5 Business Process Execution Language (BPEL)

In diesem Kapitel werden die Details der Sprache BPEL erläutert. Es kann und soll als Nachschlagewerk verstanden werden. In Kapitel 6 wird das hier erworbene Wissen zur Umsetzung unseres Fallbeispiels angewendet.

Die Business Process Execution Language für Webservices, kurz BPEL oder auch WS-BPEL, ist eine domänenspezifische, imperative Programmiersprache zur Definition von ausführbaren Geschäftsprozessen. Anders als bei der konventionellen Programmierung sind Geschäftsprozesse typischerweise lang laufende, nebenläufige Programme, deren Kontroll- und Datenfluss deshalb etwas anders ausspezifiziert werden und die von einer Workflow-Maschine interpretiert werden. Der Fokus liegt dabei auf dem sogenannten Two-Level-Programming: Hierbei werden zunächst Services in einer konventionellen Programmiersprache, wie beispielsweise Java, implementiert und über eine Schnittstelle anderen Anwendern zugänglich gemacht. Diese Services können nun auf einer höheren Abstraktionsebene zu komplexen Unternehmensanwendungen »orchestriert« werden. BPEL wird deshalb auch in die Gruppe der Sprachen zur Serviceorchestrierung eingeordnet. Daraus leitet sich ab, dass BPEL vorwiegend für sogenannte Backend-Prozesse verwendet wird, also automatisierbare Geschäftsprozesse, die möglichst ohne manuelle Intervention ausgeführt werden können (Dunkelverarbeitung). Mit den später beschriebenen Erweiterungen BPEL4People und WS-HumanTask lassen sich manuelle Schritte in die Prozessautomatik einfügen, wodurch die sogenannte Hellverarbeitung möglich wird. Beide Ansätze lassen sich auch mischen. Für reine Frontend-Prozesse, also z. B. die Maskensteuerung von mehrstufigen Formularen (»Page-Flows«), ist BPEL nur bedingt geeignet.

Beim Entwurf der Sprache hat sich das Entwicklungs- und Standardisierungskonsortium auf Webservices als Basistechnologie zur Beschreibung und Einbindung der Services festgelegt. Diese Technologie bietet mit SOAP und WSDL eine standardisierte und vor allem sehr flexible Plattform, sodass sich die BPEL-Spezifikation selbst nicht mit technischen Details, wie der Bindung von Services, Encoding von Parametern etc., beschäftigen muss. Dennoch muss auch diese Plattform konfiguriert werden. Obwohl man BPEL als eine Programmiersprache begreifen kann, wird diese nicht »programmiert«, sondern »modelliert«. Dazu verwendet man üblicherweise keinen Texteditor (obwohl man das natürlich auch kann), sondern

ein Modellierungswerkzeug, das auch die Konfiguration der Webservice-Details unterstützt. Diese übernehmen also die Erstellung und Aktualisierung der einzelnen Dateien (BPEL, WSDL, XSD), sodass sich der Entwickler auf die Geschäftslogik konzentrieren kann.

Während das typische BPEL-Anwendungsszenario die Ausführung von Geschäftsprozessen ist, kann es manchmal auch sinnvoll sein, nicht für die Ausführung bestimmte Prozessmodelle zu erstellen. Dazu werden einige Informationen, die für die Ausführung wichtig wären, einfach weggelassen. Solche »abstrakten« Prozesse können für verschiedene Anwendungen nützlich sein:

- Für Dokumentationszwecke sollen nur bestimmte Aspekte, beispielsweise die Interaktion mit genau einem Partner, sichtbar gemacht werden. Es wird also von einem ausführbaren Prozess abstrahiert.
- Ein nicht ausführbares Prozessmodell wird als Startpunkt für ein ausführbares Modell verwendet und wird dahingehend verfeinert. Dieses Vorgehen hilft beispielsweise, wenn man von einem fachlichen Modell zu einem technischen Modell kommen möchte. Zusammen mit dem Fachexperten wird das abstrakte Prozessmodell erstellt und später um die nötigen technischen Details ergänzt.
- Man möchte einem Geschäftspartner eine Art Bedienungsanleitung für einen Prozess übermitteln, damit er als Partner korrekt mit dem Prozess interagieren kann. Nicht erwünscht ist jedoch, dass der Geschäftspartner die (möglicherweise geheimen) Details über die Implementierung des Prozesses erfährt. In diesem Fall kann man eine Protokollbeschreibung in Form eines abstrakten Prozessmodells zur Verfügung stellen.

Die BPEL-Spezifikation unterscheidet explizit zwischen ausführbaren und abstrakten Prozessen und definiert, welche Details in abstrakten Modellen weggelassen werden dürfen. Die Bedeutung der einzelnen Aktivitäten und deren Zusammenspiel bleibt jedoch dieselbe. In diesem Buch beschäftigen wir uns jedoch nur mit ausführbaren BPEL-Prozessen.

5.1 BPEL-Infrastrukturen

Eine typische BPEL-Infrastruktur besteht aus mehreren Komponenten. Abbildung 5-1 zeigt eine herstellerunabhängige Referenzarchitektur einer solchen Infrastruktur. Auf der linken Seite befinden sich die Werkzeuge, mit denen Entwickler, aber auch Endanwender mit der Plattform interagieren können. Am Anfang steht natürlich die Modellierung der Geschäftsprozesse mithilfe eines Modellierungswerkzeugs. Dieses Werkzeug unterstützt den Modellierer, indem es die XML- und Webservice-Details so gut es geht vor dem Modellierer verbirgt. Das Ergebnis ist in der Regel ein Satz von BPEL-, WSDL- und XSD-Dateien. Die BPEL-Dateien enthalten die Prozessmodelle, die WSDL-Dateien sowohl die eigenen Schnittstellen

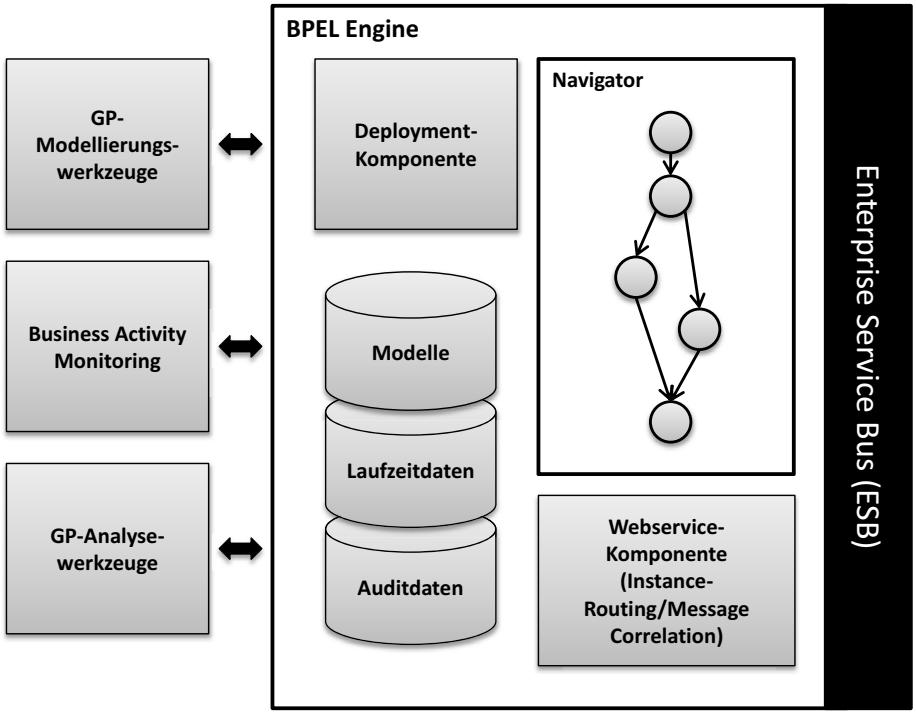

Abb. 5-1: Referenzarchitektur einer BPEL-Infrastruktur

als auch die der Partnerservices. In den XSD-Dateien werden gemeinsame Datentypen beschrieben. Dieses Paket wird nun an die BPEL-Engine, die die Prozesse dann ausführen soll, übermittelt. Diesen Schritt nennt man auch *Deployment*. Dabei müssen noch einige hersteller- und umgebungsabhängige Details festgelegt werden, beispielsweise wie und wo die BPEL-Engine die Partnerservices findet und wie detailliert die Auditlogs, also die Ausführungsprotokolle, geschrieben werden sollen. Die BPEL-Engine nimmt das Deployment-Paket entgegen und überprüft, ob alle benötigten Artefakte vorliegen und gültig sind. Dann wird es in der Modelldatenbank abgelegt. Diese Datenbank kann typischerweise auch mehrere Versionen eines Prozessmodells speichern. Das ist insbesondere bei lang laufenden Prozessen wichtig, denn wenn eine neue Version »deployed« wird, sollen zwar neue Instanzen dem neuen Prozessmodell folgen, die bereits laufenden jedoch noch nach dem alten Schema abgewickelt werden. Nach dem Deployment wartet die BPEL-Engine auf eingehende Nachrichten, die eine Instanz des Prozessmodells starten können. Trifft eine solche Nachricht ein, beginnt der Navigator die Aktivitäten des Prozessmodells entlang des Kontrollflusses abzuarbeiten. Dabei wird der aktuelle Zustand in der Laufzeitdatenbank abgelegt. Zusätzlich wird jeder Navigationsschritt als Ereignis in die Auditdatenbank geschrieben. Solche Ereignisse können z. B. die Erstellung einer Prozessinstanz, das Lesen oder Schreiben einer

Variablen oder das Betreten und Verlassen einer Aktivität dokumentieren. Über die Webservice-Komponente interagiert die Engine mit der Außenwelt, in unserer Architektur exemplarisch durch einen Enterprise Service Bus (ESB) dargestellt, der die Umsetzung der Serviceaufrufe durch verschiedene Transportprotokolle übernimmt. Nachrichten an ein Prozessmodell oder eine Prozessinstanz werden von ihm an die Engine weitergeleitet.

Die Daten in der Laufzeitdatenbank repräsentieren immer den aktuellen Zustand einer Prozessinstanz. Monitoring-Werkzeuge können darauf zugreifen und so Echtzeitberichte erstellen, während die Analysewerkzeuge aus den Auditdaten Rückschlüsse auf die Ausführung und die beteiligten Services ziehen können und so wichtiges Feedback für die kontinuierliche Verbesserung der Modelle geben können. Ebenso kann dadurch die Einhaltung von Verträgen oder Regeln überprüft werden (Compliance).

5.2 Hello World in BPEL

Eine BPEL-Prozessdefiniton kann schnell unübersichtlich werden. Wir werden uns deshalb zunächst Schritt für Schritt mithilfe von zwei einfachen Hello-World-Beispielen in die Sprache einarbeiten. Wir starten dabei mit der Implementierung eines einfachen Webservice, der keine anderen Services aufruft, sondern lediglich Daten umkopiert. Im zweiten Schritt werden wir einen Service aufrufen.

Die prinzipielle Funktionsweise des ersten er beiden BPEL-Prozesse, die wir in diesem Abschnitt entwickeln werden, ist in Abbildung 5-2 dargestellt.

Da BPEL Schnittstelle und Implementierung separiert und für die Schnittstellenbeschreibung WSDL verwendet, müssen wir bereits für ein Minimalbeispiel

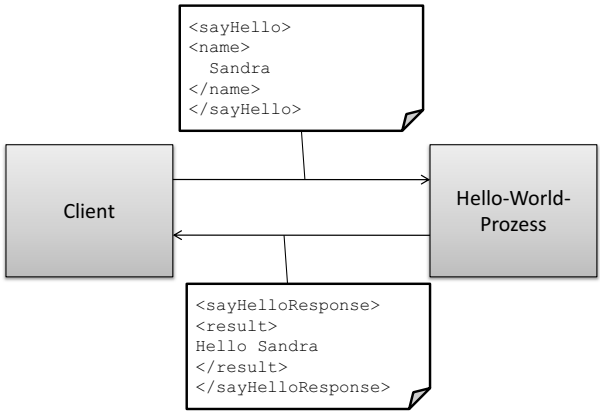

Abb. 5-2: Funktionsweise des ersten Hello-World-Prozesses

zwei Dateien (eine WSDL- und eine BPEL-Datei) erstellen, die wir im Folgenden Schritt für Schritt besprechen werden.

Eine Prozessdefinition setzt sich aus einem BPEL-Dokument sowie einem oder mehreren WSDL-Dokumenten zusammen. Die WSDL-Dokumente definieren sowohl die Schnittstellen der zu orchestrierenden Services als auch die Schnittstelle, unter der das Prozessmodell selbst erreichbar sein wird. Für die Verwendung in BPEL sind zunächst nur die abstrakten Teile der WSDL-Dokumente relevant, also die Definition von Nachrichtentypen und deren Gruppierung zu Operationen. Wie diese Operationen auf technische Transportprotokolle abgebildet werden, ist Teil der konkreten Definition; diese wird jedoch erst zum Zeitpunkt des Deployments von der ausführenden Middleware verarbeitet. Für den BPEL-Entwickler ist es daher zunächst unerheblich, ob ein Service z. B. über HTTP oder JMS zur Verfügung gestellt wird.

Abbildung 5-3 und Listing 5.1 zeigen die abstrakte Schnittstelle unseres Prozesses. Sie besteht aus zwei Teilen. Zunächst werden mit den üblichen Mitteln der WSDL eine Operation mit zwei Nachrichten definiert. Die Request-Nachricht hat ein Element mit dem Namen `<name>` und die Response-Nachricht ein Element mit dem Namen `<result>`.

Die Operation »sayHello« ist als eine Request-Response-Operation definiert, nimmt eine »sayHelloMessage« als Eingabeparameter und definiert eine »sayHelloReponseMessage« als Rückgabewert. Damit ist die Serviceschnittstelle zwar als solche ausreichend definiert, allerdings kann BPEL solche Schnittstellen (Porttypen) nicht direkt ansprechen. Stattdessen müssen die Beziehungen zwischen dem Prozess und seinen Partnerservices mithilfe von *Partnerlinks* explizit definiert werden. Sie erlauben es, später kompliziertere Muster wie Rückrufe an den Prozess etc. umzusetzen. Das bedeutet allerdings, dass auch einfache Interaktionsmuster nicht ohne diesen Mechanismus auskommen und deshalb auch unser Trivialbeispiel Partnerlinks definieren muss. Diese Tatsache macht das einfache Beispiel komplexer, als man es vielleicht erwarten würde.

Zur Definition eines Partnerlinks muss man zunächst einen sogenannten *Partnerlink-Typ* definieren. Dieser definiert mindestens eine und maximal zwei Rollen in Form von Porttypen. Ein Partnerlink ist schließlich eine konkrete Ausprägung eines Partnerlink-Typs und legt fest, welche Rolle vom Prozess selbst und welche vom Kommunikationspartner des Prozesses eingenommen wird, d. h., welcher

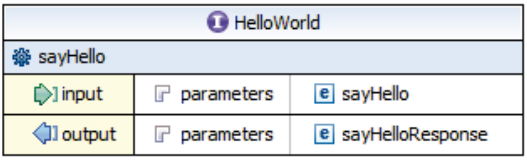

Abb. 5-3: Servicedefinition des Greeting-Service

Porttyp vom Prozess implementiert werden muss und welcher vom Kommunikationspartner des Prozesses. Der Partnerlink-Typ wird dabei, wie in Listing 5.1 gezeigt, in einer WSDL-Beschreibung definiert. Der Partnerlink selbst wird dann im BPEL-Prozess angelegt.

```
<wsdl:definitions name="HelloWorld"
  targetNamespace="http://www.bpelbuch.de/helloworld"
  xmlns:tns="http://www.bpelbuch.de/helloworld"
  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

  <wsdl:types>
    <xs:schema attributeFormDefault="unqualified"
        elementFormDefault="qualified"
        targetNamespace="http://www.bpelbuch.de/helloworld"
        xmlns:xs="http://www.w3.org/2001/XMLSchema">
      <xs:element name="sayHello">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="name" type="string" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="sayHelloResponse">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="result" type="string" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:schema>
  </wsdl:types>

  <wsdl:message name="sayHelloMessage">
    <wsdl:part name="parameters" element="tns:sayHello" />
  </wsdl:message>
  <wsdl:message name="sayHelloResponseMessage">
    <wsdl:part name="parameters"
         element="tns:sayHelloResponse" />
  </wsdl:message>

  <wsdl:portType name="HelloWorld">
    <wsdl:operation name="sayHello">
      <wsdl:input message="tns:sayHelloMessage" />
      <wsdl:output message="tns:sayHelloResponseMessage" />
```

```
42      </wsdl:operation>
43    </wsdl:portType>
44
45    <plnk:partnerLinkType name="HelloWorld">
46      <plnk:role name="HelloWorldProvider"
47              portType="tns:HelloWorld" />
48    </plnk:partnerLinkType>
49  </wsdl:definitions>
```

Listing 5.1: Hello World, WSDL-Schnittstellendefinition

Nachdem wir die Schnittstelle für unseren Hello-World-Prozess definiert haben, kommen wir zu dem eigentlichen Prozessmodell. In Abbildung 5-4 sehen wir einen Screenshot aus dem Eclipse BPEL-Designer. Da die grafische Notation von BPEL nicht standardisiert ist, fallen die Grafiken von Werkzeug zu Werkzeug leicht verschieden aus. Listing 5.2 zeigt das darunterliegende, dem standardisierten Schema entsprechende XML.

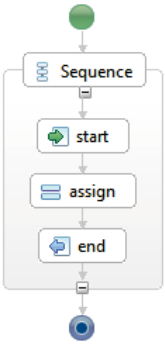

Abb. 5-4: Hello World, Screenshot

Das Wurzelelement eines jeden WS-BPEL-Prozesses ist das `<process>`-Element im BPEL-2.0-Namensraum[1]. Eindeutig identifiziert wird ein BPEL-Prozessmodell über das Attributpaar `name` und `targetNamespace`. Darüber hinaus werden noch die Namensräume für die WSDL-Definitionen, für WSDL und XML Schema definiert sowie die Standardabfragesprache auf XPath 2.0 gesetzt. Über das `<import>`-Element können externe Artefakte, typischerweise WSDL- und XSD-Dateien, importiert werden. In unserem Beispiel wird so die Hello-World-Schnittstelle dem Prozessmodell bekannt gemacht. Als Nächstes werden die Partnerlinks definiert. Da der Hello-World-Prozess nur eine Schnittstelle anbietet und keine konsumiert, reicht hier ein einfacher Partnerlink aus, der eine Rolle, nämlich die des Prozessmodells, an die Schnittstelle bindet.

[1] *http://docs.oasis-open.org/wsbpel/2.0/process/executable*

```
<bpel:process
  name="HelloWorld"
  targetNamespace="http://www.bpelbuch.de/helloworld"
  suppressJoinFailure="yes"
  xmlns:tns="http://www.bpelbuch.de/helloworld"
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/
      executable">

  <bpel:import location="HelloWorld.wsdl"
      namespace="http://www.bpelbuch.de/helloworld"
      importType="http://schemas.xmlsoap.org/wsdl/" />

  <bpel:partnerLinks>
    <bpel:partnerLink name="client"
        partnerLinkType="tns:HelloWorld"
        myRole="HelloWorldProvider" />
  </bpel:partnerLinks>

  <bpel:variables>
    <bpel:variable name="input"
        messageType="tns:sayHelloMessage" />
    <bpel:variable name="output"
        messageType="tns:sayHelloResponseMessage" />
  </bpel:variables>

  <bpel:sequence name="main">
    <bpel:receive name="Start"
        partnerLink="client" portType="tns:HelloWorld"
        operation="sayHello" variable="input"
        createInstance="yes" />

    <bpel:assign validate="no" name="Assign">
      <!-- Variable initialisieren -->
      <bpel:copy>
        <bpel:from>
          <bpel:literal>
            <tns:sayHelloResponse
                xmlns:tns="http://www.bpelbuch.de/helloworld">
              <tns:result />
            </tns:sayHelloResponse>
          </bpel:literal>
        </bpel:from>
        <bpel:to part="parameters" variable="output" />
      </bpel:copy>
      <!-- Strings zusammenführen -->
      <bpel:copy>
```

```
46          <bpel:from>
47            concat('Hello ', $input.parameters/tns:name)
48          </bpel:from>
49          <bpel:to>$output.parameters/tns:result</to>
50        </bpel:copy>
51      </bpel:assign>
52
53      <bpel:reply name="Ende" partnerLink="client"
54          portType="tns:HelloWorld"
55          operation="sayHello" variable="output" />
56    </bpel:sequence>
57  </bpel:process>
```

Listing 5.2: Hello World, WS-BPEL-Datei

In der Klassifikation der Prozessmodellierungssprachen wird BPEL in die Gruppe der Sprachen mit »explizitem Kontrollfluss« und »implizitem Datenfluss« eingeordnet. »Impliziter Datenfluss« bedeutet, dass dieser nicht grafisch über Kanten modelliert wird, sondern Aktivitäten in der Lage sind, über (globale) Variablen Daten zu lesen und zu schreiben und so untereinander Daten austauschen können. »Expliziter Kontrollfluss« bedeutet, dass die Navigation zwischen den Aktivitäten durch Kanten explizit vorgeschrieben ist und nicht etwa über Datenabhängigkeiten geregelt wird. Im `<variables>`-Element können die benötigten globalen Variablen deklariert werden. In unserem Beispiel definieren wir je eine Variable für die Input- und die Output-Message, die wir für die Eingabe und die Ausgabe des Prozessmodells verwenden.

Damit verlassen wir den Präambelbereich der Prozessdefinition und kommen nun zur Geschäftslogik. Die Logik eines Hello-World-Prozesses ist denkbar einfach. Wir möchten eine Nachricht empfangen, die irgendeinen Namen enthält. Vor den Namen hängen wir ein »Hello« an und senden eine entsprechende Nachricht zurück.

BPEL stellt verschiedene Konstrukte zur Modellierung des Kontrollflusses bereit. Diese werden im weiteren Verlauf des Kapitels ausführlich erklärt. Das einfachste Konstrukt ist die `<sequence>`. Alle in ihr enthaltenen Aktivitäten werden sequenziell abgearbeitet. In unserem Beispiel handelt es sich bei diesen Aktivitäten um ein `<receive>`, um die eingehende Nachricht zu empfangen, dann ein `<assign>`, das die Datenmanipulation vornimmt, gefolgt von einem `<reply>`, das die Antwortnachricht an den Aufrufer zurücksendet. Die Attribute `partnerLink` und `operation` der `<receive>`- und `<reply>`-Aktivitäten stellen hierbei die Verbindung zu der in WSDL beschriebenen Serviceschnittstelle her, in diesem Fall auf die Operation *hello* der Rolle »myRole« des spezifizierten Partnerlinks. Das `createInstance`-Attribut der `<receive>`-Aktivität definiert, dass diese Aktivität beim Empfang einer Nachricht eine neue Instanz des Prozes-

ses erstellt. Diese Instanz ist beendet, wenn die <reply>-Aktivität erfolgreich ausgeführt wurde und dadurch auch die <sequence> beendet wird.

Damit ist der Kontrollfluss definiert und es muss nur noch die Rückgabenachricht aufgebaut werden. Datenmanipulation wird in BPEL mithilfe der <assign>-Aktivität (siehe auch Abschnitt 5.5) realisiert. Ein <assign> kann dazu mehrere <copy>-Anweisungen enthalten, die eine Zuweisung von <from> zu <to> unter Verwendung einer definierten Expression Language (wie z. B. XPath) beschreiben.

Bevor Daten in eine Nachricht kopiert werden können, muss diese jedoch initialisiert werden. Dazu dient der erste <copy>-Block. Im zweiten Schritt wird das Wort »Hello« vor dem übergebenen Parameter angehängt. Dazu wird die XPath-Funktion concat benutzt. Der neue Wert wird wieder in die Nachricht geschrieben, was in dem <to>-Element beschrieben ist.

Hiermit ist der erste Prozess erstellt, der lediglich eine Nachricht empfängt und eine zurücksendet, aber noch keinen anderen Service aufruft. Wie dies funktioniert, wollen wir in einem zweiten einfachen Beispiel noch demonstrieren. Dazu wollen wir nicht einfach nur »Hello« an einen Prozess anfügen, sondern werden eine zufällig ausgewählte internationale Begrüßungsformel, wie »Hi«, »Hallo«, »Hello«, »Hola«, etc., von einem Webservice beziehen und dann für die Antwortnachricht verwenden. Die Aufrufreihenfolge ist in Abbildung 5-5 dargestellt.

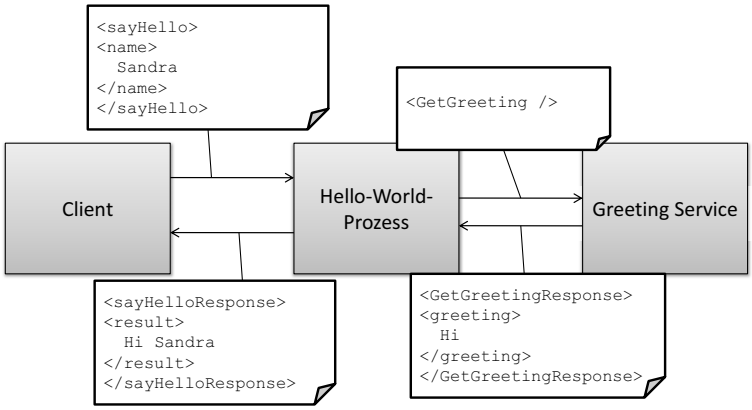

Abb. 5-5: Funktionsweise des zweiten Hello-World-Prozesses

Dazu definieren wir zunächst die Schnittstelle für den Service, der die Begrüßungsformel zurückliefern soll. Die Input-Nachricht ist dabei leer und und in der Antwort-Nachricht wird ein String mit dem Grußwort zurückgeliefert. Die WSDL ist in Listing 5.3 gezeigt.

```
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://www.bpelbuch.de/services/greeting"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    name="greeting"
    targetNamespace="http://www.bpelbuch.de/services/greeting">

  <wsdl:types>
    <xs:schema
      targetNamespace="http://www.bpelbuch.de/services/greeting">

      <xs:element name="GetGreeting">
        <xs:complexType>
          <xs:sequence />
        </xs:complexType>
      </xs:element>
      <xs:element name="GetGreetingResponse">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="greeting" type="xs:string" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:schema>
  </wsdl:types>

  <wsdl:message name="GetGreetingRequest">
    <wsdl:part element="tns:GetGreeting" name="parameters" />
  </wsdl:message>
  <wsdl:message name="GetGreetingResponse">
    <wsdl:part element="tns:GetGreetingResponse"
        name="parameters" />
  </wsdl:message>

  <wsdl:portType name="GreetingPortType">
    <wsdl:operation name="GetGreeting">
      <wsdl:input message="tns:GetGreetingRequest" />
      <wsdl:output message="tns:GetGreetingResponse" />
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

Listing 5.3: Greeting Service, WSDL-Schnittstellendefinition

Für den neuen Service muss wieder ein neuer Partnerlink-Typ und ein darauf basierender Partnerlink definiert werden. Hierbei sind aber nun die Rollen andersherum definiert als bei dem ersten Partnerlink, denn der Prozess ruft nun den

neu angelegten Service auf und wird nicht aufgerufen. Die beiden nun benötigten Partnerlink-Typen werden in die WSDL des Prozesses eingefügt. Die neue WSDL ist in Listing 5.4 dargestellt.

```
<wsdl:definitions name="HelloWorld"
  targetNamespace="http://www.bpelbuch.de/helloworld"
  xmlns:tns="http://www.bpelbuch.de/helloworld"
  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:greet="http://www.bpelbuch.de/services/greeting">

  <!-- Messages, Porttype etc. definiert wie zuvor -->

  <plnk:partnerLinkType name="HelloWorld">
    <plnk:role name="HelloWorldProvider"
        portType="tns:HelloWorld" />
  </plnk:partnerLinkType>

  <plnk:partnerLinkType name="GreetingPLT">
    <plnk:role name="GreetingService"
        portType="greet:GreetingPortType" />
  </plnk:partnerLinkType>
</wsdl:definitions>
```

Listing 5.4: Hello World (erweiterte Version), WSDL-Schnittstellendefinition

Abb. 5-6: Hello World (erweiterte Version), Screenshot

Nun können wir den Prozess, wie in Abbildung 5-6 dargestellt, entsprechend ändern. Dazu fügen wir einen neuen Partnerlink »Greeting« mit dem zuvor erstellten Partnerlink-Typ ein. Diesen nutzen wir nun im Prozessverlauf, um das Grußwort abzurufen. Hinzugekommen sind eine neue `<assign>`- und eine neue `<invoke>`-Aktivität. Neu ist die `<invoke>`-Aktivität, die benutzt wird, um einen Webservice aufzurufen. Dazu benötigen wir zwei neue Nachrichtenvariablen für die Input- und die Output-Message. Wie beim `<receive>` und `<reply>` müssen der Partnerlink und die verwendete Operation angegeben werden. Im `<assign>` wird die leere Input-Message für den Serviceaufruf initialisiert.

Durch den Serviceaufruf ändert sich das bereits vorhandene `<assign>` zur Erstellung der Rückgabenachricht ein wenig. Hier wird nun nicht mehr fest »Hello« in die Nachricht kopiert, sondern das Grußwort aus dem Serviceaufruf. Der neue Prozess in XML-Serialisierung ist in Listing 5.5 dargestellt.

```
<bpel:process name="HelloWorld2"
  targetNamespace="http://www.bpelbuch.de/helloworld"
  suppressJoinFailure="yes"
  xmlns:tns="http://www.bpelbuch.de/helloworld"
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/
      executable"
  xmlns:greet="http://www.bpelbuch.de/helloworldArtifacts"
  xmlns:gns="http://www.bpelbuch.de/services/greeting">

  <bpel:import location="greeting.wsdl"
      namespace="http://www.bpelbuch.de/services/greeting"
      importType="http://schemas.xmlsoap.org/wsdl/" />
  <bpel:import location="HelloWorld.wsdl"
      namespace="http://www.bpelbuch.de/helloworld"
      importType="http://schemas.xmlsoap.org/wsdl/" />

  <bpel:partnerLinks>
    <bpel:partnerLink name="client"
        partnerLinkType="tns:HelloWorld"
        myRole="HelloWorldProvider" />
    <bpel:partnerLink name="Greeting"
        partnerLinkType="tns:GreetingPLT"
        partnerRole="GreetingService" />
  </bpel:partnerLinks>

  <bpel:variables>
    <bpel:variable name="input"
        messageType="tns:sayHelloMessage" />
    <bpel:variable name="output"
        messageType="tns:sayHelloResponseMessage" />
    <bpel:variable name="GreetingResponse"
        messageType="gns:GetGreetingResponse" />
```

```
32      <bpel:variable name="GreetingRequest"
33          messageType="gns:GetGreetingRequest" />
34    </bpel:variables>
35
36    <bpel:sequence name="main">
37      <bpel:receive name="Start"
38          partnerLink="client"
39          portType="tns:HelloWorld"
40          operation="sayHello" variable="input"
41          createInstance="yes" />
42
43      <bpel:assign validate="no" name="GreetingVorbereiten">
44        <bpel:copy>
45          <bpel:from>
46            <bpel:literal>
47              <gns:GetGreeting />
48            </bpel:literal>
49          </bpel:from>
50          <bpel:to variable="GreetingRequest" part="parameters" />
51        </bpel:copy>
52      </bpel:assign>
53
54      <bpel:invoke name="GetGreetingAufrufen"
55          partnerLink="Greeting"
56          operation="GetGreeting"
57          inputVariable="GreetingRequest"
58          outputVariable="GreetingResponse" />
59
60      <bpel:assign validate="no" name="AntwortVorbereiten">
61        <bpel:copy>
62          <bpel:from>
63            <bpel:literal>
64              <tns:sayHelloResponse>
65                <tns:result />
66              </tns:sayHelloResponse>
67            </bpel:literal>
68          </bpel:from>
69          <bpel:to part="parameters" variable="output" />
70        </bpel:copy>
71        <bpel:copy>
72          <bpel:from>
73            concat($GreetingResponse.parameters/greeting, ' ',
74                $input.parameters/tns:name)
75          </bpel:from>
76          <bpel:to>$output.parameters/tns:result</to>
77        </bpel:copy>
```

```
78      </bpel:assign>
79
80      <bpel:reply name="Ende"
81            partnerLink="client"
82            portType="tns:HelloWorld"
83            operation="sayHello"
84            variable="output" />
85    </bpel:sequence>
86  </bpel:process>
```

Listing 5.5: Hello World (erweiterte Version), WS-BPEL-Datei

Damit kennen wir nun die grundlegende Funktionsweise von BPEL. In den folgenden Teilen dieses Kapitels werden nun die Konzepte und die weiteren Möglichkeiten besprochen, die BPEL bietet.

5.3 Prozessstruktur

BPEL-Prozesse folgen immer dem gleichen Aufbau. Sie beginnen mit einem `<process>`-Element, darin werden einige globale Eigenschaften definiert, dann folgt die Wurzelaktivität, die sich in weitere Aktivitäten aufspaltet und durch die die Geschäftslogik implementiert wird.

Notation

Neben Beispielen haben wir im Folgenden auch Codefragmente, die den syntaktischen Aufbau der BPEL-Elemente darstellen. Wir halten uns dabei an die Notation, die auch in der BPEL-Spezifikation verwendet wird:

- Ein »?« hinter einem Element oder Attribut bedeutet, dass dieses optional ist.
- Ein »*« bedeutet, dass das Element oder Attribut optional ist, aber auch mehrmals vorkommen darf.
- Ein »+« steht für einmal oder öfter.
- »|« bezeichnet eine Alternative.
- Runde Klammern gruppieren zusammenhängende Teile, meistens im Zusammenhang mit »|«.

Listing 5.6 zeigt den syntaktischen Aufbau des Wurzelelements. Eindeutig identifiziert wird ein BPEL-Prozess durch das `name`- und das `targetNamespace`-Attribut. Der »Name« eines Prozessmodells ist also ein QName. Damit nicht für jeden Abfrageausdruck explizit eine *Query* bzw. *Expression Language* gesetzt werden muss, können sie mit den Attributen `expressionLanguage` sowie `queryLanguage` prozessmodellweit festgelegt werden. Sie können später explizit überschrieben werden. Die Standardeinstellung ist XPath 1.0.