

Web Services

Donald Kossmann
Universität Heidelberg
kossmann@informatik.uni-heidelberg.de

Frank Leymann
IBM
LEY1@de.ibm.com

Zusammenfassung

Web Services gelten als die Technologie, mit der in Zukunft Softwarekomponenten innerhalb einer Organisation und zwischen Organisationen integriert werden. Unter „Web Services“ versteht man hierbei ein Bündel von Technologien zur Beschreibung von Schnittstellen und Eigenschaften von Implementierungen der Schnittstellen, Beschreibung von Datenaustauschformaten und Qualitätseigenschaften des Austauschs, Registrierung von Komponenten, Komposition von Komponenten und Sicherheit im Austausch mit Komponenten. Darüber hinaus versprechen „Web Services“ die Unterstützung eines neuen Programmier- und Systemparadigmas für die Entwicklung verteilter Systeme, in dem alle Teilnehmer autonom agieren. Dieser Artikel gibt einen Überblick über die wichtigsten „Web Services“ Technologien und wie sie auf einander aufbauen.

1. Einleitung

1.1 Motivation

In den letzten Jahrzehnten haben viele Unternehmen große Investitionen zum Aufbau Ihrer IT Infrastruktur getätigt. Als Resultat gibt es kaum eine Aufgabe in einem Unternehmen, die nicht von Software unterstützt wird. Da es nicht die eine Softwarelösung gibt, die alle Funktionen eines Unternehmens abdeckt, wurde gemäß einer best-of-breed Strategie für jede Funktion die jeweils geeignetste Software ausgewählt und einige Funktionen wurden individuell entwickelt, damit sich ein Unternehmen auch durch seine IT-Unterstützung von seiner Konkurrenz differenzieren kann. Dieses Vorgehen hat dazu geführt, dass die meisten Unternehmen sehr viele Software-Systeme von unterschiedlichen Anbietern im Einsatz haben. Teilweise sind sogar mehrere hundert Softwaresysteme in einem Unternehmen im Einsatz.

Nachdem die meisten Funktionen eines Unternehmens durch Software unterstützt werden, ist die Integration der einzelnen Software-Systeme die nächste Herausforderung. Eine solche Integration ist notwendig, um eine weitere Automatisierung und Optimierung der Abläufe, d.h. der Geschäftsprozesse, in einem Unternehmen zu erreichen. So erfordert ein Ablauf (wie z.B. in der Fertigung oder im Vertrieb) die Interaktion zwischen mehreren Software-Systemen innerhalb des Unternehmens und zunehmend auch unternehmensübergreifend (z.B. zwischen CRM System des Lieferanten und e-Procurement System des Kunden) [LR00], [LRS02]. Eine solche Integration von Software-Systemen wird auch als *Enterprise Application Integration* (EAI) bezeichnet [K02]. Eine weitere Notwendigkeit zur Integration der Software-Systeme entsteht, wenn man die Daten der einzelnen Software-Systeme verknüpfen möchte. So können zum Beispiel unterschiedliche Aspekte eines Kunden in verschiedenen Systemen verwaltet werden, und es kann für bestimmte Aufgaben (z.B. Kundenservice) notwendig sein, einen einheitlichen Zugriff auf die komplette Information zu haben. Eine solche Form der Integration nennt man *Enterprise Information Integration* (EII) [SJII], [LR02].

Leider ist die Integration von Software-Systemen nicht einfach und häufig sehr teuer. In der Vergangenheit wurde oft versucht, in isolierten Projekten eine Integration zwischen einzelnen Systemen soweit wie notwendig zu realisieren. Typische Probleme, die durch diese Vorgehensweise auftreten, sind die folgenden:

- *Plattformabhängigkeit:* Die Systeme laufen häufig auf unterschiedlichen Hardware- und Betriebssystemplattformen und sie wurden nicht für die Interaktion mit anderen Software-Systemen entworfen. Dadurch ist es nicht möglich, dass ein System einfach ein anderes System aufruft. Um die Integration trotzdem zu bewerkstelligen, wurden Adapter zwischen jeweils zwei Systemen gebaut, doch häufig können diese Adapter für eine weitere Integration mit anderen Systemen nicht wieder verwendet werden.
- *Datenformatabhängigkeit:* Ein typisches Problem beim EII sind die unterschiedlichen Datenformate und Datenmodelle, die in den unterschiedlichen Systemen verwendet werden. So kann es vorkommen, dass ein System eine relationale Datenbank verwendet, um Daten zu speichern, während die Daten eines anderen Systems in einem proprietären Format des Anbieters gespeichert werden.
- *Prozessabhängigkeit:* Häufig kann man einen Ablauf, der eine Interaktion mehrerer Softwarekomponenten erfordert, nur ganz oder gar nicht verändern. Eine lokale Änderung eines Systems (z.B. ein Upgrade) ist nicht möglich, da die Auswirkungen auf die anderen Systeme nicht abschätzbar sind. Eine besondere Herausforderung ist die Behandlung von Fehlern.
- *Management und Optimierung:* Nach einer Integration mehrerer Software-Systeme ist es häufig schwierig, Aussagen über die Sicherheit, Verfügbarkeit oder Leistungsfähigkeit des Gesamtsystems zu machen. Insbesondere ist es oft praktisch unmöglich, Garantien über Ausfallzeiten oder Antwortzeiten zu geben. Des Weiteren ist es schwierig, ein solches System zu optimieren. Techniken wie Lastbalancierung und Caching, die für einen effizienten Betrieb eines verteilten Systems unerlässlich sind, erfordern eine transparente Software-Architektur, die nach einer Integration von einzelnen Software-Systemen oft nicht mehr gegeben ist. Häufig werden Daten (z.B. ein Log aller Aufrufe) unnötigerweise redundant von mehreren Komponenten verwaltet. Viele dieser Probleme werden gegenwärtig durch Initiativen wie Autonomic Computing und Grid Computing adressiert (siehe etwa [FK04]). Besondere Optimierungsprobleme entstehen beim EII, wenn es notwendig ist, möglichst viele Berechnungen lokal durch die einzelnen Systeme durchführen zu lassen [K00].

1.2 Ansatz

Unter dem Schlagwort „Web Services“ versucht man eine Infrastruktur zur Integration von Software-Systemen zu entwickeln, den so genannten „Service Bus“ (siehe [L03b]). Anstelle von Einzelprojekten zur Integration bestimmter Komponenten erreicht man somit einen einheitlichen Ansatz. Dieser Ansatz beruht auf den folgenden Prinzipien:

- *Lose Kopplung:* Die einzelnen Systeme bleiben autonom und kommunizieren über Nachrichten miteinander. Ein gemeinsames kompilieren oder binden der einzelnen

Anwendungen zu einer großen Anwendung findet nicht statt. Hierdurch können die einzelnen Systeme unabhängig von einander weiterentwickelt werden.

- *Virtualisierung*: Die Austauschbarkeit von Komponenten (Hardware wie Software) wird versucht, durch eine *Virtualisierung* der Systeme und Ressourcen zu erreichen ([L03c], [FKN+02]). Durch diese Virtualisierung werden Kommunikationspartner häufig dynamisch bestimmt: ein System A interagiert mit einem System B, wenn System B, die aktuelle Anforderung von System A erfüllen kann.
- *Einheitliche Konventionen*: Web Services unterstützen viele unterschiedliche Datenformate, Protokolle und Mechanismen zur Zusicherung von Qualitätseigenschaften, und schränken somit die verwendeten Technologien nicht ein. Web Services Technologien bestimmen hierzu einheitliche Konventionen, wie ein Web Service die Datenformate, Protokolle und Qualitätseigenschaften zur Interaktion mit anderen Web Services festlegt.
- *Standards*: Einer der wesentlichen Gründe für den Erfolg von Web Services ist die Festlegung von Standards, an die sich alle großen Plattform-Anbieter (wie z.B. BEA, IBM und Microsoft), Anbieter von Softwareanwendungen (wie z.B. SAP und Siebel) und Anwender (z.B. Unternehmen aus der Industrie) halten.

Zur Umsetzung einer solchen Infrastruktur wird eine Middleware-Architektur verwendet [B96]. Abbildung 1 zeigt eine solche Architektur für EAI: Software-Systeme (Anwendungen) kommunizieren nicht direkt miteinander sondern sie schicken Nachrichten an einen so genannten Message Broker. Der Message Broker besitzt einen Regelsatz und bestimmt somit an welche Anwendungen eine Nachricht verschickt werden muss und ggf. wie Nachrichten hierfür transformiert werden müssen und welche Seiteneffekte eine Nachricht verursacht (z.B. Protokollierung). So könnte eine Regel besagen, dass alle Nachrichten, die eine bestimmte Produktkategorie betreffen (z.B. Aufträge für Packstoffe) von einer bestimmten Anwendung bearbeitet werden sollen. Durch Anpassung der entsprechenden Regel kann die Anwendung zur Bearbeitung von Aufträgen für Packstoffe jederzeit angepasst oder ausgetauscht werden oder die Prozesse diversifiziert werden, indem z.B. besondere Anwendungen für Packstoffe aus Karton und aus Plastik vorgesehen werden. Letztendlich leistet der Message Broker eine logische Zentralisierung der Integrationsaufgaben. Konzeptionell setzt er somit die vier Prinzipien um: (a) die lose Kopplung wird erreicht, indem Anwendungen einander nicht direkt aufrufen; (b) eine Virtualisierung wird durch die Regeln im Message Broker erreicht; (c) und (d) die einheitlichen Konventionen und Standards werden durch den Message Broker erzwungen.

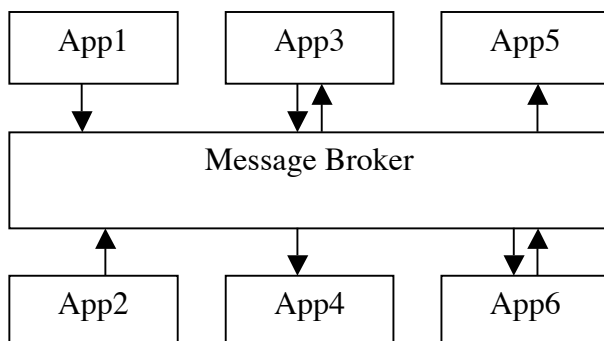


Abbildung 1: Middleware-Architektur für EAI.

1.3 Abgrenzung

Im Wesentlichen ist keines der Konzepte, die hinter Web Services stecken, neu: Es geht um bestimmte Varianten von „distributed computing“, die in der Vergangenheit bereits eingesetzt wurden. Zum Beispiel wurden Architekturen zur EAI wie die in Abbildung 1 bereits aufbauend auf CORBA [OMG95] und J2EE [S] entwickelt. XML [BPS+00], das verwendete Datenformat, basiert auf SGML, das in den 70er Jahren entwickelt wurde, und das XML Datenmodell [FMM+03] beruht auf Entwicklungen über semi-strukturierte Datenmodelle [PGW95]. Für Web Services wurden die entsprechenden Konzepte und Technologien dem Problembereich angepasst und als XML-basierte Standards formuliert. Entscheidend für den gegenwärtigen Erfolg von Web Services ist, dass die diesen Konzepten entsprechenden Technologien nun aufeinander abgestimmt sind und von allen namhaften Herstellern unterstützt werden. Die immensen Investitionen aller Beteiligten in diesem Bereich resultieren in einer weit verbreiteten Unterstützung der Web Services Standards und Standard-Vorschläge. Von großer Bedeutung ist auch der modularisierte Aufbau der Web Services Technologie und Standards: Web Services ist keine „alles oder nichts“ Entscheidung, sondern man setzt die Technologien ein, die man zur Lösung eines Problembereichs benötigt.

2. Grundlegende Web Services Technologien

2.1 Definition von Web Services

Es gibt keine anerkannte Definition des Begriffs Web Service. Das W3C definiert in [HB03] Web Services wie folgt:

„A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.“

Sehr wichtig sind uns die folgenden beiden, zusätzlichen Eigenschaften von Web Services: Identifizierbarkeit durch URIs und Autonomie. Zusammenfassend sind also die Schlüsseleigenschaften eines Web Services:

- a.) Ein Web Service wird durch einen URI identifiziert.¹
- b.) Die Schnittstelle eines Web Services ist maschinenlesbar und wird durch WSDL (siehe nächsten Abschnitt) beschrieben.
- c.) Ein Web Service kommuniziert mit anderen Softwarekomponenten durch XML Nachrichten. Der Nachrichtenaustausch kann insbesondere mit Hilfe von Internetprotokollen (z.B. HTTP oder SMTP) stattfinden.
- d.) Web Services sind autonom. Das heißt man kann nicht beeinflussen, ob und wie eine Nachricht von einem Web Service verarbeitet wird. Qualitätseigenschaften wie z.B. Antwortzeitgarantien müssen durch zusätzliche Vereinbarungen geregelt werden.

Viele Web Services entstehen derzeit im Bereich e-Business, zum Beispiel beim elektronischen Übertragen von Purchase Orders (Aufträgen) zwischen Unternehmen. Ein in

¹ URI = Uniform Resource Identifier [BFM98]. URIs sind der Standard, um Objekte im Internet eindeutig zu identifizieren. URLs wie sie in HTML zur Referenzierung von Webseiten verwendet werden sind besondere URIs.

der Öffentlichkeit sichtbares Beispiel ist Amazon. Neben der bekannten, üblichen Benutzung durch Menschen mit einem Internet Browser, bietet Amazon auch eine Web Services Schnittstelle an, die es Software-Anwendungen erlaubt, im Amazon Katalog zu suchen und Artikel zu bestellen. Mehr Informationen zu dieser Schnittstelle sind unter [Ama] verfügbar.

2.2 Basis-Technologie-Stack

Gemäß der obigen Definition von Web Services sind die Beschreibung von Schnittstellen und der Nachrichtenaustausch wesentliche Aufgaben. Hierzu wurden SOAP, WSDL und UDDI als drei Kerntechnologien entwickelt. Diese drei Technologien basieren alle auf XML und können Internetprotokolle wie http oder smtp verwenden (siehe Abb. 2). XML verstehen wir hierbei nicht nur als reines Datenformat sondern als Familie von Standards mit Datenmodell [CT01], Schema [F01], Verschlüsselung und Signatur [M]. Im folgenden möchten wir kurz SOAP, WSDL und UDDI vorstellen.

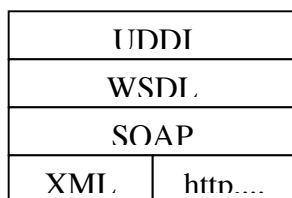


Abbildung 2: Der Basis-Web-Services-Stack

2.2.1 SOAP

SOAP² [M03] legt die Grobstruktur und die Verarbeitungsvorschriften von Nachrichten fest. Es basiert auf XML und ist wie alle Standards des W3C plattform- und programmiersprachenunabhängig. Eine SOAP Nachricht hat folgende Struktur:

```
<?xml version="1.0">
<env:Envelope xmlns:env = http://www.w3c.org/soap-envelope>
  <env:Header> ... </env:Header>*
  <env:Body>
    ...
    <env:Fault> ... </env:Fault>*
    ...
  </env:Body>
</env:Envelope>
```

Das heißt, jede Nachricht ist ein XML Dokument mit **Envelope** als Wurzelement. Jede Nachricht hat optional einen oder mehrere Header. Ein Header enthält Kontrollinformation wie z.B. eine Nachrichtennummer oder den Hinweis, dass die Nachricht eine Antwort auf eine andere Nachricht ist. Auf jeden Fall muss eine Nachricht einen Body haben. Der Body enthält die Nutzinformation, z.B. eine PurchaseOrder. Optional kann der Body auch

² Ursprünglich stand SOAP für Simple Object Access Protocoll. Seit der Version 1.2 wird auf die Verwendung dieses Akronyms verzichtet.

Fehlermeldungen enthalten. Somit regelt SOAP auch, wie Fehler als Nachrichten versendet werden und gibt das Format (Fehlercode und Fehlerbeschreibung) für Fehlermeldungen vor.

Neben dem Typ einer Nachricht definiert SOAP ein Verarbeitungsmodell. Gemäß dieses Modells gibt es neben dem Absender und Empfänger einer Nachricht noch sogenannte *Intermediaries*, die die Nachricht auf dem Weg vom Absender zum Empfänger bearbeiten können. Diese Intermediaries führen ihre Aktionen (z.B. Protokollierung, Abrechnung) typischerweise anhand der Information aus dem Header der Nachricht aus.

SOAP ist sehr flexibel einsetzbar. Zum Einen kann es dazu verwendet werden im Body XML Daten und auch nicht XML Daten (z.B. ein JPEG Bild) zu versenden. Zum Zweiten kann es mit vielen unterschiedlichen Netzwerkprotokollen (http, smtp, beep usw.) verwendet werden, aber auch mit anderen Technologien wie etwa IIOP oder Message Queuing. Zum Dritten unterstützt es viele unterschiedliche Interaktionsmuster: neben dem klassischen Request/Response Muster wie man es für RPC benötigt, kann man auch einzelne Nachrichten verschicken oder beliebige andere Muster mit Hilfe von SOAP implementieren.

2.2.2 WSDL

WSDL³ dient dazu, die Schnittstelle eines Web Services zu beschreiben. Das heißt, WSDL erlaubt die Deklaration von Operation und die Definition der Nachrichten, die ein Web Service empfangen kann und die er verschickt. Das Format, in dem Web Services beschrieben werden, ist XML. Derzeit arbeitet das W3C an WSDL Version 2.0 [BLL03]. Der derzeit gültige Standard ist WSDL 1.1 [CCM+01]. Die verwendete Terminologie dieser beiden Versionen unterscheidet sich erheblich, doch die Prinzipien sind die gleichen. Da Version 2.0 voraussichtlich erst im Herbst 2004 formal verabschiedet wird und es dann noch einige Zeit dauern wird, bis Werkzeuge hierfür entwickelt worden sind und diese Version praktisch eingesetzt wird, beschreiben wir die Konzepte von WSDL anhand der derzeit eingesetzten Version 1.1 [CCM+01].

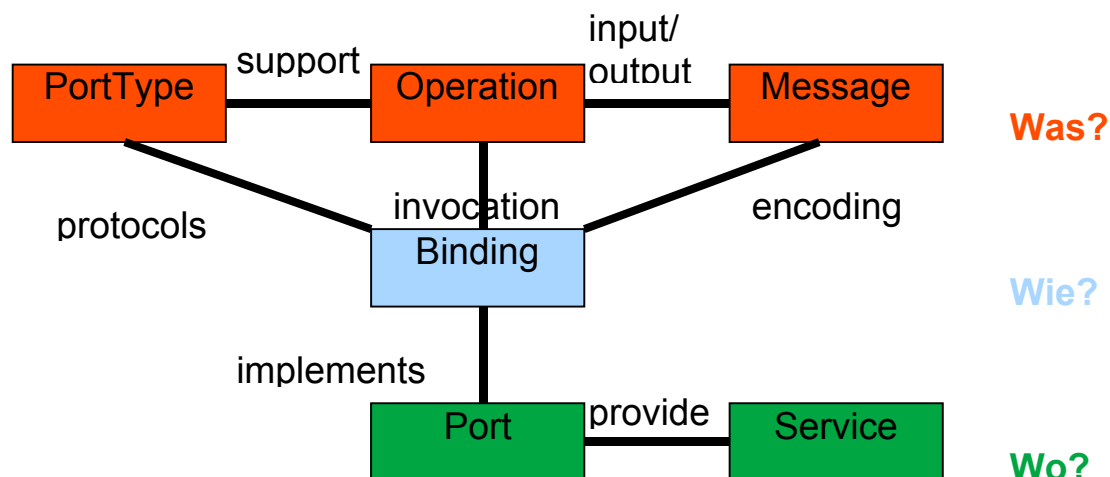


Abbildung 3: Übersicht über WSDL Konzepte

Abbildung 3 zeigt die wesentliche Konzepte einer WSDL Schnittstellenbeschreibung. Eine WSDL Beschreibung eines Web Services beantwortet drei Fragen:

³ Web Services Description Language.

- a.) **Was:** Was bietet der Web Service? D.h. welche eingehenden Nachrichten versteht der Web Service, welche ausgehenden Nachrichten (output, faults) erzeugt der Web Service und welche Operationen bietet der Web Service seinen Clienten zum Aufruf an? Ein Port Type bündelt eine Menge von Operationen, die mit denselben Mechanismen (i.e., Binding) aufgerufen werden können, d.h ein Port Type definiert die Schnittstelle eines Web Services.
- b.) **Wie:** Welche Protokolle werden zum Nachrichtenaustausch verwendet und wie werden die Nachrichten kodiert? Diese Informationen werden in so genannten Bindings spezifiziert.
- c.) **Wo:** Wie heißt der Web Service und unter welcher Internet-Adresse kann man Nachrichten an den Web Service schicken.

Wie am Anfang dieses Abschnittes erwähnt baut WSDL auf XML auf und alle Informationen werden in XML Dokumenten gehalten. Nach Version 1.1 sieht die Grobstruktur von WSDL Beschreibungen wie folgt aus:

```
<wsdl:definitions xmlns:wsdl = „http://w3.org/...“>
  <wsdl:documentation ... />
  <wsdl:types> Schema Imports </wsdl:types>
  <wsdl:message> Nachrichten </wsdl:message>
  <wsdl:portType> Operationen </wsdl:portType>
  <wsdl:binding> Protokolle und Formate </wsdl:binding>
  <wsdl:service> Service Definition </wsdl:service>
</wsdl:definitions>
```

Hier ein simples Beispiel, das zeigt, wie die Nachrichten und Operationen eines Web Services, der die Operation „add“ als einzige Operation unterstützt, mit Hilfe von WSDL beschrieben werden können.

```
<wsdl:message name="addRequest">
  <wsdl:part name="term1" type="xs:double"/>
  <wsdl:part name="term2" type="xs:double"/>
</wsdl:message>
```

```
<wsdl:message name="addResponse">
  <wsdl:part name="value" type="xs:double"/>
</wsdl:message>
```

```
<wsdl:portType name="arithmetics">
  <wsdl:operation name="add">
    <wsdl:input message="addRequest"/>
    <wsdl:output message="addResponse"/>
  </wsdl:operation>
</wsdl:portType>
```

2.2.3 UDDI

UDDI⁴ dient zur Katalogisierung von Web Services. Es liefert somit einen wesentlichen Beitrag zum Management von Web Services und zur Virtualisierung von Diensten und Ressourcen. Abbildung 4 zeigt die Rolle von UDDI bei der Nutzung von Web Services. Ein Provider (Anbieter eines Web Services) registriert seinen Web Service in einem UDDI Server. Ein Client (Nutzer) sucht nach geeignetem Web Services mit Hilfe eines UDDI Servers. Nach erfolgreicher Suche, kommuniziert der Client direkt mit dem Provider, um den Web Service zu nutzen.

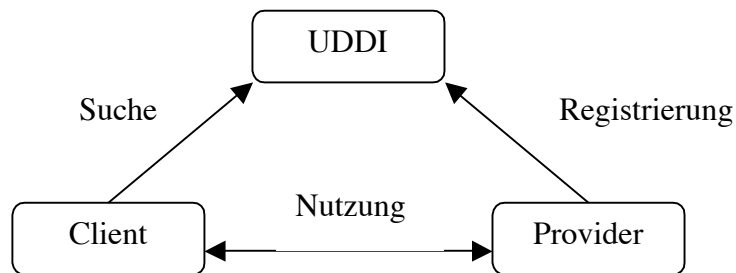


Abbildung 4: Verwendung von UDDI

Informationen über Web Services und Provider werden vom UDDI Server in sogenannten White, Yellow und Green Pages gehalten. Die Green Pages enthalten im Wesentlichen die WSDL Schnittstellen der angebotenen Web Services. White und Yellow Pages enthalten Informationen zum Provider wie z.B. Adresse, Branche, Ansprechpartner, etc.

2.3 Policies

Mit WSDL werden „lediglich“ die grundlegenden Eigenschaften eines Web Services definiert, nämlich dessen Schnittstelle und die Art und Weise seiner Erreichbarkeit (Formate, Protokolle, Aufenthaltsort). Weitere Eigenschaften wie etwa Qualitätseigenschaften (z.B. Transaktionsunterstützung, Sicherheit) oder geschäftlich relevante Informationen (z.B. Kosten, Zahlungsart) können durch *Policies* spezifiziert werden.

2.3.1 Aufbau von Policies

Hierzu wird zunächst eine Sprache zur Konstruktion von Policies definiert [Pol]: Die Grundelemente dieser Sprache sind so genannte *Assertions*. Assertions werden nicht im Rahmen des eigentlichen Policy-Standard-Stacks definiert, sondern müssen in spezifischen Anwendungsbereichen definiert werden (z.B. Assertions für Sicherheit, Transaktionen, Zahlungsmodalitäten). Daher ist eine Assertion aus Sicht eines Policy-Prozessors atomar, d.h. sie hat keine weitere Struktur, die für einen Policy-Prozessor interpretierbar wäre.

Eine Assertion hat aber sehr wohl eine Bedeutung für den Nutzer eines Web Services und für die Laufzeitumgebung des Web Services. So kann eine Assertion etwa spezifizieren, dass die Nachrichten, die mit einem bestimmten Web Service ausgetauscht werden, nach einem vorgegebenen Verfahren zu verschlüsseln sind. Diese Assertion ist selbstverständlich sowohl für den Nutzer als auch für die Laufzeitumgebung von Bedeutung, da der Nutzer etwa die

⁴ Universal Description Discovery & Integration.

Nachricht zum Web Service verschlüsseln muss, und die Laufzeitumgebung des Web Services die Nachricht entsprechend entschlüsseln muss.

Mit Hilfe von *Operatoren* können nun iterativ Policy-Ausdrücke (*statements*) aufgebaut werden. Mit Hilfe des <All> Operators wird spezifiziert, dass alle eingeschlossenen Assertions und Policy-Ausdrücke zu berücksichtigen sind. Weiterhin sind der <ExactlyOne> und der <OneOrMore> Operator definiert, die vorschreiben, dass genau eines bzw. mehrere eingeschlossene Elemente zu berücksichtigen sind. Policy-Ausdrücke können eindeutig benannt werden, so dass sie in anderen Policy-Ausdrücken referenziert werden können; das erlaubt eine einfache Wiederverwendung von häufig benötigten Policy-Ausdrücken und die Zuordnung von Policies zu einzelnen Elementen der Definition eines Web Services (siehe 2.3.2). Das Wurzelement eines Policy-Ausdrucks wird als *Policy* bezeichnet und wird verwendet, um das Verhalten von Web Services zu spezifizieren.

Die folgende Policy (in vereinfachter Syntax) spezifiziert, dass Stammkunden und Laufkundschaft unterschiedlich behandelt werden: Während Stammkunden monatlich zahlen und deren Daten geheim gehalten werden, zahlen Laufkunden für jede einzelne Nutzung des Dienstes mit der Kreditkarte und deren Daten werden weitergegeben. Da Assertions selbst nicht Teil des Policy-Stacks sind, sind die entsprechenden Assertions vom Anwender selbst definiert und in diesem Beispiel in einen eigenen Namensraum `fl` gelegt worden.

```
<Policy ... xmlns:fl="http://frank.com/policies">
  <ExactlyOne>
    <All Id="StammKunde">
      <fl:MonatlicheZahlung/>
      <fl:DatenGeheimhaltung/>
    </All>
    <All Id="Laufkundschaft">
      <fl:ZahlungProNutzung/>
      <fl:Kreditkarte/>
      <fl:DatenWeitergabe/>
    </All>
  </ExactlyOne>
</Policy>
```

2.3.2 Zuordnung von Policies

Policies können den einzelnen Elementen der WSDL Definition eines Web Services als so genannte *Attachments* zugeordnet werden [Att], indem diese Elemente im Attachment eine Policy referenzieren. So können Policies den einzelnen Nachrichten einer Operation zugeordnet werden, den einzelnen Operationen eines Web Services und schließlich dem Web Service selber. Zum Beispiel kann die Policy des `BestellDienst` Web Services zusichern, dass keine der übermittelten Daten Dritten zugänglich gemacht werden; ferner kann für die `Bestellung` Operation festgelegt werden, dass sie innerhalb einer Transaktion ausgeführt werden kann; schließlich kann spezifiziert werden, dass die Eingabedaten der `Bestellung` Operation signiert übertragen werden müssen. Die sogenannte *effektive* Policy eines Elementes setzt sich aus den Policies der „darüberliegenden“ Policies zusammen. So kann die `Bestellung` Operation nicht nur innerhalb einer Transaktion ausgeführt werden, sondern die übertragenen Daten werden auch niemals einem Dritten zugänglich gemacht.

Auch ohne die WSDL Definition eines Web Services zu ändern, können einem Web Service Policies zugeordnet werden. Insbesondere kann dies geschehen, wenn sich der Web Service

bereits im Einsatz befindet (d.h. „deployed“ ist), eine Modifikation der Definition also unvermeidbaren Aufwand mit sich bringen würde. Zu diesem Zweck können Policies deklarativen Beschreibungen (*domain expressions*) der zu dekorierenden Web Services zugeordnet werden; alle unter der Beschreibung qualifizierten Web Services werden dann mit der spezifizierten Policy versehen. Schließlich gibt es auch noch die Möglichkeit, Web Services, die in UDDI registriert sind, Policies zuzuweisen und Policies in UDDI abzulegen.

2.3.3 Implementierung von Policies

Auch der Nutzer eines Web Services hat im Allgemeinen Policies spezifiziert. Zum Beispiel kann ein Client des Bestelldienstes verlangen, dass alle seine Interaktionen innerhalb von Transaktionen ablaufen müssen. Es werden sich dann nur Implementierungen des Bestelldienstes mit entsprechend spezifizierten Policies für diesen Client qualifizieren. Wesentlich ist, dass das durch Policies spezifizierte Verhalten eines Clients oder eines Web Services im Allgemeinen nicht durch den Programmierer des Nutzers oder des Web Services selber implementiert werden muss: Es ist die Aufgabe der Laufzeitumgebung (d.h. des Service Bus) des Clients bzw. des Web Services, dieses Verhalten sicherzustellen. Somit muss sich zum Beispiel der Implementierer des Bestelldienst Web Services nicht darum kümmern, dass die Bestellung Operation in einer Transaktion läuft und dass die Eingabenaachricht entschlüsselt wird oder dass bei Beginn der Interaktion des Nutzers mit dem Web Service eine Transaktion begonnen wird. Dies wird automatisch durch die Laufzeitumgebung des Web Services bzw. des Nutzers durch Interpretation der entsprechenden Policies, die die Interaktion zwischen Nutzer und Web Service regulieren, sichergestellt.

Typischerweise werden Policies beim Deployment (Bereitstellung des Web Services) spezifiziert. Nachdem die Implementierung eines Web Services installiert wurde, werden die entsprechenden Bindings, über die der Web Service erreichbar ist, definiert und anschließend die entsprechenden WSDL Services samt ihrer Adressen spezifiziert. Dann wird das Verhalten, welches der Provider dieses Web Services garantieren will, in Form von Policies hinzugefügt. Die Laufzeitumgebung des Web Services stellt dann dieses Verhalten sicher.

2.4 Security

Eine der wesentlichen Qualitätseigenschaften der Interaktionen mit einem Web Service ist Sicherheit, insbesondere wenn diese Interaktionen über das Internet ablaufen. Jede Interaktion mit einem Web Service kann prinzipiell über mehrere Kommunikationspartner ablaufen (z.B. SOAP Intermediaries – siehe 2.2.1). Ferner können die Übertragungskanäle zwischen je zwei dieser Partner unterschiedlich sein – z.B. kann eine Nachricht vom Client zu einer Zwischenstelle über HTTPS und dann von der Zwischenstelle zum Web Service über SMTP übertragen werden. Daher kann bei der Absicherung der Qualität der Nachrichtenübermittlung zwischen Client und dem Web Service nicht auf die Qualität der beteiligten Übermittlungskanäle zurückgegriffen werden. Somit muss die Qualität der Interaktion zwischen Client und Web Service oberhalb dieser „physikalischen“ Kanäle realisiert werden.

2.4.1 Sicherheit auf der Nachrichtenebene

Daher wird Sicherheit des Nachrichtenaustauschs zwischen einem Web Service und seinen Nutzern unabhängig von jedem Transportprotokoll definiert. Dies wird durch die Definition von entsprechenden (SOAP) Headern erzielt: Solche Header spezifizieren etwa, welche Teile

einer Nachricht verschlüsselt sind und mit welchem Mechanismus. Ebenso können elektronische Unterschriften und Informationen darüber in entsprechenden Headern ausgetauscht werden. Aber auch „security token“ wie etwa Zertifikate können (verschlüsselt und signiert) sicher über Zwischenstellen ausgetauscht werden.

Hierbei greifen die entsprechenden Web Services Mechanismen selbstverständlich auf existierende Standards zurück, etwa auf XML Encryption oder auf XML Signature. Es wird also nur definiert, wie solche Standards im Web Services Umfeld zu verwenden sind.

2.4.2 Effizienter sicherer Nachrichtenaustausch

Wenn mehrere Nachrichten sicher ausgetauscht werden müssen, können die oben beschriebenen, meist auf asymmetrische Schlüssel basierenden Mechanismen effizienter gestaltet werden. So kann ein symmetrischer Schlüssel, der für die Dauer der Sitzung gültig ist, ausgehandelt und über die zuvor beschriebenen Mechanismen übertragen werden. Dies steigert die Effizienz der Sitzung zwischen dem Client und dem Web Service.

2.4.3 Vertrauen

Wenn zwei Partner im Web Services Umfeld sicher interagieren wollen, müssen sie eine Vertrauensbeziehung etablieren. D.h. die Partner müssen wechselseitig ihre *credentials* (z.B. Zertifikate) akzeptieren, obwohl sie unterschiedliche Technologien verwenden können (z.B. Kerberos oder PKI). Das bedeutet, dass die von den Partnern verwendeten Sicherheitsinfrastrukturen gefördert werden müssen. Zu diesem Zweck müssen sich die Partner auf einen Web Service einigen, dem beide vertrauen (so genannter *security token service*). Dieser Web Service akzeptiert die *credentials* eines Partners, verifiziert diese und übersetzt sie in *credentials* die der andere Partner akzeptieren kann.

2.5 Transaktionen

Eine weitere wichtige Qualitätseigenschaft ist die Transaktionalität [WV01]: Klassisch werden hierunter die bekannten ACID Eigenschaften verstanden, d.h. es können mehrere Aufrufe von Operationen eventuell verschiedener Web Services in eine Verarbeitungseinheit gebunden werden. Im verteilten Umfeld der Web Services haben wir es dann sogar mit verteilten Transaktionen zu tun, d.h. es muss strenggenommen ein Zwei-Phasen-Commit (2PC) Protokoll zwischen den beteiligten Web Services gefahren werden. Aus verschiedenen Gründen (lange Laufzeiten einzelner Operationen, Nicht-Unterstützung des 2PC durch einige beteiligte Web Services, Mischung synchroner und asynchroner Operationen usw.) sind solche verteilten Transaktionen aber nicht immer anwendbar. In diesem Falle kommen typischerweise kompensationsbasierte Transaktionsmodelle zum Einsatz, d.h. im Fehlerfalle werden automatisch Operationen aufgerufen (sogenannte „Kompensationsaktionen“), die bereits abgeschlossene Operationen zurücknehmen (siehe etwa [LR00]).

2.5.1 Agreement

Hinter dem kompensationsbasierten Transaktionsmodell liegt das Konzept der Einigung (*agreement*): Die beteiligten Web Services einigen sich am Ende der Transaktion darauf, wie die gesamte Verarbeitungseinheit geendet hat. Im traditionellen Modell verteilter Transaktionen wird über den Erfolg oder Misserfolg abgestimmt, und wenn einer der Beteiligten für Misserfolg stimmt, wird die gesamte Transaktion zurückgenommen. Es gibt

aber auch Situationen, in der ein erfolgreiches Ende bedeutet, dass einige der Beteiligten die durchgeführten Änderungen bestätigen, andere sie zurücknehmen müssen.

Ein Beispiel für eine solche Situation ist eine Angebotsermittlung, in der ein Kunde Angebote für ein Mengengut von mehreren Herstellern einholt. Die Hersteller werden die angebotenen Mengen reservieren. Der Kunde wird sich dann entscheiden, von welchen Herstellern er welche Mengen abnimmt. Die Hersteller, von denen nicht bestellt wird, müssen die Reservierungen der Teile wieder aufheben, während die Hersteller, die mit einer Lieferung beauftragt werden, die Reservierung bestätigen und die Auslieferung veranlassen müssen.

Um dieses breite Spektrum unterschiedlicher Transaktionsmodelle abdecken zu können, stehen entsprechende Basismechanismen für die Einigung am Ende von Verarbeitungseinheiten zur Verfügung (siehe z.B. [Coor02]). Diese Basismechanismen erlauben einen Kontext für eine solche Verarbeitungseinheit zu erstellen. Dieser Kontext wird über den Service Bus (die Infrastruktur) an die beteiligten Web Services propagiert. Ein derart „infizierter“ Web Service akzeptiert dann die Teilnahme an der Verarbeitungseinheit, in dem er spezifiziert, wie am Ende mit ihm das Ergebnis der Verarbeitung ausgehandelt werden soll (sogenanntes *agreement protocol*).

2.5.2 Konkrete Protokolle

Verschiedene Klassen dieser agreement Protokolle resultieren in verschiedenen Transaktionsmodellen. In [Tran02] werden zwei Klassen solcher Protokolle spezifiziert: (a) Varianten des 2 Phasen Commit Protokolls (2PC) und (b) Varianten kompensationsbasierter Protokolle. Protokolle anderer Transaktionsmodelle können entsprechend definiert werden.

3. Programmierung von Web Services

Die im vorherigen Abschnitt beschriebenen Technologien legen die Konventionen fest, mit denen Web Services Nachrichten austauschen. Es wird nicht festgelegt, wie die Web Services implementiert werden. Je nach Anwendung und Aufgabe werden unterschiedliche Programmiermodelle vorteilhaft sein und unterschiedliche Anbieter (z.B. BEA, IBM und Microsoft) bieten unterschiedliche Möglichkeiten als Teil Ihrer Plattformen (z.B. WebLogic, WebSphere und .Net) an. Darüberhinaus gibt es eine Reihe von Forschungsprojekten zu diesem Thema (unter anderem ein eigenes eines der Autoren [FGK03]).

Im folgenden werden zwei Technologien vorgestellt, die zur Implementierung von Web Services eine wesentliche Rolle spielen: XQuery für die Erstellung datenlastiger Web Services und BPEL für die Erstellung von Web Services aus bereits vorhandenen Web Services. Diese beiden Technologien werden gegenwärtig durch das W3C bzw. OASIS standardisiert.

3.1 XQuery

Salopp formuliert ist XQuery für XML Daten das, was SQL für relationale Daten ist: eine Sprache mit der man komplexe Anfragen deklarativ stellen kann. Für die Entwicklung von Web Services Anwendungen nimmt XQuery eine besondere Bedeutung ein, weil XQuery sehr gut geeignet ist, um Transformationen auf XML Daten zu spezifizieren und Regeln, wie

sie zum Beispiel in einem Message Broker (Abbildung 1) verwendet werden können, zu implementieren.

XQuery [BCF+03] wird derzeit vom W3C entwickelt und vermutlich im Herbst 2004 als Recommendation formal verabschiedet werden. XQuery ist kompatibel zu allen anderen relevanten Entwicklungen des W3C, insbesondere natürlich der XML Familie von Standards wie XML Schema, XML Infoset und Namespaces. XQuery erweitert XPath [BCF+03], das bereits in vielen Anwendungen, die XML Daten verarbeiten, eingesetzt wird (u.a. in Internet Browsern). Es ist zu erwarten, dass für Web Services Anwendungen XQuery mittelfristig XSLT, eine alternative Sprache des W3C zur Spezifikation von XML Transformationen und ebenfalls auf XPath aufbaut, ablösen wird, da XQuery viel mächtiger und einfacher ist (intuitivere Syntax und Semantik). Derzeit wird an einer XQuery Implementierung von praktisch allen großen Herstellern von Software-Plattformen gearbeitet (siehe z.B. [FHK+03]) und es gibt eine Reihe von OpenSource Projekten, z.B. Galax [F+03] und Saxon [K03].

XQuery ist eine funktionale Programmiersprache. Das heißt, XQuery erlaubt eine beliebige Komposition von komplexen Ausdrücken aus einfachen Ausdrücken. Außerdem ist XQuery geschlossen: jeder XQuery Ausdruck bekommt XML Daten als Eingabe und liefert XML Daten als Ausgabe. Des Weiteren hat XQuery ein strenges, statisches Typsystem. Das W3C war von Beginn an bemüht, die Semantik der Sprache formal zu beschreiben [DFF+03]. Das Datenmodell ist eine Erweiterung des XML InfoSets, damit auch auf Kollektionen (sogenannte Sequenzen) von XML Daten operiert werden kann (z.B. auf eine Menge von XML Dokumenten anstatt nur auf einzelnen XML Dokumenten) [FMM+03].

XQuery hat eine sehr umfangreiche Bibliothek von eingebauten Funktionen und Operatoren [MMW03], die viele Bereiche abdecken: zum Beispiel Sortieren, Aggregieren, reguläre Ausdrücke, Volltextsuche, Kalenderarithmetik und sogenannte Collations für die Verarbeitung von Dokumenten mit Sonderzeichen wie z.B. deutschen Umlauten. Darüber hinaus bietet XQuery eine Reihe von sehr mächtigen Sprachkonstrukten für existentielle und universelle Quantifizierung, Joins, Funktionen höherer Ordnung, Konstruktoren, Verarbeitung von Typen, Sichten und benutzerdefinierte Funktionen und Typen an.

Die einfachsten XQuery Ausdrücke sind Konstanten, Variablen, einfache Funktionsaufrufe oder Vergleiche. Alle XPath Ausdrücke sind ebenfalls gültige XQuery Ausdrücke. Zum Beispiel wählt der folgende Ausdruck in der PurchaseOrder mit Nummer 4711 die Produkte aller Auftragspositionen (LineItems) aus:

```
//PurchaseOrder[@number = 4711]/LineItem/Product/Name
```

Charakteristisch für XQuery sind die sogenannten FLWOR (von For, Let, Where, Order-by, Return) Ausdrücke. Sie entsprechen ungefähr den SELECT-FROM-WHERE Anweisungen von SQL. Ein FLWOR Ausdruck hat fünf Teile, die alle bis auf den letzten (Return) Teil optional sind.

- a.) **for:** Diese Klausel bindet eine oder mehrere Variablen an XML Kollektionen (i.e. Sequenzen). Diese Klausel entspricht somit der FROM Klausel in SQL.
- b.) **let:** Diese Klausel definiert Konstanten, die in den folgenden Klauseln zur Vereinfachung verwendet werden können. Für diese Klausel gibt es in SQL kein Pendant.

- c.) **where:** Diese Klausel enthält Prädikate zum Filtern von Elementen aus den XML Sequenzen der *for* Klausel. Diese Klausel entspricht somit der gleichnamigen Klausel in SQL.
- d.) **order by:** Diese Klausel bestimmt die Reihenfolge, in der die Ergebnisse zurückgeliefert werden und entspricht somit der gleichnamigen Klausel in SQL. Falls keine Reihenfolge vorgegeben wird (Ausdrücke ohne *order by*) und keine andere Funktion in der Anfrage eine Reihenfolge vorgibt, dann werden Ergebnisse immer in der ursprünglichen Reihenfolge (üblicherweise die Reihenfolge in den XML Dokumenten) zurückgegeben.
- e.) **return:** Diese Klausel konstruiert das Ergebnis und entspricht somit der SELECT Klausel in SQL. Es können beliebige andere und verschachtelte XQuery Ausdrücke in dieser Klausel verwendet werden (insbesondere auch FLWOR Ausdrücke).

Das folgende Beispiel zeigt einen XQuery Ausdruck, der als Eingabe eine XML Dokument mit einer Bibliothek bekommt. Dieses Bibliotheks-Dokument (referenziert durch die freie Variable *\$bib*) enthält eine Folge von Büchern, wobei jedes Buch einen oder mehrere Autoren hat. Als Ergebnis liefert dieser XQuery Ausdruck ein XML Dokument, in dem für jeden Autor alle Bücher aufgelistet sind, an denen dieser Autor beteiligt war. Der XPath Ausdruck *\$bib//book[author = \$x]* liefert alle Bücher des Autors, dessen Namen an die Variable *\$x* gebunden ist.

```

<AuthorList>
  { for $x in distinct-values($bib//author) return
    <author>
      <name> { $x } </name>
      { $bib//book[author = $x] }
    </author>
  }
</AuthorList>

```

Derzeit bietet XQuery nur die Möglichkeit, Anfragen zu stellen oder XML Dokumente in neue XML Dokumente zu transformieren. XQuery bietet nicht die Möglichkeit XML Dokumente oder XML Datenbanken zu manipulieren: z.B. das Einfügen von Elementen in ein Dokument, Ändern oder Löschen von Elementen oder anderen Knoten. Innerhalb des W3C gibt es bereits Vorschläge zur Erweiterung von XQuery in dieser Hinsicht [CFL+02]. An der Standardisierung dieser Vorschläge wird voraussichtlich weitergearbeitet, sobald die Arbeiten an der ersten Version von XQuery abgeschlossen sind.

3.2 BPEL

Im Wesentlichen ist BPEL (genauer: BPEL4WS – Business Process Execution Language for Web Services [BPEL03]) eine Sprache, um mit Hilfe von Workflow-Technologie [LR00] aus vorhandenen Web Services neue Web Services zu komponieren [L03a]. Hierzu definiert man mit BPEL einen Prozess, der am Ende von einem Workflowsystem ausgeführt wird, wobei die einzelnen Aktivitäten des Prozesses Operationen von Web Services sind.

Diese Web Services können von verschiedenen Partnern zu Verfügung gestellt werden, so dass mit BPEL insbesondere auch unternehmensübergreifende Geschäftsprozesse (bzw. Abläufe) definiert und ausgeführt werden können. EAI Abläufe wiederum lassen sich durch Geschäftsprozesse abbilden, die nur mit einem einzigen Partner interagieren, nämlich dem Unternehmen, dessen Anwendungen integriert werden sollen.

3.2.1 Aktivitäten

Die nachfolgende Aktivität beschreibt in BPEL, dass ein Customer die Operation `orderTrip` des `TravelService` verwenden kann, um eine Order zu übermitteln.

```
<receive partnerLink="Customer"
  portType="fl:TravelService"
  operation="orderTrip"
  variable="Order">
</receive>
```

Der Prozess wiederum kann die `bookRoom` Operation des `HotelService` benutzen, um eine Hotel Nachricht an den `HotelChain` Partner zu schicken; der Prozess erwartet eine `Confirmation` Nachricht zurück.

```
<invoke partnerLink="HotelChain"
  portType="fl:HotelService"
  operation="bookRoom"
  inputVariable="Hotel">
  outputVariable="Confirmation">
</invoke>
```

3.2.2 Kontrollfluss

Die Reihenfolge, in der Aktivitäten ausgeführt werden, kann in BPEL durch einfache Konstrukte wie `<sequence>`, `<while>` oder `<switch>`, wie sie aus Programmiersprachen bekannt sind, festgelegt werden. Um beliebige Geschäftsprozesse, die meist auch parallel ablaufende Zweige enthalten, spezifizieren zu können, erlaubt BPEL durch das `<flow>` Konstrukt Graphen zu beschreiben, deren Knoten Aktivitäten und deren Kanten Kontrollflusslinks sind.

Das folgende Beispiel schreibt vor, dass zunächst die Nachricht vom Customer empfangen werden muss bevor der Dienst der `HotelChain` in Anspruch genommen werden kann.

```
<sequence>
  <receive partnerLink="Customer" ... />
  <invoke partnerLink="HotelChain" ... />
</sequence>
```

3.2.3 Weiterführende Konstrukte

Gruppen von Aktivitäten können in einen gemeinsamen Kontext gestellt werden. Neben Datenlokalität wird so auch ein gemeinsames lokales Fehlerverhalten definiert. Insbesondere stellt solche eine Gruppe von Aktivitäten eine Verarbeitungseinheit dar, die durch Kompensationsaktivitäten zurückgesetzt werden kann (siehe 2.5). Das erlaubt eine flexible Behandlung von Ausnahme- und Fehlersituationen. Schließlich lassen sich Eventhandler

definieren, um Nachrichten in Situationen zu empfangen, die vorab nicht genau modelliert werden können.

3.2.4 Bedeutung

Die Erstellung neuer Web Services durch BPEL rückt das zweistufige Programmieren (bzw. Metaprogramming) in das Zentrum des Programmiermodells für Web Services: „Klassische“ Programmiersprachen werden verwendet, um nicht-zusammengesetzte Web Services zu erstellen (*programming in the small*); BPEL wird verwendet, um neue Web Services aus bestehenden zusammenzusetzen (*programming in the large*).

Ein BPEL Modell kann in den unterschiedlichsten Plattformen ablaufen. In diesem Sinne sind BPEL Modelle portabel. Insbesondere kann ein Prozess mit dem Werkzeug des Herstellers A definiert und in den Laufzeitumgebungen von Hersteller B oder C ausgeführt werden. BPEL Artefakte können somit ähnlich wie Java oder C# Programme handelbare Artefakte werden.

4. Zusammenfassung und Ausblick

Web Services Technologien sind Bausteine zur Lösung von Software-Integrationsproblemen. Sie setzen auf bestehenden Ansätzen in den Bereichen Verteilte Systeme, Informationssysteme und Programmiersprachen auf und versuchen, diese Ansätze in ein einheitliches Bild zusammenzufügen und zu standardisieren. Web Services Technologien sind derzeit sehr populär sowohl im akademischen Umfeld als auch in der Industrie. Enorme Investitionen werden von Anwendern und Herstellern in diese Technologien getätigt. Fast schon aus diesem Grund sind Web Services zum Erfolg verurteilt.

Obwohl das Potential von Web Services gewaltig ist, ist es aus heutiger Sicht sehr schwierig einen Ausblick zu geben. Einige Technologien und Standards befinden sich noch in der Entwicklung. Das Gleiche gilt für Entwicklungswerkzeuge, die diese Technologien unterstützen. Es gibt zwar bereits eine Reihe (positiver) Erfahrungsberichte (siehe z.B. [BC04]), doch es ist noch unklar, ob alle in diese Technologie gesetzten Erwartungen letztendlich erfüllt werden können. Dies gilt zum einen für die Performanz (Antwortzeiten), aber auch für die Effizienz bei der Entwicklung neuer Web Services und bei der Durchführung von Integrationsprojekten. Es wird sich zeigen, ob es möglich ist, die Standards so zu bündeln und zu abstrahieren, dass Entwickler sie auf leichte Weise verwenden können. Des Weiteren besteht die Notwendigkeit zur Bereitstellung von Methodiken bei der Softwareentwicklung und Durchführung von Integrationsprojekten mit Web Services – hier könnte die Forschung wesentliche Beiträge liefern. Es gibt viele Standards und neue Produkte, und dieser Artikel konnte nur einen groben Überblick über die zentralen Standards geben, während Produkte gar nicht besprochen wurden.

Referenzen

1. [Ama] Amazon Inc. Amazon Web Services. <http://www.amazon.com/webservices>.
2. [Att] Web Services Policy Attachment. <http://www-106.ibm.com/developerworks/library/ws-polatt>.
3. [B96] P. Bernstein: Middleware: A Model for Distributed System Services. Communications of the ACM, Band 39, Nummer 2, Seiten 86-98, 1996.

4. [BCO+04] M. Brandner, M. Craes, F. Oellermann und O. Zimmermann. Web Services-Oriented Architecture in Production in the Finance Industry. Erscheint in GI Informatik Spektrum, 2004.
5. [BCF+03] S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robie und J. Simeon. XQuery 1.0: An XML Query Language, 2003. <http://www.w3c.org/TR/xquery>.
6. [BFM98] T. Berners-Lee, R. Fielding und L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax, 1998. IETF RFC 2396.
7. [BLL03] D. Booth, P. Le Hegaret und C. Liu. Web Services Description Language (WSDL) Version 2.0 Part 0: Primer, 2003. <http://dev.w3.org/cvsweb/~checkout~/2002/ws/desc/wsd120/wsd120-primer.html>.
8. [BPEL03] Business Process Execution Language For Web Services V1.1, BEA, IBM, Microsoft, SAP & Siebel, 2003, <http://www-106.ibm.com/developerworks/library/ws-bpel/>.
9. [BPS+00] T. Bray, J. Paoli, C. Sperberg-McQueen und E. Maler. Extensible Markup Language (XML) 1.0 (Second Edition), 2000. <http://www.w3c.org/TR/REC-xml>.
10. [CCM+01] E. Christensen, F. Curbera, G. Meredith und S. Weerawarana. Web Services Description Language (WSDL) 1.1, 2001. <http://www.w3c.org/TR/wsd1>.
11. [CFL+02] D. Chamberlin, D. Florescu, P. Lehti, J. Melton, J. Robie, M. Rys und J. Simeon. XUpdate, 2002. <http://www.w3c.org/TR/2002/WD-xupdate-20021015>.
12. [Coor02] Web Services Coordination, BEA, IBM & Microsoft, 2002, <http://www-106.ibm.com/developerworks/library/ws-coor/>.
13. [CT01] J. Cowan und R. Tobin. XML Information Set, 2001. <http://www.w3c.org/TR/xml-infoset>.
14. [DFE+03] D. Draper, P. Fankhauser, M. Fernandez, A. Malhotra, K. Rose und J. Simeon. XQuery 1.0 and Xpath 2.0 Formal Semantics, 2003. <http://www.w3c.org/TR/xquery-semantics>.
15. [F01] D. Fallside. XML Schema Part 0: Primer, 2001. <http://www.w3c.org/TR/xmlschema-0>.
16. [F+03] M. Fernandez et al. Galax, 2003. <http://db.bell-labs.com/galax>.
17. [FGK03] D. Florescu, A. Grünhagen und D. Kossmann. XL: An XML Programming Language for Web Service Specification and Composition. Computer Networks, Band 42, Nummer 5, 2003.
18. [FHK+03] D. Florescu, C. Hillery, D. Kossmann, P. Lucas, F. Riccardi, T. Westmann, M. Carey, A. Sundararajan und G. Agrawal. The BEA/XQRL Streaming XQuery Processor. Conf. on Very Large Databases, Berlin Deutschland, Seiten 997-1008, 2003.
19. [FK04] I. Foster and C. Kesselman, The Grid: Blueprint for a new computing infrastructure, 2nd Edition, Morgan Kaufmann Publishers, San Francisco, CA, 2004.
20. [FKN+02] I. Foster, C. Kesselman, J.M. Nick and S. Tuecke, The physiology of the Grid – An open Grid services architecture for distributed systems integration, Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002, <http://www.globus.org/research/papers/ogsa.pdf>.
21. [FMM+03] M. Fernandez, A. Malhotra, J. Marsh, M. Nagy und N. Walsh. XQuery 1.0 and XPath 2.0 Data Model, 2003. <http://www.w3c.org/TR/xpath-datamodel>.
22. [HB03] H. Haas, A. Brown: Web Services Glossary. W3C Working Draft, August 2003. <http://www.w3.org/TR/2003/WD-ws-gloss-20030808/>.
23. [K00] D. Kossmann: The State of the Art in Distributed Query Processing. ACM Computing Surveys, Band 32, Nummer 4, Seiten 422-469, 2000.
24. [K02] W. Keller, Enterprise Application Integration, dpunkt.verlag 2002.

25. [K03] M. Kay. SAXON: The XSLT and XQuery Processor, 2003.
<http://saxon.sourceforge.net>.
26. [L03a] F. Leymann, Choreography: Geschäftsprozesses mit Web Services, OBJECTspektrum (6) 2003.
27. [L03b] F. Leymann, Web Services: Distributed Applications without Limits, Proc. BTW'03 (Leipzig, Germany, February 26-28, 2003), Springer 2003.
28. [L03c] F. Leymann, Web Services, Datenbank Spektrum 6 (2003).
29. [LR00] F. Leymann and D. Roller, Production Workflow - Concepts and Techniques (PTR Prentice Hall, 2000).
30. [LR02] F. Leymann, D. Roller, Flows in Information Integration, IBM Systems Journal 41(4) (2002).
31. [LRS02] F. Leymann, D. Roller, M.-T. Schmidt, Flows and Web Services: B2B aspects of business process management, IBM Systems Journal 41(2) (2002).
32. [M] M. Mactaggart. An introduction to XML encryption and XML signature.
<http://www-106.ibm.com/developerworks/xml/library/s-xmlsec.html/index.html>.
33. [M03] N. Mitra. SOAP Version 1.2 Part 0: Primer, 2003.
<http://www.w3c.org/TR/2003/REC-soap12-part0-20030624>.
34. [MMW03] A. Malhotra, J. Melton und N. Walsh. XQuery 1.0 and XPath 2.0 Functions and Operators, 2003. <http://www.w3c.org/TR/xpath-functions>.
35. [OMG95] Object Management Group (OMG): CORBA: The Common Object Request Broker Architecture. Revision 2.0, Juli 1995.
36. [PGW95] Y. Papakonstantinou, H. Garcia-Molina und J. Widom. Object Exchange Across Heterogeneous Information Sources. IEEE Conf. on Data Engineering, Taipeh, Taiwan, Seiten 251-260, 1995.
37. [Pol] Web Services Policy Framework,
<http://www-106.ibm.com/developerworks/library/ws-polfram>.
38. [S] SUN Microsystems: Java 2 Enterprise Edition Tutorial.
<http://java.sun.com/j2ee/tutorial>.
39. [SJII] IBM Systems Journal 41(4) (2002) – Special issue on Information Integration.
40. [Tran02] Web Services Transactions, BEA, IBM & Microsoft, 2002.
<http://www-106.ibm.com/developerworks/library/ws-transpec>.
41. [WV01] G. Weikum und G. Vossen. Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann Publishers, 2001.