Approaching Interoperability Testing of QoS based on WS-* Standards Implementations

Johannes Schwalb[1], Andreas Schönberger[2*], and Guido Wirtz[2]

[1] Senacor Technologies AG, Schwaig b. Nürnberg, Germany
johannes.schwalb@senacor.com
[2] Distributed Systems Group, University of Bamberg, Bamberg, Germany
{andreas.schoenberger,guido.wirtz}@uni-bamberg.de

Abstract. The maturity of Web services has increased significantly during the last years and thus allows to use this technology even for enterprise purposes. One important factor for that are the so-called WS-* standards, Web services extensions providing essential QoS properties like security and reliability.
While these features clearly are a prerequisite for enterprise use, their complexity also is a threat to the interoperability of Web services platforms. However, interoperability is a core promise of the Web services technology. So far, little research work has been put into assessing interoperability of WS-* implementations. This work targets at filling this gap by operationalizing the interoperability notion and proposing a test method targeted at WS-* standards. The approach has been validated by testing the interoperability of the WS-Security and WS-ReliableMessaging modules of two major Web services stack implementations.

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

1 Introduction

During the last years, Web services have matured significantly. Implementations of Web services standards are available on almost any platform and for almost any programming language. Major IT solution providers leverage Web services technology for a variety of application domains. Business-to-business (B2B) communities like RosettaNet[3] even propagate Web services for the implementation of inter-organizational business processes (cf. [13,14]). The most important reason for that probably is that Web services, as an interface technology, allow for the connection of software components developed on different platforms and with different programming languages in an interoperable way. Another important factor for the use of Web services in an enterprise setting is the availability of essential QoS properties like reliability or security which are brought to the

* corresponding author
[3] http://www.rosettanet.org

Web services world by a series of Web services extensions, the so-called WS-* standards. These extensions are provided as separate standards like WS-Security [7] or WS-ReliableMessaging [9] and define a rich feature set for accommodating a variety of application scenarios. However, these additional features come at the price of complexity which challenges other important QoS properties of services such as adherence to standards and the obligations imposed on service clients and service providers. In order to ensure adherence to standards and to get a grasp on the effort that results from implementing QoS features based on WS-* standards for service clients/providers, research in interoperability testing of WS-* standards is an essential need.
This paper contributes to this research field by answering the following questions:

- What is the meaning of interoperability for WS-* standards?
- What is a reasonable method for testing WS-* implementation interoperability?

The paper proceeds as follows: Section 2 pins down the notion of WS-* standards and section 3 operationalizes the notion of interoperability for the purpose of this work. A method for interoperability testing of WS-* standards is described in section 4 and evaluated for the case of WS-Security and WS-ReliableMessaging in section 5. Section 6 discusses related work and section 7 concludes and points out directions for future work.

2 WS-* Standards

For the purpose of this paper, the most important elements of a Web services stack are illustrated in figure 1. Starting at the bottom, widespread Internet protocols like HTTP, SMTP and TCP/IP are designated to be used at the transport level. In turn, the SOAP protocol [21] is designated to leverage a transport level protocol for exchanging XML messages that are packaged within SOAP containers/messages. Using SOAP, message exchanges may be performed between a SOAP sender, multiple SOAP intermediaries and an eventual SOAP receiver. At this level, which we will denote the SOAP level/messaging level in the following, WS-Addressing [20] can be used to correlate SOAP messages. On top of the SOAP level/messaging level, several Web services extensions are defined for realizing non-functional attributes such as reliability, transactional integrity or security. Those QoS standards typically are defined in terms of processing instructions for SOAP messages. For example, WS-Security describes how to use XML Signature [23] or XML Encryption [19] for signing/encrypting SOAP header or body elements. Also, QoS standards may use several SOAP messages for implementing a feature for a single payload XML message where the individual SOAP messages can be correlated using WS-Addressing. For example, WS-ReliableMessaging uses a whole protocol for exchanging a payload message (cf. fig. 2). The description of services is done using the well-known Web Services Description Language (WSDL[4], [18]). While WSDL provides the means to

[4] Although WSDL 2.0 is available since several years now, WSDL 1.1 still is more frequently employed in industry and academia.

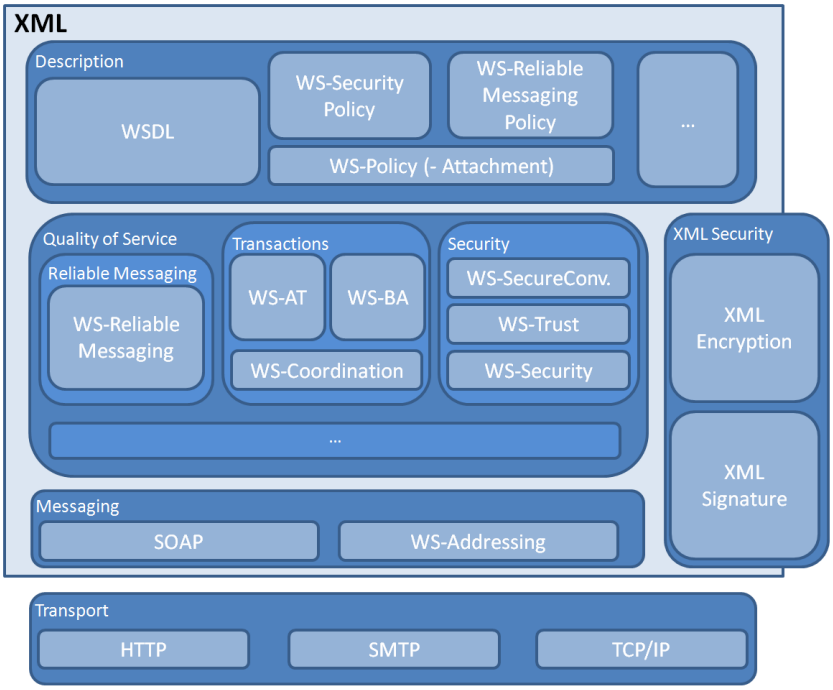

Fig. 1. Important elements of a Web services stack

describe the functionality of a Web service in terms of which messages can be exchanged and how, WS-Policy [22] provides a framework for asserting QoS properties for Web services message exchanges. Typically, dedicated policy standards like Web Services Reliable Messaging Policy Assertion [8] or WS-SecurityPolicy [11] extend the WS-Policy framework for providing specific policy expressions for the respective QoS standard. An application programmer would expose the functionality of a software component via Web services by binding its interface to a WSDL interface using a Web services stack implementation. Additionally, she would use WS-Policy expressions for requiring the Web services stack to provide QoS features as defined by the processing instructions of the QoS standard. For the purpose of this work, the term "WS-* standard" is defined to be a Web services extension that implements a QoS feature for Web services interactions by defining detailed SOAP message processing instructions. Also, a set of WS-Policy expressions for asserting the respective features is assumed to be available for a "WS-* standard".

3 Interoperability

In [24], interoperability is defined as follows:

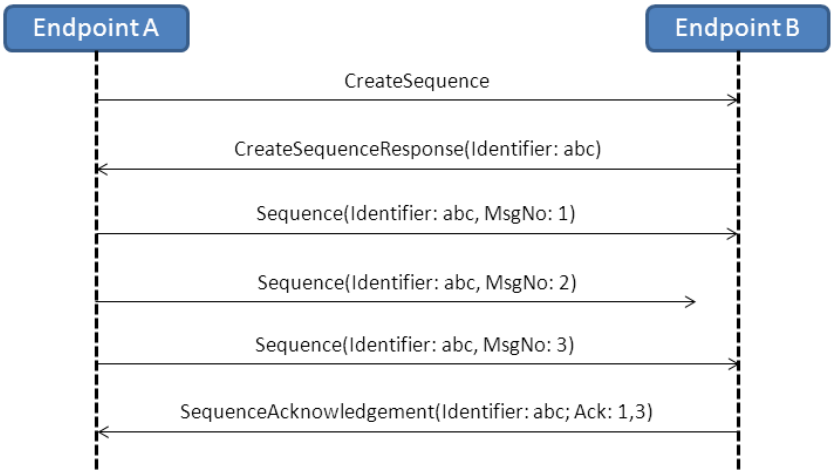

Fig. 2. Part of a WS-ReliableMessaging protocol run, adapted from [9]

> "Interoperability is the ability of two or more software components to cooperate despite differences in language, interface, and execution platform."

While this definition is good enough for an abstract characterization of interoperability in arbitrary systems, it can be refined for the purpose of WS-* interoperability testing in order to distinguish the different sources of interoperability issues between two Web services stack implementations (WS stack implementations in the following). First, one of the WS stack implementations under consideration may not know/refuse a particular WS-Policy expression that specifies a particular communication feature. Second, one of the WS stack implementations may accept a WS-Policy expression, but ignore it. Third, a WS stack implementation may deviate from one or more of the processing instructions that are specified by a WS-* standard for the implementation of a particular WS-Policy expression. Considering these sources of interoperability issues and taking into account that a Web service interaction typically takes place between a client role and a server role, we define interoperability in terms of 12 different *interoperability levels* for the purpose of this paper as follows.

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 and 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 prematurely:** 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 an unexpected fault before protocol completion.
10. **Server terminates communication prematurely:** Symmetric to level 8.
11. **Server aborts communication:** Symmetric to level 9.
12. **Protocol success:** The communication protocol has been performed successfully.

Note: The level numbers are aligned with the progress of a Web service call and do not express an order of interoperability in a semantic sense. So, level 5 does not express *better* or *lower* interoperability than level 1, but an issue at level 5 is detected at a later point in time than at level 1. Obviously, level 12 expresses full interoperability since no issues were detected.

4 Test Method

Testing WS-* implementations is a relatively new field in software testing. Although several authors have examined robustness or performance issues of Web services and even WS-* frameworks in homogeneous and heterogeneous environments (see section 6), there have only been few publications about interoperability 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, i.e., 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*, cf. [27], page 370) and are designed to cover any defined assertion/element of the WS-* standards (*structural testing*, cf. [27], page 371). Therefore, the test cases are defined as specification-based structural testing (cf. [27], page 383). However, these test cases are not able to achieve the same results as a formal analysis of the standards and implementations. Nevertheless, they are an adequate way to determine which functions are interoperable and which are not. In addition, the results of the tests do not only provide a compatibility analysis of WS-* implementations, but also an analysis of the extent the WS-* frameworks under test implement the features specified by the corresponding standards (*coverage analysis*).

Figure 3 visualizes our approach to apply *specification-based structural testing*

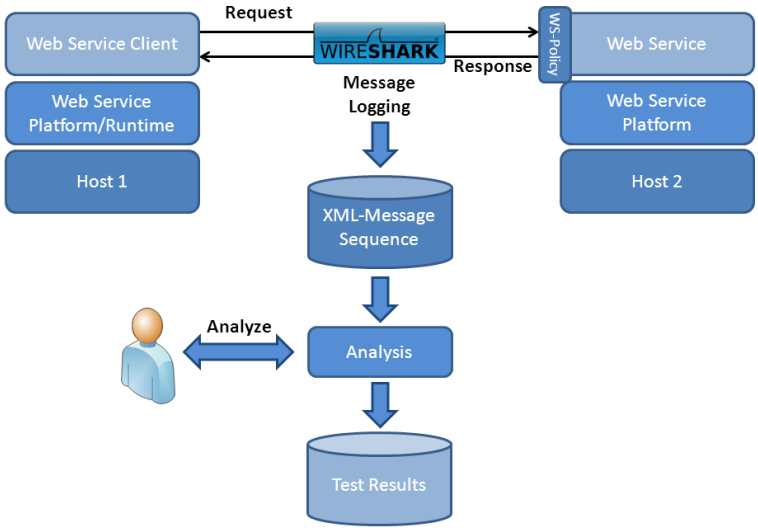

Fig. 3. Setup of test environment

to WS-* standards. In order to determine whether two WS-* framework implementations are compatible, policies (according to WS-Policy) are used to specify concrete test cases. For each test case, the WSDL of a sample Web service is extended with a WS-Policy definition to be used by the Web services stacks of the Web service client and provider for determining the number, sequence and contents of the SOAP messages to be exchanged (upper part of figure 3). The interoperability levels identified in section 3 apply to each test case. Some of the interoperability levels can be verified without examining the SOAP messages exchanged, e.g., refusal of the policy by the server. The analysis and determination of other interoperability levels require the use of network analysis tools

like Wireshark[5] that enable capturing the SOAP messages exchanged (lower part of figure 3). However, we do not check the strict conformance of SOAP messages to WS-* standards in our interoperability testing approach. Instead, SOAP messages are only analyzed for the existence of WS-* headers (for determining interoperability level 5) as well as for unexpected errors and premature terminations (for determining levels 8-11). Not checking conformance allows for the possibility of 'interoperable' communication that violates WS-* standards. However, from our experience, this is a purely theoretical limitation for heterogeneous environments.
Assuming the test setup described, sections 4.1 and 4.2 show how to derive and perform test cases for WS-* interoperability testing.

4.1 Derivation of Test Cases

The derivation of test cases for a selection of WS-* standards is a far from trivial task. This is due to the number of functionalities, intricate relationships between individual functionalities of the same standard and possible interferences between functionalities of WS-* standard combinations. In order to manage complexity, we propose to classify test cases into isolated test cases for capturing atomic functionalities, combined test cases for capturing the interactions between atomic functionalities and practicability test cases for capturing relevance for practice as described below.

Isolated Test Cases In order to provide a thorough coverage of the functionalities of a WS-* standard, the first step in deriving test cases should be the identification of the most atomic functionality of a standard that can be tested on a WS stack implementation in isolation. However, the concept of a functionality as described in a WS-* standard (and assertable via WS-Policy) and a functionality that can be tested in isolation is not the same. Considering WS-Security, a protection assertion such as signing or encrypting SOAP message parts cannot be tested without declaring assertions for a valid security binding (options for so-called Asymmetric-/Symmetric-/TransportBindings). However, the number of combinations of assertion options that must be selected for defining an executable test case is not tractable for semi-automatic processes. Considering the encryption of SOAP message parts, at least the definition of the '*encryption parts*' (3 options + number of named header elements, cf. [11, section 4.2.1]), a token type for '*initiator and recipient*' (54 options each, cf. [11, section 5.4]), an '*algorithm suite*' (23 options, cf. [11, section 6.1]) and a '*header layout configuration*' (4 options, cf. [11, section 6.7]) is necessary. This would lead to at least 14904 test cases if the same token type would be assumed to be used for initiator and recipient and several hundred thousand test cases if not. This basic calculation shows that the notion of 'atomic functionality' to be tested cannot be tied to the executability of the resulting test cases.

[5] `www.wireshark.org`

Instead, we propose to leverage the concept of a WS-Policy assertion for identifying the atomic features to be tested. For example, the number of different policy assertion options for a `X509 Token` is just 12 resulting in 12 test cases. However, for being able to execute these test cases, the `X509 Token` assertions have to be embedded within an initial executable WS-Policy configuration (that also contains algorithm suite, encryption parts definition etc.). As the configuration options of WS-Policy assertions we have tested are almost orthogonal to the options of other WS-Policy assertions, the 12 executable test cases for a `X509 Token` assertion then can be derived by permuting the `X509 Token` assertion options of the initial WS-Policy configuration only. Clearly, this approach implies giving up complete coverage of all possible configuration combinations (by tying the notion of a test case to executability) in favor of a manageable amount of test cases (by tying the notion of a test case to a WS-Policy assertion). Note that, in practice, it is not too hard to come up with an initial executable WS-Policy configuration because Web services solution providers frequently publish sample configurations for a variety of application scenarios[6].

Listing 1. Structure definition of `RMAssertion` (cf. [8])

```
<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>
```

For deriving the configuration options for a single policy assertion, we propose to make use of the assertion structure definitions that are published in the WS-Policy extension standards. Listing 1 shows the structure definition of the WS-ReliableMessaging Policy standard's `RMAssertion` assertion (note that usual regular expression operators are used to define structural constraints on the assertion). This assertion basically says that delivery semantics options `ExactlyOnce`, `AtLeastOnce` and `AtMostOnce` of WS-ReliableMessaging must be combined with either `InOrder` delivery or not.

For making use of this structure definition, we propose to first write down the policy assertion options in a configuration option matrix and then to derive the individual test cases from that matrix. This process starts with first noting the multiplicity of each WS-Policy assertion option in the matrix where the

[6] for example, see `https://wsit-docs.dev.java.net/releases/1.1/ahici.html`

available multiplicity options are derived from the policy standard's structure definition: '0/1' (none or exactly one occurrence), '1' (exactly one occurrence), '0*' (no occurrence or more), and '1*' (at least one occurrence). Then, for each structure definition constraint, expressing that two assertion options mutually exclude each other, must be combined or may be freely combined, the operators 'XOR','AND' or 'OR' are inserted into the matrix spanning the respective assertion options. Table 1 shows how an according configuration option matrix looks like for WS-ReliableMessaging Policy's `RMAssertion` feature. Table 2, in turn, shows the resulting test cases for the `RMAssertion` feature that are derived from table 1 by simply resolving the assertion options' combination operators and multiplicities.

#	Op1	Op2	Mul	Setting	Definition
1	-	-	1	Basic `RMAssertion`	[8, lines 132-134]
2	`XOR`	-	0/1	`SequenceSTR`	[8, lines 141-144] [8, lines 263-277]
			0/1	`SequenceTransportSecurity`	[8, lines 145-149] [8, lines 278-307]
3	`AND`	`XOR`	1	`ExactlyOnce`	[8, lines 166/167]
			1	`AtLeastOnce`	[8, lines 168/169]
			1	`AtMosttOnce`	[8, lines 170/171]
			0/1	`InOrder`	[8, lines 172/173]

Table 1. Configuration Matrix for WS-ReliableMessaging Policy's `RMAssertion`

#	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 2. Test cases derived for WS-ReliableMessaging Policy's `RMAssertion`

Combined Test Cases For the identification of test cases consisting of more than one functionality of a WS-* standard, structure definitions similar to the isolated test cases are not available. This means that the derivation of combined test cases must rely on the WS-* standard documentation and the test engineer's

ability to combine the isolated functionalities in a sensible way. In this process, two problems stand out:
First, the results of a test case identification process that relies on a standard specification given in prose and the ability of test engineers are ambiguous and not easily reproducible. Consequently, the test cases derived are likely to miss important combinations or to produce combinations of features that are not meant to work together. The problem is even worse when it comes to combining features of different WS-* standards where no documentation of the relationships between features may be available at all. However, alternative solutions require extra effort of the standardization organizations (cf. section 5).
Second, similar to the problem of defining executable test cases, the number of configuration options for the individual WS-* features may lead to a prohibitive amount of combined test cases. However, the solution of looking for sample configurations that can be used as starting point for permuting individual policy assertions is not applicable here because the actual subject of investigation are not the individual assertions but the interplay of assertions. At this point, complete coverage of the assertion combination possibilities cannot be the goal of testing. Instead, classes of interchangeable assertions must be defined, the most important representatives of these classes must be selected and then be combined with each other. Clearly, 'importance' must be operationalized for the purpose of testing. Considering WS-Security, the assertion classes *protection assertions*, *token assertions*, *security bindings*, *supporting tokens* and *global WS-Security options* can be identified where a tuple of representatives from the first three classes always leads to an executable test case. A test engineer then must apply some ranking algorithm for choosing between, e.g., testing all flavors of protection assertions (signing/encrypting header/body/elements/attachments) combined with a limited amount of token assertions (X509/Username/SecureConversation) and a single security binding or testing 'signing headers' only combined with all flavors of token assertions and security bindings.

Practicability Test Cases While isolated and combined test cases provide a reasonable way of achieving high coverage of WS-* standards, the application scenario that necessitates the use of WS-* standards is not respected very well. Note that, e.g., WS-Security is not designed to specify the implementation of well-known security features like privacy, integrity or authentication, but simply to specify how XML Signature and XML Encryption are to be applied in a SOAP messaging context. Therefore, test cases have to be derived that ensure application goals like privacy, integrity or authentication of message payloads. However, it is not easy to derive such test cases, in particular it is not easy to derive test cases that ensure security related goals in a distributed communication setting (see the attack on the Needham-Schroeder protocol as an example [6]; there are many others). It is the task of standardization organizations and industry communities to define and publish practicability test cases together with the corresponding WS-Policy definitions that capture the needs of 'real-world' application scenarios. This would enable straightforward interoperability test-

ing as well as validity of the application-specific WS-Policy definitions by public scrutiny.
Fortunately, for the case of WS-ReliableMessaging and WS-Security, the so-called *Secure WS-ReliableMessaging Scenario* is available as a test case of real-world relevance and complexity which has been proved independently by [1] and [3] to implement privacy, integrity and authentication. However, there is no straightforward way to identify test cases for investigating practicability without the help of standardization organizations.

4.2 Test Case Execution

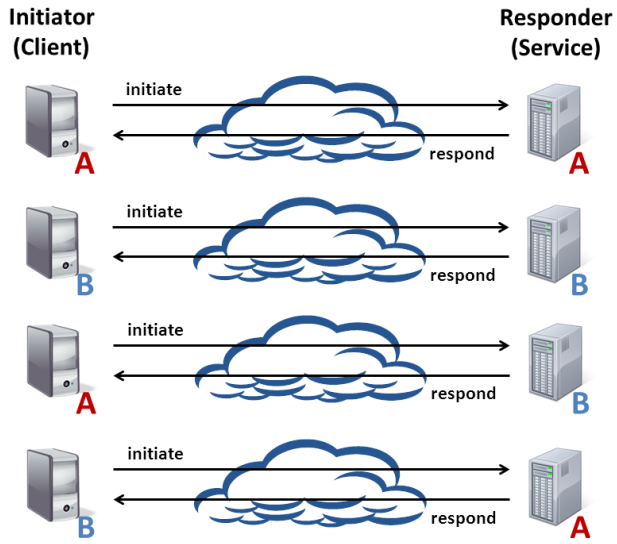

Fig. 4. Necessary Platform Configurations for Testing

For performing test cases, the different tasks that a Web service client and Web service have in implementing a WS-* feature have to be taken into account. For testing the interoperability of some WS stack implementations A and B, it makes a difference whether the client is developed using platform A and the service on platform B or the other way round. Moreover, tracing down the reasons for interoperability issues requires to know whether the respective features work in an homogeneous environment. Therefore, we propose to perform each test case in all the platform configurations displayed in figure 4. Clearly, if a test case cannot be performed in both homogeneous environments (A-A and B-B), then the heterogeneous environments are not likely to work. However, if only one of the homogeneous environments does not work, then the heterogeneous environments are worth testing. Our practical tests show (cf. [15]) that some features do not work in a homogeneous environment (A-A or B-B), but in a

heterogeneous one (B-A or A-B).
Having defined the relevant platform executions, we propose to apply the following process to isolated, combined and practicability test cases subsequently where interoperability issues detected in isolated or combined test cases reveal whether or not performing combined or practicability test cases is sensible:

1. Derive test cases for the WS-* standards under consideration as pointed out in section 4.1.
2. Develop test plan containing test cases to test all the functions determined in step 1.
3. Develop concrete implementations of the test cases in accordance with the test plan, consisting of a *Web service*, a *WS-Policy definition* assigned to this service, and a *Web service client* to invoke the service.
4. Execute the test cases:
   (a) Deploy the Web service on both platforms A and B.
   (b) Deploy the Web service client on both platforms A and B.
   (c) Invoke the Web service with the Web service client permuting the configurations of figure 4.
   (d) Document and store the test output.
5. Analyze test case documentation and test output and try to identify reasons for interoperability issues (if any).

5 Evaluation

As pointed out above, we have concentrated on WS-ReliableMessaging and WS-Security for evaluating our WS-* interoperability testing approach. Moreover, parts of the WS-SecureConversation [10] and WS-Trust [12] standards have been investigated for enabling the *Secure WS-ReliableMessaging Scenario*. As WS stack implementations, we have chosen two of the most reputable JAVA-based WS stacks, namely Oracle's (Sun's) *Metro* WS-stack that comes with the GlassFish Application Server[7] and Apache's *Axis2* WS-stack that is reused in IBM's WebSphere Application Server[8].
Due to space limitations, we concentrate on the most important results of our interoperability tests. Details are available in a technical report [15].
Looking at the number of isolated test cases that result from the procedure described in section 4.1, we have identified 170 test cases. This still seems to be a high number, but is actually pretty small compared to the number of test cases resulting from alternative procedures (cf. section 4.1). For the practical tests, 170 test cases proved to still be a manageable amount. Moreover, retrieving sample configurations from the web that employ the policy assertions under test was not possible for every single policy assertion. However, there was a sufficient amount of configurations that also allowed for deriving executable test cases that included the policy assertions under test.

[7] `http://glassfish.dev.java.net`
[8] `http://www-01.ibm.com/software/webservers/appserv/was/`

Looking at the interoperability results for Metro and Axis2, the results were disappointing. While 109 policy assertions were successfully tested (interoperability level 12, cf. section 3) in at least one of the homogeneous environments (A-A or B-B), only 47 test cases could successfully be performed in at least one of the heterogeneous environments (A-B or B-A). Most strikingly, no test case applying SOAP message encryption could successfully be executed for both heterogeneous environments. More interoperability issues for other basic functionalities resulted in a single combined test case, i.e., the combination of signed message headers with SSL-encryption. Consistently, the practicability test case (the Secure Reliable Messaging Scenario) was only performed in the two homogeneous environments and could be implemented almost as specified in [1,3].
These results clearly show that more thorough interoperability testing is needed to be done by WS-stack implementers. Interoperability projects like *Tango*[9] are helpful, but interoperability tests actually are necessary between all major WS-stack providers. Moreover, while the orthogonality of many WS-* features fosters flexibility, it hinders the efficient identification of interoperability test cases. Therefore, standardization organizations should complement WS-* standards with test plans that cover the most important/widely needed feature combinations. Also, those test plans should be linked to communication qualities like privacy, integrity or authentication so that application developers do not have to create the relationship between those qualities and implementation primitives like header/body signatures/encryptions themselves. On this account, the WS-Policy definitions shipped within the so-called *delivery package* of WS-I's[10] Reliable Secure Profile 1.0 [26] are a step into the right direction. However, for straightforward interoperability testing more than just three WS-Policy definitions are needed.

6 Related Work

In practice, the WS-I organization is dedicated to foster interoperability of Web services by creating so-called *profiles* which provide rules for creating and processing WSDL files and SOAP messages. In particular, the Basic Security Profile 1.1 [25] and the Reliable Secure Profile 1.0 [26] cover the use of security and reliability related WS-* standards. That work is different from ours in several ways. Most notably, the rules of these two WS-I profiles are tied to SOAP messages and SOAP message elements. Conversely, the test cases of our approach are tied to WS-* functionalities that are assertable via WS-Policy. WS-Policy assertions, however, may translate into several SOAP message elements or even several SOAP messages. Consistently, the WS-I profiles do not contain WS-Policy assertions for having a Web services stack produce SOAP message that may or may not comply with the WS-I rules (except for just three WS-Policy definitions in the 'Reliable Secure Profile 1.0 Test Scenarios' document). Without those assertions, testing interoperability between two WS-* implementations is far from

[9] `http://java.sun.com/developer/technicalArticles/glassfish/ProjectTango/`
[10] `http://ws-i.org/`

straightforward because it is not clear how a test engineer would make the corresponding WS-* implementations produce the relevant SOAP messages. As the WS-I rules are not tied to WS-Policy assertable functionalities, it is also not clear how a sensible coverage of WS-* standards in interoperability testing can be achieved. Finally, application goals like authentication, integrity or confidentiality are not translated into detailed message protocols in a way comparable to [1] or [3]. Put short, the WS-I resources constitute clarifications on how to process WSDL files and SOAP messages but fall short on providing an approach for deriving easy-to-use test cases of increasing complexity for interoperability testing. In so far, the WS-I resources are very valuable in finding out *why* two WS-* implementations are not interoperable but they do not streamline the process of detecting *that* two WS-* implementations are not interoperable.
Apart from WS-I, interoperability or compatibility of Web services is discussed in various ways in academia. An abundance of work concentrates on analyzing the compatibility of Web services based workflows or business processes and tries to verify/test protocol properties like deadlock-freeness or termination. Other researchers are investigating interoperability of the actual Web services stack implementations (without considering QoS) [16].
In contrast, the work presented here concentrates on interoperability testing of WS-* implementations. In the area of testing Web services/SOAs with regard to the WS-* based notion of QoS, the amount of scientific work is limited. [4] and [5] discuss challenges of SOA testing, but the QoS aspects concentrate on quantifiable factors, such as latency. Although both publications also consider building trust between service and client to be a "challenge", they do not refer to WS-Trust or WS-Security. [2] proposes a QoS Test-Bed Generator for Web Services, but only defines latency and reliability as QoS features. Although [17] provides an evaluation of the WS-Security implementation of the Axis2 WS stack, 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 as advanced communication qualities (security, reliability etc.) provided by WS-* standards and tests the compatibility of these features using different implementations.

7 Conclusion and Future Work

We have presented an approach for interoperability testing of WS-* standards that leverages the structure definitions of WS-Policy assertions for deriving a manageable amount of test cases. Our test results for two major WS stacks show that more effort needs to be put into interoperability testing by WS stack providers. Further, standardization organizations are called upon defining frequent application scenarios together with detailed WS-Policy definitions for fostering interoperability testing and ensuring relevance for practice.
Future work includes a refined approach for deriving combined test cases that takes the application scenario of WS-* usage into account. Moreover, a framework that enhances automation of WS-* interoperability testing is needed.

References

1. Backes, M., Moedersheim, S., Pfitzmann, B., Vigano, L.: Symbolic and cryptographic analysis of the secure WS-ReliableMessaging scenario. In: FOSSACS 2006. LNCS, vol. 3921, pp. 428–445. Springer
2. Bertolino, A., Angelis, G.D., Polini, A.: A QoS Test-Bed Generator for Web Services. In: ICWE 2007, Como, Italy, July 2007
3. Bhargavan, K., Corin, R., Fournet, C., Gordon, A.D.: Secure sessions for web services. ACM Trans. Inf. Syst. Secur. 10(2), 8 (2007)
4. Canfora, G., Penta, M.D.: Testing Services and Service-Centric Systems: Challenges and Opportunities. IEEE IT Pro (2), 10–17 (March/April 2006)
5. Gerardo Canfora and Massimiliano Di Penta: Service-Oriented Architecture Testing: A Survey. In: ISSSE 2006 - 2008, Salerno, Italy
6. Lowe, G.: An attack on the needham-schroeder public-key authentication protocol. Inf. Process. Lett. 56(3), 131–133 (1995)
7. OASIS: Web Services Security: SOAP Message Security 1.1 (WS-Security 2004). OASIS (February 2006)
8. OASIS: Web Services Reliable Messaging Policy Assertion (WS-RM Policy) Version 1.2. OASIS (February 2009)
9. OASIS: Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.2. OASIS (February 2009)
10. OASIS: WS-SecureConversation 1.4. OASIS (February 2009)
11. OASIS: WS-SecurityPolicy 1.3. OASIS (February 2009)
12. OASIS: WS-Trust 1.4. OASIS (February 2009)
13. RosettaNet: Multiple Messaging Services (MMS) Profile for Web Services (WS) V11.00.01. RosettaNet (August 2009)
14. RosettaNet: Message Control and Choreography (MCC) - Profile-Web Services (WS), Release 11.00.00A. RosettaNet (June 2010)
15. Schwalb, J., Schönberger, A.: Analyzing the Interoperability of WS-Security and WS-ReliableMessaging Implementations. Tech. Report: Bamberger Beiträge zur Wirtschaftsinf./Angewandten Inf. 87, Universität Bamberg (09 2010)
16. Shetty, S., Vadivel, S.: 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)
17. Shopov, M., Kakanakov, N.: Evaluation of a single WS-Security implementation. In: Proceedings International Conference on Automatics and Informatics, Sofia, Bulgaria. pp. 39–42 (October 2007)
18. W3C: Web Services Description Language (WSDL) 1.1. W3C (March 2001)
19. W3C: XML Encryption Syntax and Processing. W3C (December 2002)
20. W3C: Web Services Addressing 1.0 - Core. W3C (May 2006)
21. W3C: SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). W3C (April 2007)
22. W3C: Web Services Policy 1.5 - Framework. W3C (September 2007)
23. W3C: XML Signature Syntax and Processing (Second Edition). W3C (June 2008)
24. Wegner, P.: Interoperability. ACM Comput. Surv. 28(1), 285–287 (1996)
25. WS-I: Basic Security Profile Version 1.1. WS-I (January 2010)
26. WS-I: Reliable Secure Profile Version 1.0. WS-I (November 2010)
27. Zhu, H., Hall, P.A.V., May, J.H.R.: Software Unit Test Coverage and Adequacy. ACM Computing Surveys (CSUR) 29(4), pp. 366–427 (December 1997)