Modellierung von Geschäftsprozessen mit BPEL4WS

Seminararbeit von
Sonja Reindl

Abstract
Die Business Process Execution Language for Web Services (BPEL4WS) ermöglicht es, sowohl Geschäftsprozesse zu beschreiben, welche Web Services nutzen, als auch Geschäftsprozesse selbst zu einem Web Service zusammenzufassen. BPEL4WS liefert eine XML-basierte Sprache für die formale Spezifikation von Geschäftsprozessen. Die vorliegende Arbeit gibt eine Einführung in die Sprache BPEL4WS.

Abkürzungsverzeichnis:

BPEL	Business Process Execution Language
BPEL4WS	Business Process Execution Language for Web Services
OASIS	Organization for the Advancement of Structured Information Standards
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
UDDI	Universal Description, Discovery, and Integration
WSFL	Web Service Flow Language
WSDL	Web Service Description Language
XLANG	XML-based LANGuage
XML	eXtensible Markup Language

Inhaltsverzeichnis

1 Einleitung 3
2 Geschäftsprozesse und Business Process Executing Language 3
3 Komposition von Web Services 4
3.1 WSDL-Beschreibung 5
3.1.1 Schnittstellenbeschreibung 5
3.1.2 Nachrichten 6
3.2 Prozesse in BPEL4WS 7
3.2.1 Partner 8
3.2.2 Daten 8
3.2.3 Elementare Aktivitäten 9
3.2.4 Strukturierte Aktivitäten 10
3.2.5 Scopes 12
3.3 Installation 13
4 Beispiel für den Einsatz von BPEL 15
4.1 Schnittstelle zum Kunden 16
4.2 Schnittstelle zum Lieferanten 17
4.3 Schnittstelle des Prozesses 18
4.4 BPEL-Prozess 19
4.4.1 Partner 19
4.4.2 Daten 20
4.4.3 Fehlerbehandlung 20
4.4.4 Aktivitäten 21
5 Fazit 25
6 Literaturverzeichnis 26

1 Einleitung

Das Ziel der Softwareentwicklung besteht in der Realisierung zuverlässiger Software, die leicht zu warten und wieder verwendbar ist. Um dieses Ziel zu erreichen wurde das Konzept der Serviceorientierten Architektur (SOA) entwickelt. Funktionen werden nur einmal implementiert und dann als Service über standardisierte Schnittstellen anderen Programmen zur Verfügung gestellt. Dadurch können Redundanzen in der Softwareentwicklung vermieden und die Qualität der Software gesteigert werden, denn jeder Service muss nur einmal implementiert werden. Da Services in unterschiedlichen Programmiersprachen und auf verschiedenen Systemplattformen realisiert werden können, ist es mit SOA möglich, bereits bestehende Systeme einzubinden.
Diese Art der Softwareentwicklung ist mit BPEL4WS möglich, denn mit Business Process Execution Language for Web Services kann man sowohl Geschäftsprozesse beschreiben, die Web Services nutzen, als auch Geschäftsprozesse selbst zu einem Web Service zusammenzufassen. BPEL4WS liefert eine XML-basierte Sprache für die formale Spezifikation von Geschäftsprozessen.
Die vorliegende Arbeit basiert im Wesentlichen auf [LeRo05] und [Juri06].

Zunächst gibt es eine Einführung in BPEL4WS. Danach werden die wichtigsten Sprachelemente von BPEL vorgestellt.
Abschließend wird an einem einfachen Beispiel ein BPEL-Prozess erläutert. Dabei werden einige Sprachelement näher beschrieben.

2 Geschäftsprozesse und Business Process Executing Language

BPEL4WS (oder kurz BPEL) ist eine XML-basierte Sprache zum Beschreiben von Geschäftsprozessen. Sie vereinigt die von IBM entwickelte Web Service Flow Language (WSFL) [Wes02], die eine Unterstützung für graphenbasierte Prozessmodellierung liefert, und Microsofts XLANG [MSDN06], welche eine strukturierte Sichtweise auf Prozesse ermöglicht.
Bei BPEL unterscheidet man **ausführbare** Prozesse, die auf einer BPEL-Engine ausgeführt werden können, und **abstrakte** Prozesse. Die abstrakten Prozesse beschreiben lediglich den Nachrichtenaustausch zwischen Web Services und nicht die Details des Geschäftsprozesses selbst. Sie werden als Sicht auf einen ausführbaren Prozess verwendet und dienen dazu, das interne Verhalten des Prozesses z. B. vor einem Geschäftspartner zu verbergen. Da sowohl abstrakte als auch ausführbare Prozesse die BPEL Sprachelemente gleichermaßen nutzen, wird im Folgenden keine Unterscheidung gemacht.
BPEL-Geschäftsprozesse bieten die Möglichkeit, andere Web Services einzubinden. Der beschriebene Prozess ist selbst wieder ein Web Service, der von anderen Prozessen genutzt werden kann. Dieser so definierten Prozessebene sind keine Grenzen gesetzt. Es können stets, den fachlichen Anforderungen entsprechend, neue Funktionen zusammengestellt werden und so Geschäftsprozesse in jeder Unternehmensebene abgebildet werden. Ausführbare BPEL-Prozesse können auf einer Workflow-Engine zum Einsatz gebracht werden und sind durch diese ausführbar.
BPEL4WS nutzt die industriellen Standards [CCMW01], [OASI04] und [W3C07].

- WSDL (Web Service Description Language) ist eine XML-basierte Sprache zur Beschreibung von Schnittstellen der Web Services.

- UDDI (Universal Description, Discovery and Integration) ist ein Plattform unabhängiger und erweiterbarer Standard, um Web Services zu beschreiben, die Anbieter von Web Services zu finden und um Web Services in andere Web Services einzubinden bzw. zu neuen Web Services zusammenzusetzen.
- SOAP (Simple Object Access Protocol) gilt als einfaches Protokoll und bietet die Grundfunktionalität zur Kommunikation und den Datenaustausch zwischen Web Services.

Insbesondere zur Schnittstellenbeschreibung der einzelnen Services wird von BPEL die Web Service Description Language (WSDL) benutzt. Sie definiert eine plattform-, programmiersprachen- und protokollunabhängige XML-Spezifikation zur Beschreibung des Nachrichtenaustauschs von Web Services. Alle externen Ressourcen und Partner werden durch WSDL-Schnittstellen beschrieben. Aber nicht nur diese, sondern auch der BPEL-Prozess selbst, besitzt eine WSDL Schnittstellenbeschreibung, so dass dieser nach außen wie ein Web Service interagiert.

Die erste Version BPEL4WS Version 1.0 wurde im August 2002, die zweite BPEL4WS Version 1.1 im Mai 2003 herausgegeben. Im April 2007 ist die WS-BPEL Version 2.0 als de facto Standard von OASIS (Organization for the Advancement of Structured Information Standards) erklärt worden [OASI07]. Im Folgenden wird die BPEL4WS in der Version 1.1 [ACD+03] näher betrachtet.

3 Komposition von Web Services

Ein Geschäftsprozess ist eine Folge von geschäftlichen Tätigkeiten, um ein bestimmtes Ergebnis zu erzielen [StHa02]. Geschäftsprozesse in ihrer Gesamtheit setzen die Geschäftsaufgabe um. Dafür ist es manchmal nötig, andere Geschäftsprozesse in Form von Web Services einzubinden.
Somit spezifiziert ein Geschäftsprozess die Ausführung von involvierten Web Services in einer bestimmten Reihenfolge und den Datenaustausch zwischen den Web Services und dem Prozess. Die Beschreibung eines Geschäftsprozesses und die damit verbundene Einbindung von Web Services wird Orchestrierung genannt. Die involvierten Web Services können interne (firmeneigene) oder externe (z. B. von Geschäftspartnern verwaltete) Web Services sein.
Mit BPEL4WS ist eine, den Geschäftsregeln entsprechende, Komposition von Web Services möglich. Diese Web Services werden Partner genannt. Dabei können die Partner sowohl Kunden (Dienstnutzer), als auch Dienstanbieter sein. Abbildung 1 zeigt einen BPEL-Prozess, der vom „Kunde“ (Dienstnutzer) aufgerufen wird, und im Laufe der Ausführung des Prozesses die beiden Web Services (Dienstanbieter) einbindet. Die Schnittstellen zwischen dem Prozess und den Partnern, Web Services, werden mittels WSDL beschrieben.

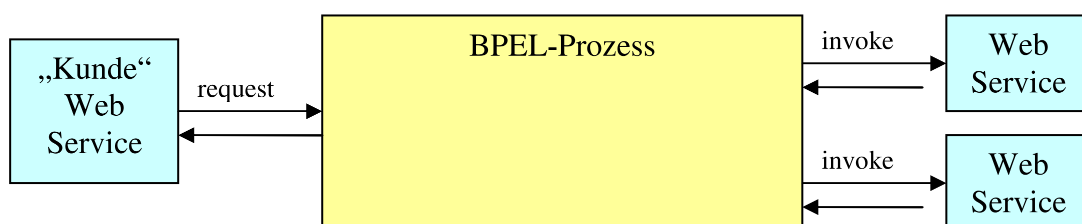


Abbildung 1: BPEL-Prozess

Bei der Beschreibung von Prozessen mit BPEL werden somit zwei verschiedene Datei-Typen benötigt:

- die **WSDL-Dateien**: sie beschreiben die Schnittstellen zu den Web Services. Dabei werden sowohl der Zugriff auf die Services (über `portTypes`) als auch die Nachrichten, die ausgetauscht werden, beschrieben.
- die **BPEL-Datei**: sie enthält die eigentliche Logik des Geschäftsprozesses und bindet andere Web Services ein.

3.1 WSDL-Beschreibung

Jeder Geschäftsprozess kann mit verschiedenen Partnern zusammenarbeiten. Dies können sowohl Kunden als auch andere Services sein, welche dem Prozess ihre Dienstleistungen zur Verfügung stellen.
Wie in Abbildung 2 zu sehen ist, ruft der „Kunde" Web Service den BPEL-Prozess auf und startet ihn somit. Danach wird der Prozess ausgeführt und ruft während dessen andere Web Services auf und empfängt Nachrichten von diesen. Zum Schluss des Prozesses wird dem „Kunde" Web Service eine Antwort-Nachricht geschickt.

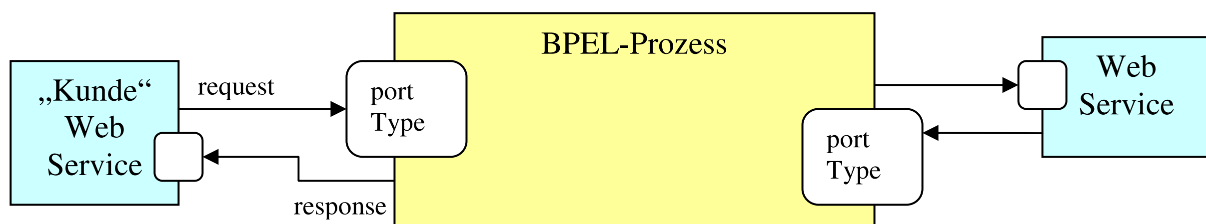


Abbildung 2: Schnittstellen zu Web Services

Die Schnittstellen zu den Web Services werden mittels WSDL beschrieben, wie das auch bei herkömmlichen Web Services der Fall ist. Dieser Umstand bietet dem Anwender den großen Vorteil, dass es keinen Unterschied macht, ob er nun mit einem Web Service im herkömmlichen Sinne oder mit einem BPEL4WS Prozess interagiert.

3.1.1 Schnittstellenbeschreibung

Das Einbinden von Services geschieht über die in WSDL definierten `portTypes` des Services. Diese `portTypes` stellen Operationen zur Verfügung, mit denen Nachrichten ausgetauscht werden können.

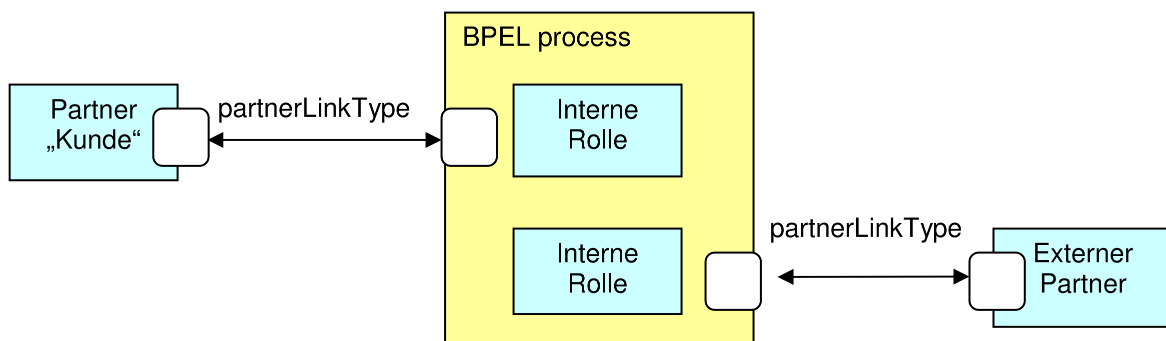


Abbildung 3: partnerLinkType

Ein Partner wird mit dem Prozess über einen `partnerLinkType` verbunden, wie in Abbildung 3 zu sehen ist. Dieser spezifiziert eine Art von Kommunikationsbeziehung zwischen zwei Partnern. Auf der einen Seite befindet sich dabei der externe Web Service (oder auch Kunde) und auf der anderen Seite der interne Partner innerhalb des BPEL-Prozesses.

Die Partner werden als Rollen (`role`) mit dazugehörigen `portTypes` im `partnerLinkType` definiert. Über jenen `portType` ist es dem jeweiligen Service möglich, die entsprechenden Nachrichten zu empfangen. Jede Rolle spezifiziert also genau einen WSDL `portType`. Die `partnerLinkTypes` und `portTypes` werden mittels WDSL außerhalb des BPEL-Prozesses definiert. Dies ist im nachfolgenden Codeausschnitt dargestellt.

```
<partnerLinkType name="nameLT">
    <role name="partnerRolle">
        <portType name="kundePT"/>
    </role>
    <role name="interneRolle">
        <portType name="processPT"/>
    </role>
</partnerLinkType>
```

3.1.2 Nachrichten

Mit BPEL beschriebene Geschäftsprozesse bestimmen den Austausch von Nachrichten zwischen Web Services. Abbildung 4 veranschaulicht den Austausch von Nachrichten zwischen dem BPEL-Prozess und zwei Partnern.

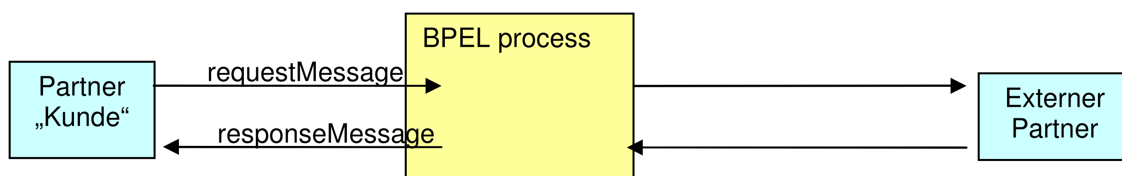


Abbildung 4: Nachrichtenaustausch zwischen Partnern

Diese Nachrichten werden in WSDL mittels des Befehls `messages` erzeugt und können dann durch die Operationen des `portTypes` versendet werden. Dazu wird beim Aufruf des Web Services nicht nur der `portType` angegeben, sondern auch die Operation, die ausgeführt werden sollen. Diese Operationen beziehen sich auf eine spezifische Eingabe- oder Ausgabe-Nachricht (`messages`).

Im folgenden Codeauschnitt werden zwei Nachrichten mit dem `messages`-Element erzeugt und anschließend die beiden `portTypes` mit ihren Operationen, die diese Nachrichten versenden, definiert.

```
<message name="requestMessage">
</message>
<message name="responseMessage">
</message>

<portType name="processPT">
        <operation name="anfragen">
                <input message="requestMessage"/>
        </operation>
</portType>
<portType name="kundePT">
        <operation name="kundeCallback">
                <input message="responseMessage"/>
        </operation>
</portType>
```

Mit den hier vorgestellten WSDL-Definitionen, `messages`, `portTypes` und `partnerLinkTypes` sind nun die Schnittstellen zu den Partnern definiert und man kann beginnen, den BPEL4WS Prozess zu beschreiben.

3.2 Prozesse in BPEL4WS

Die allgemeine Syntax eines BPEL-Prozesses ist in Abbildung 5 zu sehen. Es werden die Verbindungen zu den beteiligten Partner (Web Services) vereinbart (`partnerLinks`), die Variablen für den Prozess angegeben (`variables`) und es können Fehlerbehandlungsroutinen (`faultHandlers`) spezifiziert werden.

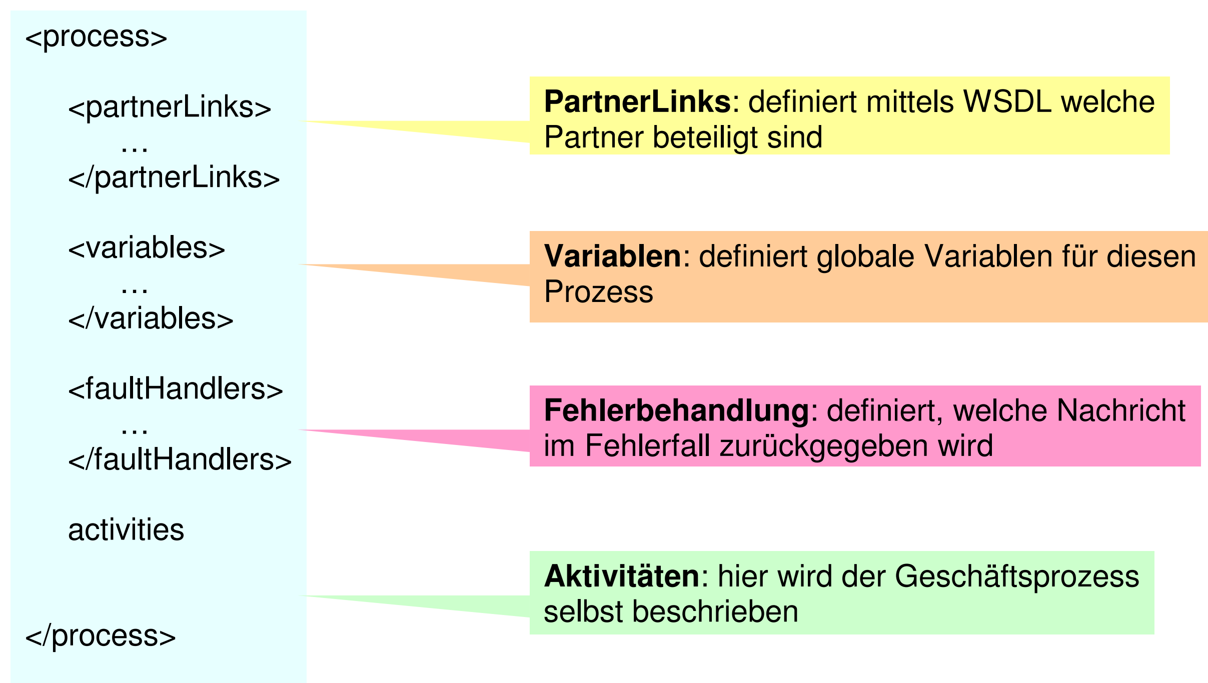


Abbildung 5: Definition eines BPEL-Prozesses

Das Herzstück des Prozesses ist die Angabe von Aktivitäten, die den Geschäftsprozess selbst beschreiben. Diese können elementar oder strukturiert sein. Mittels **elementaren** Aktivitäten

können beispielsweise Web Services aufgerufen oder auf Nachrichten von Web Services gewartet werden. Diese werden im Abschnitt 3.2.3 beschrieben. **Strukturierte** Aktivitäten können wieder elementare oder strukturierte Aktivitäten beinhalten, so dass eine Verschachtelung möglich ist. Diese werden im Abschnitt 3.2.4 näher erläutert.

Damit die Web Services genutzt werden können, müssen diese erst in den Prozess eingebunden werden. Dies wird im folgenden Abschnitt erläutert.

3.2.1 Partner

Für jeden externen Partner wird im BPEL-Prozess ein `partnerLink` eingefügt. Dort wird auch beschrieben, mit welcher Rolle des Prozesses selbst (`myRole`) dieser externe Partner (`partnerRole`) verbunden ist. Diese Rollen wurden schon vorher in WSDL mittels `partLinkType` definiert.
Durch die `partnerRole` wird beschrieben, welcher Web Service des Partners vom Prozess erwartet wird. Das `myRole` Attribut gibt den Service an, der vom Prozess selbst angeboten wird und vom Partner benutzt werden kann. Dieser ist dann nach außen hin sichtbar, während dessen die Logik des Prozesses innerhalb gekapselt ist.

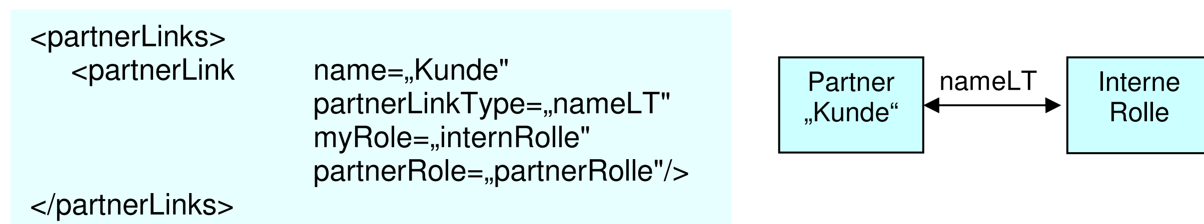
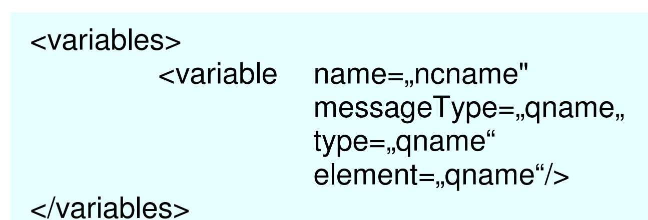


Abbildung 6: partnerLinks

Es können auch mehrere verschiedene externe Partner in einem Prozess definiert werden, die denselben `partnerLinkType` benutzen. Diese werden dann über einen eigenen `partnerLink` definiert.

3.2.2 Daten

Die Nachrichten (`messages`), die mit den Partnern ausgetauscht werden, wurden bereits in WSDL (Abschnitt 3.1) beschrieben. Für den Prozess wichtige Nachrichten werden im Prozess selbst in Variablen mit dem `variable`-Element gespeichert. Die Variablen dienen nur zur Aufnahme von Daten.



Manchmal ist es nötig, empfangene Nachrichten persistent zu speichern, damit sie nicht verloren gehen, bevor der Prozess beendet ist. Dies ist in BPEL möglich mit dem `property`-Element.

3.2.3 Elementare Aktivitäten

Durch Aktivitäten wird die eigentliche Geschäftslogik beschrieben. Die elementaren Aktivitäten sind die grundlegenden – sozusagen atomaren – Aktivitäten, welche nicht aus anderen Aktivitäten aufgebaut sind.
Die folgenden drei Aktivitäten dienen der externen Kommunikation mit Web Services (Partnern). Dies sind die Aktivitäten `receive`, `reply` und `invoke`, wie in Abbildung 7 dargestellt.

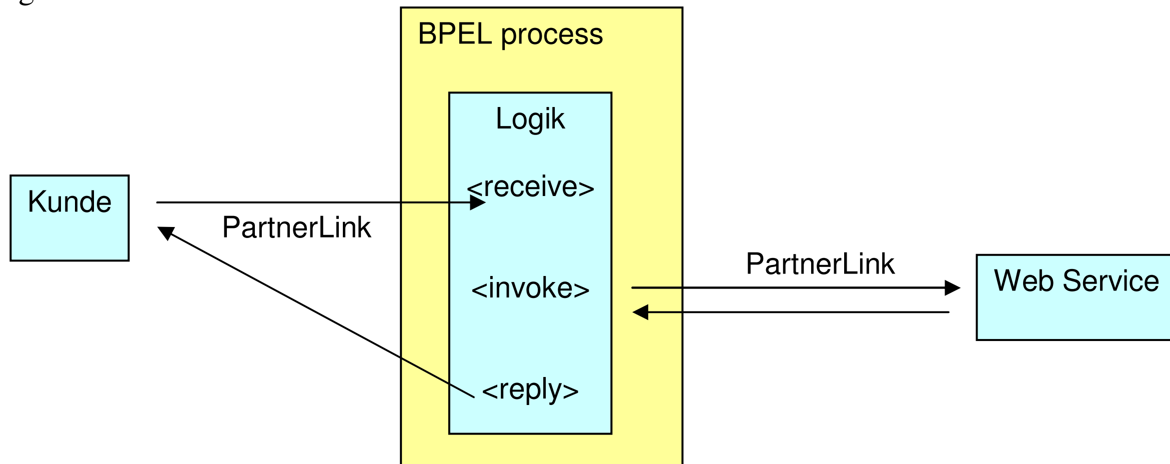


Abbildung 7: Kommunikation mit Web Services

Bei der `receive`-Aktivität wartet der Prozess auf eine Nachricht von einem Partner. Es wird der Partner mittels `partnerLink` benannt, von dem die Nachricht erwartet wird, sein `portType` und die Operation, die vom Prozess dafür bereitgestellt werden. Optional kann die Nachricht, die empfangen wurde, in einer Variablen gespeichert werden.

```
<receive  name=„receiveAnfrage“
          partnerLink=„Kunde"
          portType=„kundePT"
          operation=„opname„
          variable=„vname">
</receive>
```

Um einen Partner eine Antwort auf eine frühere Anfrage, die durch die `receive`-Aktivität entgegen genommen wurde, zu senden, wird `reply` benutzt. Der `reply`-Aktivität kann optional eine Variable zugewiesen werden.

```
<reply    name=„replyAntwort"
          partnerLink=„Kunde"
          portType=„kundePT"
          operation=„opname"
          variable=„vname">
</reply>
```

Oftmals muss ein Prozess die Dienstleistung eines anderen Service in Anspruch nehmen. Dieser wird mittels `partnerLink` eingebunden und kann dann durch die `invoke`-Aktivität aufgerufen werden. Die `invoke`-Aktivität kann entweder *synchron* oder *asynchron* aufgerufen werden. Bei der *synchronen* Variante wird auf eine Antwort von dem Web Service gewartet, bei der *asynchronen* Variante läuft der Prozess weiter und die Antwort des Web Services kann mittels einer `receive`-Aktivität empfangen werden.

synchron:

```
<invoke   name=„invokeService"
          partnerLink=„webService"
          portType=„namePT"
          operation=„opname"
          inputVariable=„iname“
          outputVariable=„oname“>
</invoke>
```

asynchron:

```
<invoke   name=„invokeService"
          partnerLink=„webService"
          portType=„namePT"
          operation=„opname"
          variable=„vname“>
</invoke>
```

In Tabelle 1 werden alle elementaren Aktivitäten kurz beschrieben, zusätzlich zu den schon erläuterten, die noch fehlenden:

Tabelle 1: Elementare Aktivitäten

<receive>	Warten auf Nachrichten von einem Partner
<reply>	Antworten auf eine Anfrage
<invoke>	Aufrufen von Web Service (Partner) Operationen
<assign>	Kopieren von Daten (Variable)
<throw>	Fehlerbehandlungen
<compensate>	Rückgängigmachen einer schon ausgeführten Aktivität
<wait>	Warten auf einen Zeitpunkt oder für eine bestimmte Zeit
<empty>	Leere Anweisung
<terminate>	Beenden des Prozesses

3.2.4 **Strukturierte Aktivitäten**

Strukturierte Aktivitäten beinhalten andere Aktivitäten und lassen so die rekursive Komposition von komplexen Prozessen zu. Mittels einer strukturierten Aktivität kann die Reihenfolge der Ausführung der Aktivitäten festgelegt werden. Dabei können die eingeschlossenen Aktivitäten selbst wieder strukturierte oder aber elementare Aktivitäten sein.
In BPEL stehen folgende strukturierte Aktivitäten zur Verfügung, welche im Folgenden näher beschrieben werden:

- `sequence`: sequenzielle Reihenfolge der Aktivitäten
- `flow`: parallele Ausführung von Aktivitäten
- `while`: Ausführung von Aktivitäten in einer Schleife
- `switch`: bedingte Ausführung von Aktivitäten
- `pick`: Empfangen von Nachrichten eines Partners
- `scope`: Bündelung von Aktivitäten zu einer Transaktion

Durch die `sequence`-Aktivität werden die Aktivitäten sequentiell in der vorgegebenen Reihenfolge abgearbeitet. Dies ist in Abbildung 8 gezeigt.

Abbildung 8: `sequence`-Aktivität

Mit dem `flow`-Konstrukt kann Nebenläufigkeit und Synchronisation von Aktivitäten ausgedrückt werden. Die in dieser Aktivität untereinander aufgelisteten Aktivitäten werden parallel ausgeführt. Durch `links` ist es möglich, die Abfolge der Ausführung der nebenläufigen Aktivitäten zu beeinflussen. Dies wird in Abbildung 9 dargestellt.

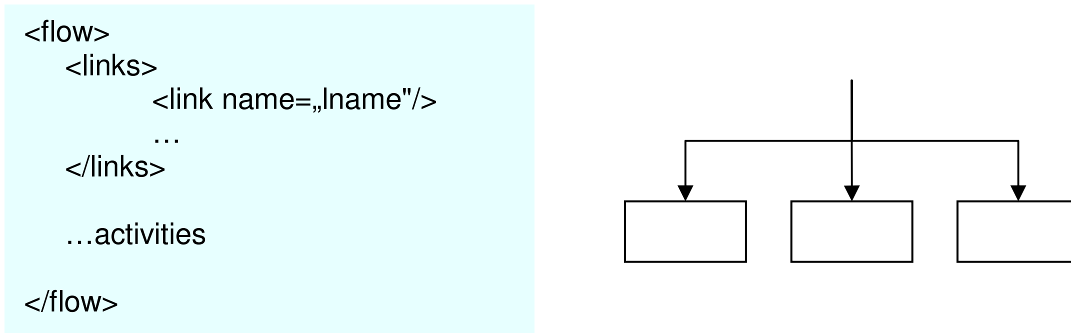


Abbildung 9: `flow`-Aktivität

In Abbildung 10 ist die `while`-Aktivität zu sehen. Sie führt die eingeschlossenen Aktivitäten solange aus wie eine boolesche Bedingung erfüllt ist.

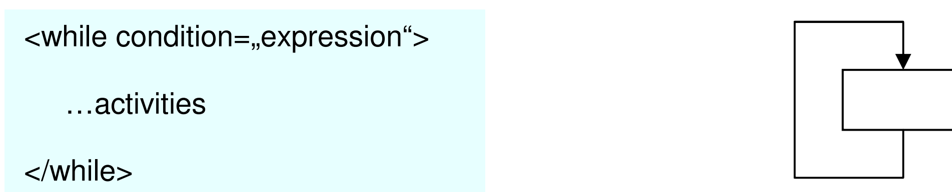


Abbildung 10: `while`-Aktivität

Mit der `switch`-Aktivität ist eine bedingte Ausführung von Aktivitäten möglich, wie in Abbildung 11 gezeigt wird.

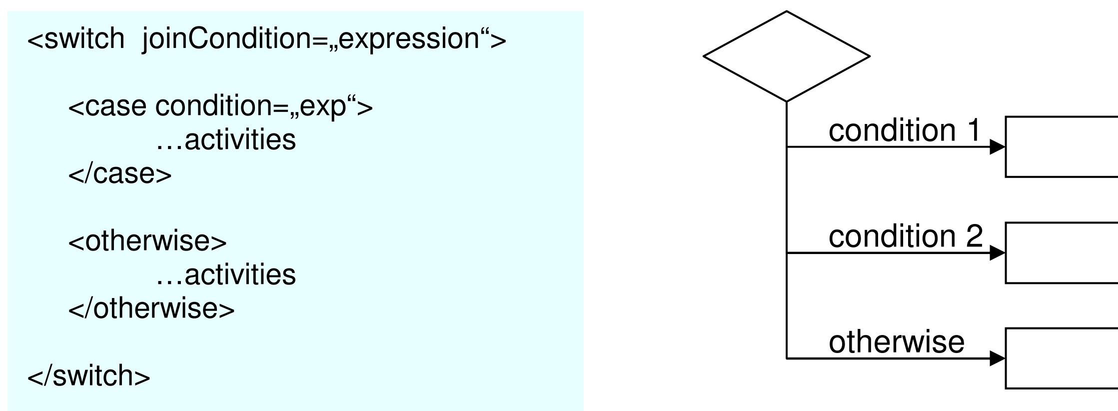


Abbildung 11: `switch`-Aktivität

Die `pick`-Aktivität ist eine Erweiterung von `receive`, da mehrere Nachrichten von einem Partner empfangen werden können. Dabei wartet der Prozess solange bis eine der Nachrichten empfangen wurde, erst dann kann der Prozess weiterlaufen.

```
<pick>
    <onMessage        partnerLink=„pLname"
                      portType=„namePT"
                      operation=„opname„
                      variable=„vname">
              …activities
    </onMessage>

    <onAlarm>
              …activities
    </onAlarm>
</pick>
```

Das `onMessage`-Element wird benutzt, um eine spezifische Nachricht eines Partners mit portType und Operation zu beschreiben. Die Struktur von `onMessage` ist dieselbe wie die der `receive`-Aktivität. Der einzige Unterschied ist die zwingend notwendig folgende Aktivität, die ausgeführt wird, wenn die Nachricht empfangen wurde. Dies kann auch eine `empty`-Aktivität sein, falls nichts zu tun ist.
Das `onAlarm`-Element bewirkt, dass die Aktivität eine gewisse Zeit wartet oder bis zu einer bestimmte Zeit den Prozess anhält. Danach wird die eingeschlossene Aktivität ausgeführt.

Die strukturierte Aktivität `scopes` wird im nächsten Abschnitt genauer behandeln, da sie zu den wichtigsten Aktivitäten gehört.

3.2.5 Scopes

Mit Hilfe dieses Konstrukts können Aktivitäten gebündelt werden und zu einer transaktionalen Einheit zusammengefasst werden. Durch einen `scope` kann einer Gruppe von Aktivitäten eine Fehlerbehandlung (`faultHandlers`), eine Ereignisbehandlung (`eventHandlers`) und eine Kompensationsbehandlung (`compensationHandler`) zugewiesen werden. Erst die Kompensationsbehandlung macht lang andauernde Transaktionen möglich, da bereits ausgeführte Aktivitäten wieder rückgängig gemacht werden können.
Ein BPEL-Prozess selbst ist schon ein `scope` und beinhaltet somit die Möglichkeit auf Fehler oder Ereignisse zu reagieren.

Ein `scope`-Konstrukt ist folgendermaßen aufgebaut:

```
<scope variableAccessSerializable=„yes|no“>
    <variables>
          …
    </variables>

    <faultHandlers>
          …
    </faultHandlers>

    <compensationHandler>
          …
    </compensationHandler>

    <eventHandlers>
          …
    </eventHandlers>

    activity

</scope>
```

Mit der `variableAccessSerializeable`-Eigenschaft können zwei parallele scopes synchronisiert werden, d.h. der Zugriff auf globale Variablen wird kontrolliert. Lokale Variablen können durch `variables` definiert werden.

Bei der Ausführung eines BPEL-Prozesses können Fehler sowohl intern als auch extern in den Services, die aufgerufen werden, auftreten. Daher bietet BPEL4WS die Möglichkeit, so genannte `faultHandlers` (Fehlerbehandlungsroutinen) zu definieren, um explizit Fehler abzufangen und entsprechende Aktivitäten auszuführen.

Damit der Prozess korrekt weiterlaufen kann, ist es manchmal nötig, schon ausgeführte Aktivitäten wieder rückgängig zu machen. Dies kann mittels einer Kompensation (`compensationHandler`) gemacht werden.

Ein ganzer Prozess, aber auch jeder einzelne `scope`, kann eine Menge von Ereignisbehandlungsroutinen besitzen (`eventHandlers`).
Diese Ereignisbehandlung gehört im Gegensatz zur Fehlerbehandlungs- und Kompensationsroutinen zu einem normalen Ablauf eines Prozesses. Es kann hier parallel zum Prozessablauf auf Ereignisse reagiert werden, ohne den Ablauf an sich zu blockieren.

3.3 Installation

Die Installation von BPEL-Prozessen auf einer Workflowmaschine nennt man deployment. Dafür wird neben dem BPEL-Prozess-Dokument und den WSDL-Dateien zusätzlich der so genannte Deployment-Descriptor benötigt. In diesem Dokument werden die benutzten Web Services mit „echten“ Webadressen verbunden. Das Auslagern dieser Beschreibung in eine eigene Datei hat den Vorteil, dass Änderungen an realen Partner-Adressen nur einer Änderung im Deployment Descriptors bedarf. In Abbildung 12 ist die Installation eines BPEL-Prozesses noch einmal dargestellt.

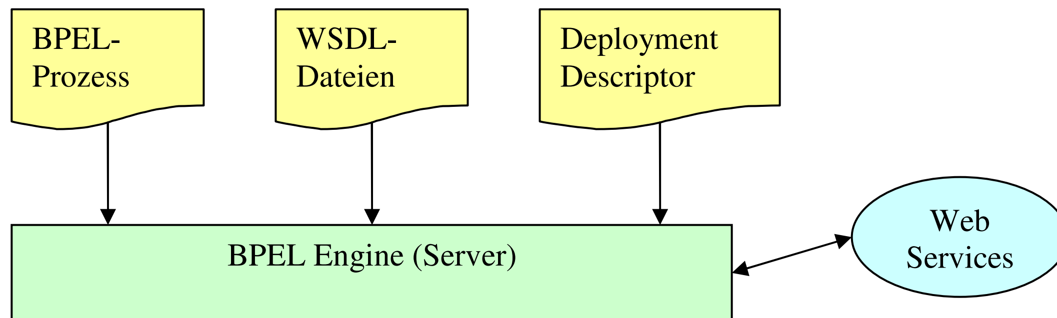


Abbildung 12: Installation eines BPEL-Prozesses

Die Beschreibung dieses Deployment-Descriptors ist abhängig davon, welche Workflowmaschine benutzt wird. Im einfachsten Fall wird der Partner direkt verbunden mit der realen Webadresse. Diese Verbindung geschieht mit dem `link` Element zum entsprechenden `partnerLink` des BPEL-Prozesses und der Zuordnung der `role`. Das `locator` Element gibt dann die reale Adresse der Services an.

```
<deploymentDescriptor>
    <link partnerLink=„partnerL“>
        <role name=„partnerService“>
         <locator type=„static“
                  service=„www.name.com“>
         </locator>
        </role>
    </link>
</deploymentDescriptor>
```

Für komplexe Fälle ist es möglich, diese Webadressen über einen Mechanismus, z. B. mittels Universal Description, Discovery and Integration auswählen zu lassen. UDDI bietet einen Verzeichnisdienst, anhand dessen Web Services, die spezielle Dienste anbieten, ermittelt werden können.

Wie schon erwähnt, kann der BPEL-Prozess auf irgendeiner BPEL-Maschine ausgeführt werden. Dafür ist es lediglich nötig den Deployment-Descriptor anzupassen. Im Folgenden werden einige BPEL-Maschinen aufgelistet:

- SAP Exchange Infrastructure – Implementierung des BPEL-Standards 1.1, grafische Modellierung über ARIS für SAP NetWeaver [SAP04].
- Oracle BPEL Process Manager – Implementierung des BPEL-Standards 1.1, mit grafischer Modellierungs/Orchestrierungs Tool für JDeveloper und Eclipse [Orac06].
- ActiveBPEL – Open Source Implementierung (GPL) von BPEL4WS 1.1 und WS-BPEL 2.0 [Acti07].
- IBM WebSphere Process Server – BPEL-Laufzeitumgebung der IBM basiert auf WebSphere Application Server, dem J2EE Server der IBM [IBM06].
- Microsoft BizTalk Server – BizTalk Server 2006 kann Prozessmodelle ausführen, die mit Modellierungswerkzeugen wie Microsoft Visio erstellt worden sind [Micr06].

4 Beispiel für den Einsatz von BPEL

Im Folgenden soll anhand eines Beispiels die Beschreibung eines Geschäftsprozesses mit BPEL4WS demonstriert werden.
Es wird ein Prozess erstellt, der es einem Kunden ermöglicht, ein Produkt zu bestellen. Für dieses Produkt sind zwei Teilprodukte von zwei verschiedenen Lieferanten nötig. Also muss der Prozess diese beiden Lieferanten als Web Service einbinden. Erst wenn beide Teilprodukte geliefert wurden, kann der Kunde benachrichtigt und das Produkt versandt werden. In Abbildung 13 ist der Prozess dargestellt.
Zur Vereinfachung wird in dem Beispiel davon ausgegangen, dass beide Lieferanten über den gleichen `portType` verfügen (wovon in der Realität natürlich nicht auszugehen ist) und deren Web Services bekannt sind. Da die Lieferung der beiden Teilprodukte längere Zeit in Anspruch nehmen kann, muss das Benachrichtigen des Kunden asynchron geschehen. Dadurch braucht der Kunde nicht unnötig lange auf eine Antwort warten.
Auch die beiden Web Services der Lieferanten sollten asynchron in den BPEL-Prozess eingebunden werden. Damit die Wartezeit nicht unnötig lange dauert, wird das Warten auf die Lieferung zeitlich begrenzt durch eine Fehlerbehandlungsroutine. Außerdem wird der Fehler abgefangen, wenn ein Lieferant nicht erreichbar ist.

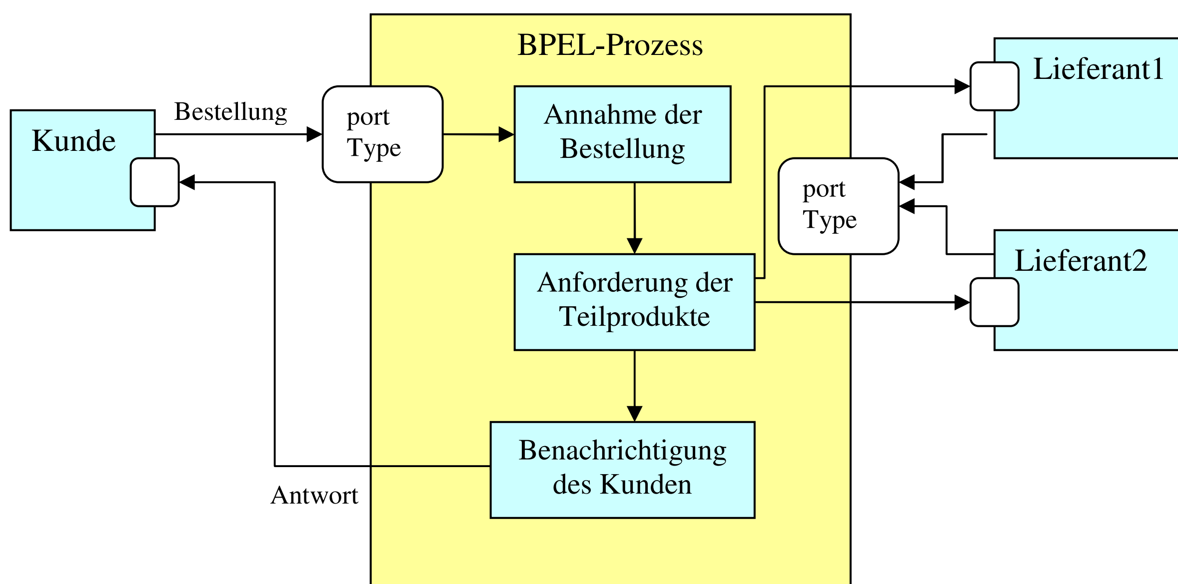


Abbildung 13: Beispiel BPEL-Prozess

Als erstes werden genaue Beschreibungen der Interfaces und der auszutauschenden Nachrichten benötigt. Diese werden in WSDL geschrieben. Für das Beispiel sind drei WSDL-Dateien nötig, da sowohl die Schnittstellen zu den Partnern beschrieben werden müssen als auch die Schnittstelle zum Prozess selbst.

Für die Beschreibung des Beispiels wurde der Oracle JDeveloper verwendet.

4.1 Schnittstelle zum Kunden

Zuerst wird die Schnittstelle zum Kunden beschrieben, also die Nachrichten, die ausgetauscht werden, und die `portTypes`, über die diese Nachrichten verschickt werden. In Abbildung 14 sind der Nachrichtenaustausch und die `portTypes` dargestellt.

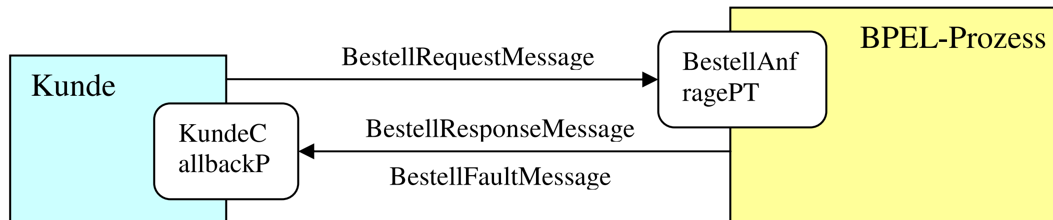


Abbildung 14: Schnittstelle zwischen Kunde und BPEL-Prozess

Durch das `message`-Element werden die Nachrichten definiert. Hier werden drei Nachrichten benötigt, eine für die Kundenanfrage (`BestellRequestMessage`), eine für die Antwort des Kunden (`BestellResponseMessage`) und eine für eine eventuelle Fehlermeldung (`BestellFaultMessage`).
Es werden zwei nun die `portTypes` definiert, einer um Nachrichten beim Prozess zu empfangen (`BestellAnfragePT`) und einer um Nachrichten beim Kunden zu empfangen (`KundeCallbackPT`). Innerhalb des `portType`-Elements werden Operationen (`operation`) definiert, die es ermöglichen, diese definierten Nachrichten zu verschicken. Beim `KundeCallbackPT portType` sind es z. B. zwei Operationen, die eine schickt eine normale Antwort (`clientCallback`) und die andere eine Fehlermeldung (`clientCallbackFault`).

```
<message name="BestellRequestMessage">
        <part name="payload" element="tns:BestellProcessRequest" type="xsd:string"/>
</message>
<message name="BestellResponseMessage">
        <part name="payload" element="tns:BestellProcessResponse" type="xsd:string"/>
</message>
<message name="BestellFaultMessage">
        <part name="error" type="xsd:string"/>
</message>

<portType name="BestellAnfragePT">
        <operation name="anfragen">
                <input message="tns:BestellRequestMessage"/>
        </operation>
</portType>
<portType name="KundeCallbackPT">
        <operation name="clientCallback">
                <input message="tns:BestellResponseMessage"/>
        </operation>
        <operation name="clientCallbackFault">
                <input message="tns:BestellFaultMessage"/>
        </operation>
</portType>
```

4.2 Schnittstelle zum Lieferanten

Nun wird die Schnittstelle zum Lieferanten beschrieben, also die Nachrichten, die zwischen dem Prozess und dem Lieferanten ausgetauscht werden, und die `portTypes`, die dafür benötigt werden. Dies ist in Abbildung 15 dargestellt.

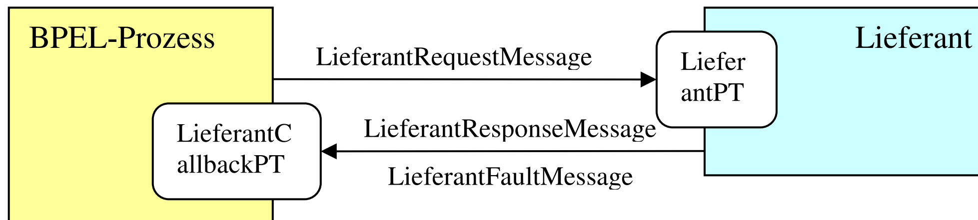


Abbildung 15: Schnittstelle zwischen Lieferant und BPEL-Prozess

Auch hier gibt es drei Nachrichten: eine, um eine Nachricht an den Lieferanten zu schicken (`LieferantRequestMessage`), eine, um die Nachricht vom Lieferanten zu empfangen (`LieferantResponseMessage`) und eine Fehlernachricht vom Lieferanten (`LieferantFaultMessage`).
Es werden zwei `portTypes` definiert: einer, um Nachrichten beim Lieferant zu empfangen (`LieferantPT`) und einer, um Nachrichten beim Prozess zu empfangen (`LieferantCallbackPT`). Beim `LieferantCallbackPT portTypes` gibt es zwei Operationen, die eine schickt eine normale Antwort (`lieferantCallback`) und die andere eine Fehlermeldung (`lieferantNotAvaliable`).

```
<message name="LieferantRequestMessage">
        <part name="payload" element="tns:LieferantProcessRequest" type="xsd:string"/>
</message>
<message name="LieferantResponseMessage">
        <part name="payload" element="tns:LieferantProcessResponse" type="xsd:string"/>
</message>
<message name="LieferantFaultMessage">
        <part name="error" type="xsd:string"/>
</message>

<portType name="LieferantPT">
        <operation name="bestellen">
                <input message="tns:LieferantRequestMessage"/>
        </operation>
</portType>
<portType name="LieferantCallbackPT">
        <operation name="lieferantCallback">
                <input message="tns:LieferantResponseMessage"/>
        </operation>
        <operation name="lieferantNotAvaliable">
                <input message="tns:LieferantFaultMessage"/>
        </operation>
</portType>
```

4.3 Schnittstelle des Prozesses

Zum Schluss wird die Schnittstelle zum Prozess selbst in WSDL beschrieben. Dazu muss als erstes in der WSDL-Datei des Prozesses die schon erstellten WSDL Informationen des Kunden und des Lieferanten bekannt gemacht werden. Dies geschieht durch den `import` der beiden WSDL-Dateien „KundeService.wsdl“ und „LieferantService.wsdl“. Zur Zuordnung der Attribute (Nachrichten und `portTypes`) wird der `namespace` zu dem entsprechenden link des Services angegeben. Damit sind auch hier die gleichen `portTypes` und die Definitionen der Nachrichten vorhanden.

```
<import location="./KundeService.wsdl"
        namespace="http://xmlns.oracle.com/KundeService"/>
<import location="./LieferantService.wsdl"
        namespace="http://xmlns.oracle.com/LieferantService"/>
```

Die Web Services (Kunde und Lieferant) müssen mit dem Prozess verbunden werden. Dies geschieht durch das `partnerLinkType`-Element. Mit `role` wird angegeben, welche Rolle der Service annimmt, der über diesen Link erreicht wird, und welcher `portType` benutzt werden soll. In Abbildung 16 sind die `partnerLinkTypes` zu sehen.

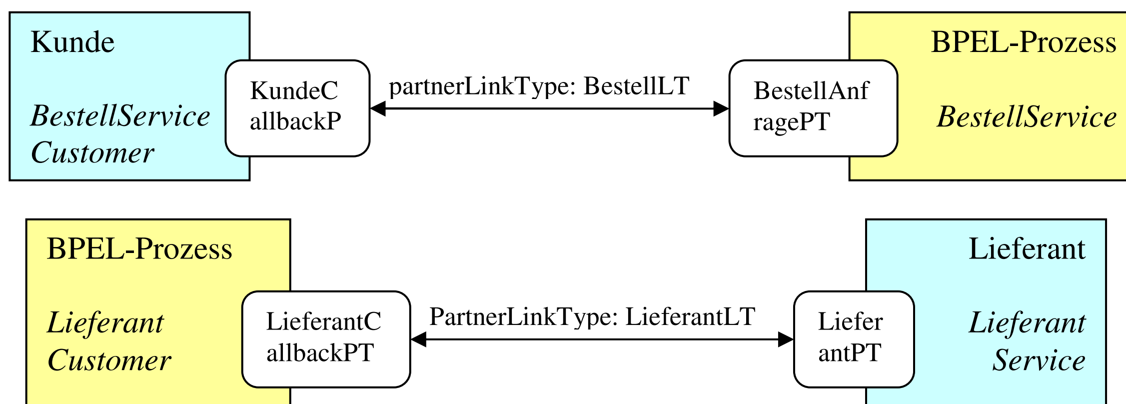


Abbildung 16: Rollen der verschiedenen Partner

Im folgenden Codefragment wird der `partnerLinkType` zwischen dem Web Service „Kunde“ und dem BPEL-Prozess beschrieben:

```
<plnk:partnerLinkType name="BestellLT">
        <plnk:role name="BestellService">
                <plnk:portType name="client:BestellAnfragePT"/>
        </plnk:role>
        <plnk:role name="BestellServiceCustomer">
                <plnk:portType name="client:KundeCallbackPT"/>
        </plnk:role>
</plnk:partnerLinkType>
```

Nun folgt der `partnerLinkType` zwischen dem Web Service „Lieferant" und dem BPEL-Prozess:

```
<plnk:partnerLinkType name="LieferantLT">
        <plnk:role name="LieferantService">
                <plnk:portType name="lieferant:LieferantPT"/>
        </plnk:role>
        <plnk:role name="LieferantCustomer">
                <plnk:portType name="lieferant:LieferantCallbackPT"/>
        </plnk:role>
</plnk:partnerLinkType>
```

Mit den vorgestellten WSDL-Dateien ist nun alles definiert und man kann beginnen, den BPEL4WS Prozess zu schreiben.

4.4 BPEL-Prozess

Der Prozess beginnt mit dem `process`–Tag, der den gesamten Prozess umschließt. Der Namensraum wird definiert und die soeben erstellten WSDL-Dateien werden eingebunden und an die Namespace-Präfixe `tns`, `lieferant` und `client` gebunden, damit man nachvollziehen kann wo entsprechend verwendete Attribute definiert sind.

```
<process name="BestellProcess"
      targetNamespace=http://xmlns.oracle.com/BestellProcess
      xmlns=http://schemas.xmlsoap.org/ws/2003/03/business-process/
      xmlns:tns=http://xmlns.oracle.com/BestellProcess
      xmlns:lieferant=http://xmlns.oracle.com/LieferantService
      xmlns:client=http://xmlns.oracle.com/KundeService>
```

4.4.1 Partner

Als nächstes wird definiert, mit welchen Partnern der BPEL4WS Prozess interagiert. In diesem Fall sind dies der Kunde und die Web Services der Lieferanten (Lieferant1, Lieferant2):

```
<partnerLinks>
        <partnerLink name="Kunde"
                        partnerLinkType="tns:BestellLT"
                        myRole="BestellService"
                        partnerRole="BestellServiceCustomer"/>

        <partnerLink name="Lieferant1"
                        partnerLinkType="tns:LieferantLT"
                        myRole="LieferantCustomer"
                        partnerRole="LieferantService"/>

        <partnerLink name="Lieferant2"
                        partnerLinkType="tns:LieferantLT"
                        myRole="LieferantCustomer"
                        partnerRole="LieferantService"/>
</partnerLinks>
```

Jeder Partner bekommt einen Namen und der `partnerLinkType` wird angegeben, über den man ihn erreicht. Mit `myRole` wird die Rolle angegeben, die der Prozess selbst in Verbindung mit dem Partner spielt, mit `partnerRole` entsprechend die Rolle des Partners.

4.4.2 Daten

Um die Nachrichten, die empfangen werden, zu speichern, müssen Variablen definiert werden. Dies geschieht mit dem `variables`-Element, es wird ein Variablen-Name angegeben und der Nachrichtentyp, der hier abgelegt werden soll. Dieser Nachrichttyp wurde schon in WSDL mit dem `messages`-Element definiert.
Es wird jeweils eine Eingabe- und Ausgabe-Variable für jeden Partner benötigt, zusätzlich wird noch eine Variable für Fehlermeldungen definiert.

```
<variables>
        <variable name="kundeInputVariable"
                messageType="client:BestellRequestMessage"/>
        <variable name="kundeOutputVariable"
                messageType="client:BestellResponseMessage"/>
        <variable name="liefer1InputVariable"
                messageType="lieferant:LieferantRequestMessage"/>
        <variable name="liefer1OutputVariable"
                messageType="lieferant:LieferantResponseMessage"/>
        <variable name="liefer2InputVariable"
                messageType="lieferant:LieferantRequestMessage"/>
        <variable name="liefer2OutputVariable"
                messageType="lieferant:LieferantResponseMessage"/>
        <variable name="lieferFaultVariable"
                messageType="lieferant:LieferantFaultMessage"/>
</variables>
```

4.4.3 Fehlerbehandlung

Mit einer Fehlerbehandlungsroutine `faultHandlers` ist es möglich, auf Fehler zu reagieren und eine entsprechende Aktivität auszuführen. In dem Beispiel werden Fehler behandelt, die beim Warten auf eine Antwort eines Lieferanten entstehen können.
Dies sind in diesem Fall ein Timeout, das Warten wird nach einer gewissen Zeit abgebrochen, und das Nichterreichen eines Lieferanten, dies führt auch zu einem Fehler. In beiden Fällen wird durch die nachfolgende `invoke`-Aktivität der Kunde benachrichtigt.

```
<faultHandlers>
        <catch faultName="tns:callbackTimeout" faultVariable="lieferFaultVariable">
                <invoke partnerLink="Kunde"
                        portType="KundeCallbackPT"
                        operation="clientCallbackFault"
                        inputVariable="lieferFaultVariable"/>
        </catch>
        <catch faultName="tns:LieferantNotAvalaible" faultVariable="lieferFaultVariable">
                <invoke partnerLink="Kunde"
                        portType="KundeCallbackPT"
                        operation="clientCallbackFault"
                        inputVariable="lieferFaultVariable"/>
        </catch>
</faultHandlers>
```

4.4.4 Aktivitäten

Nun beginnt der eigentliche Ablauf des Geschäftsprozesses. Für diesen Prozess wird die strukturierte Aktivität `sequence` als Haupt-Aktivität verwendet, welche dafür sorgt, dass alle in sie eingeschachtelten Aktivitäten (egal ob strukturiert oder elementar) sequentiell in der angegebenen Reihenfolge ausgeführt werden.
Nun kann die Komposition der Web Services beginnen. Die erste Aktivität in einem BPEL-Prozess muss eine `receive-` oder eine `pick`-Aktivität sein. Beide Aktivitäten erwarten eine Nachricht von einem Partner.
Das Sequenz-Diagramm in Abbildung 17 verdeutlicht den Ablauf des Beispiels: Der Kunde startet den BPEL-Prozess, indem er eine Bestell-Anfrage sendet. Dann werden parallel die beiden Web Services der Lieferanten aufgerufen. Dies geschieht asynchron und die Lieferanten schicken ihrerseits die Antworten auf die Anfragen. Zum Abschluss wird der Kunde mittels der Bestell-Antwort informiert.

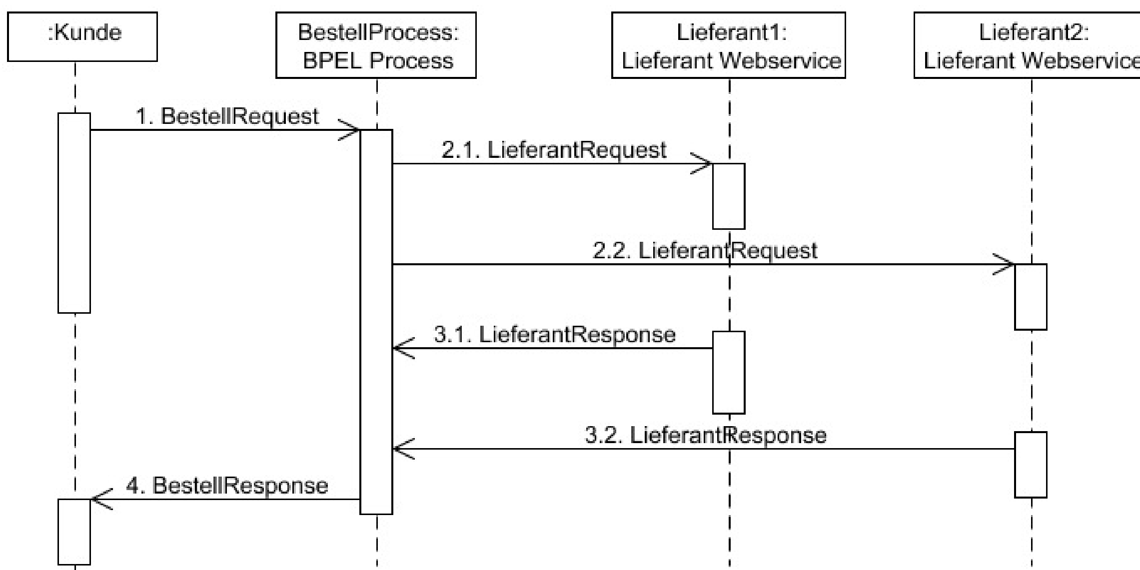


Abbildung 17: Sequenz-Diagramm des BPEL-Prozesses

In dem Beispiel ist `receive` die erste elementare Aktivität, die ausgeführt wird. Über Attribute wird ihr ein Name gegeben, der Partner, der zu verwendende `portType` und die auszuführende `operation` angegeben. Zusätzlich wird noch der Name der `variable` spezifiziert, in den die zu empfangende Nachricht gespeichert werden soll, hier „*`kundeInputVariable`*", wie oben definiert.
Da dies die Aktivität ist, die den Prozess startet, wird das Attribut `createInstance` auf „*yes*" gesetzt. Dadurch wird eine neue Instanz des Bestell-Prozesses gestartet.
Mit diesem `receive` wird also eine `BestellRequestMessage` des Kunden in `variable` „*`kundeInputVariable`*" gespeichert.

```
<sequence name="main">

        <receive name="receiveBestellung"
                 partnerLink="Kunde"
                 portType="client:BestellAnfragePT"
                 operation="anfragen"
                 variable="kundeInputVariable"
                 createInstance="yes"/>
```

Nun ist also eine die Anfrage eines Kunden angekommen, der ein Produkt bestellen möchte. Zur Vereinfachung wird davon ausgegangen, dass dieses Produkt dem Prozess bekannt ist, und auch aus welchen beiden Teilprodukten es besteht.
Also müssen, wie schon beschrieben, die beiden Web Services der Lieferanten aufgerufen werden. Abbildung 18 verdeutlicht das Einbinden der Lieferanten. Da dieses Einbinden parallel geschehen soll, wird die `flow`-Aktivität benutzt. Diese umschließt die Aufrufe (`invoke`) und das Warten auf eine Antwort (`pick`) der beiden Lieferanten.

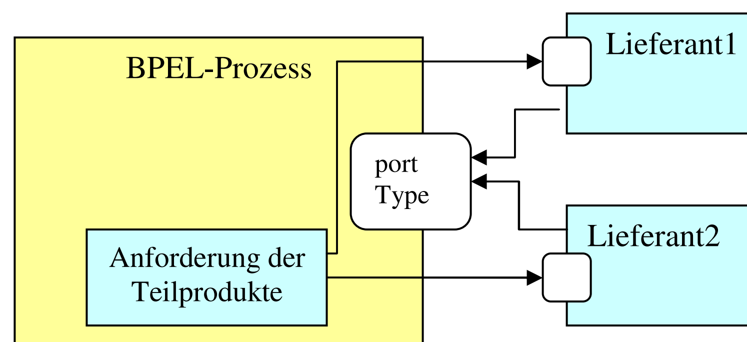


Abbildung 18: Einbinden der Lieferanten

Die `invoke`- und `pick`-Aktivität der einzelnen Lieferanten werden von einer `sequence`-Aktivität umgeben, um die Reihenfolge vorzugeben. Im folgenden Codefragment sind der Aufruf und das Warten auf Nachrichten von Lieferant1 implementiert.

```
<flow>
        <sequence>
                <invoke name="invokeLieferant1"
                         partnerLink="Lieferant1"
                         portType="lieferant:LieferantPT"
                         operation="bestellen"
                         inputVariable="liefer1InputVariable"/>

                <pick name="pickLieferant1">
                        <onMessage partnerLink="Lieferant1"
                                      portType="lieferant:LieferantCallbackPT"
                                      operation="lieferantCallback"
                                      variable="liefer1OutputVariable">
                                <empty/>
                        </onMessage>
                        <onMessage partnerLink="Lieferant1"
                                      portType="lieferant:LieferantCallbackPT"
                                      operation="lieferantNotAvaliable"
                                      variable="lieferFaultVariable">
                                <throw faultName="tns:lieferantNotAvaliable"
                                       faultVariable="lieferFaultVariable"/>
                        </onMessage>
                        <onAlarm for="'P1DT'">
                                <throw faultName="tns:callbackTimeout"
                                       faultVariable="lieferFaultVariable"/>
                        </onAlarm>
                </pick>
        </sequence>
```

Zunächst wird die `invoke`-Aktivität ausgeführt, also dem Partner Lieferant1 wird eine Nachricht geschickt. Wie auch bei `receive` wird über Attribute Name, Partner, etc. angegeben. Da dies ein asynchroner Aufruf ist, also keine direkte Antwort erwartet wird, wird der `invoke`-Aktivität nur die `inputVariable` (*liefer1InputVariable*) mitgegeben.
Die nachfolgende `pick`-Aktivität kann mehrere Nachrichten empfangen. Dies ist erforderlich, um auf eventuelle Fehler reagieren zu können. Das erste `onMessage`-Element reagiert auf die Operation „*lieferCallback*“ und nimmt die empfangende Nachricht in der Variablen *liefer1OutputVariable* entgegen. Danach läuft der Prozess normal weiter, d. h. `empty` bewirkt nichts und die Aktivität `pick` wird beendet.
Das nächste `onMessage`-Element reagiert auf die Operation „*lieferantNotAvaliable*“ und nimmt eine Fehlermeldung in der Variablen *lieferFaultVariable* in Empfang. Danach wird die `throw`-Aktivität ausgeführt, die den Fehler `lieferantNotAvaliable` produziert. Dieser wird in der Fehlerbehandlungsroutine (`faultHandlers`) des Prozesses, wie schon weiter oben beschrieben, behandelt.
Das `onAlarm`-Element bewirkt mit den angegebenen Attributen nach einer Wartezeit von einem Tag („P1DT“) den Fehler `callbackTimeout`, der auch in der Fehlerbehandlungsroutine abgefangen wird.

Nun folgen die Aktivitäten des Partners Lieferant2, die ganz ähnlich aussehen:

```
        <sequence>
                <invoke name="invokeLieferant2"
                        partnerLink="Lieferant2"
                        portType="lieferant:LieferantPT"
                        operation="bestellen"
                        inputVariable="liefer2InputVariable"/>

                <pick name="pickLieferant2">
                        <onMessage partnerLink="Lieferant2"
                                        portType="lieferant:LieferantCallbackPT"
                                        operation="lieferantCallback"
                                        variable="liefer2OutputVariable">
                                <empty/>
                        </onMessage>
                        <onMessage partnerLink="Lieferant2"
                                        portType="lieferant:LieferantCallbackPT"
                                        operation="lieferantNotAvaliable"
                                        variable="lieferFaultVariable">
                                <throw faultName="tns:lieferantNotAvaliable"
                                        faultVariable="lieferFaultVariable"/>
                        </onMessage>
                        <onAlarm for="'P1DT'">
                                <throw faultName="tns:callbackTimeout"
                                        faultVariable="lieferFaultVariable"/>
                        </onAlarm>
                </pick>
        </sequence>
</flow>
```

Danach kann die `flow`-Aktivität wieder beendet werden.

Da beide Lieferanten kontaktiert wurden, wird jetzt mittels einer `invoke`-Aktivität der Kunde informiert. Zuvor wird durch die `assign`-Aktivität die Nachricht „*Die Bestellung wurde verschickt*“ in die `kundeOutputVariable` eingefügt. Natürlich ist es möglich, weitere Informationen an den Kunden zu senden. Darauf wird aber hier nicht weiter eingegangen.
Durch diese `invoke`-Aktivität wird also der Kunde informiert. Danach kann die `sequence` beendet werden und der gesamte Prozess ist am Ende angelangt

```
                <assign>
                        <copy>
                                <from expression="string('Die Bestellung wurde verschickt')"/>
                                <to variable="kundeOutputVariable"/>
                        </copy>
                </assign>
                <invoke name="callbackKunde" partnerLink="Kunde"
                        portType="client:KundeCallbackPT" operation="clientCallback"
                        inputVariable="kundeOutputVariable"/>
        </sequence>
</process>
```

Zum Abschluss des Beispiels wird in Abbildung 19 der mit Oracle JDeveloper erstellte BestellService-Prozess gezeigt.

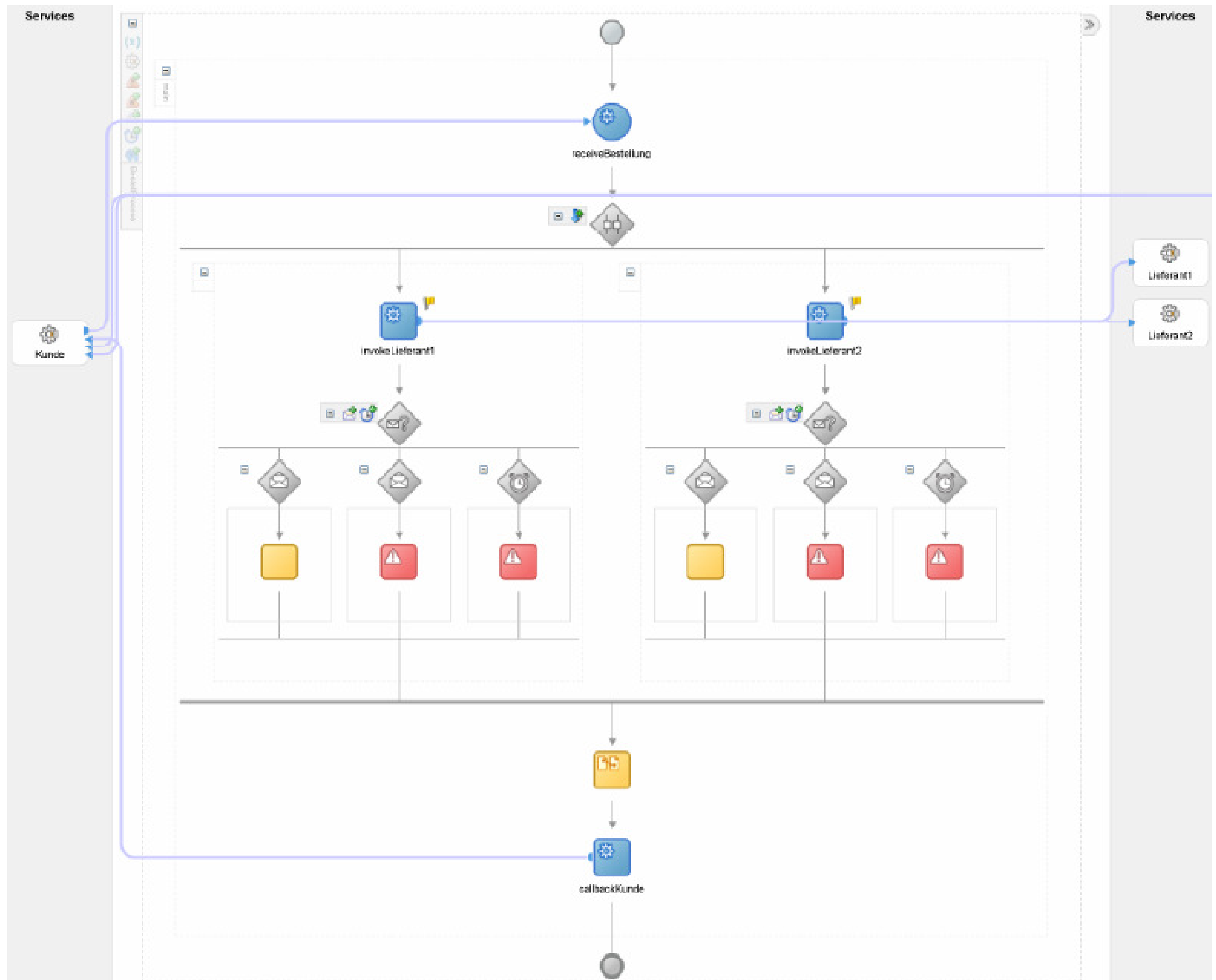


Abbildung 19: BPEL-Prozess: BestellService (erstellt mit Oracle JDeveloper)

5 Fazit

In dieser Seminararbeit wurde aufgezeigt, dass es über BPEL4WS möglich ist, Geschäftsprozesse unter Zuhilfenahme von Web Services zu erstellen. Hierbei stellt der resultierende Geschäftsprozess wiederum einen Web Service dar, welcher als solches in einem weiteren Geschäftsprozess Verwendung finden kann. Somit kann dieses Konzept als geschlossen gelten.

Hinzu kommt, dass die Unterstützung dieser Spezifikation weit vorangeschritten ist. Wie in Abschnitt 3.1 erwähnt stellen einige Firmen Worklow-Engines zur Verfügung und bieten Entwicklungswerkzeuge zum Beschreiben von BPEL-Prozessen an.

In den nächsten Jahren wird BPEL4WS unter dem Namen WS-BPEL wohl an Bekanntheit zunehmen und Unterstützung vieler Anbieter gewinnen, da die Organization for the Advancement of Structured Information Standards (OASIS) die Spezifikation WS-BPEL Version 2.0 im April 2007 als Standard bestätigt hat [OASI07].

Den Anbietern bietet BPEL die Möglichkeit, ihre Geschäftsprozesse flexibel abzubilden, um auf Veränderungen des Marktes schnell reagieren zu können.

6 Literaturverzeichnis

[Acti07]
Active Endpoints: Active BPEL Open Source Engine Version 3.1. http://www.active-endpoints.com/active-bpel-engine-overview.htm, 2007.

[ACD+03]
Andrews, T.; Curbera, F.; Dholakia, H.; Goland, Y.; Klein, J.; Leymann, F.; Liu, K.; Roller, D.; Smith, D.; Thatte, S.; Trickovic, I.; Weerawarana, S.: Business Process Execution Language for Web Services Version 1.1. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf, 2003.

[CCMW01]
Christensen, E.; Curbera, F.; Meredith, G.; Weerawarana, S.: Web Services Definition Language (WSDL) 1.1. *W3C*, http://www.w3.org/TR/wsdl, 2001.

[IBM06]
IBM Corporation: IBM WebSphere Process Server Version 6.0.2. http://www-306.ibm.com/software/integration/wps, 2006

[Juri06]
Juric, M. B.: Business Process Execution Language for Web Services 2nd Edition, *PACKT PUB*, 1/2006.

[LeRo05]
Leymann, F.; Roller, D.: Modeling business processes with BPEL4WS. *Information Systems and E-Business Management*, *4* (2005), 265-284.

[Micr06]
Microsoft Corporation: Microsoft BizTalk Server. http://www.microsoft.com/biztalk, 2006.

[MSDN06]
Microsoft Developer Network: XLANG/s Language. http://msdn2.microsoft.com/en-us/library , Microsoft Corporation, 2006.

[Orac06]
Oracle Corporation: Oracle BPEL Process Manager 10.1.3.1.0. http://www.oracle.com/technology/products/ias/bpel, 2006.

[OASI04]
OASIS: Universal Description, Discovery and Integration (UDDI). http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm#uddiv3, 2004.

[OASI07]
OASIS, WS-BPEL Version 2.0. http://www.oasis-open.org/news/oasis-news-2007-04-12.php (2007).

[SAP04]
SAP AG: SAP NetWeaver Exchange Infrastructure XI. http://www.sap.com/platform/netweaver/components/xi, 2004.

[StHa02]
Stahlknecht, P.; Hasenkamp, U.: Einführung in die Wirtschaftsinformatik. *Springer Verlag*, Berlin 2002.

[Wes02]
Weslay, A.: Web Service Flow Language (WSFL 1.0), IBM Software Group. http://www-128.ibm.com/developerworks/webservices/library/ws-wsfl1, 2002.

[W3C07]
W3C: SOAP Version 1.2. http://www.w3.org/TR/SOAP12, 2007.