Service-orientierte Software-Architekturen

Prof. Dr. U. Hoffmann
FH Wedel

Web Services Business Process Execution Language 2.0 (WS-BPEL 2.0)

Das letzte Mal

Geschäftsprozessmanagement

- **Modellierung**
  - Entwurfsstrategien
  - Werkzeuge
  - erster Blick auf BPEL
  - Standards
  - Alternative Strategien
- **Orchestrierung und Choreographie**

Gliederung

Ein genauerer Blick auf WS–BPEL 2.0

WS-BPEL 2.0

Beziehungen zwischen Prozessen

Verhalten von Prozessen

- Kommunikation
- Kontrollstrukturen
- Parallelverarbeitung
- Datenmanipulation
- Ausnahmebehandlung
- Ereignisbehandlung

Sonstiges

WS-BPEL 2.0

- Zweite Version von BPEL (vorher BPEL4WS 1.1)
- Standardisiert von OASIS:
  Web Services Business Process Execution Language Version 2.0, OASIS Standard 11 April 2007
- BPEL–Einführung von OASIS: BPEL 2.0 Primer
- online unter `www.oasis-open.org/committees/wsbpel/`
- basiert auf vielen anderen XML-Standards
  - XML-Namespaces
  - XML-Schema
  - XPath
  - WSDL
  - ...

Struktur von BPEL–Dokumenten

```
<process name="BeispielProzess"
   targetNamespace="http://fh-wedel.de/ss08/soa/beispiel/"
   xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable" />
```

Ein BPEL–Dokument beschreibt einen Geschäftsprozess

- Ausführbare Prozesse
- Abstrakte Prozesse (teilweise Beschreibung mit offenen Datails)

Struktur von BPEL–Dokumenten

```
<scope name="Scope">
  <!-- local definitions  -->
</scope>
```

- Die Sichtbarkeiten von Definitionen kann mittels `scope` eingeschränkt werden.

Quelle: Beispiele dieser Folien aus OASIS BPEL 2.0 Primer

Partner–Links zu Geschäftspartnern

```
<partnerLinks>
   <partnerLink name="ClientStartUpLink"
      partnerLinkType="wsdl:ClientStartUpPLT"
      myRole="Client" />
</partnerLinks>
```

Partner–Links definieren Kommunikationskanäle zwischen Prozessen

- Verbindung zu einem WSDL–Port (über einen BPEL-PartnerLinkType)

Zusätzliche Angaben in der WSDL:

```
<plnk:partnerLinkType name="EmployeeServiceType">
  <plnk:role name="EmployeeServiceProvider"
             portType="emp:EmployeeInterface" />
</plnk:partnerLinkType>
```

Zustand eines BPEL–Prozesses

Der Zustand eine Geschäftsprozesses wird durch die Werte seiner Variablen bestimmt.

Definition von Variablen:

```
<variables>
   <variable name="myVar1"
        messageType="myNS:myWSDLMessageDataType" />
   <variable name="myVar2"
             element="myNS:myXMLElement" />
   <variable name="myVar3"
             type="xsd:string" />
   <variable name="myVar4"
             type="myNS:myComplexType" />
</variables>
```

- Variablen sind typisiert:
  - WSDL–Nachrichten
  - einfache XML–Schema–Typen
  - komplexe XML–Schema–Typen
  - XML–Schema–Elemente

Empfangen von Nachrichten externer Partner

```
<receive name="ReceiveRequestFromPartner"
   createInstance="yes"
   partnerLink="ClientStartUpPLT"
   operation="StartProcess" ... />
```

`receive-Aktivität`

- atomare Aktivität um Nachrichten von externen Partnern zu erhalten
- `createInstance`: soll ein neuer Prozess erzeugt werden oder verarbeitet der aktuelle Prozess diese Nachricht?

Senden von Antworten

```
<reply name="ReplyResponseToPartner"
   partnerLink="ClientStartUpPLT"
   operation="StartProcess" ... />
```

`reply`–Aktivität

- definiert Antworten auf frühere Anfragen
- Reguläre Antworten oder Fehler–Antworten (`faultName`–Attribut)

Stellen von Anfragen

Anfrage/Antwort:

```
<invoke name="RequestResponseInvoke"
   partnerLink="BusinessPartnerServiceLink"
   operation="RequestResponseOperation"
   inputVariable="Input"
   outputVariable="Output" />
```

Einweg:

```
<invoke name="OneWayInvoke"
   partnerLink="BusinessPartnerServiceLink"
   operation="OneWayOperation"
   inputVariable="Input" />
```

- Aufruf einer Operation eines Web–Services eines Partners
- Aufruf von Anfrage/Antwort–Operationen
- Aufruf von Einweg–Operationen

Hintereinanderausfürhung: `sequenz`–Aktivität

```
<sequence name="InvertMessageOrder">
   <receive name="receiveOrder" ... />
   <invoke name="checkPayment" ... />
   <invoke name="shippingService" ... />
   <reply name="sendConfirmation" ... />
</sequence>
```

Hintereinanderausführen von Aktivitäten

- Die Teilaktivitäten werden nacheinander ausgeführt.

Fallunterscheidung: `if-else`-Aktivität

```
<if name="isOrderBiggerThan5000Dollars">
 <condition>
    $order &gt; 5000
 </condition>
 <invoke name="calculateTenPercentDiscount" ... />
 <elseif>
  <condition>
     $order &gt; 2500
  </condition>
  <invoke name="calculateFivePercentDiscount" ... />
 </elseif>
 <else>
   <reply name="sendNoDiscountInformation" ... />
 </else>
</if>
```

Fallunterscheidung

- Ausdruckssprache: XPath (default)

Schleifen: `while`– und `RepeatUntil`–Aktiviät

```
<while>
   <condition>
      $iterations &gt; 3
   </condition>
   <invoke name="increaseIterationCounter" ... />
</while>

<repeatUntil>
   <invoke name="increaseIterationCounter" ... />
   <condition>
      $iterations &gt; 3
   </condition>
</repeatUntil>
```

Aktivitäten wiederholen, bis eine Bedingung erfüllt ist.

Schleifen: `foreach`–Aktivität

```
<forEach parallel="no" counterName="N" ...>
   <startCounterValue>1</startCounterValue>
   <finalCounterValue>5</finalCounterValue>
   <scope>
      <documentation>
          check availability of each item ordered
      </documentation>
      <invoke name="checkAvailability" ... />
   </scope>
</forEach>
```

- feste Anzahl von Schleifendurchläufen (final-start+1)
- `forEach` muss ein `scope`–Element enthalten
- Sequentielle oder parallele Abarbeitung möglich

Nebenläufigkeit: `flow`–Aktivität

```
<flow ...>
   <links> ... </links>
   <documentation>
      check availability of a flight, hotel and
      rental car concurrently
   </documentation>
   <invoke name="checkFlight" ... />
   <invoke name="checkHotel" ... />
   <invoke name="checkRentalCar" ... />
</flow>
```

Nebenläufige Ausführung von Aktivitäten

- Synchronisation zwischen Aktivitäten durch `link`
- `source` muss beendet sein, bevor `target` gestartet wird.
- bedingte Links: `target` startet nur, wenn Bedingung erfüllt

Nebenläufigkeit: `flow`–Aktivität mit Links

```
<flow ...>
   <links>
      <link name="checkFlight-To-BookFlight" />
   </links>
   <documentation>
      check availability of a flight, hotel and
      rental car concurrently
   </documentation>
   <invoke name="checkFlight" ...>
      <sources>
         <source linkName="checkFlight-To-BookFlight" />
      </sources>
   </invoke>
   <invoke name="checkHotel" ... />
   <invoke name="checkRentalCar" ... />
   <invoke name="bookFlight" ...>
      <targets>
         <target linkName="checkFlight-To-BookFlight" />
      </targets>
   </invoke>
</flow>
```

Nebenläufigkeit: `flow`–Aktivität mit Bedingung 1

```
<flow ...>
   <links>
      <link name="request-to-approve" />
      <link name="request-to-decline" />
   </links>
   <receive name="ReceiveCreditRequest"
      createInstance="yes"
      partnerLink="creditRequestPLT"
      operation="creditRequest"
      variable="creditVariable">
      <sources>
         <source linkName="request-to-approve">
            <transitionCondition>
               $creditVariable/value &lt; 5000
            </transitionCondition>
         </source>
         <source linkName="request-to-decline">
            <transitionCondition>
               $creditVariable/value >= 5000
            </transitionCondition>
         </source>
      </sources>
   </receive>
   <invoke name="approveCredit" ...>
      <targets>
         <target linkName="request-to-approve" />
      </targets>
   </invoke>
   ...
   </flow>
```

- `transitionCondition` muss erfüllt sein, damit Link wirkt.

Nebenläufigkeit: `flow`–Aktivität mit Bedingung 1

```
<flow ...>
     ...
    <invoke name="approveCredit" ...>
      <targets>
         <target linkName="request-to-approve" />
      </targets>
   </invoke>
   <invoke name="declineCredit" ...>
      <targets>
         <target linkName="request-to-decline" />
      </targets>
   </invoke>
</flow>
```

- Entweder `approveCredit` oder `declineCredit` wird aufgerufen.

Nebenläufigkeit: `flow`–Aktivität mit Bedingung 2

```
<flow ...>
   <links>
      <link name="request-to-approve" />
      <link name="request-to-decline" />
      <link name="approve-to-notify" />
      <link name="decline-to-notify" />
   </links>
   <receive name="ReceiveCreditRequest"
      createInstance="yes"
      partnerLink="creditRequestPLT"
      operation="creditRequest"
      variable="creditVariable">
      <sources>
         <source linkName="request-to-approve">
            <transitionCondition>
               $creditVariable/value &lt; 5000
            </transitionCondition>
         </source>
         <source linkName="request-to-decline">
            <transitionCondition>
               $creditVariable/value >= 5000
            </transitionCondition>
         </source>
      </sources>
   </receive>
   <invoke name="approveCredit" ...>
      <source linkName="approve-to-notify" />
      <targets>
         <target linkName="request-to-approve" />
      </targets>
   </invoke>
  ...
```

Nebenläufigkeit: `flow`–Aktivität mit Bedingung 2

```
  ...
   <invoke name="declineCredit" ...>
      <source linkName="decline-to-notify" />
      <targets>
         <target linkName="request-to-decline" />
      </targets>
   </invoke>

   <reply name="notifyApplicant" ...>
      <targets>
         <joinCondition>
            $approve-to-notify or $decline-to-notify
         </joinCondition>
         <target linkName="approve-to-notify" />
         <target linkName="decline-to-notify" />
      </targets>
      </invoke>
      </reply>
</flow>
```

- `joinCondition` muss erfüllt sein, damit ein Prozess startet

Datenmanipulationen: `assign`–Aktivität

```
<assign>
 <copy>
  <from variable="TimesheetSubmissionFailedMsg" />
  <to variable="EmployeeNotificationMsg" />
 </copy>
</assign>
```

`assign`–Aktivität

- enthält mehre `copy`–Aktivitäten
- verschiedene Arten von `copy`–Aktivitäten:
  - Kopieren zwischen Variablen
  - Kopieren von Teilen (mittels XPath selektiert)
  - Kopieren von Partnerlinks
  - Kopieren von Properties
  - Kopieren von konstanten XPath-Ausdrücken (ohne Kontextknoten)

Datenmanipulationen: `assign`–Aktivität mit Selektion

```
<assign>
 <copy>
  <from variable="Input" part="operation1Parameter">
    <query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
       creditCardInformation
    </query>
  </from>
  <to variable="CreditCardServiceInput" />
 </copy>
</assign>
```

- default `QueryLanguage` ist XPath 1.0

Ausnahmebehandlung: `faultHandlers`

```
<faultHandlers>
   <catch faultName="BookOutOfStockException"
      faultVariable="BookOutOfStockVariable">
      ...
   </catch>
   <catchAll>...</catchAll>
</faultHandlers>
```

`faultHandlers` können

- einem Prozess,
- einem `scope` oder
- einem `invoke`–Aufruf

zugeordnet sein.

Ausnahmebehandlung mit eingeschränkter Sichtbarkeit

```
<scope>
 <faultHandlers>
  <catch faultName="xyz:anExpectedError">...</catch>
  <catchAll><!-- deal with other errors -->
          ...
  </catchAll>
 </faultHandlers>
 <sequence>
   <!-- do work -->
 </sequence>
</scope>
```

Ausnahmebehandlung in `scope`

- ist nur im Gültigkeitsbereich von `scope` definiert.

Ausnahmebehandlung: Aktivität beim Beenden

```
<scope>
 <terminationHandler>
  <!-- clean up resources in case
       of forced termination -->
 </terminationHandler>
 <sequence>
   <!-- do work -->
 </sequence>
</scope>
```

Termination-Handler

- wird abgearbeitet, wenn alle Unteraktivitäten beendet sind (oder beendet werden).

Kompensation

```
<scope name="S1">
   <faultHandlers>
      <catchAll>
         <compensateScope target="S2" />
      </catchAll>
   </faultHandlers>
   <sequence>
      <scope name="S2">
         <compensationHandler>
            <!-- undo work -->
         </compensationHandler>
         <!-- do some work -->
      </scope>
      <!-- do more work -->
      <!-- a fault is thrown here;
           results of S2 must be undone -->
   </sequence>
</scope>
```

- `compensate` führt die Handler der Unter-`scopes` aus
- `compensateScope` führt den Hander eines `scope`s aus.

Ereignisbehandlung: `pick`–Aktivität

```
<pick>
 <onMessage partnerLink="buyer"
    operation="inputLineItem"
    variable="lineItem">
 <!-- activity to add line item to order -->
 </onMessage>
 <onMessage partnerLink="buyer"
    operation="orderComplete"
    variable="completionDetail">
    <!-- activity to perform order completion -->
 </onMessage>
 <onAlarm>
   <for>'P3DT10H'</for>
   <!-- handle timeout for order completion -->
 </onAlarm>
</pick>
```

- `pick` trifft eine Auswahl aus vielen Ereignissen
- `onMessage` erwartet das Eintreffen einer Nachricht
- `onAlarm` wartet Zeit ab

Ereignisbehandlung: Mehrere Ereignisse bearbeiten

```
<flow>
   <links>
      <link name="buyToSettle" />
      <link name="sellToSettle" />
   </links>
   <receive name="receiveBuyerInformation" createInstance="yes" ...>
      <sources>
         <source linkName="buyToSettle" />
      </sources>
      <correlations>
         <correlation set="tradeID" initiate="join" />
      </correlations>
   </receive>
   <receive name="receiveSellerInformation" createInstance="yes" ...>
      <sources>
         <source linkName="sellToSettle" />
      </sources>
      <correlations>
         <correlation set="tradeID" initiate="join" />
      </correlations>
   </receive>
   <invoke name="settleTrade" ...>
      <targets>
         <joinCondition>$buyToSettle and $sellToSettle</joinCondition>
         <target linkName="buyToSettle" />
         <target linkName="sellToSettle" />
      </targets>
   </invoke>
   ...
</flow>
```

- Mehrere Anfragen in beliebiger Reihenfolge werden erwartet.

Ereignisbehandlung: Asynchrone Ereignisse

```
<process name="purchaseOrderProcess" ...>
   ...
   <eventHandlers>
      <onEvent partnerLink="purchasing"
         operation="queryOrderStatus" ...>
         <scope>...</scope>
      </onEvent>
      <onEvent partnerLink="purchasing"
         operation="cancelOrder" ...>
         <scope>...</scope>
      </onEvent>
   </eventHandlers>
   ...
</process>
```

- `onEvent` behandelt Ereignisse ohne Synchronisation

Verzögerung eines Prozesses

```
<wait>
   <!-- wait for three days
          and ten hours to go by ... -->
   <for>'P3DT10H'</for>
</wait>
```

- `wait` verzögert den aktuellen Prozess um die angegebene Zeit.
- XPath–Zeitspannen und –Zeitpunkte

Weitere WS-BPEL–Konzepte

- Korrelation von Nachrichten
- paralleles `forEach`
- `empty`, `exit`
- XML–Validierung (`validate`)
- Nebenläufiges Ändern der Daten (Isolation)
- Dynamische Ermittlung der Partner
- Erweiterbarkeit durch Angeben der Ausdrucks– und Abfragesprache

Zusammenfassung

WS-BPEL 2.0

Beziehungen zwischen Prozessen

Verhalten von Prozessen

- Kommunikation
- Kontrollstrukturen
- Parallelverarbeitung
- Datenmanipulation
- Ausnahmebehandlung
- Ereignisbehandlung

Sonstiges

- Fragen?
- nächste Woche:

  Organisatorische Aspekte und SOA im Unternehmenskontext