

Freie Universität Berlin  
Fachbereich Mathematik und Informatik  
Institut für Informatik

Seminararbeit  
Advanced Topics in Networking

# **BPEL4WS**

## **Business Process Execution Language for Web Services**

Riad Djemili  
([djemili@inf.fu-berlin.de](mailto:djemili@inf.fu-berlin.de))

WiSe 2003/04

# Inhaltsverzeichnis

<b>1 Übersicht</b>	<b>3</b>
<b>2 Webdienste</b>	<b>3</b>
2.1 Terminologie . . . . .	4
<b>3 Kombination von Webdiensten</b>	<b>5</b>
<b>4 Dokumentaufbau</b>	<b>6</b>
4.1 Umgebungen . . . . .	7
<b>5 Kommunikation zwischen Diensten</b>	<b>7</b>
5.1 Deklarieren von Partnerbeziehungen . . . . .	7
5.2 Ausgehende Webdienstaufrufe . . . . .	8
5.3 Eingehende Webdienstaufrufe . . . . .	9
<b>6 Kontrollabläufe</b>	<b>9</b>
6.1 Sequentieller Ablauf . . . . .	9
6.2 Nebenläufiger Ablauf . . . . .	10
<b>7 Zustände</b>	<b>10</b>
7.1 Variablen . . . . .	11
7.2 Eigenschaften . . . . .	11
7.3 Korrelationsmengen . . . . .	12
<b>8 Ausnahmebehandlung</b>	<b>13</b>
8.1 Fehlerfälle . . . . .	13
8.2 Kompensation . . . . .	14
<b>9 Ereignisbehandlung</b>	<b>15</b>
<b>10 Weitere Aktivitäten</b>	<b>15</b>
<b>11 Konklusion</b>	<b>16</b>
<b>Literatur</b>	<b>17</b>

Webdienste ermöglichen auf standardisierte Art, verteilte Komponenten zu entwickeln und zu veröffentlichen. Moderne Geschäftsvorgänge, die die Funktionalität mehrerer verteilter Dienste kombinieren, rücken damit in praktikable Nähe. Allerdings ist die Beschreibung dieser Prozesse noch nicht einheitlich geregelt. Mehrere Spezifikationen wurden lanciert, um diese allgemein erkannte Lücke zu schließen.

Eine neue Initiative basiert auf der Kooperation von Microsoft, IBM und BEA : BPEL4WS. Sie nutzt WSDL-Informationen als Grundlage, um mittels spezieller Konstrukte, Prozesse als eigene Webdienste zu beschreiben. Dabei können sowohl abstrakte Geschäftsprotokolle, als auch ausführbare Prozesse definiert werden.

Diese Arbeit gewährt einen Überblick über diese neue Sprache.

## 1 Übersicht

Einleitend soll in Abschnitt 2 der Begriff Webdienst, sowie die assoziierten Spezifikationen eingeordnet werden. Danach erläutert der Abschnitt 3 die Kombination (Orchestration) von verschiedenen Webdiensten und gibt einen kurzen Überblick über die kürzlich aufgekommenen Spezifikationen.

Schließlich folgt der Schwerpunkt dieser Arbeit. In dem Abschnitt 4 wird der grundlegende Aufbau eines BPEL4WS-Dokuments gemäß der aktuellen Spezifikationsversion 1.1 [1] beschrieben. Abschnitt 5 beschreibt das Spezifizieren der Kommunikationsbeziehungen und Abschnitt 6 die elementarsten Ablaufreihenfolgen. Die folgenden Abschnitte beschreiben das Halten von Zuständen (Abschnitt 7), die Ausnahmebehandlung (Abschnitt 8), die Ereignisbehandlung (Abschnitt 9) und zuletzt weitere Aktivitäten in Abschnitt 10.

Abschließend folgt mit Abschnitt 11 die Konklusion.

## 2 Webdienste

Der Begriff Webdienst findet in den Medien starke Verwendung, ist jedoch meist nur vage definiert. Allgemeine konzeptionelle Definitionen beschreiben Webdienste im Wesentlichen als verteilte lose-gekoppelte Komponenten, die plattformunabhängig mittels Web-Standards kommunizieren. Restriktivere und spezifizierende Definitionen betrachten Webdienste, als verteilte Anwendungen, die mittels der Spezifikationen von SOAP, WSDL und UDDI entwickelt wurden. Das *World Wide Web Consortium* (W3C) verwendet folgende Definition.

*Ein Webdienst ist ein Software-System, entwickelt um dialogfähige Maschine-zu-Maschine Interaktion über ein Netzwerk zu unterstützen. Sie hat eine, in einem maschinen-verarbeitbaren Format (im speziellen WSDL) beschriebene, Schnittstelle. Andere System interagieren mit dem Webdienst in einer durch ihre Beschreibung vorgeschriebenen Art, typischerweise übertragen durch HTTP mit XML Serialisierung und in Verbindung mit anderen Web-verwandten Standards. [7]*

Obwohl nicht in dieser Definition aufgeführt, wird UDDI meist ebenfalls als elementarer Bestandteil von Webdiensten betrachtet. Die Grundpfeiler von Webdiensten sind zusammenfassend.

### Simple Object Access Protocol (SOAP)

SOAP [6] ist ein leichtgewichtiges und erweiterbares Kommunikationsprotokoll zum Austausch von Nachrichten im XML-Format. Es hat sich als meist verbreitetes Nachrichtenprotokoll für Webdienste durchgesetzt.

SOAP Nachrichten können beliebige Daten transportieren, wobei ein besonderer Schwerpunkt auf der Codierung von Remote Procedure Calls (RPCs) liegt, d.h. der Übermittlung von Operationsaufrufen. Obwohl eigentlich protokollunabhängig wird die Verwendung von SOAP häufig gleichgesetzt mit der Verwendung des HTTP-Protokolls auf der unterliegenden Transportebene.

### Web Service Definition Language (WSDL)

Die Kommunikation zwischen verteilten Diensten, erfordert eine genaue Übereinkunft über die Form derselben. Dies beinhaltet nicht nur das technische Kommunikationsprotokoll, sondern auch spezielle Informationen zu jeder Operation, wie die Art und Menge der Argumente, Rückgabewerte oder Fehlerausnahmen. Diese Übereinkunft geschah in der Vergangenheit meist mittels unstandardisierter Dokumentation.

WSDL [4] ist eine Sprache zur Definition von maschinen-verarbeitbaren Spezifikation von verteilten Diensten in Form von XML-Dokumenten. WSDL definiert dazu auf abstrakter Ebene die angebotenen Operationen, die über so genannte bindings an konkrete Protokolle gebunden werden.

### Universal Description, Discovery and Integration (UDDI)

UDDI [8] erlaubt Anbietern ihre angebotenen Internetdienste (insbesondere Webdienste) mitsamt Beschreibung in einem globalen Katalog abzulegen. Die Katalogisierung kann dabei eine Vielzahl von Bereichen, wie z.B. Geschäftsfeld, Lokalität oder Sprache umfassen.

Dies erlaubt Nachfragern gesuchte Webdienste, zentral und auf standardisierte Art zu finden und durch die hinterlegte Spezifikation dieser Dienste (idealerweise in WSDL), schnell und dynamisch in Kontakt zu treten.

## 2.1 Terminologie

Im folgenden eine Übersicht über wichtige Begriffe im Zusammenhang mit Webdiensten.

**Nachrichten** sind Träger von primitiven oder speziellen Typen und entsprechen den versendbaren Argumenten bzw. empfangbaren Rückgabewerten. Sie werden als XML-Dokumente codiert.

**Porttypen** kapseln die angebotenen abstrakten Operationen. Man unterscheidet vier Arten von Operationen:

- *One-way*. Der Endpunkt erhält eine Nachricht.
- *Request-Response*. Der Endpunkt erhält eine Nachricht und sendet eine Rückgabenachricht.
- *Solicit-response*. Der Endpunkt sendet eine Nachricht und empfängt eine Rückgabenachricht.
- *Notification*. Der Endpunkt sendet eine Nachricht.

Operationen mit Rückgabewert bzw. Fehlerausnahme werden synchron genannt, da der Ablauf der Programmlogik zuerst auf den Rückgabewert bzw. die Signalisierung eines Fehlers durch den anderen Dienst wartet. Operationen ohne Rückgabewert werden asynchron genannt.

**Endpunkte** bezeichnen Assoziationen zwischen Bindungen und Netzwerkadressen, in Form einer URI (Uniform Resource Identifier).

### 3 Kombination von Webdiensten

Die Effekte von Geschäftsvorgängen vollziehen sich meist noch innerhalb von lokalen Instanzen, die für die Abwicklung des gesamten Prozesses verantwortlich sind. Die losere und dynamische Verteilung auf mehrere spezialisierte Dienstleister ist bisher höchstens mit proprietären eigenentwickelten Architekturen realisiert worden.

Mit verteilten komponentenbasierten Webdiensten und standardisierter Kommunikation zwischen ihnen, wächst jedoch auch der Wunsch nach komplexeren Geschäftsprozesse, die sich über die Verantwortlichkeit eines Unternehmens flexibel hinweg vollziehen.

Beispielhaft ist in Abbildung 1 eine Reisebuchung skizziert. Der Kunde beginnt den Geschäftsvorgang durch den Aufruf einer Operation am Reisebüro-Webdienst. Dieser Webdienst bucht mit den übermittelten Informationen selbstständig Flug- und Hotelreservierungen mittels anderer Webdienste. Schließlich wird bei einem Webdienst eines Bankinstituts, eine entsprechende Rechnung übermittelt und eine Rückmeldung an den Kunden geschickt

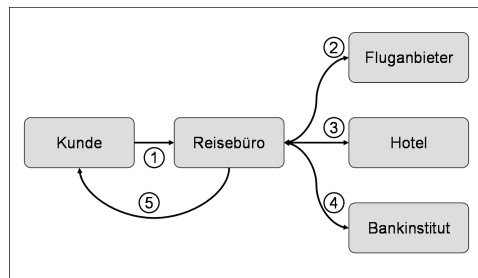


Abbildung 1: Eine Reisebuchung mit mehreren Webdiensten.

Die Webdienst Spezifikationen erlauben Webdienste auf plattformunabhängige und standardisierte Art zu beschreiben, bekannt zu machen und mit ihnen zu kommunizieren. Die Beschreibungsmöglichkeiten von WSDL bleiben jedoch nur auf die atomaren Operationen beschränkt. Die Kombination (so genannte Orchestration) von mehreren Webdiensten geschieht jedoch noch mittels heterogener Ansätze, meist unter der Einsatz klassischer Programmiersprachen, wie beispielsweise Java oder C#. Sie sind jedoch weder auf die speziellen Aspekte der Beschreibung von Geschäftsvorgängen ausgerichtet, noch können mit ihnen die Kommunikationsbeziehungen wirkungsvoll und maschinenverarbeitbar dokumentiert werden.

Deshalb existiert eine allgemeine Nachfrage nach standardisierten Ansätzen zur Beschreibung und Umsetzung von Geschäftsvorgängen, in denen mehrere Webdienste kommunizieren müssen. Mehrere Initiativen haben sich zum Ziel gesetzt, diese Lücke zu schließen. Peltz [9] identifiziert die drei Wichtigsten, als WSCI, BPML und BPEL4WS.

### Web Services Choreography Interface (WSCCI)

WSCCI [3] ist eine Spezifikation, die durch Sun, BEA, SAP und Intalio entwickelt wurde. Sie dient der Spezifikation von so genannten Geschäftsprotokollen, d.h. nach außen sichtbaren Kommunikationsregeln. Dabei dokumentiert jedes WSCCI Dokument auf Basis von WSDL nur die Interaktion eines bestimmten Webdienstes, so dass die Umschreibung eines gesamten Geschäftsvorgangs mehrere Dokumente umfassen kann.

### Business Process Management Language (BPML)

BPML [2] ist eine auf WSCCI aufbauende Spezifikation, die durch die *Business Process Management Initiative* (bestehend unter anderem aus Sun und Intalio) entwickelt wurde. BPML dient der Entwicklung von auf *Business Process Management Systems* (BPMS) ausführbaren Prozessen. Insbesondere erweitert BPML die Spezifikation von WSCCI und ermöglicht ausführbare Vorgänge zu spezifizieren.

### Business Process Execution Language for Web Services (BPEL4WS)

BPEL4WS [1] ist eine Vereinigung, der ehemals unabhängigen Kompositionsspezifikationen XLANG [11] von Microsoft und WSFL [10] von IBM. Sie erlaubt auf Basis von WSDL die Definition sowohl von abstrakten Geschäftsprotokollen, als auch ausführbaren Prozesses.

Die folgende Arbeit befasst sich mit der BPEL4WS-Spezifikation in der aktuellen Version 1.1 (kurz BPEL).

## 4 Dokumentaufbau

BPEL ist eine imperative Programmiersprache für die Beschreibung von Geschäftsvorgängen, an denen mehrere Webdienste beteiligt sind. Sie hat eine XML-Notation und setzt als Grundlage WSDL-Beschreibungen ein. Dadurch kann sie ohne technische Umwege unmittelbar auf die WSDL-Informationen über Nachrichtentypen oder andere Webdienste zugreifen.

Zudem stellt BPEL den beschriebenen Prozess selber als regulären Webdienst bereit. Neue Prozesse werden durch das Aufrufen speziell ausgezeichneter Webdienstoperation des Prozesses instanziiert (vgl. Abschn. 5.3).

BPEL unterscheidet zwischen der Spezifikation von abstrakteren Geschäftsprotokollen und ausführbaren Prozessen. Beide Anwendungen haben eine gemeinsame Schnittmenge von Sprachelementen, die den Grossteil von BPEL ausmacht. Die Syntax wird je nach Anwendungsziel weiter spezifiziert. So benötigen ausführbare Prozesse zusätzliche Informationen, wie z.B. genaue Endpunktadressen der beteiligten Webdienste.

Jede BPEL Beschreibung hat den folgenden Grundaufbau.

```
<process name="ProcessName">
  <partnerLinks>..</partnerLinks>
  <variables>..</variables>
  <correlationSets>..</correlationSets>
  <faultHandlers>..</faultHandlers>
  <compensationHandler>..</compensationHandler>
  <eventHandlers>..</eventHandlers>

  activity
```

```
|</process>
```

Das Dokument besteht aus einem process-Element, welches mit dem name-Attributen zu spezifizierenden Geschäftsvorgang benennt. Bis auf eine Aktivität sind alle Kindelemente optional. BPEL unterscheidet zwischen so genannten Basisaktivitäten und strukturellen Aktivitäten. Basisaktivitäten sind atomare Operationen, die elementare Effekte, wie die Kommunikation mit anderen Diensten bereitstellen. Ihre XML-Elemente sind:

```
receive, reply, invoke, assign, throw, terminate, wait,  
empty, scope, compensate
```

Strukturelle Aktivitäten beschreiben die Ablaufreihenfolgen und Ablaufbedingungen ihrer gekapselten Elemente, welche einfach Basisaktivitäten oder weitere strukturelle Aktivitäten sein können. Erst sie erlauben die mächtige Komposition von Basisaktivitäten. Strukturelle Aktivitäten sind in BPEL die Elemente:

```
sequence, switch, while, pick, flow
```

## 4.1 Umgebungen

Die globale Umgebung beinhaltet alle Elemente des Wurzelements *process*. Unterumgebungen mit lokalem Namensraum lassen sich mit dem scope-Element bilden. Das scope-Element kann, bis auf das partnerLinks-Element, alle Elemente enthalten, die das process-Element enthalten darf. Insbesondere lassen sich so Variablen lokal deklarieren oder die Ausnahmebehandlung in der Zuständigkeit auf bestimmte Bereiche einschränken. Semantisch entspricht dies Bereichen, die in imperativen Programmiersprachen durch Klammersetzung entstehen.

## 5 Kommunikation zwischen Diensten

Die Beschreibung unternehmensübergreifender Prozesse, heißt im Wesentlichen Aussagen über die Kommunikationsbeziehungen der beteiligten Partner zu tätigen. Die Arten von Beziehungen werden als Typen definiert, die mehrmals instanziiert werden können. Die Kommunikation ist dabei oft bidirektional, d.h. Dienste greifen auf entfernte Dienste zu und stellen gleichzeitig eigene bereit.

### 5.1 Deklarieren von Partnerbeziehungen

PartnerLinkType-Elemente erzeugen Kommunikationstypen, indem sie den beteiligten Porttypen Rollennamen zuweisen. Handelt es sich um eine einseitige Kommunikationsbeziehung, so reicht die Angabe einer Rolle. Rufen sich die Instanzen gegenseitig auf, so dass beidseitige Anforderungen an die Porttypen bestehen, werden zwei Rollen spezifiziert. Die tatsächlichen Instanzen dieser Typen sind die PartnerLinks. Ihre Attribute weisen, den in der Kommunikationsbeschreibung deklarierten Rollen, tatsächliche Webdienste zu.

In ausführbaren Prozessen müssen zudem konkrete Endpunkte den Rollen in Partnerlinks zugewiesen werden. Dieser Aspekt ist aber nicht Teil der BPEL Dokumente und bleibt somit den einzelnen BPEL-Implementationen überlassen.

```
|<partnerLinkType name="BuyerSellerLink">
```

```

    <role name="Buyer">
      <portType name="BuyerPT"/>
    </role>
    <role name="Seller">
      <portType name="SellerPT"/>
    </role>
  </partnerLinkType>

  <partnerLinks>
    <partnerLink name="buying" partnerLinkType="BuyerSellerLink"
      myRole="Buyer" partnerRole="Seller"/>
  </partnerLinks>

```

Listing 1: *Definition und Instanziierung eines Beziehungstyps, bei der der Prozess als Käufer auftritt.*

Webdienste können im Laufe eines Geschäftsprozesses auch unterschiedliche Rollen annehmen. Das Partner-Element erlaubt eine Zusammenfassung aller PartnerLinks, an der ein Dienst beteiligt ist.

```

<partners>
  <partner name="Shop">
    <partnerLink name="Seller"/>
    <partnerLink name="Shipper"/>
  </partner>
</partners>

```

Listing 2: *Der Webdienst eines Ladens nimmt hier zu unterschiedlichen Zeitpunkten, die Rollen von Verkäufer und Versender einer Ware an.*

Ist die Art der Kommunikation zwischen den Diensten erfasst, so kann die eigentliche Kommunikation erfolgen.

## 5.2 Ausgehende Webdienstaufrufe

Das invoke-Element dient zum Aufrufen von Operationen, die durch anderer Webdienste bereitgestellt werden. Durch die Angabe des PartnerLinks, Porttypen und der Operation wird die aufzurufende entfernte Operation referenziert. Eventuelle Argumente bzw. Rückgabewerte werden über die Variablenattribute angegeben, welche nur Nachrichtentypen haben dürfen. Zur Erinnerung können Nachrichten, Träger einer beliebigen Anzahl von Werten sein und alle Operationsargumente immer als eine Nachricht versandt.

```

<invoke partnerLink="buying" portType="SellerPT"
  operation="buy" inputVariable="itemid"
  outputVariable="response"/>

```

Listing 3: *Ruft eine verkaufe-Operation mit der itemid-Nachricht beim Verkäufer-Webdienst auf.*

## 5.3 Eingehende Webdienstaufufe

Da der Prozess selber auch ein regulärer Webdienst ist, können nicht nur entfernte Dienste genutzt, sondern auch eigene Operationen angeboten werden. Eingehende Dienstaufufe sind zudem die einzige Möglichkeit einen neuen BPEL Prozess zu instanziiieren.

Receive ist ein blockierende Eingangsoperation, d.h. der Programmablauf fährt erst dann fort, wenn ein Aufruf an den Dienst erfolgt ist, der dann den Effekt der Operation definieren kann. Ist das createInstance-Attribut explizit auf *yes* gesetzt, so wird bei Aufruf der Operation eine neue Instanz des Prozesses an dieser Stelle des Programmlaufs erzeugt.

```
<receive partnerLink="selling" portType="SellerPT"
  operation="getamount" variable="itemid"
  createInstance="yes"/>
```

Listing 4: *Der Prozess wird, durch den Aufruf der Operation buy, begonnen.*

Ist die zu definierende Operation eine synchrone Operation, so erwartet der aufrufende Dienst die Rückgabe eines Wertes (oder Signalisierung eines Fehlers). Dazu dient das reply-Element, welches semantisch analog zur return-Anweisung aus klassischen Programmiersprachen, Rückgaben zurückliefert. Erst mit ihr werden synchrone Operationen abgeschlossen.

```
<reply partnerLink="selling" portType="SellerPT"
  operation="buy" variable="price"/>
```

Listing 5: *Der Aufrufer der Operation buy erhält als Rückgabe einen Preis.*

## 6 Kontrollabläufe

In klassischen imperativen Programmiersprachen ist die Komposition von Effekten durch sequentielle Abarbeitung die elementarste Ablaufreihenfolge. Gerade in verteilten Systemen wird jedoch häufig ein eher paralleler Ablauf bevorzugt, da die Zuverlässigkeit und Latenz einzelner Aktionen nicht bekannt ist und daher das parallele Ausführen nicht gegenseitig bedingender Aktionen in höherer Effizienz resultieren kann.

BPEL stellt beide Ablaufarten als elementare Konstrukte bereit.

### 6.1 Sequentieller Ablauf

Das Sequenz-Element erlaubt eine Zusammenfassung von Aktivitäten, die sequentiell abgearbeitet werden. Eine Aktivität wird erst dann ausgeführt, wenn all ihre vorhergehenden Aktivitäten abgearbeitet wurden.

```
<sequence>
  <!-- activity 1 -->
  <!-- activity 2 -->
</sequence>
```

Listing 6: *Erlaubt die sequentielle Abarbeitung von Aktivitäten.*

## 6.2 Nebenläufiger Ablauf

Das flow-Element ermöglicht die nebenläufige Ausführung ihrer gekapselten Aktivitäten. Zudem können zusätzliche Kausalitätsbeziehungen (auch Synchronisationsbedingungen genannt) zwischen nebenläufigen Aktivitäten definiert werden. Das links-Element deklariert dazu link-Namen, die als Vorbedingung bzw. Effekt anderer Aktivitäten genutzt werden können. Links entsprechen semantisch Korrelationspfeilen, die Abhängigkeiten zwischen eigentlich parallelen und daher unabhängigen Aktionen spezifizieren.

Soll eine Aktion *A* einer Aktion *B*, in einem nebenläufigen Umfeld, zwingend vorhergehen, so ist sie die Quelle eines solchen Links und kapselt das source-Element, wobei das linkName-Attribut einen der zuvor streng statisch getypten link-Namen angibt. Analog wird Aktion *B*, mittels des target-Elements, als das Ziel dieses Links spezifiziert. Aktivitäten, die ein target Element inne haben, müssen solange mit der Ausführung warten, bis die Aktivität, in deren Element das gleichnamige source-Element ist, ausgeführt wurde.

```
<flow>
  <links>
    <link name="AtoB">
  </links>

  <!-- invoke operation A -->
  <invoke ..>
    <source linkName="AtoB"/>
  </invoke>

  <sequence>
    <receive ../>

    <!-- invoke operation B -->
    <invoke ..>
      <target linkName="AtoB"/>
    </invoke>
  </sequence>
</flow>
```

Listing 7: Die invoke-Operation *B* wird erst dann ausgeführt, wenn invoke-Operation *A* abgearbeitet wurde.

Eine Aktivität kann auch das Ziel mehrerer Kausalitätspfeile sein. Das joinCondition-Attribut gibt dann einen Ausdruck an, der die Relation der eingehenden Pfeile angibt. Wird keine Bedingung definiert, so reicht standardmäßig die Erfüllung eines einzelnen Zielpfeils zum Ausführen der Aktivität. Durch die Nutzung der Operation `getLinkStatus`, mit dem der Status eingehender Pfeile abgefragt werden kann, können auch komplexere Bedingungsausdrücke realisiert werden.

## 7 Zustände

In lang anhaltenden Vorgängen zwischen verschiedenen Diensten, müssen sitzungsrelevante Zustände, die z.B. aus Webdienstaufrufen resultieren, gehalten werden können. Dies kann mit Hilfe von Variablen erfolgen. Eigenschaften erlauben den Zugriff auf interne Teile von Nachrichten. Korrelationsmengen fassen Identifikati-

onseigenschaften zusammen.

## 7.1 Variablen

Da BPEL eine strenge statische Typsicherheit besitzt, müssen Variablen vor der ersten Benutzung bestimmten Typs deklariert werden. Dies geschieht über das `variable`-Element, welches als erlaubte Typen WSDL Nachrichtentypen (`messageType`-Attribut), XML Schema primitiver Typ (`type`-Attribut) oder XML Schema Elemente (`element`-Attribut) annehmen kann.

```
<variables>
  <variable name="itemid" type="xsd:int"/>
</variables>
```

Listing 8: *Eine Variable itemid wird als xsd-Typ int deklariert.*

Variablen können nicht nur als Ziel von Rückgabewerten aus Webdienstaufrufen benutzt werden, sondern auch explizit zugewiesen werden.

```
<assign>
  <copy>
    <from>1000</from>
    <to variable="maxprice">
  </copy>
</assign>
```

Listing 9: *Eine Zuweisung, die als Maximalpreis den Wert 1000 festlegt.*

Quellen von Zuweisungen können Variableninhalte, Endpunkt-Referenzen, Ausdrücke (ausgewertet mittels XPath [5]) oder Literale sein. Ziele von Zuweisungen können Variableninhalte oder PartnerLinks sein.

## 7.2 Eigenschaften

Datenabstraktion ist eines der wesentlichen Aspekte moderner Softwareentwicklung: Der äußere Zugriff auf Daten soll soweit wie möglich eingeschränkt werden, um eine losere Kopplung zwischen interner und äußerer Repräsentation zu erreichen.

Geschäftsprozesse kommunizieren mittels Nachrichten, deren Inhalte für sie idealerweise unsichtbar sind. In bestimmten Fällen kann der Ablauf eines Prozesses jedoch vom Inhalt einer solchen Nachricht abhängig sein. So genannte Eigenschaften erlauben deshalb den öffentlichen Zugriff auf interne Inhalte von Nachrichten.

Eigenschaften werden über ein `property`-Element als Eigenschaft eines bestimmten Typs deklariert. Um der Eigenschaft nun ein internes Feld einer Nachricht zuzuweisen, wird ein Alias definiert. Der Zugriff auf die Eigenschaft wird damit zum Stellvertreter des tatsächlichen internen Attributs. Eigenschaften sind vor allem aus Programmiersprachen zur komponentenbasierten Entwicklung bekannt (z.B. Visual Basic oder C#).

```
<property name="userID" type="xsd:string"/>

<propertyAlias propertyName="userID"
  messageType="orderDetails" part="identification"
```

```
| query="/credentials/">
```

Listing 10: Die Eigenschaft *userID* erlaubt öffentlichen Zugriff auf die Identifikationsnummer in der Bestellnachricht.

Um die Eigenschaft einer bestimmten Variable nun auszulesen, dient die Operation `getVariableProperty ('variableName', 'propertyName')`, welche innerhalb von Ausdrücken benutzt werden kann.

### 7.3 Korrelationsmengen

Da Dienste (insbesondere der BPEL Prozess) meist von mehreren Aufrufern genutzt werden, müssen sich diese eindeutig identifizieren lassen, um Nachrichten den korrekten Sitzungen zuordnen zu können.

Üblicherweise benutzt man dafür zusätzliche eindeutige Aufrufargumente, wie z.B. Kundennummern, die beim Aufruf des Dienstes Auskunft über den Aufrufer geben. Gemeinsam haben diese speziellen Nachrichteninhalte, dass sie, nachdem sie initialisiert wurden, für alle weiteren Kommunikationen mit einem Dienst die eindeutige und korrekte Zuweisung der Nachricht zu einer Sitzung ermöglichen sollen.

Unter Korrelationsmengen, versteht BPEL, Mengen von Eigenschaften, die diese konstanten Identifikationsmerkmale von Nachrichten bilden. Sie werden einmalig in Folge einer Aufrufrückgabe oder direkten Zuweisung initialisiert und dann für eine Reihe von Kommunikationen als sitzungsrelevante Identifikation verwendet.

Eine Menge von Eigenschaften wird als Leerzeichen-separierte Liste angegeben und durch das `correlationSet`-Element als neuer Typ bezeichnet. Das `correlationSets`-Element kapselt alle `correlationSet`-Elemente.

```
| <correlationSets>  
|   <correlationSet name="userid"  
|     properties="username , userpw"/>  
| </correlationSets>
```

Listing 11: Die Korrelationsmenge *userid* soll Benutzer sicher identifizieren können. Dazu setzt sie sich aus *Benutzername* und *Passwort* zusammen.

Diese getypten Korrelationsmengen werden nun in Verbindung mit Kommunikationsaktivitäten verwendet. Diese sind *invoke*, *receive*, *reply*, sowie *onMessage* bei *pick* und der Ereignisbehandlung. All diese Operationen können das Element *correlations* kapseln, welches die Korrelationsmengen angibt, die hier zur Identifikation benutzt werden sollen.

Bei eingehenden Nachrichten (d.h. auch Rückgaben von Operationsaufrufen bei entfernten Diensten) haben Korrelationsmengen unterschiedliche Bedeutungen. Ist das *initiate*-Attribut wahr gesetzt, so wird die Korrelationsmenge mit den Werten aus der Nachricht gesetzt. Andernfalls jedoch dient die Korrelationsmenge dazu eingehende Nachrichten automatisch der richtigen lokalen Prozessinstanz zuzuweisen. Demjenigen BPEL Prozess nämlich, der genau die erhaltene Korrelationsmenge erwartet. Das *pattern*-Attribut gibt bei einer synchronen *invoke*-Operation an, ob die Korrelationsmengen sich auf die Argumente oder den Rückgabewert beziehen.

```
| <sequence>  
|   <invoke partnerLink="ordering" portType="ShopPT"
```

```

    operation="login" inputVariable="id">

    <correlations>
      <correlation set="session" initiate="yes"
        pattern="in"/>
    </correlations>
  </invoke>

  <invoke partnerLink="ordering" portType="ShopPT"
    operation="order" outputVariable="itemid">

    <correlations>
      <correlation set="session" initiate="no"
        pattern="out"/>
    </correlations>
  </invoke>
</sequence>

```

Listing 12: Bei einem Bestellsystem wird eine Sitzung durch den Aufruf der Operation *login* eröffnet. Sie liefert eine eindeutige Identifikationsmenge zurück, die benutzt wird, um die eigentliche Bestellung vorzunehmen.

## 8 Ausnahmebehandlung

Operationen stehen gerade in verteilten Systemen unter der Gefahr nicht korrekt abzulaufen. BPEL unterscheidet zwei Konzepte zur Behandlung von Fehlern.

In frühen Programmiersprachen wurden Fehlerfälle als zahlencodierter Rückgabewert von Operationen bekannt gemacht. Modernere Programmiersprachen nutzen dazu so genannte Ausnahmebehandlungen: Benannte Fehlerfälle werden über spezielle Rückgabekonstrukte gemeldet und in einen alternativen Programmablauf von passend deklarierten Ausnahmebehandler abgearbeitet (meist mittels der Schlüsselwörter *try* und *catch*). Dieses Konzept findet sich als Fault-Handler in BPEL wieder.

Tritt ein Fehler auf, so kann ein Abbruch des ganzen Prozesses eine Konsequenz sein. Nicht nur muss dann die fehlgeschlagene Aktion behandelt werden, sondern evtl. auch Effekte vorheriger eigentlich *erfolgreich* ausgeführter Operationen rückgängig gemacht werden. Da die Effekte eines BPEL Geschäftsprozesses sich auch über die Verantwortlichkeit entfernter Komponenten vollziehen, ist dies kein trivialer Vorgang. BPEL bietet ein spezielles Konstrukt für Rücknahmevorgänge, namentlich Compensation-Handler.

### 8.1 Fehlerfälle

Faults sind getypte Fehlerfälle, welche durch den Mechanismus zur Fehlerbehandlung bearbeitet werden. Ergänzende Informationen können über den eigentlichen Rückgabewert angegeben werden. Fehler können auf folgende Arten signalisiert werden:

- Bei synchronen Aufrufen entfernter Webdienste können Fehler signalisiert werden.

- Analog zum Aufruf entfernter Webdienste, können Fehler dem Aufrufer einer bereit gestellten Operation signalisiert werden. Dazu wird das `fault`-Attribut des `reply`-Elementes gesetzt.

```
<reply partnerLink="selling" portType="SellerPT"
      operation="buy" faultName="OutOfStockFault"/>
```

- Das `throw`-Element dient zum expliziten Signalisieren von Fehlern, innerhalb des lokalen Prozesses.

```
<throw faultName="NoDatabaseConnectionFault"/>
```

Fehler werden vom `faultHandler`-Element der Umgebung behandelt. Jedes ihrer `catch`-Elemente ist für einen speziellen Fehlertyp deklariert und wird bei Auftreten des jeweiligen Fehlers ausgeführt. Allgemeine Fehlerbehandlung ist durch das optionale `catchAll`-Element möglich.

```
<faultHandlers>
  <catch faultName="NoDatabaseConnectionFault">
    <compensate/>
  </catch>
  <catchAll>
    <terminate/>
  </catchAll>
</faultHandlers>
```

Listing 13: *Konnte keine Datenbankverbindung aufgebaut werden, soll die Aktion kompensiert werden (vgl. Abschn. 8.2). Bei anderen Fehlern soll der Prozess terminiert werden (vgl. Abschn. 10).*

Das `invoke`-Element erlaubt die `catch`-Elemente auch direkt zu kapseln und somit ad-hoc eine lokale Fehlerbehandlung zu definieren. Fehlt in einer Umgebung die Angabe eines Fehlerbehandlers, so wird eine implizite vordefinierte Fehlerbehandlung angenommen, die

1. Alle Compensation-Handler in der Umgebung in umgekehrter Reihenfolge aufruft.
2. Den Fehler an die höher umschließenden Umgebungen weiterreicht.

## 8.2 Kompensation

Mittels dem `compensationHandler`-Element lassen sich Rückläufe definieren. Häufig beinhaltet dies den Aufruf von speziell bereitgestellten Rücknahmeoperationen an den entfernten Webdiensten, da die Effekte sich meist dort niederschlagen.

```
<compensationHandler>
  <invoke partnerLink="bidding" portType="AuctionPT"
        operation="undoBid" outputVariable="productID">
    <correlations>
      <correlation set="visitorID" initiate="no"
        pattern="out"/>
    </correlations>
  </invoke>
```

```
| </compensationHandler>
```

Listing 14: *Als Folge eines Fehlerfalls, wird das getätigte Auktionsgebot durch den Aufruf einer speziell bereitgestellten Operation wieder zurückgenommen.*

Rückläufe werden über das `compensate`-Element initiiert, welches nur innerhalb von Kompensations- oder Fehlerbehandlern erscheinen darf.

Das optionale `scope`-Attribut kann gezielt Umgebungen bestimmen, andernfalls wird die aktuelle Umgebung bearbeitet. Kompensationsbehandler können dabei nur für komplett erfolgreich abgearbeitete Umgebungen aufgerufen werden. Dies sorgt für eine strikte konzeptionelle Trennung der Zuständigkeiten von Kompensations- und Fehlerbehandlern.

Wird für eine Umgebung keine Kompensationsbehandlung definiert, so werden `compensation`-Aufrufe einfach an die höher einschließenden Umgebungen weitergeleitet.

## 9 Ereignisbehandlung

Ereignisse sind spezielle eingehende Operationsaufrufe oder abgelaufene Zeitlimits. Sie werden bei Eintreten unmittelbar und unabhängig, vom weiter regulär ablaufenden Prozess, behandelt. Meist dienen sie der Realisierung von, zur Laufzeit des Geschäftsvorgangs verfügbaren, Abbruchoperationen.

```
| <eventHandlers>  
|   <onMessage partnerLink="buying" portType="shopPT"  
|     operation="cancel">  
|     <terminate/>  
|   </onMessage>  
| </eventHandlers>
```

Listing 15: *Der Aufruf der cancel-Operation wird als Ereignis aufgefasst und resultiert in dem Abbruch des Geschäftsvorgangs (vgl. Abschn. 10).*

## 10 Weitere Aktivitäten

**switch** Das `switch`-Element erlaubt eine Fallunterscheidung, bei dem der Programmablauf bearbeitet wird, dessen Fallbedingung erfüllt wird. Optional kann mittels `otherwise` auch ein alternativer Ablauf definiert werden, der zur Geltung kommt, wenn keiner der gegebenen Fälle eingetreten ist.

Semantisch entspricht das `Switch`-Element, klassischen `switch`-Anweisungen aus imperativen Sprachen, bei dem jeder Fall mit einem `'break'` beendet wurde, d.h. die alternativen Programmabläufe für die Fälle schließen sich gegenseitig.

```
| <switch>  
|   <case condition="amout>0">  
|     <invoke .. />  
|   </case>  
|   <otherwise>  
|     <receive .. />  
| </switch>
```

```

    </otherwise>
  </switch>

```

Listing 16: *Eine einfache Fallunterscheidung, die if-Semantik nachbildet.*

**pick** Das Verhalten gleichzeitig wartender receive-Operationen ist nicht definiert. Das pick-Element erlaubt deshalb die Kombination einer switch und receive-Semantik. Die Attribute der onMessage-Elemente sind analog zu dem des receive-Elements und ermöglichen das Warten auf den ersten Aufruf einer der eingehenden Operationen. Es ist also auch möglich neue Prozesse zu instanziiieren (vgl. Abschn. 5.3). Das Behandeln der Aufrufe ist, wie bei der normalen switch-Anweisung gegenseitig ausgeschlossen.

```

<pick>
  <onMessage partnerLink="selling" portType="SellerPT"
    operation="getPrice" variable="itemid">
    <!-- activities -->
  </onMessage>
  <onMessage partnerLink="selling" portType="SellerPT"
    operation="buy" variable="itemid">
    <!-- activities -->
  </onMessage>
</pick>

```

Listing 17: *Das Warten auf einen Aufruf von buy oder getPrice.*

Zusätzlich kann durch ein onAlarm-Element, auch die maximale Wartezeit angegeben werden.

**while** Die gekapselten Anweisungen des while-Elements werden solange ausgeführt, wie die Schleifenbedingung erfüllt bleibt.

```

<while condition="bpws:getVariableData('counter')<10">
  <!-- some activities -->
</while>

```

Listing 18: *Eine einfache Schleife.*

**wait** Das wait-Element blockiert den Programmablauf für eine beliebige Zeitdauer.

```

<wait until="'2004-01-09T00:00+01:00'"/>

```

Listing 19: *Eine Warteweisung, die bis zum 9.1.2004 wartet.*

**terminate** Das terminate-Element beendet die Instanz eines Geschäftsprozesses, d.h. alle laufenden Aktivitäten werden abgebrochen. Diese Operation steht nur in Beschreibungen von ausführbaren Geschäftsprozessen zur Verfügung.

**empty** Das empty-Element ist eine leere Aktivität, die keinerlei Effekte oder Rückgaben hat. Aus syntaktischen Gründen kann der Gebrauch eines solchen Elements notwendig sein.

## 11 Konklusion

Bisherige proprietäre Ansätze zur Orchestration von Webdiensten widersprechen dem Gedanken von plattformunabhängiger Webdienstkommunikation. Das Fehlen

einer speziellen und standardisierten Beschreibungssprache ist deshalb allgemein als Lücke der aktuellen Webdienst Spezifikationen erkannt worden.

BPEL4WS fügt sich fließend in die aktuellen Webdienste ein, indem es WSDL als Basis nutzt. Zudem verbindet es bereits jetzt die Erfahrung aus zwei Spezifikationen in einem einzelnen, übersichtlichen und ausgereiften Ansatz. Schlussendlich könnte nicht zuletzt auch die Tatsache, dass mit Microsoft und IBM zwei sehr große Marktmächte hinter dem Entwurf stehen, den Ausschlag zugunsten von BPEL4WS geben.

## Literatur

- [1] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. *Business Process Execution Language for Web Services, Version 1.1*. BEA Systems, IBM International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems, May 2003. Am 9.1.2004 unter <ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf>.
- [2] Assif Arkan. *Business Process Modeling Language*. Business Process Management Initiative, November 2002. Am 9.1.2004 unter <http://www.bpmi.org/specifications.esp>.
- [3] A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takasci-Nagy, I. Trickovic, and S. Zimek. *W3C Note. Web Service Choreography Interface (WSCI) 1.0*. W3C, August 2002. Am 9.1.2004 unter <http://www.w3.org/TR/wsci/>.
- [4] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. *W3C Note. Web Services Description Language (WSDL) 1.1*. W3C, March 2001. Am 9.1.2004 unter <http://www.w3.org/TR/wsdl>.
- [5] James Clark and Steve DeRose. *W3C Recommendation. XML Path Language (XPath) Version 1.0*. W3C, November 1999. Am 9.1.2004 unter <http://www.w3.org/TR/xpath>.
- [6] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jaceques Moreau, and Hernik Frystyk Nielsen. *W3C Recommendations. SOAP Version 1.2*. W3C, June 2003. Am 9.1.2004 unter <http://www.w3.org/2000/xp/Group/>.
- [7] Hugo Haas and Allen Brown. W3C working draft. Web Services Glossary., August 2003. Am 9.1.2004 unter <http://www.w3.org/TR/ws-gloss/>.
- [8] OASIS. *UDDI Version 3.0.1*, October 2003. Am 9.1.2004 unter <http://uddi.org/pubs/uddi-v3.0.1-20031014.pdf>.
- [9] Chris Peltz. Web Services orchestration. A review of emerging technologies, tools, and standards., January 2003.
- [10] Frank Prof. Dr. Leymann. *Web Services Flow Language (WSFL 1.0)*. IBM, May 2001. Am 9.1.2004 unter <http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>.
- [11] Satish Thatte. *XLANG : Web Services for Business Process Design*. Microsoft, 2001. Am 9.1.2004 unter [http://www.gotdotnet.com/team/xml\\_wsspecs/xlang-c/default.htm](http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm).