WS-FM 2004 Preliminary Version

Specifications using XQuery Expressions on Traces

Marcus Venzke [1]

Telematics Department
Technische Universität Hamburg-Harburg
Hamburg, Germany

Abstract

This paper contributes to the interoperability of web services by proposing the flexible specification technique SXQT in conjunction with the automatic validation, a straightforward approach for detecting non-conformance. SXQT allows different levels of abstraction by specifying individual requirements on a web service's protocol, i.e. its SOAP messages and behaviour including SOAP modules. The automatic validation detects non-conformance by comparing occurring message sequences with the protocol's specification.

Key words: Automatic Validation, Choreography, Specification Technique, SXQT, Web Services, XQuery

1 Introduction

Specifications play a critical role for the interoperability of web service applications. They describe the protocol between a web service provider and its clients (web clients). The independent implementation in different enterprises requires specifications to be precise to allow hazard free interworking, but still leaving room for a broad range of implementations.

The common Web Service Description Language (WSDL) [16] only allows sparse specifications. It is restricted describing type and structure of exchanged messages. Expected protocol behaviour, such as required orderings of messages, cannot be specified, but is crucial for successful interworking.

Protocol behaviour can be specified in BPML [2] and BPEL4WS [20]. These specify web services in great detail, allowing specifications to be executed. In that sense they are more programming than specification languages. Implementers are restricted to compile specifications into implementations, not having room for design decisions (cp. [17]).

[1] Email: `venzke@tu-harburg.de`

BPML and BPEL4WS as well as WSCL [3] and WSCI [1] do not allow specifying SOAP modules [12]. SOAP modules are general purpose, infrastructure related protocols, whose data units are conveyed in the SOAP header. Examples are the WS-* protocols [18]. For developing a web service's protocol, application specific parts (conveyed in SOAP body) are composed with required SOAP modules [5]. With the specification techniques stated above these cannot be specified, because their models are based on the level of WSDL's port types, abstract interfaces that are independent of SOAP. Thus there is no notion of the SOAP header and its content cannot be specified.

This paper proposes the specification technique SXQT that allows specifying application specific protocols, SOAP modules and relations in between. Different levels of abstraction can be chosen for only specifying those requirements needed for interworking between a web service provider and its clients.

Having precise specifications there is still the risk of incompatibility due to non-conforming implementations. The issue is frequently addressed by testing of implementations. Unfortunately testing can only ensure conformance for a limited number of test cases. Furthermore if having misunderstood the specification, developers will not find resulting non-conformance. Program verification goes beyond by formally proving the implementation's conformance. But finding such proves for non-trivial implementations is extremely elaborate.

As contribution to this issue the paper proposes the automatic validation, which allows detecting non-conformance of occurring message sequences. When used for testing, non-conformance is detected independent of the developer's understanding. Permanently used in production it is a straightforward approach to detect every occurring violation. SXQT is developed to facilitate the automatic validation.

2 SXQT

SXQT (Specifications using XQuery expressions on Traces) is based on predicates that constrain the structure of and values in XML documents. Every requirement on structure and values is expressed individually as a predicate that returns true, if the requirement is fulfilled. To specify a protocol's behaviour, the temporal ordering of SOAP messages is transformed into an XML data structure called trace. The idea is similar to Hoare's specifications of simple event sequences in [13].

2.1 Traces

The transformation of temporal ordering into the trace is based on observing messages sent or received by a web service provider or client. Such events are considered to be atomic, without duration and cannot occur simultaneously to other events. Absolute time is not considered.

A conceptual observer records events in the order of their observation until some point in time into the trace, as illustrated in figure 1. It starts with an empty trace, an XML document containing only a root element. For every event the SOAP message and additional observable meta-data is recorded. It is appended as child element to the trace's end. The resulting data structure is exemplified in figure 2, depicting a trace with four events.

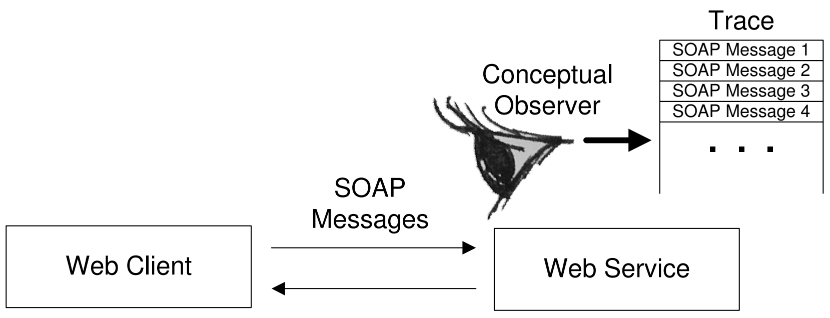

Fig. 1. Observer recording trace

```
<tra:Trace
    xmlns:tra="http://ti5.tu-harburg.de/venzke/2002101/traces"
    xmlns:env="..." xmlns:wsat="...">
  <tra:Message to="Service" operation="1">
    <env:Envelope>
      <env:Header> ... </env:Header>
      <env:Body>
        <wsat:Prepare />
      </env:Body>
    </env:Envelope>
  </tra:Message>

  <tra:Message to="Client" operation="1">
    <env:Envelope> ... </env:Envelope>
  </tra:Message>

  <tra:Message to="Service" operation="2"> ... </tra:Message>
  <tra:Message to="Client"  operation="2"> ... </tra:Message>
</tra:Trace>
```

Fig. 2. Sample trace

Making the trace an XML document is an obvious choice and allows specifying temporal ordering and data structuring with the same techniques. The hierarchical nature of XML enables embedding the SOAP message's hierarchies into the single hierarchy of the trace. No additional formalism is needed to represent temporal ordering. Moreover representing temporal ordering and message structuring in the same way enables their specification with the same techniques. SXQT uses predicates for this purpose.

2.2 SXQT Expressions

SXQT's predicates are called SXQT expressions. An SXQT expression maps a trace to true or false. It needs to be designed to return true, if the requirement it verbalises is fulfilled by the trace. It is expressed in the XML Query Language (XQuery) [6], which provides its syntax and the language's semantics.

XQuery is well suited for specifications in the web service field. A formal semantics is provided in [10]. Being in the standardisation at the World Wide Web Consortium (W3C) and a successor of SQL for XML databases, future developers in the web service field will know it. Tools will be available. It provides means to handle XML structured data and express predicates. User-defined functions allow abstracting and reusing partial expressions. Finally a standard library [14] contains many standard functions useful for SXQT expressions.

To facilitate the construction of predicates SXQT provides an additional library. Its functions simplify the construction of SXQT expressions, hiding the structure of the trace and SOAP messages. This also increases readability.

2.3 A sample SXQT expression

As illustration the following will discuss an example SXQT expression [2]. It verbalises a requirement on message ordering of the two-phase commit protocol used in WS-Transaction [7] to achieve atomicity. The requirement states that if a Rollback message is observed, a Commit message must not have been observed before for the same transaction.

```
every $m
in opr:restrict(opr:tr(), xs:QName("wsat:Rollback"))
satisfies
  empty(opr:restrict(myfn:EventsInTxBefore($m),
                     xs:QName("wsat:Commit")))
```

Fig. 3. Sample SXQT expression

As most SXQT expressions, the example in figure 3 uses XQuery's universal quantifier (`every - in - satisfies`) to ensure that a predicate holds true for a specific type of messages in the trace. In the example it is the Rollback messages for which the predicate must hold true, that there is no Commit message earlier in the trace that belongs to the same transaction.

After the keyword `in`, the sequence of all Rollback messages is determined. All messages in the trace are accessed with SXQT's function `opr:tr`. These are restricted to the Rollback messages using the function `opr:restrict`.

For every Rollback message the predicate after the keyword `satisfies` is evaluated with the variable $m referencing it. It is given to the func-

[2] More examples are given in [22].

tion `myfn:EventInTxBefore`, which determines earlier messages in the same transaction. These are restricted to Commit messages. Since the requirement declines Commit messages prior to the Rollback message, no message should be returned. This is checked with XQuery's standard function `empty`.

The function `myfn:EventInTxBefore` is specially defined for specifying WS-Transaction. By abstracting the partial expression to determine earlier messages in the same transaction, it greatly increases readability and allows its reuse in several SXQT expressions. Similar functions can be defined for other specifications to determine all messages of a conversation between a web service and one web client to facilitate specifying the conversation's dynamics.

The function `myfn:EventInTxBefore` relates a SOAP module with the protocol's application specific part. Messages of WS-Transaction are required to carry a transaction identifier in the SOAP header. It is used by the function to determine messages belonging to the same transaction. Relating to SOAP modules gets possible by regarding SOAP messages as a whole. A model based on WSDL's port types, as used for BPEL4WS, BPML, WSCL, and WSCI, would not have allowed this.

2.4 Individual Requirements

Expressing requirements individually is a major strength of SXQT. It allows specifying only those requirements needed for interoperation. Different parties can express their requirements independently, potentially covering only a single SOAP module or the application specific part of protocol. SXQT expressions can be combined with WSDL, describing requirements not expressed by it.

Specifications based on automata do not allow expressing requirements individually. A single, closed model (the automata) has to be constructed that fulfils all requirements. While easing the implementation this obstructs reasoning about individual requirements. Furthermore adding or suspending a requirement entails reconstructing the entire model. By expressing requirements individually SXQT allows focusing on these and makes adding and suspending of requirements trivial.

Specifying individual requirements also allows different levels of abstraction. Only requirements needed for a specific purpose (e.g. interoperation) have to be expressed. Protocol development can start with major requirements. After getting a better understanding, additional requirements can be added.

A specification's SXQT expressions may originate from different parties. In this manner each party can specify its requirements on a protocol independently. Standard requirements are formalised once and added to many specifications. An example is the requirement of WS-Coordination [8] that messages need to contain a coordination context in their SOAP header. This is required for all protocols using WS-Coordination such as WS-Transaction.

The requirement may be formalised only once and then added to the specifications of all these protocols.

In this way SXQT can be used to specify SOAP modules independent of a specific web service's protocol. The SOAP module's requirements are described as SXQT expressions. Requirements can include assumptions on other SOAP modules or the SOAP body. When specifying a web service's protocol all SXQT expressions of required SOAP modules are added to its specification.

SXQT expressions can be combined with WDSL to avoid duplicate specification. WSDL cannot be replaced by SXQT, because it is a de facto standard and a lot of tools read WSDL, e.g. proxy generators. In addition it allows specifying some requirements more concisely, such as the structure and types of SOAP messages. Thus only those requirements not expressed in WSDL should be expressed as SXQT expressions. These should be added to a WSDL document using extensibility elements, extensions allowed for WSDL [3] . Extended WSDL documents, describing a web service's protocol including behaviour, are given to developers as reference for implementing the web service provider and interoperable clients.

A risk of incompatibility remains, if implementations do not conform to the specification. The issue is independent of the specification technique used. The paper contributes to the issue by proposing the automatic validation, which makes detecting non-conformance straightforward.

3 Automatic Validation

The automatic validation is the process of checking, if occurring message sequences conform to the protocol specification. The key idea is to directly compare observed message sequences to the specification. Non-conformance can then be signalled to developers, administrators, users, or the implementation. This has similarities to program checkers for side-effect free programs [4] [4] . SXQT is developed to be well suited for this purpose.

Automatic validation supports developers creating conforming implementations. It can be applied from first prototypes, even if specifications are in early state and contain only major requirements. While testing, it frees developers from deciding, if implementations behave as specified. Even if non-conformance is originated from developers having misunderstood the specification, it will be detected. Being permanently used in production environments developers are informed about any violation or it is known that none ever occurred.

Deviations occurring in production environments after a long period of conformance are a hint to developers that the protocol might have changed

[3] See [9], section 6.

[4] We have compared this in [21] and [22].

and web clients need to be updated. The autonomous evolution of web service providers and clients in different enterprises can lead to changes of the web service's protocol without notification of its users. A web client only needs to be updated, if it can observe a violation against the specification it was implemented for. Thus validating against this specification allows detecting protocol changes that require updates.

The automatic validation also allows protecting a web service provider against non-conforming, compromising SOAP messages. Such messages may arise form implementation errors or malicious users, who try to intrude the web service provider. This may compromise the provider if not all necessary checks are implemented. Using the automatic validation, the checks can be performed more reliably. A message is then validated before being delivered to the web service provider. In case of non-conformance it is not delivered, but answered to the web client with an error message.

3.1 Validator

The entity that performs the automatic validation is called validator. As depicted in figure 4 it consists of an observer and a core validator. The observer is capable of observing SOAP messages sent or received by a web service provider or client. It can be seen as concrete implementation of SXQT's conceptual observer. Thus it records SOAP messages in the order of their observation into the trace (cp. section 2.1). The core validator checks, if the message sequence represented by the trace conforms to the specification.

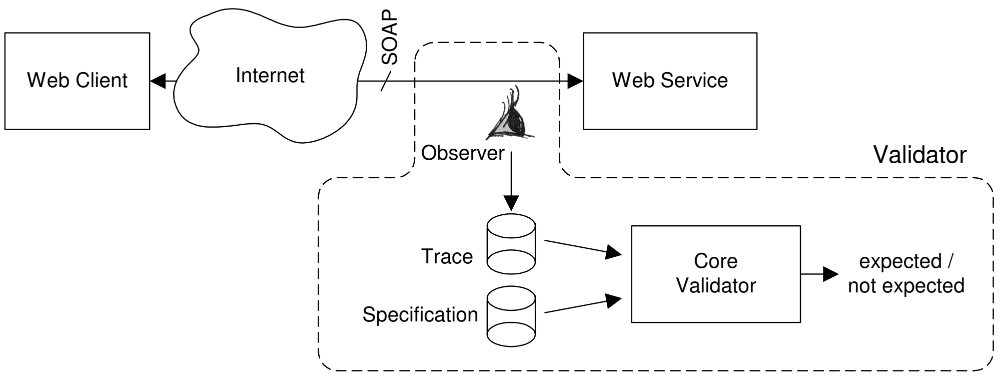

Fig. 4. Basic structure of validator

As discussed in [22] validators differ in where and how they observe messages and when the validation is performed. The observation can take place on the side of the web service or client. Observers can be implemented in separate processes as HTTP proxies [11] or in the process of the web service provider or client respectively. When observing a message it may be validated immediately or only recorded into the trace for later validation. The latter approach has less impact on performance but is not applicable to all usage scenarios.

3.2 Validation against SXQT Specifications

SXQT makes the automatic validation straightforward and implementable using standard software in substantial parts. Validation against SXQT expressions can be performed using standard XQuery processors, validation against XML Schema definitions using standard schema validators.

The straightforwardness of validation against SXQT expressions results from SXQT's model and the choice for XQuery. To validate, that an observed trace conforms to an SXQT expression, the expression just need to be evaluated with the trace as parameter. For conformance this must return true. Since SXQT expressions are XQuery expressions the evaluation can be performed using standard XQuery processors available from different vendors (see [15]). This facilitates the validator's implementation.

For extended WSDL documents the validator must also consider requirements expressed in WSDL. This includes definitions in XML Schema [19] used for defining data structures of SOAP messages. Validation against the latter can be performed using schema validators also available from different vendors (see [19]).

4 Conclusion

The automatic validation and SXQT are a contribution to the web service interoperability. The specification technique SXQT allows specifying a protocol's SOAP messages and required behaviour at different levels of abstraction. Requirements are verbalised individually as predicates called SXQT expressions. These may originate from different parties expressing their requirements including once formalised standard requirements. Behaviour is described by transforming temporal ordering into an XML data structure, allowing the same technique (predicates) to specify message structures and temporal ordering. Considering entire SOAP messages enables to specify SOAP modules including assumptions on other SOAP modules and the SOAP body. The choice for XQuery makes SXQT easy to learn for developers in the web service field.

SXQT is well suited for the automatic validation, a straightforward approach for checking, if occurring message sequences conform the specification. It supports developers to create conforming implementations, by detecting non-conformance in test environments for early prototypes or final implementations or in production environments.

References

[1] Arkin, Assaf, et. al., *Web Service Choreography Interface (WSCI) 1.0.* W3C Note. World Wide Web Consortium (W3C), August 2002. http://www.w3.org/TR/2002/NOTE-wsci-20020808/ (Accessed: 8 Dec 03)

[2] Arkin, Assaf, *Business Process Modelling Language*. Draft in last call. The Business Process Management Initiative (BPMI.org), 2002.
http://www.bpmi.org/bpml-spec.esp (Accessed: 13 Feb 03)

[3] Banerji, Arindam, et. al., *Web Services Conversation Language (WSCL) 1.0*. W3C Note. World Wide Web Consortium (W3C), March 2002.
http://www.w3.org/TR/2002/NOTE-wscl10-20020314/ (Accessed: 8 Dec 03)

[4] Blum, Manuel and Kannan, Sampath, *Designing programs that check their work*. In: Journal of the ACM, Vol. 42, Nr. 1, S. 269-291, Januar 1995.
ftp://ftp.cis.upenn.edu/pub/kannan/jacm.ps.gz (Accessed: 21 Dec 02)

[5] Box, Don, *Understanding GXA*. Microsoft Corporation, July 2002.
http://msdn.microsoft.com/library/en-us/dngxa/html/understandgxa.asp (Accessed: 15 Dec 03)

[6] Boag, Scott, et. al., *XQuery 1.0: An XML Query Language*. W3C Working Draft. World Wide Web Consortium (W3C), November 2003.
http://www.w3.org/TR/2003/WD-xquery-20031112/ (Accessed: 11 Dec 03)

[7] Cabrera, Felipe, et. al., *Web Service Transaction (WS-Transaction)*. BEA Systems, International Business Machines Corporation, Microsoft Corporation, August 2002.
http://www-106.ibm.com/developerworks/library/ws-transpec/
(Accessed: 6 Feb 04)

[8] Cabrera, Felipe, et. al., *Web Services Coordination (WS-Coordination)*. BEA Systems, International Business Machines Corporation, Microsoft Corporation, September 2003.
http://www-106.ibm.com/developerworks/library/ws-coor/
(Accessed: 6 Feb 04)

[9] Chinnici, Roberto, et. al., *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. W3C Working Draft. World Wide Web Consortium (W3C), November 2003.
http://www.w3.org/TR/wsdl20/ (Accessed: 6 Feb 04)

[10] Draper, Denise, et. al., *XQuery 1.0 and XPath 2.0 Formal Semantics*. W3C Working Draft. World Wide Web Consortium (W3C), November 2003.
http://www.w3.org/TR/2003/WD-xquery-semantics-20031112/
(Accessed: 11 Dec 03)

[11] Fielding. R., et. al., *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616. Internet Engineering Task Force (IETF), Juni 1999.
http://www.ietf.org/rfc/rfc2616.txt (Accessed: 23 Jul 03)

[12] Gudgin, Martin, et. al., *SOAP Version 1.2 Part 1: Messaging Framework*. W3C Recommendation. World Wide Web Consortium (W3C), June 2003.
http://www.w3.org/TR/2003/REC-soap12-part1-20030624/
(Accessed: 18 Jul 03)

[13] Hoare, C.A.R, *Communicating Sequential Processes.* Prentice-Hall, London, 1985.

[14] Malhotra, Ashok, et. al., *XQuery 1.0 and XPath 2.0 Functions and Operators.* W3C Working Draft. World Wide Web Consortium (W3C), November 2003.
http://www.w3.org/TR/2003/WD-xpath-functions-20031112/
(Accessed: 11 Dec 03)

[15] Marchiori, Massimo, *XML Query (XQuery).* Website. World Wide Web Consortium (W3C), December 2003.
http://www.w3.org/XML/Query (Accessed: 12 Dec 03)

[16] Marsh, Jonathan, Le Hgaret, Philippe, *Web Services Description Working Group.* Website. World Wide Web Consortium (W3C), December 2003.
http://www.w3.org/2002/ws/desc/ (Accessed: 12 Dec 03)

[17] Meredith, L.G., Bjorg, Steve, *Contracts and Types.* In: Communications of the ACM, Vol. 46, No. 10, October 2003.

[18] Microsoft, *Web Services Specifications Index Page.* Website. Microsoft Corporation, Redmond, 2003.
http://msdn.microsoft.com/library/en-us/dnglobspec/html/
wsspecsover.asp (Accessed: 10 Dec 03)

[19] Sperberg-McQueen, C. M., Thompson, Henry, *XML Schema.* Website. World Wide Web Consortium (W3C), November 2003.
http://www.w3.org/XML/Schema (Accessed: 12 Dec 03)

[20] Thatte, Satish, et. al., *Business Process Execution Language for Web Services Version 1.1.* BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems, 2003.
http://www-106.ibm.com/developerworks/library/ws-bpel/
(Accessed: 30 Jan 04)

[21] Venzke, Marcus, *Automatic Validation of Web Services.* In: Electronic proceedings of the 8th CaberNet Radicals Workshop. October 2003.
http://www.newcastle.research.ec.org/cabernet/workshops/radicals/
2003/papers/Venzke_Cabernet03_030731.pdf (Accessed: 6 Feb 04)

[22] Venzke, Marcus, *Spezifikation von interoperablen Webservices mit XQuery.* Doctoral thesis. Technische Universität Hamburg-Harburg, 2003.
http://doku.b.tu-harburg.de/volltexte/2004/51/ (Accessed: 6 Feb 04)