von andrew doble und
andreas elting

VERSIONIERUNG VON SERVICES: SOA FÜR FORTGESCHRITTENE

Obwohl viel über „Service Oriented Architecture" (SOA) geredet wird, haben nur wenige Unternehmen wirklich eine solche implementiert. Mit der Umsetzung einer SOA zu beginnen, ist eine Sache. Aber gerade in den ersten Phasen eines solchen Projektes kann man eine Reihe von Fehlern machen, die sich später rächen. Der Artikel diskutiert eines der Probleme, denen sich ein SOA-Architekturteam stellen muss und das nicht zu verachten ist: Wie kann man sicherstellen, dass eine Änderung an einem Service keinen (potenziell sehr teuren) nachteiligen Effekt auf die Anwender (Clients) des Service hat?

SOA hat sich in den letzten Jahren zu einem echten Hype entwickelt – und das aus gutem Grund. IT-Abteilungen müssen sich mit unterschiedlichen Fragestellungen herumschlagen:

- steigende Integrationskosten und
- immer größer werdender Druck, auf sich ändernde Geschäftsprozesse mit IT-Unterstützung schnell zu reagieren.

Da ist ein Heilbringer wie SOA mehr als willkommen. Die Marktakzeptanz von Standards wie SOAP und der Erfolg des Word Wide Web Consortiums (W3C) und OASIS haben die IT-Fachleute schnell davon überzeugt, dass die nötige Technologie zur Umsetzung einer SOA bereits heute existiert und weit verfügbar ist. Die meisten Tool-Hersteller auf diesem Sektor demonstrieren heutzutage, wie mit einem Klick auf einen einzigen Knopf ihrer Benutzungsoberfläche schnell ein *Web-Service (WS)* erzeugt werden kann.

Hinter all dieser Euphorie liegt jedoch die schwere Aufgabe, für ein Unternehmen wirklich funktionierende Systeme zu entwickeln, die den komplexen Anforderungen des Geschäfts oder der Organisation genügen. Einige dieser Anforderungen sind konzeptionelle Fragestellungen wie die, wie die geforderte Geschäftsfunktionalität in Services strukturiert werden soll und ob dies überhaupt geht. Andere Fragestellungen beschäftigen sich damit, wie eine SOA betrieben und gewartet werden kann. Obwohl viele Tool-Hersteller jetzt damit beginnen, auch Produkte anzubieten, die sich mit dem Betrieb von SOA beschäftigen (WS-Management-Werkzeuge), gibt es zur Zeit am Markt nur wenige Hersteller, die sich mit der Fragestellung beschäftigen, wie eine SOA werkzeuggestützt weiterentwickelt werden kann.

Die Weiterentwicklung einer SOA steht anfangs (leider) nicht ganz oben auf der Prioritätenliste eines Teams, das sich mit der Einführung einer SOA beschäftigt. Das Team konzentriert sich darauf nachzuweisen, dass die geplante SOA entweder die IT-Kosten senken kann, die Flexibilität bei Änderungen der Geschäftsprozesse erhöht oder beides. Um dies zur erreichen, greifen die Entwickler tief in die Trickkiste der WS-Standards und -Tools. Über den Nachweis, mit einem Knopfdruck schnell einen WS erzeugen zu können, wird suggeriert, dass dies auch für die Erzeugung ganzer SOA-Architekturen ähnlich schnell gelingen wird. Die Weiterentwicklung der SOA in Betracht zu ziehen und dem Management deren Auswirkungen aufzuzeigen, erscheint dem Team nicht notwendig. Durch die Verwendung von Web-Services sind viele Entwickler davon überzeugt, dass die Systeme, die sie entwickeln, lose gekoppelt sind (eines der Hauptversprechen der SOA) und dass die Weiterentwicklung damit kein Problem darstellen wird. Das böse Erwachen kommt dann, wenn einer der bereits implementierten Services geändert werden muss. In diesem Fall muss sich der Entwickler folgende Fragen stellen:

- Müssen alle Clients dieses Service (in SOA-Terminologie die *Service Consumer*) jetzt auch geändert werden?
- Wenn ja, wie kann dies inkrementell ausgeführt werden? Schließlich wird es gerade in größeren Unternehmen nicht möglich sein, alle Clients auf einen Schlag zu ändern.

In einem lose gekoppelten System sollte dies kein Problem sein. Existiert dieses Problem also beim Einsatz von WSs im Rahmen einer SOA eventuell gar nicht?

Kompatibilitätsfragen

Am besten nähern wir uns der Fragestellung, wie und warum ein Dienst, der als WS implementiert wurde, weiterentwickelt wird, anhand eines Beispiels:

Ein Einzelhändler ist in den letzten Jahren immer mehr gewachsen und war mit der Reaktionszeit der IT-Abteilung auf die neuen Anforderungen sehr unzufrieden. Die Umsetzungszeit in IT-Lösungen war einfach zu lang. Außerdem verfolgte das Unternehmen eine aggressive Übernahmestrategie und kaufte in der Vergangenheit andere Unternehmen auf. Die IT-Abteilung brauchte nach Meinung des Managements viel zu lange, um die erworbenen Systeme zu integrieren. Um diese Situation zu verbessern, schlug die IT-Abteilung vor, die vorhandene IT-Landschaft in eine SOA zu überführen, bekam das nötige Geld und beauftragte ein Entwicklungsteam mit der Umsetzung.

So weit war die umgesetzte SOA ein großer Erfolg. Als ein Beispiel dieses Erfolgs sei der Service genannt, der die Rabatte für bestimmte Kunden ermittelt. Zuerst wurde dieser nur für die Erstellung der Rechnungen eingesetzt. Nun konnte derselbe Service im neuen web-basierten Bestellsystem, dem Callcenter-System (für eine Online-Bestellung) und sogar für die neuen

die autoren

Andrew Doble (E-Mail: adoble@csc.com) ist Chief System Architect bei CSC Ploenzke mit den Arbeitsschwerpunkten Konzeption und Erstellung von agilen unternehmensweiten Architekturen, SOA und Businessprozess-Management.

Andreas Elting (E-Mail: Andreas.Elting@csc.com) ist Solution Architect bei CSC Ploenzke und spezialisiert auf Vorgehensmodelle zur Erstellung von Prozess- und IT-Architekturen, MDA und UML 2.0 im Rahmen von Businessprozess-Management.

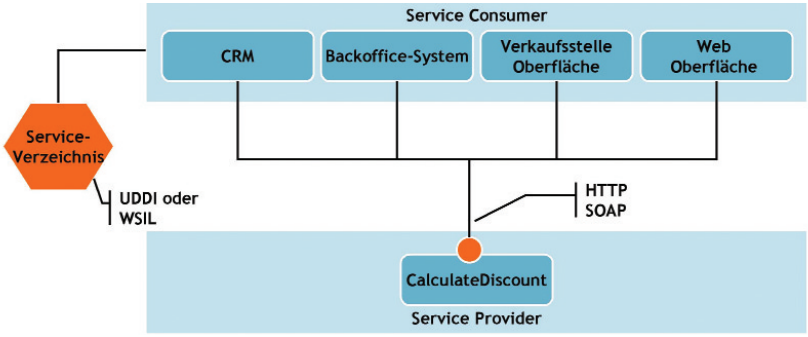

Abb. 1: Ein erster Wurf der SOA (Ausschnitt)

Verkaufsstellen des Einzelhändlers genutzt werden. In jeder dieser neuen Verkaufsmöglichkeiten wurden Kosteneinsparungen erreicht, da der gleiche Code wieder verwendet werden konnte (CalculateDiscount), der zudem als WS implementiert wurde. Abbildung 1 zeigt einen Teil der SOA für diese Fragestellung.

Um den Rabatt (*Discount*) zu berechnen, sendet der Client eine Nachricht an die Operation CalculateDiscount, deren vereinfachtes Schema in Listing 1 angegeben ist.

Der Einzelhändler will nun auch Geschäftskunden ansprechen und benötigt eine komplett unterschiedliche Methode, um den Rabatt für diesen Kreis zu berechnen. Um dieses Problem zu lösen, entwickelt das IT-Team eine neue Operation CalculateCommercialDiscount, die den gleichen Nachrichtentyp verwendet wie vorher. Neue Operationen zu einem Service hinzuzufügen ist rückwärtskompatibel. Daher funktionieren die Aufrufe der Privatkunden wie vorher und die sukzessive Migration zum neuen WS kann erfolgreich durchgeführt werden. Kurz gesagt: Eine Weiterentwicklung der SOA wurde erfolgreich durchgeführt.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/
            XMLSchema">
  <xs:element name="customer-id" type="xs:integer"/>
</xs:schema>
```

Listing 1

Erfreut über diesen Erfolg will der Einzelhändler nun bestimmte Zielgruppen – z. B. Teenager, ältere Menschen, bestimmte Einkommensgruppen, kleine Unternehmen, große Unternehmen – gezielter ansprechen. Jede dieser Zielgruppen soll eine eigene Rabattstruktur bekommen. Dem Entwicklungsteam wird schnell klar, dass es zu dem Service nicht einfach neue Operationen hinzufügen kann, da sonst eine Explosion von Operationen ins Haus stünde. So wurde beschlossen, eine einzige Operation CalculateDiscount einzusetzen, die Nachricht aber um eine Typangabe des Kunden zu erweitern. Die vorhandene Operation CalculateCommercialDiscount soll ausgemustert werden.

Das neue XML-Schema ist in Listing 2 angegeben.

Einem XML-Schema ein neues Element hinzuzufügen, ist ebenfalls eine rückwärtskompatible Änderung. Das bedeutet, dass die Dienstanbieter (*Service Provider*), die dieses Element nicht benötigen, es einfach nicht beachten werden (wenn sie geeignet entwickelt wurden). Dies hat den zusätzlichen Vorteil, dass die Clients nicht alle zur gleichen Zeit geändert werden müssen.

Das Entfernen der alten Operation CalculateCommercialDiscount ist jedoch nicht rückwärtskompatibel. Um die Mühe zu vermeiden, die Clients sofort alle ändern zu müssen, beschließt das Entwicklungsteam, die alte Operation erst mal bestehen zu lassen und die Umstellung über einen längeren Zeitraum hinweg vorzunehmen (analog zu den depricated Operationen in Java). Obwohl dies es dem Team gestattet, die neue Funktionalität einzuführen, muss es die alte damit weiter warten. Das wird erst mal in Kauf genommen.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/
            XMLSchema">
  <xs:element name="customer-id" type="xs:integer"/>
  <xs:element name="customer-type" type="xs:string"/>
</xs:schema>
```

Listing 2

In einem Gespräch mit der Marketingabteilung des Unternehmens stellt sich heraus, dass sich die Ansprache bestimmter Zielgruppen noch weiter und spezifischer auf mehr Gruppen ausweiten wird. Damit würde der im XML-Schema definierte CustomerType immer komplexer werden. Das Team beschließt, den Typ nun über einen complexType statt über ein einfaches Element zu implementieren (siehe Listing 3)

Nun kommt es zum bösen Erwachen. Die Änderung eines XML-Schemas in diesem Ausmaß ist keine bloße Erweiterung mehr und damit schon gar nicht rückwärtskompatibel. Das bedeutet, dass keiner der bestehen Clients noch funktionieren wird, was das Entwicklungsteam in eine Zwickmühle bringt. Die Entwickler könnten fordern, dass alle Clients geändert werden, um mit dem neuen XML-Schema klarzukommen, was eine Verzögerung der neuen Marketingstrategie bedeuten würde. Dies ist jedoch schwer zu vertreten, da die IT-Abteilung mit dem Versprechen, über eine SOA schnell und flexibel auf sich ändernde Geschäftsprozesse reagieren zu können, das Budget für die Umsetzung der SOA überhaupt genehmigt bekommen hatte. Eine solche Situation kann dazu führen, dass das Management schnell zur Überzeugung kommt, dass SOA eine Entwicklung ist, die zwar im Elfenbeinturm funktioniert, aber nicht zum Unternehmenserfolg beiträgt. Das Management ist also versucht, alle Projekte, die sich damit beschäftigen, einzustellen bzw. gar nicht erst zu starten.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/
            XMLSchema">
  <xs:element name="customer-id" type="xs:integer"/>
  <xs:complexType name="customer-type">
    ...
  </xs:complexType>
</xs:schema>
```

Listing 3

Versionierung von Services

Was hätte das Entwicklungsteam tun können, um die Situation zu vermeiden? Eine – zugegebenermaßen radikale – Lösung wäre es, mit einem XML-Schema zu starten, das es gestattet, eine beliebige Datenstruktur zu verwenden, z. B. über

```
<xsd:any namespace="##any" minOccurs="0"/>
```

als einzige Elementspezifikation im XML-Schema. Dies garantiert, dass das System wirklich lose gekoppelt ist. Ein solcher Ansatz hat aber viel Nacheile. Ein paar seien hier genannt:

- Die Clients können so verändert werden, dass sie jede XML-Struktur verwenden, die die Entwickler für nötig halten. Das kann dazu führen, dass die Dienstanbieter die Anfrage nicht mehr bearbeiten können. Damit entsteht eine andere Art von Rückwärtsinkompatibilität im System
- Die Antworten des Dienstanbieters können ebenfalls frei strukturiert sein, was zum umgekehrten Effekt führt: Die

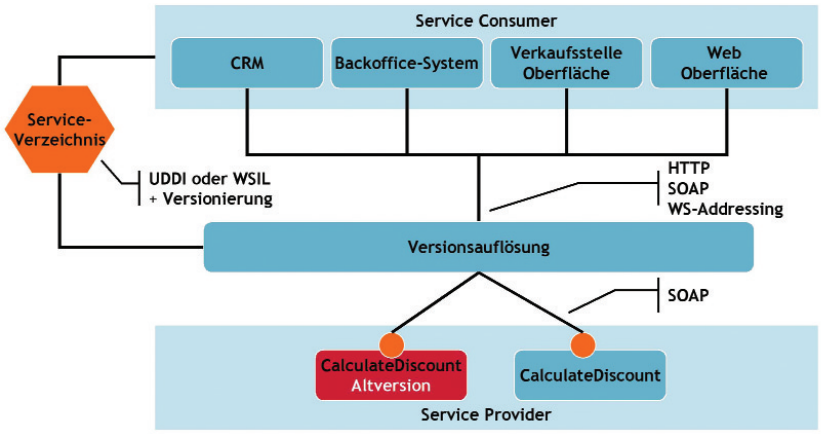

Abb. 2: Die verbesserte SOA mit Versionierungsverfahren (Ausschnitt)

Nutzer eines Service können die Antwort nicht verarbeiten und müssen geändert werden. Auch das ist keine Verbesserung der Situation.

- Entwickler nutzen das XML-Schema unter anderem, um automatisch das Frontend für die Dienstanbieter und die Dienstnutzer (*Service Consumer*) zu erzeugen. Das muss nun manuell erledigt werden, was sich nachteilig auf die Erstellungsdauer auswirkt.

Eine andere Lösungsmöglichkeit wäre es, einen Versionierungsmechanismus bereits in den frühen Phasen der SOA in Betracht zu ziehen. Dies ermöglicht es dem Entwicklungsteam, einen WS rückwärtskompatibel zu ändern und es den Clients dennoch zu gestatten, die alte Version zu nutzen.

Aus Architektursicht bedeutet dies:

- Die Clients müssen die Version des Service spezifizieren, den sie nutzen wollen.
- Ein Verzeichnis (*Directory*) wird benötigt, das es erlaubt, den richtigen Endpunkt eines Service aus der Versionsinformation abzuleiten. Dies könnte eine Erweiterung der bestehenden *Directory*-Technologien wie *UDDI (Universal Description, Discovery and Integration)* oder *WSIL (Web Services Inspection Language)* sein. Es könnte auch über einen eigenen Verzeichnisdienst implementiert werden.
- Der Versionsauflösungsmechanismus fungiert als ein *Gateway* zu den tatsächlichen Diensten. Der Mechanismus nutzt die Versionsinformation des Clients und die Endpunktinformation aus dem Verzeichnis, um die korrekte Version des zu verwendenden Service zu ermitteln.

Abbildung 2 zeigt den relevanten Teil der nötigen Architektur.

Diese Vorgehensweise hätte das Entwicklungsteam aus unserem Beispiel gar nicht erst in die Zwickmühle gebracht. Rückwärtskompatible Änderungen hätten an jedem Service vorgenommen werden können, solange er korrekt versioniert ist. Clients verwenden die für sie „richtige" Version und können sukzessive weiter entwickelt werden, um die neuen Dienste zu nutzen.

Nachteil dieser Lösung ist, dass es derzeit keine explizite Unterstützung für eine solche Versionierung bei WSs auf dem Markt gibt. Um dieses Problem zu lösen, wird vorgeschlagen, dass die Namensräume (*Namespaces*), die sich auf die Nachrichtentypen (*MessageTypes*) in der WSDL beziehen, versioniert werden. In unserem CalculateDiscount-Beispiel könnte die WSDL so aussehen wie in Listing 4 angedeutet.

Die Versionierungsinformation findet sich als Datumsangabe in „2005/02/09", das in den Namensraum eingearbeitet wurde. Jeder Client, der eine spezielle Version des Service nutzen will, muss dies zu dem Namensraum der Nachricht hinzufügen, die an den Service gesendet werden soll. Der Versionierungsmechanismus liest den Namensraum aus, extrahiert die Versionsinformation, ermittelt den zugehörigen Endpunkt des Service und modifiziert den SOAP-Endpunkt so, dass die korrekte Version der Operation aufgerufen wird.

Was ist schlecht an dieser Vorgehensweise? Das Hauptproblem besteht darin, dass die Versionierung über eine Namenskonvention erreicht wird. Eine solche in einem großen Unternehmen zu etablieren, hat sich in der Vergangenheit als sehr schwer herausgestellt. Unternehmen (und ihre IT-Landschaften) wachsen organisch, d. h. ursprünglich isolierte Inseln werden über Firmenakquisitionen zusammengeführt. Auch das Management kann darauf hinwirken, dass durch eine Zusammenlegung von Abteilungen neue Lösungen entstehen müssen, um die IT-Kosten zu reduzieren.

Oben gezeigte Namenskonvention könnte sich zudem bei einer internationalen Akquisition als schwierig erweisen. Europäische Unternehmen interpretieren die Datumsinformation als den 9. Februar, US Unternehmen könnten diese als den 2. September lesen. Verkompliziert wird die Situaion noch dadurch, dass sowohl die europäische als auch die amerikanische Organisation bereits Versionierungsverfahren etabliert haben könnten, die jetzt zusammengeführt werden müssen. Dies könnte sich als nicht gerade einfach erweisen. SOA ist dabei keine große Unterstützung.

Im nächsten Abschnitt stellen wir eine mögliche Lösung über einen relativ neuen Standard bei WSs vor.

Der WS-Adressierungsstandard

In einfachen SOA-Lösungen wird als einziges Transportprotokoll HTTP eingesetzt. Das

```
<types>
  <schema
targetNamespace="http://retailer.com/2005/
            02/09/DiscountRequest.xsd"
     xmlns="http://www.w3.org/2000/10/XMLSchema">
    <xs:element name="customer-id" type="xs:integer"/>
    <xs:element name="customer-type" type="xs:string"/>
  </schema>
</types>
```

Listing 4

bedeutet, dass Ziel- und Antwort-Adressaten relativ klar sind. Die URI des HTTP-Anfrage ist die Zieladresse der Nachricht und die Antwort erfolgt verpackt über das HTTP-Anwortpaket. Dieses einfache Modell funktioniert allerdings dann nicht mehr, wenn viele Transportmedien eingesetzt werden, um SOAP-Nachrichten zu übermitteln. Ein Beispiel dafür könnte sein, dass die Nachrichten über ein *Message-Queueing*-System transportiert werden. Die Antwortadressen könnten in diesem Fall über *JMS (Java Message Services)* spezifiziert werden oder in die Nachricht selbst eingebettet sein.

Den Transportmechanismus selbst zur Adressbestimmung nutzen, entspricht folgendem Beispiel: Sie versenden einen Brief in einem nicht adressierten Umschlag und sagen dem Postboten, wohin er den Umschlag liefern soll und wohin und wie die Antwort gesendet werden soll. Das könnte zwischen München und Frankfurt funktionieren. Wenn der Brief aber zwischen Paris und München verkehren soll und der Postbote kein Französisch spricht,

dann würde die Information mit hoher Wahrscheinlichkeit verloren gehen.

Ein weiterer Grund, warum das einfache Adressierungsschema erweitert werden muss, liegt dann vor, wenn mehr als eine Anfrage an die gleiche Instanz eines WS gesendet werden soll. In den Zeiten vor dem WS-Adressierungsstandard gab es keinen geregelten Weg, dies zu erreichen. Eine typische Lösung bei einem Einkaufssystem wäre es, etwas wie ?cart-id=4711 an die URL anzuhängen, was aber nicht funktioniert, wenn die Nachricht über verschiedene Transportmechanismen verschickt wird. In diesem Fall gibt es keinen standardisierten Weg, den Einkaufskorb zu repräsentieren.

Wie geht nun der WS-Adressierungsstandard mit diesem Problem um? Ein Schlüsselelement der Spezifikation ist die Referenzierung des Endpunkts. Betrachten wir wieder den WS aus dem Einkaufsbeispiel. Bei der Erzeugung eines Einkaufwagens könnte ein WS einen WS-Adressierungsendpunkt zurückliefern, der sowohl den Ort des Einkaufswagenservices als auch zusätzliche Informationen über den Einkaufswagen zurückliefert. Eine SOAP-Antwort könnte wie in Listing 5 angedeutet aussehen.

Die Referenz auf den Endpunkt ist also mehr als eine einfache URL. Zusätzlich zur Adresse kann eine Menge von Referenzeigenschaften (ReferenceProperties) spezifiziert werden. In diesem Fall ist dies die Referenz auf den Einkaufswagen, der gerade angelegt wurde.

Wie kann der gerade erhaltene Endpunkt in einer SOAP-Nachricht eingesetzt werden? Die WS-Adressierung spezifiziert hierfür eine Menge von Regeln, wie die Elemente des Endpunkts auf einen SOAP-Aufruf abgebildet werden. Listing 6 zeigt hierfür ein Beispiel anhand des Einkaufkorbs.

Nachdem die Referenz auf den Endpunkt empfangen wurde, können nachfolgende Aufrufe des Service den Endpunkt verwenden und so sicherstellen, dass der Aufruf an die korrekte Instanz des Service geht.

Einsatz der WS-Adressierung zur Versionierung

Eine interessante Möglichkeit für die Versionierung von WSs besteht darin, die oben genannten Referenzeigenschaften eines Endpunkts zu nutzen, um diesen näher zu beschreiben. Auf das Beispiel des Einzelhändlers übertragen könnte die Referenz auf eine spezielle Version des Rabattservice so aussehen, wie in Listing 7 angegeben.

Der Namensraum wsv bezieht sich auf dien aktuellen Namensraum der *Tags*, die

```
<soap:body>
  <retailer:CartReference>
<wsa:Address>http://retailer.com/CartService</
            wsa:Address>
    <wsa:ReferenceProperties>
      <retailer:cart-id>4711</retailer:cart-id>
    </wsa:ReferenceProperties>
  </retailer:CartReference>
</soap:body>
```

Listing 5

verwendet wurden, um die Version des WS zu beschreiben. Dieser Namensraum könnte beispielsweise vom Entwicklungsteam des Versionierungssystems entwickelt worden sein. Auf diese Weise ist es relativ einfach für den Auflösungsmechanismus, die Versionierungsinformation zu extrahieren, den zugehörigen Endpunkt des WS im Service-*Directory* zu finden und die Ziel-URL geeignet anzupassen.

Die WS-Adressierung auf diese Weise einzusetzen hat eine Reihe von Vorzügen:

- Jede Unklarheit darüber, was die einzelnen Teile des Versionsschemas bedeuten, ist schnell aufgelöst.
- Wenn weitere Versionierungsschemas hinzukommen, erhalten sie andere Namensräume. Das Versionierungssystem kann diese dann entweder selbst bearbeiten oder an ein weiteres System übergeben, das die *Tags* in diesem Namensraum versteht und verarbeitet. Auf diese Weise kann die SOA weiterentwickelt werden, ohne vorhandene Clients zu beeinflussen.

Fazit

Wir empfehlen jedem Team, das sich mit der Entwicklung einer SOA beschäftigt, sich sehr

```
<soap:header>
  <wsa:To>http://retailer.com/CartService</wsa:TO>
  <retailer:cart-id>4711</retailer:cart-id>
 . . .
</soap:header>
<soap:body>
. . .
</soap:body>
```

Listing 6

früh mit dem Thema der Versionierung von Services zu beschäftigen. Obwohl der Einsatz der WS-Adressierung eine Teillösung des WS-Versionsproblems darstellt, bleibt die Tatsache, dass die Versionierung von Web-Services ein Thema ist, mit dem sich die diversen Gremien bisher nur wenig beschäftigt haben. Da dies der Fall ist, bieten nur sehr wenig Tool-Verkäufer Lösungen für das Versionierungsproblem an – und wenn, dann handelt es sich um proprietäre Ansätze. Daher sollten Entwickler versuchen, das Versionierungssystem so zu erstellen, dass es offen für Veränderungen ist. Das bedeutet unter anderem,

- sicherzustellen, dass Clients ihren Dienstanbietern die Version in einer Weise übergeben, die später leicht auf andere Versionierungsschemata abgebildet werden kann,
- die Details des tatsächlich implementierten Versionierungsverfahren hinter einem Gateway zu verbergen.

```
<wsv:ServiceReference>
<wsa:Address>http://retailer.com/DiscountService</
            wsa:Address>
    <wsa:ReferenceProperties>
      <wsv:version>
        <wsv:year>2005</wsv:year>
        <wsv:month>2</wsv:month>
        <wsv:day>9</wsv:day>
      </wsv:version>
    </wsa:ReferenceProperties>
</wsv:ServiceReference>
```

Listing 7

Dieser Beitrag hat eine Möglichkeit skizziert, wie eine Auflösung von Versionsinformationen durchgeführt werden kann. Allerdings bleibt noch eine Reihe von Problemen bestehen. Wie können z. B. bestehende WS-Verzeichnisse so erweitert werden, dass eine Versionierung mit ihnen möglich ist? Oder sind in diesem Fall neue Verzeichnisse nötig? Wie können Versionsabhängigkeiten zwischen Clients und Dienstanbietern verwaltet werden? Können WSs so gebaut werden, dass sie einerseits die eben beschriebenen Fragestellungen lösen und andererseits immer noch wirklich lose gekoppelt sind? Sie sehen, es gibt noch viel zu tun. ■

Literatur & Links

[Bro04] K. Brown, M. Ellis, Keep your Web services current with WSDL and UDDI, 30.1.04, siehe www-128.ibm.com/developerworks/webservices/library/ws-version/

[Chi03] R. Chinnici et al., Web Services Description Language (WSDL) Version 1.2,Part 1: Core Language, W3C Working Draft, 11.6.03, siehe www.w3.org/TR/2003/WD-wsdl12-20030611

[Dav04] D. Davis, The hidden impact of WS-Addressing on SOAP, 6.4.04, siehe www.106.ibm.com/developerworks/library/ws-address.html

[Gud05-a] M. Gudgin et.al., Web Services Addressing 1.0 – Core Draft, 15.2.05, siehe www.w3.org/TR/2005/WD-ws-addr-core

[Gud05-b] M. Gudgin et. al., Web Services Addressing 1.0 – SOAP Binding, Draft 15.2.05, siehe www.w3.org/TR/ws-addr-soap

[Gud05-c] M. Gudgin et. al., Web Services Addressing 1.0 - WSDL Binding, Draft 15.2.05, siehe www.w3.org/TR/ws-addr-wsdl