Garantierte Zustellung

WS-ReliableMessaging: Zuverlässige Zustellung von Web Services-Nachrichten

WS-ReliableMessaging definiert ein transportunabhängiges Protokoll zur gesicherten Zustellung von Nachrichten im Web Services-Umfeld und stellt damit einen wichtigen Baustein zum Erreichen des hoch gesteckten Zieles einer „losen Kopplung“ dar.

von Stefan Tilkov

Auf den ersten Blick ist die Menge der Standards, die im Web Services-Umfeld existieren, aktualisiert werden und neu entstehen, schier unüberschaubar. Manchmal kann man das Gefühl bekommen, man müsse nur die Buchstabenkombination „WS“ mit einem beliebigen technischen Begriff per Bindestrich verbinden und habe damit schon eine neuen Spezifikation – die irgendein Hersteller wahrscheinlich auch schon per Veröffentlichung zum Standard erklärt hat.

Ganz so schlimm, wie sie sich zunächst darstellt, ist die Situation bei näherem Hinsehen aber gar nicht. Insbesondere die von IBM und Microsoft – in wechselnden Partnerschaften mit Tibco, BEA, SAP und anderen – herausgegebenen Spezifikationen ergänzen sich mehr und mehr zu einer modularen Gesamtarchitektur, deren Komponenten je nach Bedarf kombiniert werden können, um eine konkrete Aufgabenstellung zu unterstützen.

Ein wichtiges Themengebiet ist dabei die asynchrone Kommunikation, der wir uns in diesem Beitrag widmen wollen.

Messaging vs. RPC

Viele Anwender beginnen ihre Web Services-Experimente mit Beispielszenarien, die sich genauso gut auch mit alternativen Technologien wie z.B. RMI, CORBA oder DCOM umsetzen ließen. Dazu trägt vor allem der in der Vergangenheit von den Werkzeuganbietern stark in den Vordergrund gedrängte Interaktionsstil der synchronen Request/Response-Kommunikation bei.

In diesem Modell ruft ein Dienstnutzer eine Operation eines Dienstanbieters auf und wartet darauf, von diesem nach Durchführung einer Aktion eine Antwort zu erhalten. Dieses synchrone, RPC (Remote Procedure Call)-orientierte Modell ist gut für ein klassisches Client/Server-System geeignet, jedoch nur sehr eingeschränkt für eine realistische Kommunikation in einem kooperativen Integrationsszenario, geschweige denn in einem B2B-Umfeld.

Der Grund dafür ist, dass die Teilnehmer in einer Service-orientierten Architektur (SOA) möglichst unabhängig voneinander sein sollten. Wird ein System durch die Nichtverfügbarkeit eines anderen ausgebremst, weil es synchron auf eine Antwort wartet, ist die Gesamtverfügbarkeit aller Systeme unter Umständen nur so hoch wie die des anfälligsten und schwächsten Teilsystems. Schon in unternehmensinternen Integrationszenarien, insbesondere aber im B2B-Umfeld, kommen eventuell Netzprobleme und -ausfälle als Risikofaktoren hinzu.

Sowohl Anwender als auch Anbieter von Web Services-Lösungen folgen daher zunehmend einem Trend weg von der synchronen, RPC-ähnlichen Kommunikation und hin zu einem asynchronen, eher nachrichten- bzw. dokumentenorientierten Interaktionsstil. Das Problem beim Einsatz des allseits gegenwärtigen HTTP-Protokolls ist, dass dieses keine zuverlässige, asynchrone Kommunikation unterstützt. Diese ist jedoch für unternehmenskritische Anwendungen eine Voraussetzung – Nachrichten können verloren gehen, in ungeordneter Reihenfolge oder doppelt zugestellt werden usw.

Eine Möglichkeit zur Unterstützung von zuverlässiger, asynchroner Kommunikation ist der Einsatz einer darauf ausgelegten Transportschicht, wie sie z.B. von JMS-

Implementierungen wie WebSphere MQ oder Tibco mitgebracht wird. In diesen Fällen wird HTTP durch ein anderes Protokoll ersetzt (oder ein anderes Protokoll durch HTTP „getunnelt"). Standardisiert ist diese Variante jedoch nicht bzw. nicht im gleichen Umfang wie z.B. SOAP oder HTTP: So mag JMS zwar in der Java-Welt durchaus ein gangbarer Weg sein, .NET-Anwender an ein JMS-System anzuschließen ist jedoch alles andere als einfach. In B2B-Szenarien ist ohnehin kaum ein anderes Protokoll als HTTP denkbar.

WS-ReliableMessaging [1] setzt genau an dieser Stelle an und definiert ein auf Bestätigungen (Acknowledgements) basierendes Protokoll, das zuverlässige Nachrichtenübertragung auch für Transportmechanismen unterstützt, die selbst nicht zuverlässig sind.

Listing 1

```
<?xml version="1.0" encoding="UTF-8"
standalone="yes"?>
<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:wsa="http://schemas.xmlsoap.org/ws/2004/
                              03/addressing">
 <e:Header>
  <wsa:MessageID>uuid:fbc714e0-94fb-11d8-8569-
                    f90c13908569</wsa:MessageID>
  <wsa:To>http://destination.example.org/
                              TestService</wsa:To>
  <wsa:Action>http://destination.example.org/
                TestService#SomeAction</wsa:Action>
  <wsa:From>
   <wsa:Address>http://source.example.org/
                        TestConsumer</wsa:Address>
  </wsa:From>
  <wsa:ReplyTo>
   <wsa:Address>http://source.example.org/
            TestConsumerReplyHandler</wsa:Address>
  </wsa:ReplyTo>
  <wsa:FaultTo>
   <wsa:Address>http://source.example.org/
          TestConsumer/FaultHandler</wsa:Address>
  </wsa:FaultTo>
 </e:Header>
 <e:Body>
  ...
 </e:Body>
</e:Envelope>
```

Erweiterbarkeit

Möglich wird die Erstellung von erweiterten Spezifikationen für Web Services durch eines der Kernmerkmale des SOAP-Protokolls: die für unterschiedlichste Zwecke nutzbaren SOAP-Header. Darüber können Informationen als Teil von Nachrichten übertragen werden, ohne dass dazu die logische Schnittstelle eines Dienstes angepasst werden muss. Mit anderen Worten: Für den Anbieter und Nutzer eines Dienstes bleibt die Schnittstelle – was die funktionalen Aspekte angeht – unverändert; nichtfunktionale Anforderungen werden praktisch „unter der Haube" abgebildet. Diesen Mechanismus nutzen sowohl WS-ReliableMessaging als auch das von diesem verwendete WS-Addressing-Protokoll.

WS-Addressing als Basis

WS-Addressing [2] ist eine relativ einfache, aber ausgesprochen wichtige Spezifikation – neben WS-ReliableMessaging setzen z.B. auch WS-Eventing [3] und WS-Notification [4] darauf auf. Die Spezifikation definiert, wie Web Service-Adressen unabhängig von der verwendeten Transportschicht in SOAP-Nachrichten ausgedrückt werden. Eine solche Funktionalität ist z.B. notwendig, um Callbacks auf Basis von Web Services umzusetzen – in diesem Fall ist die Adresse eines Dienstes Bestandteil einer Nachricht, die einem anderen Dienst übermittelt wird. Neben der Identifikation von Diensten spezifiziert WS-Addressing auch, wie Nachrichten mithilfe von Message-IDs eindeutig identifiziert werden können. Am einfachsten lässt sich die Anwendung von WS-Addressing an einer Beispiel-SOAP-Nachricht erläutern (Listing 1).

Listing 2

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<e:Envelope
 xmlns:e="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/
                              addressing">
 <e:Header>
  <wsa:MessageID>uuid:ccd1e600-9506-11d8-9372-
                    c829e1c09372</wsa:MessageID>
  <wsa:To>http://source.example.org/
                    TestConsumerReplyHandler</wsa:To>
  <wsa:Action>http://source.example.org/
                TestConsumerReplyHandler#SomeAction
                              </wsa:Action>
  <wsa:RelatesTo>uuid:fbc714e0-94fb-11d8-8569-
                    f90c13908569</wsa:RelatesTo>
 </e:Header>
 <e:Body>
  ...
 </e:Body>
</e:Envelope>
```

Interessant sind dabei die XML-Elemente mit dem Präfix *wsa*. Zunächst enthält die SOAP-Nachricht eine eindeutige Identifikation (*wsa:MessageID*), die verwendet werden kann, um später die Nachricht als solche zu referenzieren (z.B. um eine Antwortnachricht mit der Anfrage zu korrelieren). *wsa:To* und *wsa:Action* drücken transportunabhängig aus, an welchen Endpoint die Nachricht gesendet werden soll. Schließlich sind noch drei so genannte Endpoint Reference-Elemente enthalten:

- *wsa:From* ist der Endpoint, von dem die Nachricht gesendet wurde,
- *wsa:ReplyTo* ist der Endpoint, an den die Antwort geschickt werden soll und
- *wsa:FaultTo* ist der für die Behandlung von Fehlern zuständige Endpoint.

Im Beispiel sind unterschiedliche Endpoints für die Behandlung von Antworten und Fehlern zuständig. In vielen Fällen wird dafür derselbe Service zuständig sein. Ebenso ist denkbar, dass keine Antwort erwünscht wird (*From*-, *ReplyTo*- und *FaultTo*-Elemente sind optional, nur *wsa:To* und *wsa:Action* sind für WS-Addressing-konforme Nachrichten zwingend erforderlich). WS-Addressing spezifiziert, wie diese Elemente auf SOAP-Nachrichtenfelder abgebildet werden. Am einfachsten ist dies wiederum anhand einer Beispielnachricht zu erklären (Listing 2).

Zu erkennen ist, dass sich diese Antwort auf die Nachricht aus Listing 1 bezieht (die ID im *wsa:RelatesTo*-Element ist die Message-ID der ersten Nachricht) und das *wsa:To*-Element die Adresse aus dem *wsa:ReplyTo*-Element enthält. WS-Addressing ermöglicht also, diese Antwortnachricht nicht nur synchron zu übermitteln – z.B. in der HTTP-Response bei Verwendung von HTTP als Transportschicht –, sondern auch asynchron, zu einem späteren Zeitpunkt und über denselben oder einen anderen Transportkanal.

Sequenzen, Message-IDs, Acknowledgements

WS-ReliableMessaging nutzt einen bewährten Mechanismus, um die sichere Übertragung von Nachrichten auch in Fehlersituationen zu gewährleisten (bzw. Fehler zumindest sicher zu erkennen): das Versenden von Bestätigungen (Acknowledgements). Ziel ist dabei, Versender und Empfänger der Nachricht möglichst wenig zu belasten und insbesondere sicherzustellen, dass die nichtfunktionale Anforderung sicherer Nachrichtenübertragung keine Auswirkung auf die Umsetzung funktionaler Anforderungen hat.

Das Modell, das WS-ReliableMessaging zugrunde liegt, ist in Abbildung 1 dargestellt. Für Dienstnutzer und -anbieter erfolgt die Kommunikation auf Basis der nach den funktionalen, logischen Anforderungen konzipierten Nachrichten. Die Zuverlässigkeit der Nachrichtenübertragung wird von der darunter liegenden Infrastruktur übernommen. Dabei handelt es sich typischerweise um eine Web Services-Ablaufumgebung, wie z.B. einen Application Server, die .NET-Runtime oder eine Stand-alone-Implementierung.

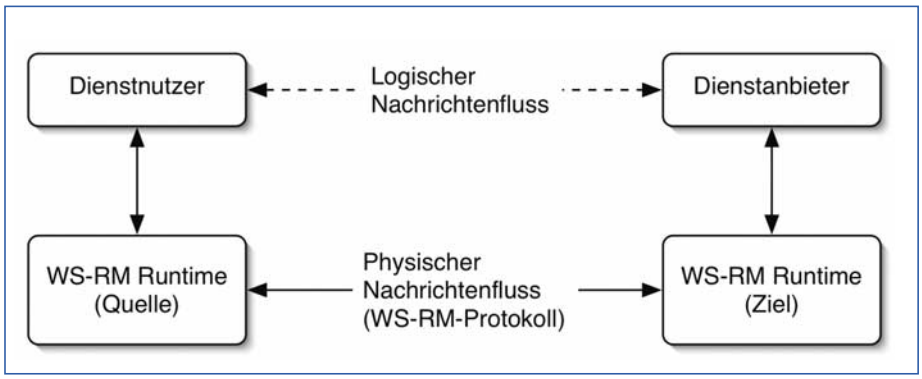

Abb. 1: Konzeptionelles Modell

Die Spezifikation legt – wie im Web Services-Umfeld üblich – hier keine Details der Implementierung und auch kein Programmiermodell fest, sondern standardisiert nur das Nachrichtenformat, das auf der Transportschicht – „auf dem Draht" – verwendet wird.

Konzeptionell werden in der Spezifikation drei unterschiedliche Auslieferungsgarantien (Delivery Assurances) definiert: *AtMostOnce*, *AtLeastOnce* und *ExactlyOnce*. Diese legen fest, ob eine Nachricht höchtens, mindestens oder genau einmal zugestellt werden soll. Jeder dieser drei Strategien kann optional noch mit der Zusicherung *InOrder* kombiniert werden, um sicherzustellen, dass Nachrichten in derselben Reihenfolge zugestellt werden, in der sie versendet wurden. (In der ersten Version der Spezifikation vom März 2003 wurden diese Auslieferungsgarantien auf WS-PolicyAssertion-Ausdrücke abgebildet; in der aktuellen Version vom März

Listing 3

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<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:wsa="http://schemas.xmlsoap.org/ws/2004/03/
                                        addressing"
 xmlns:wsrm="http://schemas.xmlsoap.org/
                                        ws/2004/03/rm"
 xmlns:wsu="http://schemas.xmlsoap.org/
                                        ws/2002/07/utility">
 <e:Header>
  <wsrm:Sequence>
   <wsu:Identifier>
    http://source.example.org/
          b9b4dcd0-9506-11d8-a776-b87b2450a776
   </wsu:Identifier>
   <wsrm:MessageNumber>1</
                                        wsrm:MessageNumber>
   <wsu:Expires>2005-04-23T11:15:13.827+02:00
                                        </wsu:Expires>
  </wsrm:Sequence>

  <wsa:MessageID>uuid:fbc714e0-94fb-11d8-8569-
                        f90c13908569</wsa:MessageID>
  <wsa:To>http://destination.example.org/
                                        TestService</wsa:To>
  <wsa:Action>http://destination.example.org/
                    TestService#SomeAction</wsa:Action>
  <wsa:From>
   <wsa:Address>http://source.example.org/
                                TestConsumer</wsa:Address>
  </wsa:From>
  <wsa:ReplyTo>
   <wsa:Address>http://source.example.org/
          TestConsumerReplyHandler</wsa:Address>
  </wsa:ReplyTo>
  <wsa:FaultTo>
   <wsa:Address>http://source.example.org/
                        TestConsumer/FaultHandler</
                                        wsa:Address>
  </wsa:FaultTo>
 </e:Header>
 <e:Body>
  ...
 </e:Body>
</e:Envelope>
```

Listing 4

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<e:Envelope xmlns:e="http://schemas.xmlsoap.org/
                                        soap/envelope/"
 xmlns:wsa="http://schemas.xmlsoap.org/
                                        ws/2004/03/addressing"
 xmlns:wsrm="http://schemas.xmlsoap.org/
                                        ws/2004/03/rm"
 xmlns:wsu="http://schemas.xmlsoap.org/
                                        ws/2002/07/utility">
 <e:Header>
  <wsrm:SequenceAcknowledgement>
   <wsu:Identifier>
    http://source.example.org/
          b9b4dcd0-9506-11d8-a776-b87b2450a776
   </wsu:Identifier>
   <wsrm:AcknowledgementRange Lower="1"
                                        Upper="1"/>
  </wsrm:SequenceAcknowledgement>
  <wsa:MessageID>uuid:ccd1e600-9506-11d8-9372-
                        c829e1c09372</wsa:MessageID>
  <wsa:To>http://source.example.org/
                TestConsumerReplyHandler</wsa:To>
  <wsa:Action>
   http://schemas.xmlsoap.org/
          ws/2003/03/rm#SequenceAcknowledgement
  </wsa:Action>
  <wsa:RelatesTo>uuid:fbc714e0-94fb-11d8-8569-
                        f90c13908569</wsa:RelatesTo>
 </e:Header>
 <e:Body/>
</e:Envelope>
```

2004 ist der betreffende Abschnitt nicht mehr enthalten. Grund dafür ist, dass diese Varianten konzeptionell zwar weiterhin existieren, ihre Umsetzung jedoch keinerlei Auswirkungen auf das Protokoll selbst hat und deswegen an dieser Stelle auch nicht standardisiert werden muss.)

Nachrichten werden zu einer Sequenz zusammengefasst. Diese Sequenz – die im Extremfall auch nur eine einzige Nachricht beinhalten kann – ist es, für die bestimmte Auslieferungsstrategien zum Einsatz kommen können. Nachrichtensequenzen erhalten eine eindeutige Identifikation. Diese wird entweder vom Sender (der Normalfall) oder vom Empfänger (auf Anforderung) erzeugt. Nachrichten innerhalb einer Sequenz müssen fortlaufend durchnummeriert werden.

Die Erweiterung der SOAP-Nachricht aus Listing 1 um die WS-ReliableMessaging-Informationen ist in Listing 3 dargestellt. Die Nachricht enthält zunächst Referenzen auf die Namespaces der WS-ReliableMessaging-Spezifikation (Präfix *wsrm*) und den Utility-Namensraum, der von vielen Web Services-Spezifikationen für gemeinsam genutzte Elemente verwendet wird. Der *wsrm:Sequence*-Header in der SOAP-Nachricht identifiziert zunächst die Sequenz, zu der die Nachricht gehört. Das *wsrm:MessageNumber*-Element zeigt, dass dies die erste Nachricht der Sequenz ist und das *wsu:Expires*-Element gibt an, bis zu welchem Zeitpunkt die Sequenz gültig ist.

Die WS-ReliableMessaging-Implementierungen auf Sender- und Empfängerseite stellen nun sicher, dass Nachrichten quittiert werden. Dazu definiert die Spezifikation das Format für ein Acknowledgement, das der Empfänger dem Sender zur Bestätigung zusendet. Der Sender wartet auf die Bestätigung und sendet die Nachricht gegebenenfalls so lange noch einmal, bis er die Bestätigung erhält.

Zur Vermeidung unnötiger Netzwerkkommunikation werden dabei verschiedene Strategien eingesetzt. Zunächst einmal bestätigt der Empfänger nicht jede Nachricht einzeln, sondern kann in einem Acknowledgement ganze (nicht zwingend zusammenhängende) Bereiche quittieren – also zum Beispiel zurückmelden, dass Nachrichten 1-3, 5 und 10-12 korrekt eingegangen sind. Ein Acknowledgement kann durchaus auch in eine Antwortnachricht eingebettet werden („Piggybacking") – trotz asynchroner Kommunikation ist immer noch ein Request/Response-Interaktionsstil denkbar. Auch negative Bestätigungen sind möglich – in diesem Fall werden nicht die empfangenen Nachrichten quittiert, sondern die fehlenden Nachrichten aufgelistet.

Der Sender wartet einen konfigurierbaren Zeitraum lang auf die Bestätigung, bevor er Nachrichten erneut sendet. Die Strategie, gemäß welcher er das Senden wiederholt, ist konfigurierbar.

Eine Bestätigung ist in Listing 4 dargestellt. Eine Sequenz wird terminiert, wenn eine Nachricht einen *wsrm:LastMessage*-Header enthält.

Listing 5

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<wsp:PolicyAttachment xmlns:wsrm="http://schemas.
                        xmlsoap.org/ws/2004/03/rm"
      xmlns:wsp="http://schemas.xmlsoap.org/
                        ws/2002/12/policy">
 <wsp:AppliesTo>
  <wsrm:SequenceRef Match="Prefix">
   <wsu:Identifier>http://source.example.org
                        </wsu:Identifier>
  </wsrm:SequenceRef>
 </wsp:AppliesTo>
 <wsp:PolicyReference
          Ref="http://example.org/wsrmPolicy.xml"/>
</wsp:PolicyAttachment>
```

Implementierungen

Zum Zeitpunkt der Erstellung dieses Beitrags existierten folgende WS-ReliableMessaging-Implementierungen:

- Microsoft stellt für MSDN-Subscriber als Teil der Indigo-Messaging-Infrastruktur [7] eine WS-ReliableMessaging-Implementierung bereit. Für die nächste Version der Web Services Extensions (WSE) – einem Toolkit, mit dem Microsoft Funktionen aus der Indigo-Infrastruktur bereits vorab zur Verfügung stellt – ist die Unterstützung der aktuellen Version der Spezifikation geplant. Eine experimentelle Implementierung findet sich außerdem in [8].
- IBM bietet mit dem Emerging Technology Toolkit [9] eine Preview-Implementierung einer Vielzahl von Web Services-Standards an. Diese ist primär darauf ausgelegt, der Entwicklergemeinde bereits jetzt eine Umgebung zur Verfügung zu stellen, die das Experimentieren mit den Standards erlaubt.
- Die Version 5.0 des Systinet WASP Server for Java [10] unterstützt WS-ReliableMessaging (zum Zeitpunkt der Erstellung noch in Beta).
- Bei der Apache Software Foundation findet sich eine Open Source-Implementierung, allerdings noch in einem sehr frühen Stadium [11].

Standardisierung, WS-ReliableMessaging und WS-Reliability

Web Services werden von den Herstellern als Lösung aller Interoperabilitätsprobleme angepriesen. Auf Basis klar definierter Standards, an die sich alle halten, können Anwender auf beliebige Kombinierbarkeit von Produkten unterschiedlichster Partner vertrauen. – So viel zur Theorie.

In der Realität handelt es sich bei vielen der Standards – unter anderem auch bei all denen, die in diesem Beitrag erwähnt werden – nur um Spezifikationen. Insbesondere Microsoft und IBM veröffentlichen mehr oder weniger detaillierte Beschreibungen von Protokollen, ohne diese jedoch im Rahmen eines Standardisierungsprozesses bei einem der dafür vorgesehenen Gremien einzureichen.

Beide Hersteller haben zwar entsprechende Absichtserklärungen veröffentlicht, wollen aber offensichtlich zum einen die Kontrolle über die Spezifikationen behalten und zum anderen – wobei man ihnen dieses kaum verübeln kann – die schnelle Weiterentwicklung der Technologie garantieren.

Leider führt dies in manchen Fällen zu Überschneidungen, wie z.B. im Fall WS-ReliableMessaging und WS-Reliability [12]. WS-ReliableMessaging stammt von IBM, Microsoft, BEA und Tibco; WS-Reliability wurde von Sun, Oracle, Fujitsu, Hitachi und NEC veröffentlicht, die ihre Variante mittlerweile bei der OASIS zur Standardisierung eingereicht haben. Um die Verwirrung zu komplettieren, verwendet das OASIS-Komitee die Abkürzung WSRM, die äußerst leicht zu Verwechslung mit der für WS-ReliableMessaging üblichen Abkürzung WS-RM führen kann.

In allen wesentlichen Aspekten sind beide Spezifikationen – zumindest aus Anwendersicht – nahezu gleichwertig (eine sehr gute Analyse findet sich unter [13]). Es besteht durchaus Grund zur Hoffnung, dass beide Standards langfristig konsolidiert werden. Im Web Services-Umfeld scheint es nicht zu gewagt, auf IBM und Microsoft zu setzen und die Prophezeiung zu riskieren, dass der Name der konsolidierten Spezifikation wahrscheinlich WS-ReliableMessaging sein wird.

Policy

Bereits mehrfach erwähnt, aber noch nicht näher erläutert wurde die Konfigurierbarkeit des WS-ReliableMessaging-Protokolls. Hierzu wird ein weiterer Baustein der WS-Architektur eingesetzt: Das Framework WS-Policy [5]. Dieses definiert *Policy Assertions*, d.h. Zusicherungen über ein bestimmtes Verhalten, die deklarativ mit Web Services assoziiert werden. Wie diese Verknüpfung geschieht, ist in der WS-Policy-Attachment-Spezifikation definiert. Die Art und Weise, wie die Assertions formuliert werden, legt die WS-PolicyAssertion-Spezifikation fest.

Eine umfassendere Erläuterung von WS-Policy würde den Rahmen dieses Beitrags sprengen, daher soll auch hier ein Beispiel zur Verdeutlichung genügen:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<wsp:Policy xmlns:wsrm="http://schemas.xmlsoap.org/
    ws/2004/03/rm" xmlns:wsp= "http://schemas.xmlsoap.
                                org/ws/2002/12/policy"
                                Name="DefaultProfile" >
 <wsrm:InactivityTimeout Milliseconds="86400000"
                            wsp:Usage="wsp:Observed"/>
 <wsrm:BaseRetransmissionInterval Milliseconds="3000"
                            wsp:Usage="wsp:Observed"/>
 <wsrm:ExponentialBackoff wsp:Usage="wsp:Observed"/>
 <wsrm:AcknowledgementInterval Milliseconds="1000"
                            wsp:Usage="wsp:Observed"/>
</wsp:Policy>
```

Die Informationen in dieser Konfiguration legen fest, dass eine Sequenz nach 24 Stunden Inaktivität (86.400.000 Millisekunden) automatisch terminiert wird, Nachrichten nach drei Sekunden ohne Bestätigung erneut übertragen und nach jeweils eine Sekunde eine Bestätigung für die bis dahin eingegangenen Nachrichten versandt wird. Außerdem werden erneute Sendeversuche immer mehr verzögert (*ExponentialBackoff*), um ein bereits überlastetes Netzwerk nicht noch mehr zu belasten.

WS-Policy-Ausdrücke können im Falle von WS-ReliableMessaging sowohl mit einzelnen Sequenzen als auch mit allen Sequenzen, deren Identifikationen ein gemeinsames Präfix haben, verknüpft werden. Dies wird in Listing 5 dargestellt.

Einsatzgebiete

WS-ReliableMessaging kann unterschiedlich umgesetzt werden. Eine Implementierung der einfachsten Form setzt „nur" das Protokoll um, nimmt den Diensten und Dienstnutzern die Aufgabe ab, sich selbst um das Verwalten von Acknowledgements, Timeouts, erneuten Versuchen usw. zu kümmern und stellt zumindest sicher, dass Fehler bei der Zustellung von Nachrichten erkannt werden. Da das Protokoll auf den Web Services-Technologien SOAP und HTTP aufsetzt, kann es problemlos auch über Unternehmensgrenzen hinweg eingesetzt werden.

Anbieter von Messaging-Systemen – seien es bereits etablierte oder neu im Web Services-Umfeld entstehende – können über das WS-ReliableMessaging-Protokoll Nachrichtenversender und -empfänger unabhängig von der Technologie, mit der diese implementiert sind, anbinden. In diesem Fall können Fehler nicht nur erkannt, sondern gegebenenfalls auch vom Messaging-System behoben werden.

Der letzte – und vielleicht wohl wichtigste – Einsatzbereich ist die Interoperabilität zwischen Messaging-Systemen der unterschiedlichen Anbieter, die WS-ReliableMessaging als herstellerübergreifenden Standard nutzen können, um ihre jeweiligen Systeme aneinander zu koppeln. Mittlerweile sind die ersten Implementierungen der Spezifikation verfügbar (siehe Kasten „Implementierungen" auf Seite 40); die wichtigsten Anbieter haben die Unterstützung klar geplant.

Fazit

Wie die meisten Web Services-Standards adressiert WS-ReliableMessaging kein neues Problem mit einem bahnbrechenden, neuen Lösungsansatz, sondern definiert eine Abbildung erprobter Problemlösung in die Web Services-Protokollwelt. Durch die Standardisierung auf Basis des Kommunikationsprotokolls wird eine Verknüpfung mit einer bestimmten Technologie vermieden und die Interoperabilität unterschiedlichster Systeme ermöglicht. WS-ReliableMessaging stellt somit eine wesentliche Komponente für die lose gekoppelte Kommunikation zwischen autarken Systemen dar. ●

Links & Literatur

[1] msdn.microsoft.com/ws/2004/03/ws-reliablemessaging/

[2] www-106.ibm.com/developerworks/library/specification/ws-add/

[3] msdn.microsoft.com/library/en-us/dnglobspec/html/ws-eventing.asp

[4] www-106.ibm.com/developerworks/library/specification/ws-notification/

[5] www-106.ibm.com/developerworks/library/ws-polfram/

[6] msdn.microsoft.com/library/en-us/dnglobspec/html/ws-secureconversation.asp

[7] msdn.microsoft.com/Longhorn/understanding/pillars/Indigo/default.aspx

[8] www.brains-n-brawn.com/default.aspx?vDir=cfreliable

[9] www.alphaworks.ibm.com/tech/ettk

[10] www.systinet.com/products/java_ws

[11] ws.apache.org/ws-fx/sandesha/

[12] otn.oracle.com/tech/webservices/htdocs/spec/ws-reliability.html

[13] xml.coverpages.org/Chappell-WSRelSOAP.pdf