



mitp

Friedrich
Kiltz

Java Webservices

Teil A

Bestimmung der Mitte und Stecken der Grenzen

In diesem Teil:

- **Kapitel 1**
Überblick Webservices 23
- **Kapitel 2**
Kontext von Webservices 33

Überblick Webservices

Mit dem Begriff »Webservices« wird nicht immer das Gleiche verbunden. Dieses Kapitel definiert die Webservices, wie sie diesem Buch zugrunde liegen, differenziert die Arten von Webservices und stellt die Aufgaben heraus, die Webservices lösen können und lösen sollten. Zum Abschluss des Kapitels finden Sie noch eine Übersicht der Message Exchange Patterns.

1.1 Was sind Webservices?

Die Frage nach der Erklärung des Begriffs Webservice muss man unter verschiedenen Aspekten betrachten. Ein Aspekt ist die Frage nach dem Warum. Warum sollte man Webservices einsetzen? Was können Webservices, das alternative Konzepte nicht können und was sind die alternativen Konzepte? Im Mittelpunkt des Buches steht die Frage nach dem Wie. Dies liegt an der Zielgruppe dieses Buches, was uns zum dritten Aspekt unserer Frage bringt: Wer ist an der Konzeption und Realisierung von Webservices beteiligt und wie stellt sich seine Sicht dar?

Damit können wir die recht komplexe Frage »Was sind Webservices?« in drei hoffentlich einfachere Fragen aufteilen:

- **Warum** braucht man Webservices?
- **Wie** werden Webservices realisiert?
- **Wer** ist an der Konzeption und Nutzung von Webservices beteiligt?

1.1.1 Warum braucht man Webservices?

Wenn wir davon ausgehen, dass niemand aus Jux und Tollerei ein neues Softwarekonzept wie Webservices entwickelt, sondern dass ganz besondere Aufgaben, die mit vorhandenen Techniken nicht – oder nicht gut – zu lösen waren, der Grund für diese Entwicklung waren, geben uns diese Aufgaben gute Hinweise auf das Wesen von Webservices.

Die zentrale Aufgabe der Webservices ist es, ein verteiltes System bereitzustellen, welches heterogene Programme verbinden kann. Die Heterogenität der Programme bezieht sich dabei sowohl auf die Plattform (Windows, Linux/Unix, Mac) als auch auf unterschiedliche Programmiersprachen (Java, Cobol, C etc.). Diesen Ansatz gibt es schon: CORBA.

CORBA hat das Problem, dass die Beteiligten sich auf eine Objektstruktur einigen müssen. Dies führte zu dem Versuch, ein gemeinsames fachliches Objektmodell einzuführen. Ein Objekt *Kunde* kann doch gar nicht so unterschiedlich sein – oder doch? Ich denke, dass der Otto-Versand doch weit größere Anforderungen an das Objekt *Kunde* hat, als z.B. der lokale Heizöllieferant. Ein weiteres Problem ist, dass CORBA sehr feingranular ist und auf die Ebene der Setter und Getter zurückgeht und die Objekte verteilt »per Remote« anbietet. Dieser Remotezugriff führt zu einer schlechten Skalierbarkeit des Programms. Eine Nutzung der Objekte per Value ist eine Erweiterung von CORBA, aber eigentlich nicht der Grundgedanke von CORBA.

Die Notwendigkeit von Webservices entstand also daraus, dass unterschiedliche Systeme mit unterschiedlichen Objektmodellen gut skalierbar verbunden werden sollen. Gute Skalierbarkeit bedeutet, dass bei einem steigenden Datenverkehr der Ressourcenverbrauch proportional steigt.

Typische Beispiele für die Notwendigkeit von Webservices erkennen wir in folgenden repräsentativen Beispielen:

- Das Webservice-Angebot von Amazon ermöglicht es vielen Nutzern, die Dienste von Amazon in ihre Umgebungen einzubinden und kann damit einen größeren Umsatz erzielen.
- Die Anbindung von Alt- und Neusystemen, wie z.B. einer Host-Anwendung in Cobol mit einer neuen Java-Webapplikation, kann als Insellösung oder als ein Baustein einer SOA-Landschaft teurere Neuentwicklungen vermeiden und vorhandene Ressourcen nutzen.
- International tätige Unternehmen, die mit einer Vielfalt von unterschiedlichen Unternehmen zusammenarbeiten, können ihre Geschäftsprozesse mit Webservices optimieren. Als Beispiel hierfür kann man Traudel als Importeur erlesener Waren mit seinen eingangs erwähnten Anwendungsfällen betrachten.

1.1.2 Wie werden Webservices realisiert?

Aus technischer Sicht ist ein Webservice die Realisierung einer verteilten Anwendung mit HTTP(S) als Transportprotokoll, XML für die Nutzlast (Daten und Zusatzinformationen) im SOAP-Format, XML-Schema für die Beschreibung der Struktur der Daten und WSDL für die Definition der Schnittstelle. Dabei ist es egal, welche Programmiersprache und welche Container benutzt werden. Durch HTTP als Transportprotokoll und XML als Datenformat ist eine Unabhängigkeit weitestgehend sichergestellt.

Als Transportprotokoll kann neben HTTP(S) auch SMTP, TCP oder JMS genutzt werden. Ebenso können die Daten auch in anderen Formaten als XML übertragen werden, z.B. mit JSON. Auch eine binäre Kodierung mit dem Hessian-Protokoll ist verbreitet.

Die folgenden Kapitel befassen sich hauptsächlich mit der Realisierung von Webservices und den dafür notwendigen Technologien. Deshalb möchte ich hier die Realisierung nur aus einer Höhe von 10.000 Fuß betrachten. Grob gesehen, haben wir zwei Hauptaufgaben bei der Realisierung:

- Erzeugung des Webservices
- Nutzung des Webservices

Bei der Erzeugung des Webservices ist die WSDL die zentrale Einheit der Synchronisation von Client und Server oder – besser passend im Webservice-Umfeld – Provider als der Anbieter des Services und Consumer als der Nutzer des Services. Sie enthält die Informationen über die zu übertragenden Datentypen (Parameter und Rückgabewerte) sowie die möglichen Operationen und darüber, wie man diese aufrufen kann. Diese Informationen kann man sowohl auf Provider- als auch auf Consumer-Seite in eine Programmstruktur (z.B. Methode) umwandeln.

Bei der Nutzung des Webservices ist SOAP von zentraler Bedeutung. SOAP umschließt die Daten und die benötigten Informationen, damit der Provider weiß, was er mit den Daten tun soll und der Consumer weiß, woher und warum er diese Daten bekommt. Zusätzlich muss der Provider seine Dienste über ein Transportprotokoll zur Verfügung stellen. Dies tut er meist mit einem Webserver per HTTP. Der Consumer muss nun in der Lage sein, das Transportprotokoll zu nutzen und einen Request zusammenzustellen, ihn zu versenden und bei Bedarf seine Response auszuwerten.

SOAP und WSDL basieren auf XML und somit dreht sich bei der Realisierung von Webservices vieles um die Erzeugung und Verarbeitung von XML.

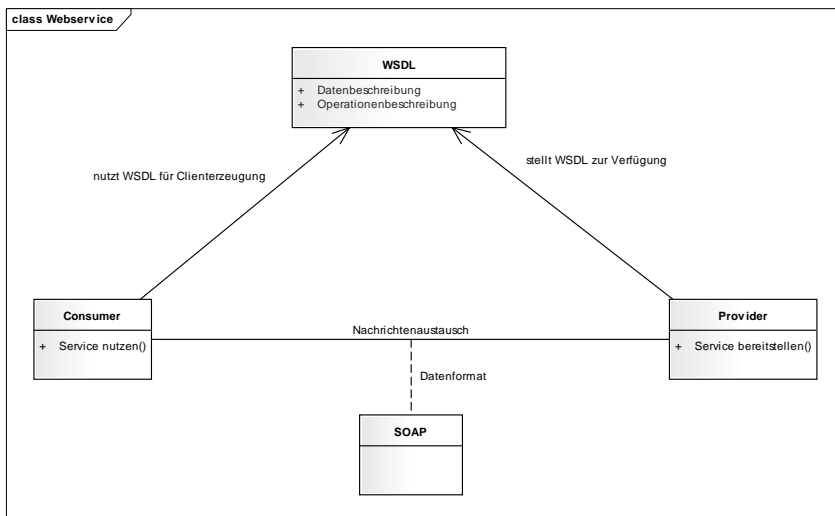


Abb. 1.1: Die Hauptakteure des Webservices aus technischer Sicht

Was soll nun aus technischer Sicht das Revolutionäre sein? Durch die Nutzung von HTTP hat man ein Transportprotokoll, das von fast jedem verstanden wird und das fast jeder nutzen kann. Die Übertragung der Daten per XML hat den gemeinsamen Nenner, dass jeder Textdateien verarbeiten kann und dass man mit XML auch sehr komplexe Datenstrukturen beschreiben und formatieren kann.

Webservices basieren aus technischer Sicht auf fünf Basis-Standards:

- **XML:** Die Datenstruktur der Nutzdaten und Beschreibungsdaten
- **HTTP:** Ein mögliches, viel genutztes Transportprotokoll
- **WSDL:** Die Beschreibung des Services
- **SOAP:** Die übergeordnete Struktur der Nutzdaten
- **UDDI:** Der Verzeichnisdienst

Für weitere Aufgabenbereiche rund um Webservices haben sich weitere Standards gebildet, die meist mit *WS-* beginnen. Berühmte Vertreter sind dabei *WS-Addressing*, *WS-Policy*, *WS-Security*. Diese Standards werden hauptsächlich von den Organisationen W3C, OASIS und WS-I herausgegeben.

Nicht zu vergessen ist REST, einen Architekturstil, den man flapsig auch *Webservice-Light-Technologie* nennen kann. Hier beschränkt man sich auf das Transportprotokoll HTTP (das sowieso in den meisten Fällen genutzt wird) und das Kommunikationsmuster Request/Response (das in sehr vielen Fällen Anwendung findet) und spart sich den SOAP-Umschlag (der in vielen Fällen keinen großen Vorteil bietet). REST kann mit einfachen Bordmitteln einen Großteil der Anforderungen abdecken. Deshalb wird es in diesem Buch auch ausführlich beschrieben.

1.1.3 Wer ist an der Konzeption und Nutzung von Webservices beteiligt?

Wenn man einen Webservice aus verschiedenen Perspektiven betrachtet, sollte man sich auch ansehen, wer diese Perspektiven einnimmt. Die Betrachtung sollte bei der Ermittlung des Bedarfs ansetzen. Aus den Bereichen der Geschäftsprozessanalyse und -modellierung ermitteln die Betriebswirtschaftler die Aufgaben, die von den Webservices erfüllt werden sollen.

Um Wildwuchs zu vermeiden und die Vielzahl der Aufgaben und Lösungsansätze zu koordinieren, benötigen wir Architekten, die eine saubere SOA-Landschaft aufbauen und stabile Schnittstellen beschreiben.

Softwareentwickler setzen nun die Schnittstellen um und nutzen hierzu Frameworks, um die Webservices zu realisieren und zu pflegen.

Nicht vergessen sollte man die Administratoren, die u.a. das Deployment und das Monitoring zu ihren Aufgabenbereichen zählen.

Alle diese Gruppen haben unterschiedliche Anforderungen an Webservices. Die Betriebswirtschaftler und die Architekten sollten sich auf alle Fälle von Bernd

Oestereich et al. das Buch »Objektorientierte Geschäftsprozessmodellierung mit der UML« [Oestereich04] ansehen, das ihnen sehr hilfreich bei der Beschreibung der Aufgaben sein kann. Architekten sei das Buch »SOA in der Praxis« von Nicolai Josuttis [Josuttiso8] ans Herz gelegt. Das vor Ihnen liegende Buch ist im Kern für Softwareentwickler und Programmierer gedacht, welche die Aufgaben mit den Technologien umsetzen.

1.1.4 Der gemeinsame Nenner

Zusammenfassend kann man sich die Definition aus Wikipedia vor Augen führen:

*»Das World Wide Web Consortium definiert die Bereitstellung eines Webservices als Unterstützung zur Zusammenarbeit zwischen verschiedenen Anwendungsprogrammen, die auf unterschiedlichen Plattformen und/oder Frameworks betrieben werden. Ein Webservice oder Webdienst ist eine Software-Anwendung, die mit einem Uniform Resource Identifier (URI) eindeutig identifizierbar ist und deren Schnittstelle als XML-Artefakt definiert, beschrieben und gefunden werden kann. Ein Webservice unterstützt die direkte Interaktion mit anderen Software-Agenten unter Verwendung XML-basierter Nachrichten durch den Austausch über internetbasierte Protokolle.«
(s. [wikipedia])*

Aus dieser Definition kommt noch ein zusätzlicher wichtiger Aspekt in den Vordergrund, nämlich die »Unterstützung zur Zusammenarbeit zwischen verschiedenen Anwendungsprogrammen«. Dies heißt nichts anderes, als dass die Webservices eine Verknüpfung von Programmen darstellen. Der Nutzer eines Webservices ist nie ein Benutzer (Mensch), sondern immer ein anderes System.

1.2 Arten von Webservices

Ein schöner Ansatz der Definition von Webservices ist eine Klassifizierung davon. Analog zur Klassifizierung von Services von [Josuttiso8] sollte man eine sinnvolle Einteilung von Webservices vornehmen.

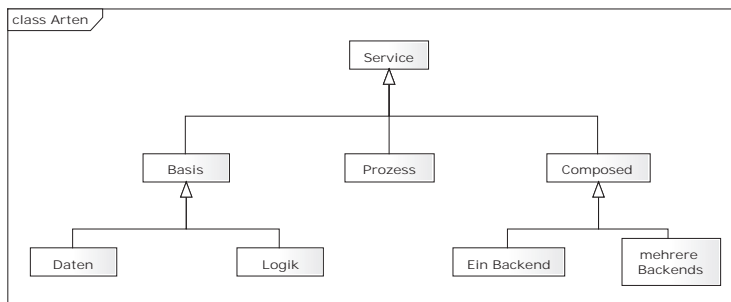


Abb. 1.2: Einteilung der Services nach [Josuttiso8]

1.2.1 Basis-Services

In den »Basis-Services« wird jeweils eine fachliche Funktionalität zur Verfügung gestellt. Normalerweise sind diese Services kurzlaufend, zustandslos und synchron. Sie entsprechen dem normalen Request-Response-Pattern. Basis-Services werden in »Basis-Daten-Services« und »Basis-Logik-Services« unterschieden. »Basis-Daten-Services« lesen oder schreiben Daten auf dem Backend, sie sind für die Verwaltung von Daten auf dem Backend zuständig. Typische Beispiele für einen Basis-Daten-Service wären:

- Erzeugen/Absenden einer Bestellung
- Registrierung eines Kunden
- Löschen, Ändern oder Suchen eines Artikels
- Authentifizierung eines Benutzers
- Lieferung eines Belegs (Rechnung, Lieferschein, Bestellung)

Die »Basis-Logik-Services« greifen nicht auf Daten zu, sondern präsentieren Geschäftsregeln. Typische Beispiele für Basis-Logik-Services wären:

- Validierung von Daten
- Berechnung der Restschuld eines Kredits
- Ermittlung von Konditionen (Rabatt bei einer bestimmten Bestellsumme)

1.2.2 Composed-Services

»Composed-Services« setzen sich aus mehreren (Basis-)Services zusammen. Die Komposition neuer Services aus vorhandenen Services nennt man Orchestrierung. Composed-Services sind aber normalerweise immer noch kurzlaufend und zustandslos. In den meisten Fällen sind Composed-Services Kompositionen von unterschiedlichen Backend-Systemen. So kann eine komplexe Bestellung u.a. aus folgenden Services bestehen:

- Kernsystem: Fakturierung
- Prüfung der Verfügbarkeit: Lagerhaltung/Lieferant
- Prüfung der Kreditwürdigkeit: Schufa
- Prüfung der Lieferzeit: Spediteur

Composed-Services können sich auch nur auf ein Backend-System beziehen. Hierbei kann es sich mitunter auch um einen Adapter- oder Wrapper-Service handeln, der verschiedenen Nutzern unterschiedliche Schnittstellen anbietet.

Ein Problembereich von Composed-Services ist die Transaktion. Bei mehreren Backends wird normalerweise kein Rollback, sondern eine Kompensation vorgenommen. Vgl. hierzu Kapitel 14 Absatz 2 Transaktionen.

1.2.3 Prozess-Services

»Prozess-Services« repräsentieren meist ganze Workflows, bei denen es sich um langlaufende und zustandsbehaftete Services handelt. Hierbei ist es auch möglich, dass durch Interaktionen Unterbrechungen stattfinden. Josuttis führt hier in [Josuttis08] als typisches Beispiel den Abschluss einer Versicherung an. Die Initiierung kann z.B. von dem zu Versichernden über ein Web-Portal ausgehen. Dann kann ein Sachbearbeiter mit einem Back-Office-Platz evtl. die Daten prüfen und ergänzen oder im Dialog Rückfragen klären.

Prozess-Services sind meiner Ansicht nach aber eher ein Aspekt der SOA, weniger ein Problem zur Realisierung von Webservices. Denn dieser Prozess wird aus Sicht der Realisation in mehrere Basis- oder Composed-Services unterteilt.

1.3 Aufgaben rund um Webservices

Um die Planung, die Realisierung und den Betrieb von Webservices entstehen unterschiedliche Aufgaben. Die wichtigsten Aufgaben kann man in folgende Gruppen zusammenfassen:

- **Definition der Schnittstelle:** Die Schnittstelle fasst die angebotenen Funktionen (Services) zusammen und definiert den Namen, die Parameter, den Rückgabewert und die Ausnahmen (Faults, Exceptions), die bei dem Service auftreten können.
- **Definition der Datentypen:** Die Datentypen für die Parameter und Rückgabewerte werden mit einem XML-Schema definiert. Dies können einfache vordefinierte Datentypen oder komplexe Strukturen sein.
- **Realisierung der Webservice-Schnittstelle durch den Service:** Der Service sucht die Implementation in Form eines Spring- oder EJB-Services und vermittelt zwischen dem, was der Webservice erwartet und was die eigentliche Implementation uns anbietet.
- **Publizierung:** Je nach der eingesetzten Technik, dem benutzten Framework und der Plattform, in der der Webservice laufen soll, muss der Webservice mit seinen dazugehörigen Komponenten deployed werden.
- **Erstellung der Consumer:** Ist der Webservice deployed und erreichbar, kann ein Client erzeugt werden. Die Art und Weise dieser Erstellung hängt natürlich von der eingesetzten Technik und dem Framework für den Client ab, die völlig unterschiedlich zur eingesetzten Technik des Webservices sein kann.

1.4 Granularität von Webservices

Die Granularität eines Webservices ist im Normalfall größer als in einer durch EJB verteilten Anwendung. Ein Webservice sollte im Mindesten einen zusammenhängenden fachlichen Service repräsentieren. Teilschritte zur Sammlung der Daten müssen dann auf der Consumer-Seite durchgeführt werden.

Während es bei einer EJB-Anwendung denkbar ist, dass für einen Bestellvorgang die Zusammensetzung der Bestellung (Hinzufügen eines Artikels, Zuordnen des Kunden, Auswahl der Lieferadresse etc.) in einzelnen Remote-Aufrufen realisiert wird, wird in einem Webservice nur die komplette Bestellung als Remote-Aufruf durchgeführt. Die Remote-Aufrufe für das Suchen der Artikel- oder Kundendaten bleiben dabei unberührt.

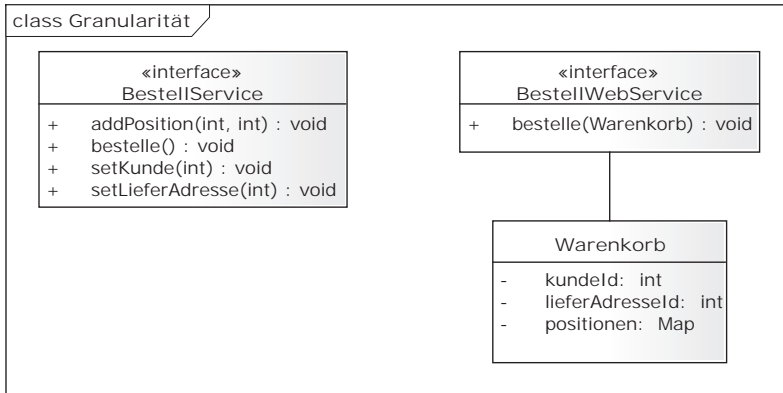


Abb. 1.3: Unterschiedliche Schnittstelle für (EJB-)Service und Webservice

Die Schnittstelle des `BestellService` könnte von einer Stateful Session Bean realisiert werden. Hier werden die Informationen gesammelt und durch `bestelle()` wird die Bestellung ausgeführt. Bei dem `BestellWebService` werden die kompletten Daten für die Bestellung als Parameter mit übergeben. Hierdurch ergibt sich eine andere Aufgabenverteilung für den Client bzw. Consumer. Auch bei einem EJB-Service sollte eine zustandslose Vorgehensweise aus Performance- und Skalierbarkeitsgründen vorgezogen werden.

1.5 Message Exchange Patterns

In diesem Abschnitt zeige ich eine kleine Zusammenfassung der »Message Exchange Patterns« (MEP). MEP sind ein generelles Konzept, um die Kommunikation zwischen verschiedenen Systemen zu beschreiben. Hierbei übernimmt ein

Partner die Rolle des Consumers (Nutzer) und einer die Rolle des Producers (Anbieter).

Folgende MEP können unterschieden werden:

Request/Response: Die klassische Kommunikation stellt sich so dar, dass der Consumer seine Anfrage stellt und auf das Ergebnis des Producers wartet.

One-Way: Sollte der Consumer keine Antwort erwarten, kann er seine Nachricht absenden und den Vorgang als abgeschlossen betrachten. Dieses MEP haben wir typischerweise beim Versand von E-Mails.

Zweimal One-Way statt Request/Response: Sofern man auf die Response nicht warten muss und auch keine genaue Zuordnung einer »Antwort« auf eine »Frage« benötigt, kann man auch den Request/Response in zwei One-Ways mit getauschten Rollen umändern. Sollte unsere Firma Edeltraud eine Bestellung von Halloumi in Zypern vornehmen und eine Benachrichtigung über den Liefertermin erwarten (um den Käse im Hafen abzuholen), so muss die Anfrage (Bestellung von Traudel) nicht unbedingt auf die Antwort (Nennung des Lieferzeitpunkts) warten. Der Zypriot, der zum Zeitpunkt des Bestellungseingangs noch am Strand war (ich weiß, ein Klischee, aber ein schönes), kann nach seiner Rückkehr ins Büro und in Absprache mit der Käserei den Lieferzeitpunkt zu einem späteren Zeitpunkt zurücksenden.

Request/Callback: Sofern in obigem Beispiel eine Zuordnung zwischen Bestellung und Bekanntgabe des Lieferzeitraums schwierig ist oder weitere Aktionen auslöst werden sollen, kann man beim Aufruf eine Adresse für eine Callback-Funktion angeben, die bei der Antwort aufgerufen werden soll. Damit hat man ein Request/Response-MEP, bei dem der Consumer nicht auf die Antwort warten muss, sondern über den Eingang der Antwort informiert wird. Dieses Pattern nennt man auch häufig asynchrones Request/Response.

Publish/Subscribe: Aus dem Bereich des Java Messaging Services (JMS) ist dieses MEP bekannt, bei dem man sich für ein Thema (Topic) registrieren kann und bei Nachrichten zu diesem Thema informiert wird. Traudel könnte sich dabei bei einem Service über betrügerische Onlinekunden registrieren und erhält Informationen darüber, wenn ein Kunde als »Schwarzes Schaf« auf der Liste erscheint.