Ausgabe 3.06 Mai / Juni

Mit CD!

Deutschland € 5,50
Österreich € 6,00 Schweiz sFr 10,80

S&S

entwickler magazin

Enterprise Technologies & Business Solutions

www.entwickler-magazin.de

der entwickler

Visual Studio 2005

Ready for .NET 2.0

C++-Bibliothek wxWidgets vorgestellt

LinuX Enterprise

OpenSolaris

Suns Linux-Alternative

Python auf Nokia

Python for S60 jetzt Open Source

Datenbanken magazin

Firebird Embedded

Automatisiertes Deployment

Zugriffskontrolle

Oracle Label Security

Enterprise ARCHITEKTUR MAGAZIN

SOA-Plattformen

BEA AquaLogic, IBM WebSphere, SAP NetWeaver, Oracle Fusion

Die Wahl des richtigen Web Service

XML magazin & WEB SERVICES

Session Management mit Cocoon

XML Query Test Suite

XQuery-Tools testen

27 Seiten extra

Let‘s talk about SOA

- Die wichtigsten Plattformen
- SOA zwischen Hype und Business

Mobile Datenbanken

- Die Wahl der richtigen Architektur

Virtualisierung

- QEMU & Co. im Vergleich

CD-Inhalt

Intel C++ Compiler für Windows (Testversion),
SVG Editor 1.0 for Eclipse IDE,
Porcupine 0.0.6,
Opera 8.5,
Apache Cocoon 2.1.8,
XMLEspresso 5.0,
AquaQuery 1.0

D31141F

Ein Überblick über die Web-Services-Technologien und ihre wichtigsten Standards und Spezifikationen

TITELTHEMA

Eine gute Wahl treffen

Web Services gewinnen zunehmend an Akzeptanz und Verbreitung. Gerade im Zusammenhang mit serviceorientierten Architekturen (SOA) und Interoperabilität sind sie nicht mehr aus der Entwicklung (stark) verteilter Systeme wegzudenken. Sie werden zu Recht als prädestinierte Technologie zur Implementierung von Services innerhalb einer SOA gesehen. Was aber bildet überhaupt die Basis für Web Services? Wie werden die für Enterprise-Anwendungen notwendigen Infrastrukturdienste mit Web Services realisiert? Wie wird Interoperabilität erreicht und welche Standards und Spezifikationen existieren? Zahlreiche Fragen, die Grund genug sind, sich einen Überblick über die Web-Services-Landschaft zu verschaffen.

von Marcel Tilly und Hartmut Wilms

Web Services sind, einfach gesprochen, Dienste, die über Standard-Webprotokolle wie TCP, SMTP und vor allem HTTP(S) angeboten werden. Der Siegeszug des World Wide Web, das allgegenwärtige Transport- oder auch Applikationsprotokoll HTTP und XML als anerkannte und allseits unterstützte Architektur für Dokumente sind u.a. verantwortlich für den Erfolg von Web Services. Ein weiterer Erfolgsfaktor sind die Standardisierungen im Web-Services-Kontext, die Interoperabilität und eine konsistente unternehmensweite Architektur innerhalb von verteilten (heterogenen) Systemen erst möglich machen. Standards werden durch Web-Services-Spezifikationen definiert. Frei nach dem Motto „Standards sind gut – wir haben auch einen“ existieren in der Web-Services-Welt mehrere Gremien und Organisationen, die zum Teil unterschiedliche oder auch gegensätzliche Standards entwickeln. Allen Spezifikationen ist jedoch gemein, dass sie keine Vorgaben für die Implementierung von so genannten Web-Service-Servern oder gar Web Services an sich liefern, sondern sich voll und ganz auf das Protokoll konzentrieren.

Die Vielzahl der existierenden Spezifikationen lässt einen leicht schwindeln und fragen, wie man die Entwicklung von Web Services in den Griff bekommen soll. Glücklicherweise lassen sich die Spezifikationen in aufeinander aufbauende Schichten einteilen, von denen allein die unterste (Basis-)Schicht obligatorisch ist (Abbildung 1). Services werden entsprechend der Basisspezifikationen und Teilen der erweiterten Spezifikationen, z.B. Messaging, implementiert. Infrastrukturdienste, die der Erfüllung nichtfunktionaler Anforderungen wie Sicherheit und Zuverlässigkeit dienen, werden durch so genannte WS-Spezifikationen der zweiten Generation abgedeckt. Durch die Erweiterbarkeit des Simple Object Access Protocol (SOAP) können solche Dienste leicht nach Bedarf auf Nachrichtenebene hinzugefügt werden. Zuletzt können Services zur Implementierung komplexer Geschäftsvorfälle zusammengesetzt und zur Realisierung von Geschäftsprozessen orchestriert werden.

Basisspezifikationen

Allgemein anerkannt sind vier Spezifikationen, auf die sich das ganze Web-Services-Gebilde stützt. XML, natürlich auch XML Namespaces, XML Infoset und XML Schema als Datenformat und SOAP als Kommunikationsprotokoll sind dabei über jegliche Diskussion erhaben. Die Web Services Description Language (WSDL) wird häufig als notwendiges Übel betrachtet, gerade die in Arbeit befindliche Version 2.0 ist umstritten [1], und dient zur Beschreibung der Schnittstelle. Das Universal Discovery and Directory Interface (UDDI) als Verzeichnisdienst

Abb. 1: Schichtenmodell der Web-Services-Spezifikationen

zum Finden [2] der unzähligen Services hingegen fristet ein eher trostloses Dasein. Dennoch bilden diese Spezifikationen ein konsistentes Bild für die Verwendung von Web Services im Speziellen bzw. Services im Allgemeinen (Abbildung 2).

Ein Consumer sucht in einem UDDI-Verzeichnis nach bestimmten Kriterien einen Service, der funktional und nichtfunktional seinen Anforderungen entspricht. Diese Anforderungen sind in der WSDL und der evtl. zugehörigen Policy (Verfahrensweise), auf die später noch eingegangen wird, festgelegt. WSDL, Policy und die Beschreibung der Service-Semantik bilden zusammen den Service-Vertrag (Contract). Die WSDL, hinterlegt im UDDI Verzeichnis, ermöglicht dem Consumer eine SOAP-Anfrage (Request) zu formulieren und an den Endpunkt des Providers zu schicken. Die SOAP-Anfrage wird in den meisten Fällen durch ein Framework auf der Consumer-Seite erzeugt. Hierzu wird zumeist statisch eine Proxy-Klasse aus der WSDL generiert, z.B. mit dem Apache-Axis-Tool WSDL2Java [3]. Die notwendigen Informationen sind in folgende WSDL-Elemente aufgegliedert:

- <*types*>-Element: XML Schema der Datentypen
- <*message*>-Element: Elemente der Operationen
- <*portType*>-Element: Service-Schnittstellen (Interface)
- <*binding*>-Element: Kommunikationsprotokoll-Mapping (SOAP über HTTP etc.)
- <*service*>-Element: Service-Endpunkte (URL)

Der Request kann zwar auch programmatisch via XML-API erzeugt werden, aber bei der Verwendung von Web Services wird gerne auf die Proxy-Klassen zurückgegriffen, weil die Implementierung dadurch einfacher von der Hand geht.

Eine SOAP-Anfrage repräsentiert eine Nachricht, die der Message-Metapher der Service-Interaktion innerhalb des serviceorientierten Paradigmas entspricht. Services kommunizieren über Nachrichten(-Dokumente) miteinander, dabei beinhalten SOAP-Nachrichten ein Wurzelelement, <*soap:envelope*>. Dieses enthält ein optionales <*soap:Header*>-Element und ein <*soap:body*>-Element. Der SOAP-Header ist für die Übertragung von nichtfunktionalen Informationen (Security, Reliability, Acknowledgement ...) vorgesehen. Der SOAP-Body hingegen enthält die eigentlichen Daten. Eine SOAP-Anfrage, die eine Bestellung abschickt, könnte wie in Listing 1 dargestellt aussehen.

Im Beispiel werden im SOAP-Header zusätzlich Authentifizierungsdaten („Max Mustermann“ mit Passwort „Test“) mitgegeben. Im SOAP-Body wird eine *getTicket*-Nachricht an den Service verschickt, die eine *Order* beinhaltet, bei der ein *Customer* („Ein Fußballfan“) für ein *Event* („WM-Endspiel“ in „Berlin“) ein Ticket bestellt.

Natürlich kann man mit SOAP auch größere oder binäre Daten innerhalb einer SOAP-Nachricht verschicken. Bisher konnte man zwischen SOAP with Attachment (SwA) und WS-Attachment wählen. Mittlerweile gibt es zu den beiden genannten Spezifikationen aber Alternativen: XOP und MTOM. Diese beiden Spezifikationen werden zukünftig zur Übertragung binärer Daten zum Einsatz kommen.

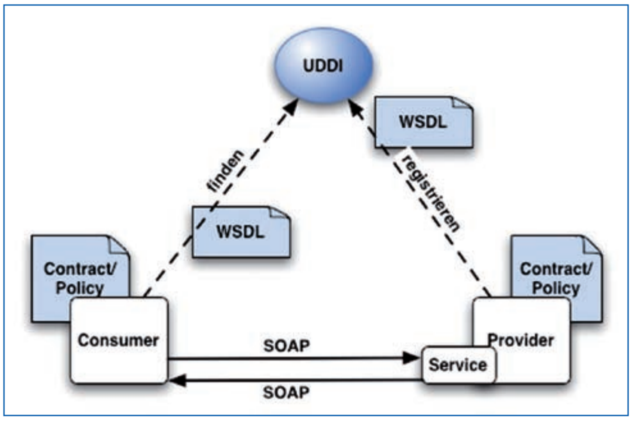

Abb. 2: Basisspezifikationen im Zusammenspiel

Essenzielle Erweiterungen

SOAP ist ursprünglich mit dem Ziel geschaffen worden, einfach und flexibel zu sein. Wie einfach eine SOAP-Nachricht sein kann, haben wir ja bereits gesehen, die Flexibilität wird durch den SOAP-Header erreicht. Dieser ist speziell für Erweiterungen der Basisspezifikationen gedacht und dient vor allem dafür, die SOAP-Nachricht hinsichtlich nichtfunktionaler Anforderungen zu erweitern. In Listing 1 wurde der Header benutzt, um die Authentifizierungsdaten zu übertragen. Durch die strikte Trennung zwischen funktionalen Daten im SOAP-Body und nichtfunktionalen Daten im SOAP-Header, kann ein Service mit nichtfunktionalen Erweiterungen bedacht werden, ohne die Service-Schnittstelle tatsächlich zu ändern.

Der SOAP-Header kann für verschiedenste Erweiterungen verwendet werden. Eine sehr wichtige Funktionalität bei der Arbeit mit nachrichten- bzw. dokumentenorientierten Technologien ist das asynchrone Verschicken solcher Nachrichten. Verwendet man SOAP über HTTP, so ist es einfach, sich in einem klassischen Client-Server-Szenario zu bewegen. Diese Art

Web Services zu benutzen, unterscheidet sich nur geringfügig von der herkömmlichen Verwendung von RPC mit CORBA oder RMI und ist in der Regel synchron. Möchte man stattdessen auch über HTTP Web Services asynchron verwenden, muss man auf eine essenzielle Erweiterung zurückgreifen: WS-Addressing.

Diese beim W3C als Recommendation verabschiedete Spezifikation dient dazu, Informationen über die Aktion, die ausgeführt werden soll, den URI des Endpunktes, den URI des Empfängers im Erfolgs- und den URI des Empfängers im Fehlerfall in der SOAP-Nachricht mitzuschicken. Dadurch kann erreicht werden, dass der Consumer nicht direkt auf die Antwort warten muss, sondern der Service die Antwort asynchron an den mitgegebenen Empfänger verschickt. Das hört sich jetzt möglicherweise extrem hypothetisch an, aber ein Beispiel macht die Funktionsweise sicher deutlich (siehe Listing 2).

Listing 1

SOAP-Nachricht einer Ticket-Bestellung für das WM-Endspiel in Berlin inkl. SOAP-Header für Authentification

```
<e:Envelope xmlns:d=“http://www.w3.org/2001/
                                      XMLSchema“
  xmlns:e=“http://schemas.xmlsoap.org/soap/
                                       envelope/“
  xmlns:i=“http://www.w3.org/2001/
                              XMLSchema-instance“
  xmlns:wn0=“http://innoq.com/wsdl/com/innoq/
                                            ws/“>
  <e:Header>
    <ns0:authentication xmlns:ns0=
              “http://innoq.com/TicketSaleTypes“>
      <ns0:username>MAX MUSTERMANN
                                  </ns0:username>
      <ns0:password>TEST</ns0:password>
    </ns0:authentication>
  </e:Header>
  <e:Body>
    <wn0:getTicket>
      <wn0:order i:type=”wn0:TicketOrder”>
        <wn0:event i:type=”wn0:Event”>
          <wn0:eventName i:type=”d:string”>
                        WM-Endspiel</wn0:eventName>
          <wn0:location i:type=”d:string”>Berlin
                                   </wn0:location>
        </wn0:event>
        <wn0:customer i:type=”d:string”>
                         Ein Fußballfan</wn0:name>
      </wn0:order>
    </wn0:getTicket>
  </e:Body>
</e:Envelope>
```

Einer Nachricht wird in diesem Fall eine eindeutige *MessageID* zugeordnet, damit im Response- oder Fehlerfall Bezug darauf genommen werden kann. Ebenso beinhaltet der WS-Addressing-Header Informationen wohin (*ReplyTo*) eine Antwort geschickt werden soll (*FaultTo* gilt für eine Fehlernachricht).

Im Grunde genommen sollte WS-Addressing zu den Basis-Spezifikationen gehören. Eine Adressierung auf Nachrichtenebene ist für die serviceorientierte Message-Metapher unabdingbar. Dies wird schnell deutlich, wenn Nachrichten nicht nur innerhalb von Punkt-zu-Punkt-Verbindungen gesendet, sondern über mehrere Hops (Knoten im Netzwerk) übertragen werden. Gegebenenfalls könnte dann zwischen den Knoten sogar das Transportprotokoll gewechselt werden, ohne die Adressinformationen zu verlieren.

Safety first

Natürlich ist das Thema Sicherheit auch bei Web Services von tragender Bedeutung. Möchte man nur eine transportbasierte Verschlüsselung seiner Nachricht, so ist am einfachsten HTTPs – die sicherere Variante von HTTP – zu verwenden. Ist man hingegen aber gezwungen, Teile seiner Nachricht zu verschlüsseln und das sogar noch mit unterschiedlichen Schlüsseln, weil nur Teile der Daten von unterschiedlichen Partnern gelesen werden sollen, dann muss man zu alternativen Lösungen greifen. Neben den Basisspezifikationen ist die bei OASIS verwaltete WS-Security-Spezifikation in der WS-Gemeinde eine der anerkanntesten Spezifikationen.

Grundsätzlich baut WS-Security auf den beiden existierenden und etablierten Spezifikationen XML-Signature und XML-Encryption auf, die Verfahren zum Signieren und Verschlüsseln beschreiben. WS-Security verwendet diese Spezifikationen und erweitert den SOAP-Header, damit sicherheitsrelevante Informationen in der Nachricht transportieren werden können. Darüber hinaus definiert WS-Security, wie Benutzeridentifikationsdaten (User Credentials) über ein Token, sowohl XML-basiert oder binär, übermittelt werden können. Binäre Tokens können in einem BinarySecurityToken-Element übermittelt werden. Durch die Verbindung mit SAML [4], X509 [5]- und Kerberos-SecurityToken [6] lassen sich sicherheitsrelevante Web Services anbieten. Die Unterstützung in den einzelnen Frameworks und Servern ist diesbezüglich recht gut ausgereift. Verständlicherweise existiert nur eine prototypische SAML-Unterstützung in .NET [7], da Microsofts Interesse an SAML eher gering ist.

Garantiert zuverlässig

Sicher mag die Anwendung ja nun sein, allerdings ist der Transport der Nachrichten noch nicht zuverlässig. In einer Unternehmensanwendung müssen Nachrichten häufig garantiert zugestellt werden, z. B. muss bei der nächtlichen Übertragung von Daten in das Bestandssystem nachvollziehbar sein, welche Daten übermittelt und welche verlorengegangen sind. In dem Bereich der transportunabhängigen Nachrichtenübertragung scheint sich eine Spezifikation durchzusetzen: WS-Reli-

Listing 2

SOAP-Nachricht mit WS-Addressing-Header

```
<e:Envelope ...>
  <e:Header>
    <wsa:Action>http://localhost:8080/
                   TicketSale/getTicket</wsa:Action>
    <wsa:From>
      <wsa:Address>http://localhost:4712/
                AMS_Web/Sender.ashx</wsa:Address>
    </wsa:From>
    <wsa:MessageID>urn:uuid:15f8bb0d-b8df-4e83-
                a26304b2c2bdbd28</wsa:MessageID>
    <wsa:ReplyTo>
     <wsa:Address>http://localhost:4712/AMS_Web/
                     Receiver.ashx</wsa:Address>
    </wsa:ReplyTo>
    <wsa:FaultTo>
     <wsa:Address>http://localhost:4712/AMS_Web/
                        Fault.ashx</wsa:Address>
    </wsa:FaultTo>
    <wsa:To>http://localhost:8080/innoQ.com/
                             TicketSale</wsa:To>
  </e:Header>
  <e:Body>
    <!--siehe Listing 1 -->
</e:Envelope>
```

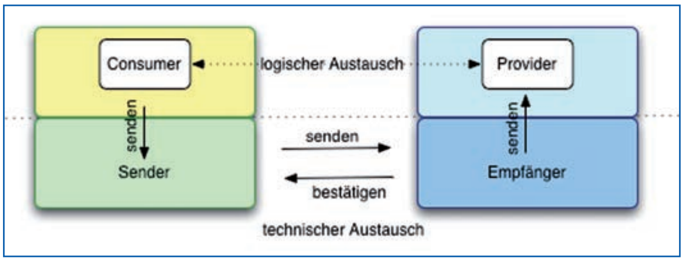

Abb. 3: Modell des Austauschs von Nachrichten bei WS-ReliableMessaging

ableMessaging. Diese ist gerade an OASIS zur Standardisierung übergeben worden. Der interessante Aspekt ist, dass es bereits einen verabschiedeten Standard hierzu bei OASIS gab: WS-Reliability. Allerdings zeigt sich bei diesem Thema ganz deutlich, welche Hersteller im Web-Service-Umfeld den Takt angeben. Grundsätzlich sind die Spezifikationen von Microsoft und IBM diejenigen mit der größten Durchsetzungskraft. So ist auch zu erwarten, dass die Vereinigung von WS-Reliability und WS-ReliableMessaging eher wie die ursprüngliche WS-ReliableMessaging-Spezifikation von Microsoft, IBM, BEA und Tibco aussieht. Die Spezifikation zu WS-ReliableMessaging beschreibt ein Protokoll für den zuverlässigen Austausch von Nachrichten in einer verteilten Umgebung zwischen verschiedenen Services. Dabei ist die Art und Weise der Übertragung transportunabhängig. Es kann also mit verschiedensten Transportprotokollen benutzt werden und ist somit die Web-Service-Alternative zu den hauptsächlich proprietären Varianten, wie JMS oder MQSeries. WS-ReliableMessaging verwendet WS-Addressing und definiert ein Protokoll, um Nachrichten zu identifizieren und zu verfolgen. Diese Nachrichten werden im Sinne einer gesicherten Übermittlung verwaltet. Die Zustellung ist genau zwischen zwei Services (einer Quelle und einem Ziel) festgelegt. Dabei kann auch eine Folge (Sequence) von Nachrichten verschickt werden. Die einzelnen Nachrichten innerhalb einer Sequence werden durchnummeriert. Die letzte Nachricht wird dabei speziell markiert (*lastMessage="true"*) oder es wird alternativ bereits beim Erzeugen der Sequence angegeben, wie viele Nachrichten verschickt werden sollen. Für erfolgreich empfangene Nachrichten wird eine Empfangsbestätigung (Acknowledgement) verschickt. Wie wird das aber nun erreicht?

Bei WS-RM kommt eine zusätzliche Schicht zum Einsatz, die den logischen Nachrichtenaustausch zwischen Consumer und Provider vom technischen Austausch zwischen Sender und Empfänger trennt. Sender und Empfänger sind hier Bestandteile einer WS-RM-Runtime. Der Consumer schickt seine Nachricht also nur logisch an den Provider. In Wirklichkeit wird die Nachricht an den Sender geschickt, der für die garantierte Zustellung sorgt. Ist der Empfänger nicht verfügbar, versucht der Sender so lange wie angegeben die Nachricht zu schicken. Auf der Provider-Seite nimmt dementsprechend ein Empfänger zunächst die Nachricht entgegen, bevor er diese an den Provider weiterleitet (Abbildung 3).

Verträge und Verfahrensweisen

Wie bereits erwähnt, ist es durchaus sinnvoll, sich einen Service nicht nur nach seinen funktionalen Aspekten auszusuchen,

BPEL4WS	Business Process Execution Language for Web Services ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf
BPML	Business Process Modeling Language ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf
CDL4WS	Choreography Definition Language for Web Services http://www.w3.org/TR/ws-cdl-10/
MOWS	Management of Web Services ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf
MUWS	Management Using Web Services ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf
MTOM	SOAP Message Transmission Optimization Mechanism http://www.w3.org/TR/soap12-mtom
OASIS	Organization for the Advancement of Structural Information Standards http://www.oasis-open.org
SAML	Security Assertion Markup Language http://docs.oasis-open.org/security/saml/v2.0/saml-2.0-os.zip
SOAP	früher: Simple Object Access Protocol; seit SOAP1.2 wird das Akronym nicht mehr aufgelöst http://www.w3.org/TR/soap
SwA	SOAP with Attachments http://www.w3.org/TR/SOAP-attachments
UDDI	Universal Description, Discovery and Integration http://www.w3.org/TR/soap
W3C	World Wide Web Consortium http://www.w3c.org
WS-A	Web Service Addressing http://www.w3.org/TR/soap
WS-Attach	WS-Attachments http://msdn.microsoft.com/library/en-us/dnglobspec/html/draft-nielsen-dime-soap-01.txt
WS-CAF	Web Services Composite Application Framework http://www.w3.org/TR/soap
WSCI	Web Service Choreography Interface http://www.w3.org/TR/wsci
WSDL	Web Service Description Language http://www.w3.org/TR/soap
XML	eXtensible Markup Language http://www.w3.org/TR/soap
XOP	XML-binary Optimized Packaging http://www.w3.org/TR/xop10

Tabelle 1: Standards und Spezifikationen

sondern auch nach seinen nichtfunktionalen. Ein Consumer und ein Service sollen einen Vertrag eingehen, der auch oder der gerade die nichtfunktionalen Anforderungen mit abdeckt.

WS-Policy (WSP) erweitert einen Web Service um die Beschreibung seiner Anforderungen, Fähigkeiten und Zusicherungen in nichtfunktionaler Hinsicht. Eine Policy kann festlegen, welche Sicherheitsregeln oder welche Zustellungsmöglichkeiten ein Web Service unterstützt oder von seinem Kontrahenten erwartet. Eine Policy ist somit eine Verfahrensweise, die Provider (und Consumer) für eine erfolgreiche Kommunikation definieren. Die Policy eines Web Service kann z.B. festlegen, dass eine gültige Signatur vorliegen muss oder die Größe einer Nachricht nicht überschritten werden darf.

Zusammen mit WS-PolicyAssertions und WS-PolicyAttachments bildet WS-Policy ein Framework zum Festlegen dieser Richtlinien für einen Service. Dabei ist das Framework so gestaltet, dass es durch andere Web-Service-Spezifikationen erweitert werden kann. Eine Policy ist dabei eine Sammlung von verschiedenen Zusicherungen (Assertions). Eine solche Assertion definiert eine bestimmte Anforderung, Eigenart, Verhalten oder Regel, wie mit dem Web Service kommuniziert werden kann bzw. soll. Eine Assertion kann optional sein. Ebenso können Assertions in Alternativen (policy alternatives) gruppiert sein. Damit gibt es für einen Consumer die Möglichkeit, aus einer Reihe von Alternativen zu wählen, z.B. welches Security-Token verwendet werden soll. Über eine Verbindung (Attachment) kann eine Policy mit einem Service (oder im WS-Policy-Sprachgebrauch einem „Subjekt“) verbunden werden. Eine Policy kann dabei wie folgt aussehen:

```
<wsp:Policy xmlns:wsp=“...“>
 <wsp:TextEncoding
  wsp:Usage=”wsp:Required” Encoding=”utf-8”/>
 <wsp:Language wsp:Usage=”wsp:Required”
  Language=”de”/>
 <wsp:SpecVersion wsp:Usage=”wsp:Required”
  URI=”http://www.w3.org/TR/2000/
                              NOTE-SOAP-20000508/” />
 <wsp:MessagePredicate wsp:Usage=”wsp:Required”>
  count(wsp:GetHeader(.)/wsse:Security) = 1
 </wsp:MessagePredicate>
 <wsp:MessagePredicate wsp:Usage=”wsp:Required”>
  count(wsp:GetBody(.)/*) = 1
 </wsp:MessagePredicate>
 ...
</wsp:Policy>
```

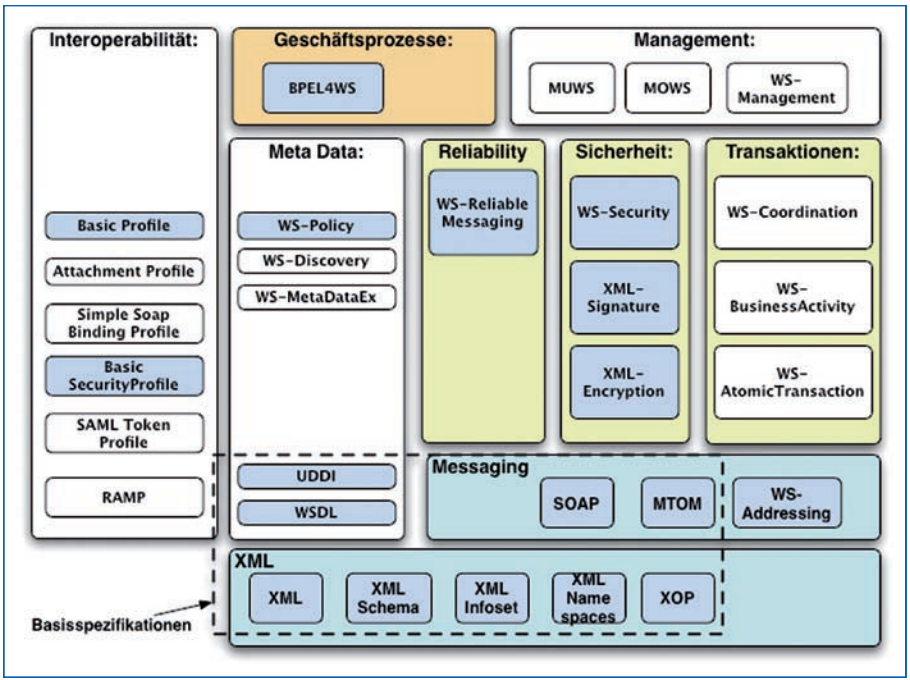

Abb. 4: Die wichtigsten Web-Services-Spezifikationen im Überblick (blau hinterlegt sind die bereits etablierten Spezifikationen)

In dieser Policy wird Folgendes definiert:

1. UTF-8 Encoding wird von einem möglichen Subjekt gefordert.
2. Deutsch wird als natürliche Sprache verwendet.
3. SOAP wird in der Version 1.1 unterstützt.
4. Über den XPath-Ausdruck *count(wsp:GetHeader(.)/wsse:Security) = 1* mit der Built-in Methode *wsp:GetHeader* wird geprüft, ob in der SOAP-Nachricht genau ein Security-Header vorhanden ist.
5. Über den XPath-Ausdruck *count(wsp:GetBody(.)/*) = 1* mit der Built-in-Methode *wsp:GetBody* wird geprüft, dass der SOAP-Body genau ein Element enthält und damit konform zum WS-I Basic Profile ist.

… und der ganze Rest

Über die bereits erwähnten Spezifikationen hinaus gibt es noch eine Vielzahl weiterer – nach aktueller Zählung mehr als 70 – Spezifikationen, die sich im Web-Services-Umfeld tummeln [8]. Wie wir bereits gesehen haben, gibt es durchaus konkurrierende Spezifikationen. Das liegt daran, dass kein Anbieter einen funktionalen Bereich kampflos einem Konkurrenten überlässt. Dennoch lässt sich eine Faustregel finden: Die Spezifikationen, die von Microsoft und IBM unterstützt werden, haben große Chancen sich durchzusetzen, denn kein Anbieter kann es sich leisten, nur eingeschränkt mit .NET und Java interoperabel [9] zu sein.

Um ein komplettes Bild für eine unternehmensweite Anwendung zu bekommen, sollten bisher unerwähnte Bereiche durchaus noch eingeordnet werden. Denn welche neue Technologie bemüht sich nicht, möglichst alle Bereiche der etablierten Technologien abzudecken. So gibt es auch im Web-Services-Umfeld Spezifikationen, die sich dem Thema Trans-

Geprüfte Interoperabilität

Die WS-I bietet Web Service Testing Tools an, die Nachrichten- und Schnittstellenbeschreibung prüfen und in gefälliger Form einen Report zum Thema Interoperabilität erstellen. Diese Tools können sowohl in einer C# als auch in einer Java-Variante kostenlos heruntergeladen werden [10].

aktionen annehmen. Hierzu gibt es die drei Spezifikationen WS-Coordination, WS-BusinessActivity und WS-AtomicTransaction. Die Koordinierung der an einer BusinessActivity beteiligten Services übernehmen Services, die in der Spezifikation WS-Coordination festgelegt werden. Eine BusinessActivity ist eine langlebige Transaktion, während die WS-AtomicTransaction-Spezifikation, kurz gesagt, zur Abbildung eines 2-Phasen-Commits dient.

Zur Abbildung von Geschäftsprozessen und Orchestrierung von Services gibt es ebenfalls konkurrierende Spezifikationen. Neben BPEL4WS existieren noch CDL4WS, WS-CAF und BPML. Hier scheint allerdings bereits eine Entscheidung zu Gunsten von BPEL4WS gefallen zu sein, schließlich sind schon entsprechende Tools und die richtige Herstellerunterstützung (IBM, Oracle und Microsoft) vorhanden.

So lassen sich die Spezifikationen mit den größten Chancen zu einem recht gut strukturierten Bild zusammensetzen (Abbildung 4).

Fazit

Die Web-Services-Technologie basiert auf einem soliden Fundament und wird bereits vielerorts eingesetzt. Die Auswahl der einzusetzenden erweiterten Spezifikationen oder Standards muss aufgrund der jeweiligen Systemanforderungen und Use Cases erfolgen. Im Falle von unternehmensintern offerierten Services kann man experimentierfreudiger sein als im Falle von Services, die von externen, vielleicht wechselnden bzw. unterschiedlichen Klienten konsumiert werden. Im zweiten Fall stellt sich auch die Frage, ob es für den Service Provider opportun ist, dem Konsumenten seine Standards zu diktieren und es ihm zu überlassen, diesen zu entsprechen. In vielen Fällen dürften Anbieter und Konsument beiderseits an einer möglichst schnellen und reibungslosen Interaktion interessiert sein. Sich über die Details eines Standards erst abstimmen zu müssen, kann einen beträchtlichen Aufwand darstellen.

Interoperabilität und weitestgehend problemlose Interaktion verspricht in jedem Fall WS-I Profil-Konformität. Es existieren bereits verschiedene Testtools, die die Profil-Konformität und Interoperabilität von Web Services überprüfen (siehe Kasten „WS-Interoperabilität“).

WS-Interoperabilität

Die Kommunikation zwischen verschiedenen Plattformen und Programmiersprachen steht bei der Benutzung von Web Services häufig im Vordergrund. Doch bereits in den Anfängen wurde erkannt, dass es durchaus Probleme geben kann, wenn Daten von der einen Plattform auf die andere transportiert werden sollen. Das liegt zum einen an den eher wenig restriktiven Spezifikationen SOAP und WSDL, deren Fokus immer eine hohe Flexibilität ist, was meist auf Kosten der Restriktionen und damit dem Interpretationsspielraum ging, zum anderen am XML Schema, das sich nicht immer einfach auf das plattformspezifische Typsystem abbilden lässt. Aus diesem Grund haben die Hersteller zunächst die Soapbuilder-Initiative ins Leben gerufen, die Tests definiert, sodass anhand einer Interoperabilitätsmatrix abgelesen werden konnte, in welchen Punkten die Plattformen der Wahl zusammenpassen und wo es Probleme gibt. Häufige Fehler waren die Abbildung von Listen, Datumstypen oder die Übermittlung von Null-Werten.

Wenig später gründete sich eine weitere Initiative, die Web Service Interoperability Group (WS-I), die sich nun gezielt das Thema Interoperabilität auf die Fahnen geschrieben hat. Die WS-I entwickelte Richtlinien (Profile), die definieren, welche Vorgaben zu erfüllen sind, wenn ein Service interoperabel sein soll. Hierbei kümmert sich die WS-I nicht nur um die Basisspezifikationen SOAP und WSDL, sondern auch um die Security- und ReliableMessaging-Spezifikationen. Derzeit gibt es die folgenden Profile beim WS-I:

- Basic Profile: Basis-Richtlinien zur Verwendung von interoperablen Web Services.
- Attachments Profile: Erweitert das Basic Profile für die Verwendung von Web Services mit Attachments.
- Simple SOAP Binding Profile: Spezifiziert die Serialisierung des Envelopes.
- Basic Security Profile: Interoperabilitätsprofil bei Verwendung der X.509-Token und Username-Token.
- REL Token Profile: Interoperabilitätsprofil zur Verwendung der Rights Expression Language (REL) Sicherheitstoken.
- SAML Token Profile: Interoperabilitätsprofil zur Verwendung von SAML Token innerhalb von WS-Security.
- Conformance Claim Attachment Mechanism (CCAM): Definiert das Hinzufügen von Konformitätsbereichen zu Web-Service-Artefakten.
- Reliable Asynchronous Messaging Profile (RAMP): Spezifiziert B2B-Integrationsszenarien mit Web Services.

Marcel Tilly *ist Principal Consultant bei der innoQ Deutschland GmbH. Sein Arbeitsschwerpunkt ist derzeit die Konzeption und Umsetzung von serviceorientierten Architekturen sowie der Einsatz generativer Softwareentwicklung in Projekten.*

Hartmut Wilms, *Senior Software Architect bei der innoQ Deutschland GmbH, ist seit mehr als 10 Jahren als Software Consultant und Entwickler in der objektorientierten Softwareentwicklung tätig. Zurzeit beschäftigt er sich schwerpunktmäßig mit serviceorientierten Architekturen, Web Services und Model Driven Development.*

Links & Literatur

[1] Marcel Tilly: WSDL2.0 – Neuerungen der Spezifikation, in: XML & Web Services Magazin 1.2005

[2] Stefan Tilkov: UDDI Revisited – Mit der Version 3 wird UDDI erwachsen, in: XML & Web Services Magazin 2.2004

[3] Web Services – Axis: ws.apache.org/axis/

[4] OASIS Security Services (SAML) TC: www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

[5] Public-Key Infrastructure (X.509): www.ietf.org/html.charters/pkix-charter.html

[6] Kerberos: web.mit.edu/kerberos/www/

[7] SAML STS Quickstart: www.gotdotnet.com/codegallery/codegallery.aspx?id=8da852b9-2c0d-4eb7-a2de-77222a4075f6

[8] Web-Services-Standards – Ein Überblick, Poster: www.innoq.com/soa/ws-standards/poster/

[9] Christian Weyer, Marcel Tilly: Java und .NET – Interoperabilität jenseits von Theorie und Spezifikation, in: Enterprise Programming 1.2005

[10] WS-I Testing Tools: www.ws-i.org/deliverables/workinggroup.aspx?wg=testingtools