Automatic Validation of Web Services

Marcus Venzke

Telematics Department,

Technische Universität Hamburg-Harburg, Germany

http://www.ti5.tu-harburg.de/staff/venzke/

(PhD thesis submitted: 14th July 2003)

Abstract — *This paper proposes the concept of "automatic validation" for web services. It allows to check if occurring message flows conform to the interface's specification. A validator reads an SXQT specification, observes and records exchanged messages and checks their conformance to the specification. This allows detecting non-conforming message-flows, before compromising a system's dependability.*

1 Introduction

Web services allow interoperation of loosely coupled components using web technology. Instead of distributing a component's code, its services are provided over networks such as the Internet via software interfaces based on XML messages. Components can be implemented by different parties based on a variety of architectures, languages and platforms.

The autonomy of component development puts risk on the dependability of assembled systems. Developers providing or using web services in different enterprises need to agree on interface specifications. These specifications have to be obeyed. Violations need to be detected, before compromising the system's dependability, even if they occur after a period of conformance due to the evolution of components.

A common approach for ensuring conformance to a specification is verification. It requires a formal model of a component's implementation to allow a mathematical proof of conformance to the specification, which is a second formal model of the component. Besides requiring two formal models the effort for creating proofs for complex components is frequently an issue.

This paper proposes an alternative approach for ensuring conformance to the specification of interfaces of web services: automatic validation. The key idea is to directly compare observed message flows exchanged via an interface with its specification. This task requires only the specification as formal model and can be applied automatically. It is facilitated for web services due to the explicit nature of XML messages exchanged over their interfaces.

2 Web Services

Web services expose their interfaces with web technology. Combining the standard encoding XML [Bray 00] for messages with the well-understood transport protocol HTTP [Fielding 99] into the protocol SOAP [Gudgin 03a, Gudgin03b] they stand for simplicity and interoperability between different platforms.

Web services can be regarded as web applications accessible by programmatic clients (web clients). Traditional web applications have proven their abilities to allow application access in an interoperable manner via the Internet and enterprise-wide intranets. However access is restricted to humans using web browsers, mainly due to the use of HTML combining data with formatting information. Web services resolve this issue by replacing HTML with XML for request and response messages transmitted over HTTP. The protocol SOAP adds conventions for the message exchange and the structuring of messages (SOAP messages). This mode of interaction is illustrated in Figure 1.

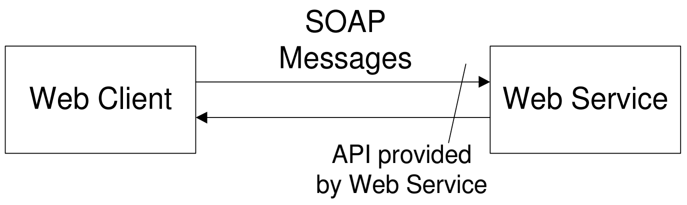

Figure 1: Communication between web client and web service

The Web Services Description Language (WSDL) [Chinnici 03, Gudgin 03c, Moreau 03] is used to describe software interfaces realised in this manner. It describes interfaces in terms of exchanged messages. Interfaces are defined as sets of operations, which mainly consist of two message types for the operation's request and response messages. XML fragments in these messages are declared in XML-Schema [Thompson 01, Brion 01], the common type system for XML.

Using web services for providing interfaces over the web leads to interoperable solutions and does not require complex infrastructures. Implementations can be created using widely available infrastructures for web applications plus support for XML. More specialized infrastructures are also available, generally for common component architectures (e.g. J2EE [Sun 03], .NET [Microsoft 03]). Interoperability is permitted due to the use of web technology and WSDL's clear message exchange semantics for describing interfaces. The consensus for web services in industry and research leads to a wide range of tools and applications for different platforms.

3 Automatic Validation

For ensuring that messages exchanged via a web service's interface conform to the specification, this paper proposes automatic validation. It is based on the idea of program checkers for side-effect free programs. Program checkers verify if a program returns the expected result. This is enhanced for web services, where a validator observes and records messages and compares them with a specification.

Program checkers are described in [Blum 95]. A program checker C is a program that is started after a program P, whose results are to be checked. C gets the parameters given to and the result returned by P. In many cases checking an available result is much simpler and less error-prone than calculating it form the parameters. If the result is what C expects, it is confirmed by C and returned as final result. Otherwise no final result is given and an error is reported. This allows the creation of partial correct functions without verification. [Gaul 99] and [Bartsch 01] show how this can be applied for the creation of correct compiler back-ends or expert systems respectively.

The automatic validation for web services differs form program checkers in what needs to be checked and how it is realized. Checking a web service's interface requires more than checking a single result. It requires checking if the flow of several SOAP messages conforms to the specification. This includes checking the order of SOAP messages as well as of contained values. In contrast to program checkers the validator is not implemented for a specific interface to be checked, but is general-purpose and reads the specification to check against.

As depicted in Figure 2 the validator consists of an observer and a core validator. The observer is capable of observing SOAP messages exchanged by a web client's or web service's implementation. SOAP messages are recorded in the order of their observation into the so-called trace. The core validator uses the trace to check if the observed sequence of SOAP messages conforms to the specification. Non-conformance is then signalled to users, administrators or the application.

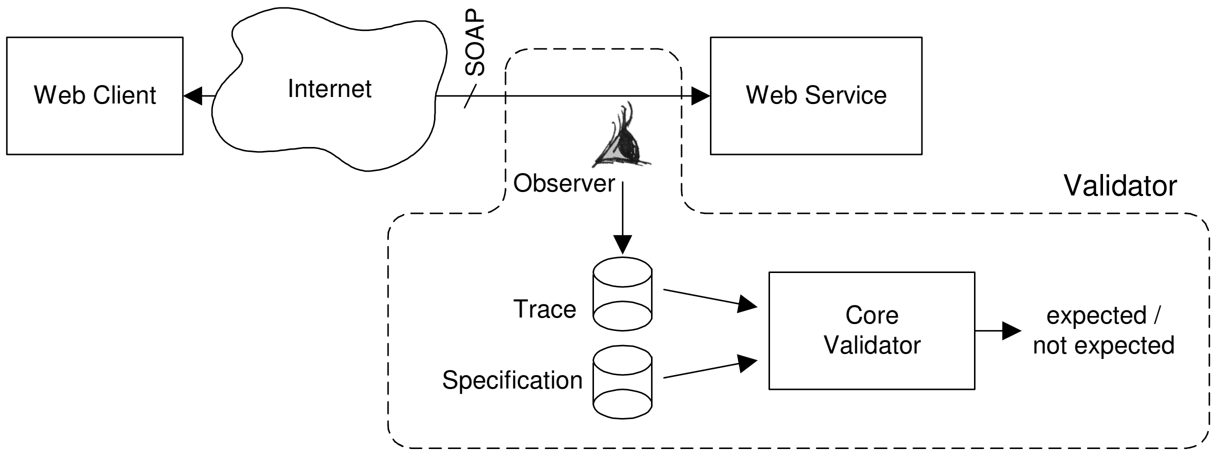

Figure 2: Basic structure of validator

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 client or web service. This leads to different sequences of observed messages to be validated. Observers can be implemented in separate processes as HTTP proxies [Fielding 99] or in the process of web client or web service respectively. The validation can be performed automatically whenever a message is observed or can be started manually after recording the whole trace.

The use of WSDL as the only specification would lead to weak checks. Only requirements stated in the specification can be checked. WSDL allows describing which message types belong to an operation. This includes the description of the structure and types of simple values in SOAP messages. However requirements on the ordering of operation calls or on relations between several values in one or several SOAP messages cannot be specified. To be able to express and validate such requirements the specification technique SXQT has been developed.

4 Specification Technique SXQT

The abbreviation SXQT stands for "Specifications using XQuery expressions on Traces". Based on a specification technique of C.A.R Hoare in [Hoare 85], with major enhancements for specifying interfaces of web services, SXQT uses predicates of first order logic to constrain which message flows are allowed.

The model behind SXQT considers an observation of a SOAP message as an atomic event. The SOAP message and additional observable meta-data are parameters of the event. Analogous to the validator's observer a conceptual observer records events in the order of their observation until some point in time into the so-called trace, as illustrated in Figure 3. A trace thus formalises an observed sequence of SOAP messages. An understanding of absolute time and simultaneous events is excluded.

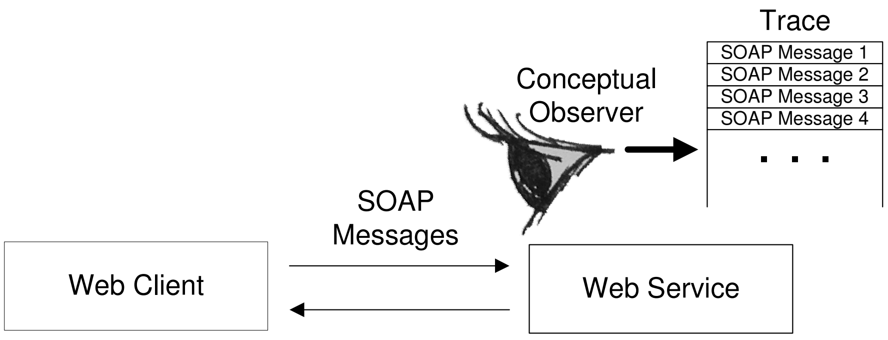

Figure 3: Observer recording a trace

Every requirement on an interface is expressed as a predicate (SXQT expression) that constrains observable traces. An SXQT expression has a trace as parameter and returns the logical value true for expected traces and false if a trace contradicts with the requirement. SXQT expressions are formulated in the standardized language XQuery [Boag 02] and added to the WSDL description of the web service's interface.

A specification in SXQT is well suited for the automatic validation. To check if an observed trace conforms to an SXQT expression, the SXQT expression just needs to be evaluated with the trace as parameter using a common XQuery processor. The evaluation must return true for all SXQT expressions of a specification. In addition the validator checks conformance to the description in WSDL and the SOAP standard.

5 Applications of Automatic Validation

The automatic validation has different applications that increase a system's dependability. It gives hints to developers, who may find the reason as being errors in implementations of web services or web clients or changes of interfaces. It can also be used to protect web services against non-conforming, compromising SOAP messages.

To get hints for debugging developers use a validator in the component's test phase or its production environment. In the test phase only a limited number of test cases can be analysed. In the production environment the automatic validation allows to detect all deviations to the specification that occur in production use.

Deviations that occur in production environments after a long period of conformance can be a hint that an interface has changed. The autonomous evolution of components in different enterprises can lead to changes of web service's interfaces without its users being informed. Such changes are relevant if the evolved component violates against the specification used as reference for the implementation of its web clients. By validating against this specification such violations can be detected.

The automatic validation can protect a web service by rejecting non-conforming SOAP messages. Such messages might arise by implementation errors or malicious users, who try to intrude the web service's implementation. Messages can compromise the implementation if not all necessary checks have been implemented. The automatic validation can automate such checks. Non-conforming messages are not delivered to the web service's implementation but rejected and answered to the web client with an error message.

6 Conclusion

The paper has introduced to automatic validation, which checks, if message flows exchanged over a web service's interface conform to the specification. It can be performed with a general-purpose validator that reads the specification, observes and records messages and checks their conformance to the specification.

Using a validator non-conformance can be detected, before putting risk on the system's dependability. While testing or in production environments it warns developers immediately, e.g. in case of implementation errors or relevant changes of interfaces. In addition a web service can be protected reliably against non-conforming, compromising messages.

References

[Bartsch 01] BARTSCH, Roy; GOERIGK, Wolfgang: "*Mechanical a-posteriori Verification of Results: A Case Study for a Safety Critical AI System*". In: KHATIB, Lina; PECHEUR, Charles: *Model-Based Validation of Intelligence - Papers from 2001 AAAI Spring Symposium.* AAAI Press, Menlo Park, March 2001. http://www.informatik.uni-kiel.de/~wg/Berichte/AAAI-2001.ps.gz (last accessed: 16th December 02)

[Blum 95] 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 (last accessed: 21st December 02)

[Boag 02] BOAG, Scott; et. al..: *XQuery 1.0: An XML Query Language*. W3C Working Draft. World Wide Web Consortium (W3C), November 2002. http://www.w3.org/TR/2002/WD-xquery-20021115/ (last accessed: 6th May 03)

[Bray 00] BRAY, Tim; u. a.: *Extensible Markup Language (XML) 1.0 (Second Edition)*. W3C-Recommendation. World Wide Web Consortium (W3C), Oktober 2000. http://www.w3.org/TR/2000/REC-xml-20001006 (last accessed: 24th July 03)

[Brion 01] BRION, Paul V.; MALHOTRA, Ashok: *XML Schema Part 2: Datatypes*. W3C Recommendation. May 2001. http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/ (last accessed: 18th July 03)

[Chinnici 03] CHINNICI, Roberto, et. al.: *Web Services Description Language (WSDL) Version 1.2 Part 1: Core Language*. W3C Working Draft. World Wide Web Consortium (W3C), June 2003. http://www.w3.org/TR/2003/WD-wsdl12-20030611/ (last accessed: 18th July 03)

[Fielding 99] 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 (last accessed: 23rd July 03)

[Gaul 99] GAUL, Thilo; et. al.: "*Conctruction of Verified Software Systems with Program-Checking: An Application To Compiler Back-Ends*". In: PNUELI, A.; TRAVERSO; P: *Electronic Proceedings of the Federated Logics Conference 99 Workshop on Runtime Result Verification*, 1999.

[Gudgin 03a] 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/ (last accessed: 18th July 03)

[Gudgin 03b] GUDGIN, Martin; et. al.: *SOAP Version 1.2 Part 2: Adjuncts*. W3C Recommendation. World Wide Web Consortium (W3C), June 2003. http://www.w3.org/TR/2003/REC-soap12-part2-20030624/ (last accessed: 18th July 03)

[Gudgin 03c] GUDGIN, Martin; et. al.: *Web Services Description Language (WSDL) Version 1.2 Part 2: Message Patterns*. W3C Working Draft. World Wide Web Consortium (W3C), June 2003. http://www.w3.org/TR/2003/WD-wsdl12-patterns-20030611/ (last accessed: 18th July 03)

[Hoare 85] HOARE, C.A.R: *Communicating Sequential Processes*. Prentice-Hall, London, 1985.

[Microsoft 03] MICROSOFT. *Microsoft .NET*. Website. Microsoft Corporation, Redmond, 2003. http://www.microsoft.com/net/ (last accessed: 21st July 03)

[Moreau 03] MOREAU, Jean-Jacques; SCHLIMMER, Jeffrey: *Web Services Description Language (WSDL) Version 1.2 Part 3: Bindings*. W3C Working Draft. World Wide Web Consortium (W3C), June 2003. http://www.w3.org/TR/2003/WD-wsdl12-bindings-20030611/ (last accessed: 18th July 03)

[Sun 03] Sum: Java 2 Platform, Enterprise Edition (J2EE). Website. Sun Microsystems, Santa Clara, 2003. http://java.sun.com/j2ee/ (valid: 21st July 03) http://www.informatik.uni-kiel.de/~wg/Berichte/RTRV99.ps.gz (last accessed: 16th December 02)

[Thompson 01] THOMPSON, Henry S.; et. al.: *XML Schema Part 1: Structures*. W3C Recommendation, World Wide Web Consortium (W3C), Mai 2001. http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/ (last accessed: 18th July 03)

Note: After every URL the date is given in brackets, when the URL was valid.