemann U., Malischewski C.: Tool-Based Business Process Modeling Using the SOM Approach
Nr. 20 (1994)	Ferstl O.K., Sinz E.J.: From Business Process Modeling to the Specification of Distributed Business Application Systems - An Object-Oriented Approach -. 1st edition, June 1994
	Ferstl O.K., Sinz E.J. : Multi-Layered Development of Business Process Models and Distributed Business Application Systems - An Object-Oriented Approach -. 2nd edition, November 1994
Nr. 21 (1994)	Ferstl O.K., Sinz E.J.: Der Ansatz des Semantischen Objektmodells zur Modellierung von Geschäftsprozessen
Nr. 22 (1994)	Augsburger W., Schwab K.: Using Formalism and Semi-Formal Constructs for Modeling Information Systems
Nr. 23 (1994)	Ferstl O.K., Hagemann U.: Simulation hierarischer objekt- und transaktionsorientierter Modelle
Nr. 24 (1994)	Sinz E.J.: Das Informationssystem der Universität als Instrument zur zielgerichteten Lenkung von Universitätsprozessen
Nr. 25 (1994)	Wittke M., Mekinic, G.: Kooperierende Informationsräume. Ein Ansatz für verteilte Führungsinformationssysteme
Nr. 26 (1995)	Ferstl O.K., Sinz E.J.: Re-Engineering von Geschäftsprozessen auf der Grundlage des SOM-Ansatzes
Nr. 27 (1995)	Ferstl, O.K., Mannmeusel, Th.: Dezentrale Produktionslenkung. Erscheint in CIM-Management 3/1995
Nr. 28 (1995)	Ludwig, H., Schwab, K.: Integrating cooperation systems: an event-based approach
Nr. 30 (1995)	Augsburger W., Ludwig H., Schwab K.: Koordinationsmethoden und -werkzeuge bei der computergestützten kooperativen Arbeit
Nr. 31 (1995)	Ferstl O.K., Mannmeusel T.: Gestaltung industrieller Geschäftsprozesse
Nr. 32 (1995)	Gunzenhäuser R., Duske A., Ferstl O.K., Ludwig H., Mekinic G., Rieder H., Schwab H.-J., Schwab K., Sinz E.J., Wittke M: Festschrift zum 60. Geburtstag von Walter Augsburger
Nr. 33 (1995)	Sinz, E.J.: Kann das Geschäftsprozeßmodell der Unternehmung das unternehmensweite Datenschema ablösen?
Nr. 34 (1995)	Sinz E.J.: Ansätze zur fachlichen Modellierung betrieblicher Informationssysteme - Entwicklung, aktueller Stand und Trends -
Nr. 35 (1995)	Sinz E.J.: Serviceorientierung der Hochschulverwaltung und ihre Unterstützung durch workflow-orientierte Anwendungssysteme
Nr. 36 (1996)	Ferstl O.K., Sinz, E.J., Amberg M.: Stichwörter zum Fachgebiet Wirtschaftsinformatik. Erscheint in: Broy M., Spaniol O. (Hrsg.): Lexikon Informatik und Kommunikationstechnik, 2. Auflage, VDI-Verlag, Düsseldorf 1996

Nr. 37 (1996)	Ferstl O.K., Sinz E.J.: Flexible Organizations Through Object-oriented and Transaction-oriented Information Systems, July 1996
Nr. 38 (1996)	Ferstl O.K., Schäfer R.: Eine Lernumgebung für die betriebliche Aus- und Weiterbildung on demand, Juli 1996
Nr. 39 (1996)	Hazebrouck J.-P.: Einsatzpotentiale von Fuzzy-Logic im Strategischen Management dargestellt an Fuzzy-System-Konzepten für Portfolio-Ansätze
Nr. 40 (1997)	Sinz E.J.: Architektur betrieblicher Informationssysteme. In: Rechenberg P., Pomberger G. (Hrsg.): Handbuch der Informatik, Hanser-Verlag, München 1997
Nr. 41 (1997)	Sinz E.J.: Analyse und Gestaltung universitärer Geschäftsprozesse und Anwendungssysteme. Angenommen für: Informatik ’97. Informatik als Innovationsmotor. 27. Jahrestagung der Gesellschaft für Informatik, Aachen 24.-26.9.1997
Nr. 42 (1997)	Ferstl O.K., Sinz E.J., Hammel C., Schlitt M., Wolf S.: Application Objects – fachliche Bausteine für die Entwicklung komponentenbasierter Anwendungssysteme. Angenommen für: HMD – Theorie und Praxis der Wirtschaftsinformatik. Schwerpunkheft ComponentWare, 1997
Nr. 43 (1997):	Ferstl O.K., Sinz E.J.: Modeling of Business Systems Using the Semantic Object Model (SOM) – A Methodological Framework - . Accepted for: P. Bernus, K. Mertins, and G. Schmidt (ed.): Handbook on Architectures of Information Systems. International Handbook on Information Systems, edited by Bernus P., Blazewicz J., Schmidt G., and Shaw M., Volume I, Springer 1997
	Ferstl O.K., Sinz E.J.: Modeling of Business Systems Using (SOM), 2nd Edition. Appears in: P. Bernus, K. Mertins, and G. Schmidt (ed.): Handbook on Architectures of Information Systems. International Handbook on Information Systems, edited by Bernus P., Blazewicz J., Schmidt G., and Shaw M., Volume I, Springer 1998
Nr. 44 (1997)	Ferstl O.K., Schmitz K.: Zur Nutzung von Hypertextkonzepten in Lernumgebungen. In: Conradi H., Kreutz R., Spitzer K. (Hrsg.): CBT in der Medizin – Methoden, Techniken, Anwendungen -. Proceedings zum Workshop in Aachen 6. – 7. Juni 1997. 1. Auflage Aachen: Verlag der Augustinus Buchhandlung
Nr. 45 (1998)	Ferstl O.K.: Datenkommunikation. In. Schulte Ch. (Hrsg.): Lexikon der Logistik, Oldenbourg-Verlag, München 1998
Nr. 46 (1998)	Sinz E.J.: Prozeßgestaltung und Prozeßunterstützung im Prüfungswesen. Erschienen in: Proceedings Workshop „Informationssysteme für das Hochschulmanagement“. Aachen, September 1997
Nr. 47 (1998)	Sinz, E.J.:, Wismans B.: Das „Elektronische Prüfungsamt“. Erscheint in: Wirtschaftswissenschaftliches Studium WiSt, 1998
Nr. 48 (1998)	Haase, O., Henrich, A.: A Hybrid Respresentation of Vague Collections for Distributed Object Management Systems. Erscheint in: IEEE Transactions on Knowledge and Data Engineering
Nr. 49 (1998)	Henrich, A.: Applying Document Retrieval Techniques in Software Engineering Environments. In: Proc. International Conference on Database and Expert Systems

	Applications. (DEXA 98), Vienna, Austria, Aug. 98, pp. 240-249, Springer, Lecture Notes in Computer Sciences, No. 1460
Nr. 50 (1999)	Henrich, A., Jamin, S.: On the Optimization of Queries containing Regular Path Expressions. Erscheint in: Proceedings of the Fourth Workshop on Next Generation Information Technologies and Systems (NGITS'99), Zikhron-Yaakov, Israel, July, 1999 (Springer, Lecture Notes)
Nr. 51 (1999)	Haase O., Henrich, A.: A Closed Approach to Vague Collections in Partly Inaccessible Distributed Databases. Erscheint in: Proceedings of the Third East-European Conference on Advances in Databases and Information Systems – ADBIS'99, Maribor, Slovenia, September 1999 (Springer, Lecture Notes in Computer Science)
Nr. 52 (1999)	Sinz E.J., Böhnlein M., Ulbrich-vom Ende A.: Konzeption eines Data Warehouse-Systems für Hochschulen. Angenommen für: Workshop „Unternehmen Hochschule“ im Rahmen der 29. Jahrestagung der Gesellschaft für Informatik, Paderborn, 6. Oktober 1999
Nr. 53 (1999)	Sinz E.J.: Konstruktion von Informationssystemen. Der Beitrag wurde in geringfügig modifizierter Fassung angenommen für: Rechenberg P., Pomberger G. (Hrsg.): Informatik-Handbuch. 2., aktualisierte und erweiterte Auflage, Hanser, München 1999
Nr. 54 (1999)	Herda N., Janson A., Reif M., Schindler T., Augsburger W.: Entwicklung des Intranets SPICE: Erfahrungsbericht einer Praxiskooperation.
Nr. 55 (2000)	Böhnlein M., Ulbrich-vom Ende A.: Grundlagen des Data Warehousing. Modellierung und Architektur
Nr. 56 (2000)	Freitag B, Sinz E.J., Wismans B.: Die informationstechnische Infrastruktur der Virtuellen Hochschule Bayern (vhb). Angenommen für Workshop "Unternehmen Hochschule 2000" im Rahmen der Jahrestagung der Gesellschaft f. Informatik, Berlin 19. - 22. September 2000
Nr. 57 (2000)	Böhnlein M., Ulbrich-vom Ende A.: Developing Data Warehouse Structures from Business Process Models.
Nr. 58 (2000)	Knobloch B.: Der Data-Mining-Ansatz zur Analyse betriebswirtschaftlicher Daten.
Nr. 59 (2001)	Sinz E.J., Böhnlein M., Plaha M., Ulbrich-vom Ende A.: Architekturkonzept eines verteilten Data-Warehouse-Systems für das Hochschulwesen. Angenommen für: WI-IF 2001, Augsburg, 19.-21. September 2001
Nr. 60 (2001)	Sinz E.J., Wismans B.: Anforderungen an die IV-Infrastruktur von Hochschulen. Angenommen für: Workshop „Unternehmen Hochschule 2001“ im Rahmen der Jahrestagung der Gesellschaft für Informatik, Wien 25. – 28. September 2001

Änderung des Titels der Schriftenreihe *Bamberger Beiträge zur Wirtschaftsinformatik* in *Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik* ab Nr. 61

Note: The title of our technical report series has been changed from *Bamberger Beiträge zur Wirtschaftsinformatik* to *Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik* starting with TR No. 61

Nr. 75 (2008)	Karsten Loesing (Hrsg.): Extended Abstracts of the Second *Privacy Enhancing Technologies Convention* (PET-CON 2008.1). Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 75, Bamberg University, April 2008. ISSN 0937-3349.
Nr. 76 (2008)	Gregor Scheithauer and Guido Wirtz: Applying Business Process Management Systems? A Case Study. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 76, Bamberg University, May 2008. ISSN 0937-3349.
Nr. 77 (2008)	Michael Mendler, Stephan Scheele: Towards Constructive Description Logics for Abstraction and Refinement. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 77, Bamberg University, September 2008. ISSN 0937-3349.
Nr. 78 (2008)	Gregor Scheithauer and Matthias Winkler: A Service Description Framework for Service Ecosystems. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 78, Bamberg University, October 2008. ISSN 0937-3349.
Nr. 79 (2008)	Christian Wilms: Improving the Tor Hidden Service Protocol Aiming at Better Performances. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 79, Bamberg University, November 2008. ISSN 0937-3349.
Nr. 80 (2009)	Thomas Benker, Stefan Fritzemeier, Matthias Geiger, Simon Harrer, Tristan Kessner, Johannes Schwalb, Andreas Schönberger, Guido Wirtz: QoS Enabled B2B Integration. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 80, Bamberg University, May 2009. ISSN 0937-3349.
Nr. 81 (2009)	Ute Schmid, Emanuel Kitzelmann, Rinus Plasmeijer (Eds.): Proceedings of the ACM SIGPLAN Workshop on *Approaches and Applications of Inductive Programming* (AAIP'09), affiliated with ICFP 2009, Edinburgh, Scotland, September 2009. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 81, Bamberg University, September 2009. ISSN 0937-3349.
Nr. 82 (2009)	Ute Schmid, Marco Ragni, Markus Knauff (Eds.): Proceedings of the KI 2009 Workshop *Complex Cognition*, Paderborn, Germany, September 15, 2009. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 82, Bamberg University, October 2009. ISSN 0937-3349.
Nr. 83 (2009)	Andreas Schönberger, Christian Wilms and Guido Wirtz: A Requirements Analysis of Business-to-Business Integration. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 83, Bamberg University, December 2009. ISSN 0937-3349.
Nr. 84 (2010)	Werner Zirkel and Guido Wirtz: A Process for Identifying Predictive Correlation Patterns in Service Management Systems. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 84, Bamberg University, February 2010. ISSN 0937-3349.
Nr. 85 (2010)	Jan Tobias Mühlberg und Gerald Lüttgen: Symbolic Object Code Analysis. Bamberger Beiträge zur Wirtschaftsinformatik und Angewandten Informatik Nr. 85, Bamberg University, February 2010. ISSN 0937-3349.