Seminararbeit

Serviceorientierte Softwarearchitektur (SOA)

Thema:

Geschäftsprozessmodellierung mit BPEL4WS: Aufbau und Beispiel

Christoph Forster
Winf 2370

Erstellt von:	Christoph Forster Bornkampsweg 3 22761 Hamburg Tel. (0170) 4329749
Erarbeitet im:	5. Semester
Vortrag:	29.11.2006 – 14:00
Abgabe:	04.12.2006
Dozent (FH Wedel):	Prof. Dr. Sebastian Iwanowski Fachhochschule Wedel Feldstraße 143 22880 Wedel Tel. (04103) 8048- 63

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis verwendeter Codebeispiele

Abkürzungsverzeichnis

API	Application Programming Interface
B2B	business to business
CORBA	Common Object Request Broker Architecture
EAI	Enterprise Application Integration
EPK	Ereignis-Prozess-Kette
OASIS	Organization for the Advancement of Structured Information Standards
RPC	Remote Procedure Call
SOA	Service oriented architecture
SOAP	ursprünglich: Simple Object Access Protocol
UDDI	Universal Description, Discovery and Integration
WS-CDL	Web Services Choreography Description Language
WSDL	Web Service describtion language
WSFL	Web Services Flow Language
WS-*	sämtliche Spezifikationen der Web Service Architektur
XML	Extensible Markup Language

1. Einleitung und Übersicht

Unternehmensabläufe werden heutzutage immer komplexer und aufwendiger. Gleichzeitig werden immer kürzere Reaktions- und Anpassungszeiten gefordert. Moderne IT-Infrastrukturen sind unabdingbar geworden, um diesen Zielen gerecht zu werden.
Nicht immer gelingt es jedoch, betriebswirtschaftliche Anforderungen mit den Möglichkeiten und Angeboten der Informatik zur vollständigen Zufriedenheit in Einklang zu bringen sowie Ressourcen so zielgerichtet wie möglich einzusetzen.

Erstrebenswert ist eine informationstechnische Architektur, in der betriebswirtschaftliche Aspekte von den konkreten technischen Details abstrahiert werden können.
So können sich betriebswirtschaftlich orientierte Fachkräfte um die Aufgaben aus Sicht der Geschäftslogik kümmern und technisch orientierte Fachkräfte die konkreten Implementierungen übernehmen.

BPEL4WS stellt ein Werkzeug dafür zur Verfügung.
Es handelt sich dabei um eine XML-basierte Sprache zur formalen Spezifikation von Geschäftsprozessen und -interaktionsprotokollen. Sie stellt damit die Möglichkeit zur Verfügung, mehrere Dienste einer SOA, welche als Webservices implementiert sind, zu kombinieren. Man spricht von der „Orchestrierung von Webservices“, der Geschäftsprozess übernimmt die Rolle des „Dirigenten“ für die verschiedenen Services.
Somit ist nicht nur die automatische Prozessintegration möglich.
Auch ist es damit sehr einfach und schnell möglich, veränderte Geschäftsabläufe im System zu berücksichtigen oder auf Ressourcen ganz unterschiedlicher Herkunft zuzugreifen (z.B. B2B-Integration).

Diese Seminararbeit beschäftigt sich im Weiteren näher mit der Sprache BPEL4WS.

Dazu wird zunächst der Begriff des Geschäftsprozesses näher erläutert.

Eine Betrachtung von BPEL4WS im Gesamtkonzept folgt in den nächsten beiden Abschnitten.
Einerseits liegt der Betrachtungsschwerpunkt auf der Sprachmotivation und der Entwicklung, andererseits auf der Architektur der Webservices.
Nachdem diskutiert wurde, wozu BPEL4WS spezifiziert worden ist, folgt ein näherer Blick auf die Umsetzung der Strategien.
Nach einer Zusammenfassung von WSDL werden die in BPEL4WS verwendeten Sprachkonzepte und Sprachelemente im Detail betrachtet.
Nach der theoretischen Betrachtung erfolgt ein Überblick über BPEL4WS in der praktischen Anwendung. Aktuell verfügbare BPEL-Tools werden vorgestellt und ein Sprach-Beispiel zur Erläuterung herangezogen.
Nach kurzen Ausblicken über BPEL-Spracherweiterungen folgt ein abschließendes Fazit.

2. Der Geschäftsprozess

2.1. theoretische Begriffserläuterung

Der zentrale Gedanke hinter BPEL4WS, das „Leitmotiv", ist der Geschäftsprozess.
Ein Geschäftsprozess ist die Zusammensetzung aus „Prozess" und „Geschäft".
Ein Geschäft ist generell eine Tätigkeit, deren Ziel es ist, Produkte oder Dienstleistungen zu verkaufen, um Gewinn zu erwirtschaften.

Ein Prozess ist eine Vorgangs- bzw. Tätigkeitsabfolge.

Ein Geschäftsprozess ist demzufolge eine Abfolge von Vorgängen zur Erreichung eines Resultates im betriebswirtschaftlichen Sinne.

Der Geschäftsprozess kann dabei das gesamte Geschäft oder einen Teilschritt davon darstellen.
Ein Geschäftsprozess kann Unterprozesse enthalten oder auch andere Prozesse anstoßen.

2.2. Beispiel

Die Unternehmung habe den Zweck, Kredite zu vergeben. Sie erwirtschaftet ihren Gewinn über die Zinsen, die für die Kredite zu zahlen sind.
Neben anderen Nebenfunktionen wie Marketing und Verwaltung gehören Kreditvergabe und Kreditbetreuung zu den Unternehmensfunktionen / -prozessen der oberen Ebene, direkt unter dem Hauptprozess „Kreditinstitut betreiben“.

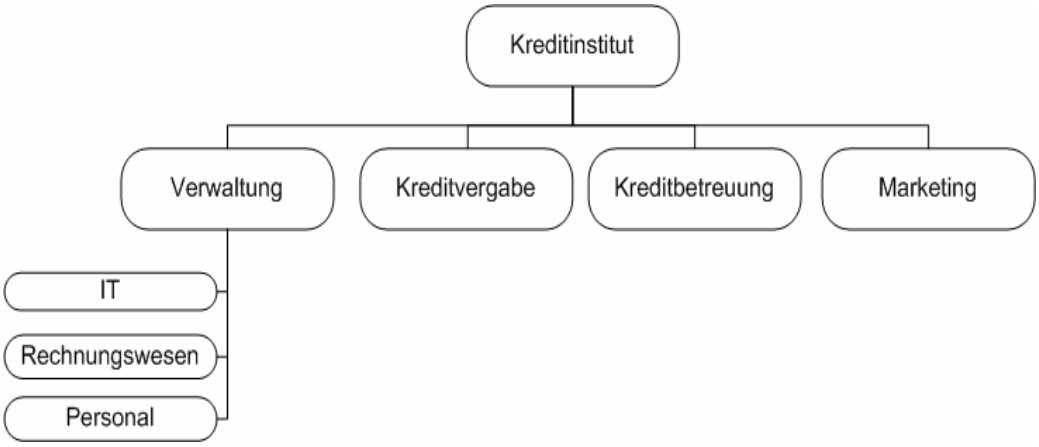

Abbildung 1: Ausschnitt eines beispielhaften Funktionsbaums

So wie sich die Verwaltungsfunktion in Unterfunktionen wie IT, Rechnungswesen und Personalmanagement aufspalten lässt, kann man auch z.B. die Kreditvergabe weiter granulieren.

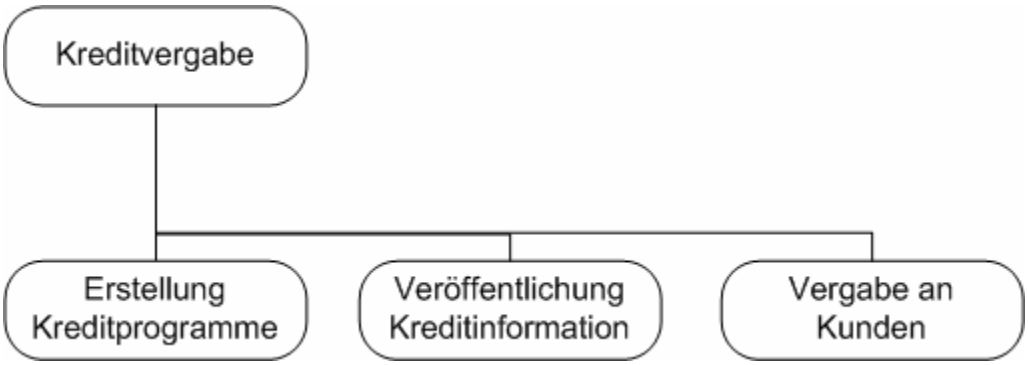

Abbildung 2: beispielhafte Wertschöpfungskette

Bevor grundsätzlich Kredite vergeben werden können, müssen entsprechende Ablaufprogramme entwickelt werden, welche z.B. Laufzeiten und Zinssätze

festlegen. Diese Kreditprogramme werden nun veröffentlicht. (dies könnte über eine Homepage oder die Weiterleitung an das Marketing geschehen. Es soll hier aber nicht näher betrachtet werden.)
Zum Schluss steht die konkrete Vergabe der Kredite an Kunden.
Diese soll nun detaillierter betrachtet werden.

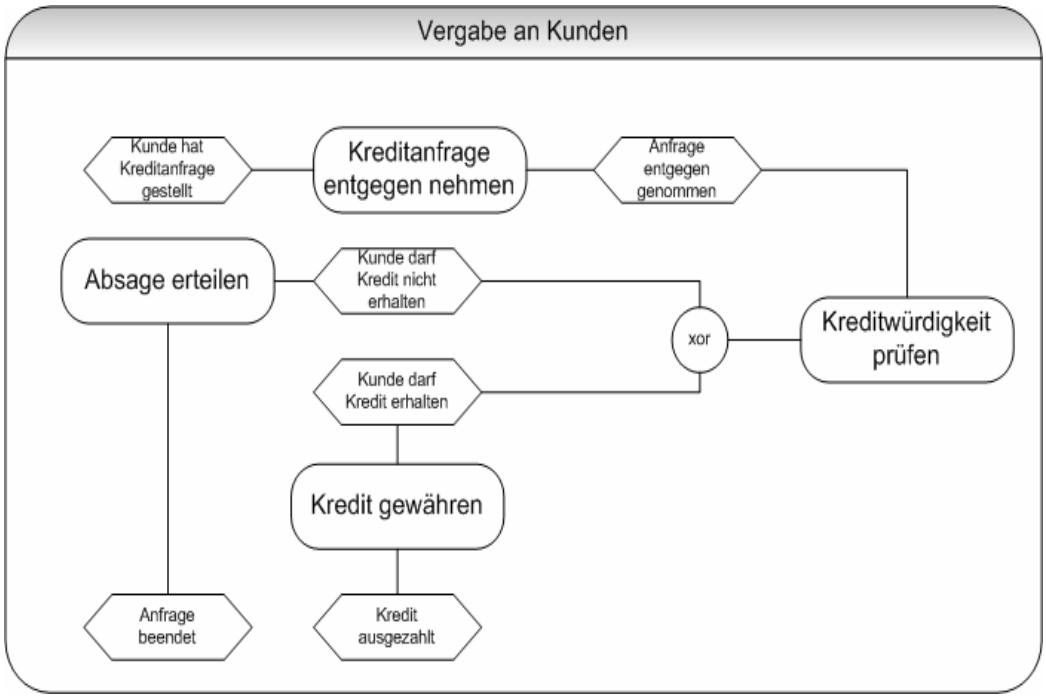

Abbildung 3: Beispiel-Ereignis-Prozess-Kette (EPK)

Immer abwechselnd stehen Ereignisse und Prozesse. Nachdem ein Kunde eine Kreditanfrage gestellt hat, wird diese zunächst vom Unternehmen entgegen genommen. Ist dies geschehen, so wird die Kreditwürdigkeit geprüft. Entweder das Ergebnis ist positiv oder es ist negativ. Eine Absage wird bei negativem Ergebnis erteilt. Ansonsten wird der Kredit gewährt.
Auch diese Geschäftsprozesse lassen sich zum Teil noch weiter granulieren.

Sie sind dabei Teil des Prozesses „Vergabe an Kunden“ und sind zu einander Vorgänger und Nachfolger.
Wichtig ist, dass man sich bei der Betrachtung einer bestimmten Abstraktionsebene nicht direkt um die anderen Ebenen zu kümmern braucht. Für die Logik des in der Ereignis-Prozess-Kette aufgezeigten Geschäftsprozess macht es keinen Unterschied, in welchem Kontext sie insgesamt steht.

Genauso irrelevant ist hier, wie die einzelnen Teilschritte konkret umgesetzt sind. Für diesen Prozess ist nur wichtig, dass sie umgesetzt werden.

Die meisten dieser im Beispiel vorgestellten Prozesse beinhalten konkrete Aktionen, welche real durchgeführt werden müssen. Kontakt muss z.B. konkret aufgenommen werden. Viele Prozesse finden jedoch auch eine Abbildung in der IT. Ein Informationssystem im Hintergrund ist also unabdingbar. Dieses sollte nach Möglichkeit ganzheitlich das komplette Unternehmen abdecken.

3. Konzept und geschichtliche Entwicklung von BPEL4WS

Den ursprünglichen Antrieb zur Entwicklung der Web Services stellt der EAI-Bereich.
Im Kern ging es darum, standardisierte Kommunikation zwischen verschiedenen Unternehmens-Anwendungen zu ermöglichen, standardisierte Schnittstellen zu spezifizieren und so ganzheitliche Informationssysteme zu ermöglichen.
Der Geschäftsprozess stellt dementsprechend den zentralen Gedanken des Zusammenspiels dar.
Logisch erscheint daher, nicht bloß einzelne Dienste abrufen zu können, sondern diese zu Geschäftsprozessen zu kombinieren.

Die Abbildung von Geschäftsprozessen in Informationssystemen sollen auf eine höhere Abstraktionsebene gebracht werden, was eine bessere Trennung von konkreten Implementierungen der Teilfunktion erlaubt. Somit wird die Geschäftslogik zentral und nicht mehr verteilt definiert, wodurch es leichter möglich wird, den Überblick zu behalten und auf veränderte Geschäftsabläufe reagieren zu können und gleichzeitig die Stärken einer verteilten Systemarchitektur zur Umsetzung der Aufgaben zu nutzen. Teilabläufe können so bei Bedarf viel leichter z.B. ausgetauscht werden.
Auch die Kommunikation im B2B-Bereich wird sehr erleichtert. Diese wird – gerade im Blick auf erhöhte Konzentration auf Kernkompetenzen und damit verbundenes Outsourcing - immer wichtiger.

Als verteilte Systeme und serviceorientierte Architekturen aufkamen, gab es für diese Aufgaben – wenn überhaupt vorhanden – jeweils nur neu entwickelte proprietäre Lösungen, in „klassischen“ Programmiersprachen / Frameworks wie Java oder .Net programmiert.
Im Hinblick auf eine verbesserte Standardisierung hatten sowohl Microsoft mit XLANG als auch IBM mit WSFL um die Jahrtausendwende herum Konzepte entwickelt.
Im Juli 2002 führten beide Firmen, unterstützt von BEA, die Konzepte zusammen und veröffentlichten BPEL4WS 1.0:
Business Process Execution Language for Web Services.
Die Version 1.1 wurde im Mai 2003 von folgenden Firmen vorgestellt:
Microsoft, IMB, Siebel Systems, SAP, BEA.
Im April 2003 wurde die Spezifikation an OASIS zur Standardisierung übergeben.
Die kommende Spezifikation ist die Version 2.0. Am 14. September 2004 beschloss das entsprechende OASIS-Komitee eine Umbenennung zu WS-BPEL um einheitlich mit den restlichen WS-*-Standards zu sein. Seitdem gewinnt auch für die aktuelle Version der Name „WS-BPEL“ bzw. „BPEL“ an Verwendungshäufigkeit.

4. BPEL4WS als Teil der Web Services-Architektur

Der derzeitig in der Informatik-Welt favorisierte Ansatz zur Errichtung einer Service Orientierten Architektur (SOA) sind die Web Services. Basis der Webservices sind eine Vielzahl unterschiedlicher Spezifikationen und Standards.

Für Transport und Encoding ist in erster Linie das XML-basierte SOAP zuständig.
Quality of Service stellen z.B. WS-Transaction und WS-Coordination sicher.
Eine Schnittstellenbeschreibung liefert WSDL, UDDI übernimmt die Funktion eines Verzeichnisdienstes.
Weitere Protokolle sollen Sicherungs- und Authentifizierungsaufgaben übernehmen.

Auf Basis dieser Standards ist es möglich, einzelne Web Services mit einander kommunizieren zu lassen, da eine gesicherte Kommunikation gewährleistet ist.
Um jedoch ein Zusammenspiel dieser Services zu erreichen, bedarf es einer weiteren logischen Schicht. Im Zuge der Standardisierung stellt die Web Services-Spezifikations-Familie für diese Aufgabe den (de-facto-) Standard zur Verfügung: BPEL4WS bzw. WS-BPEL.

BPEL4WS ermöglicht die Orchestrierung (die Zusammenführung) mehrerer Web Services zu einem Prozess, wobei ein BPEL4WS-Dokument selbst wieder einen Web Service darstellt, der wiederum von Clients, also auch wieder von BPEL4WS-Dokumenten, verwendet werden kann.
Der BPEL-Geschäftsprozess ist so gleichzeitig Requester und Service-Anbieter.

Damit steht WS-BPEL auf der zweithöchsten Abstraktions-Stufe der Web Services-Standards („orchestration service layer“).
Nur WS-CDL, was für die Choreographie von Webservices zuständig ist, liegt logisch gesehen darüber. Diese Sprache soll in dieser Seminararbeit jedoch nicht näher betrachtet werden.

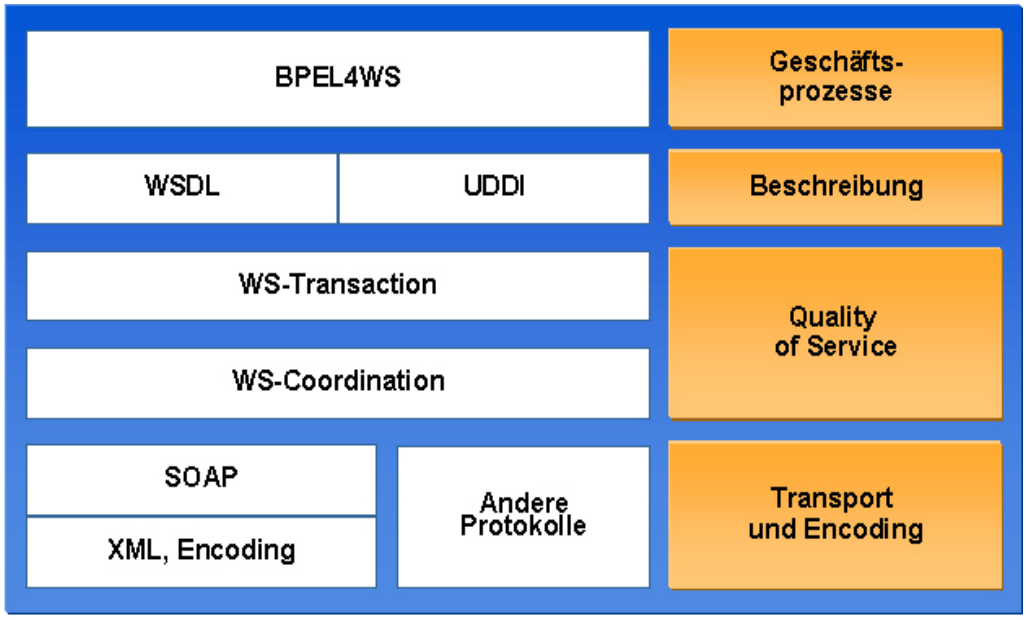

Abbildung 4: WS-Spezifikationen nach Andreas Spall, Aprill 2005

5. Beschreibung des BPEL4WS-Service per WSDL

5.1. Überblick

In der Web Services-Technologie wird auf Trennung von Servicebeschreibung (Interface) und Serviceimplementierung gesetzt. So muss jemand, der einen Dienst aufrufen möchte, keine Informationen über die interne Realisierung jenes Dienstes haben.
Die Aufgabe der Beschreibung wird dabei durch WSDL übernommen.

Wie alle Web Services setzt auch der in WS-BPEL programmierte Dienst als Grundlage WSDL-Beschreibungen ein, weswegen zunächst ein kurzer Überblick über WSDL gegeben werden soll.

WSDL-Dokumente enthalten einen abstrakten Teil bestehend aus <types>-, <message>- und <portType>-Elementen, welcher die Schnittstelle unabhängig von Transportprotokoll und konkreter Implementierung beschreibt, und einen konkreten Teil bestehend aus <binding>- und <service>-Elementen, welcher konkret die Erreichbarkeit des Web Services über URI und Protokoll definiert sowie Serialisierungs- und Codierungsinformationen enthält.

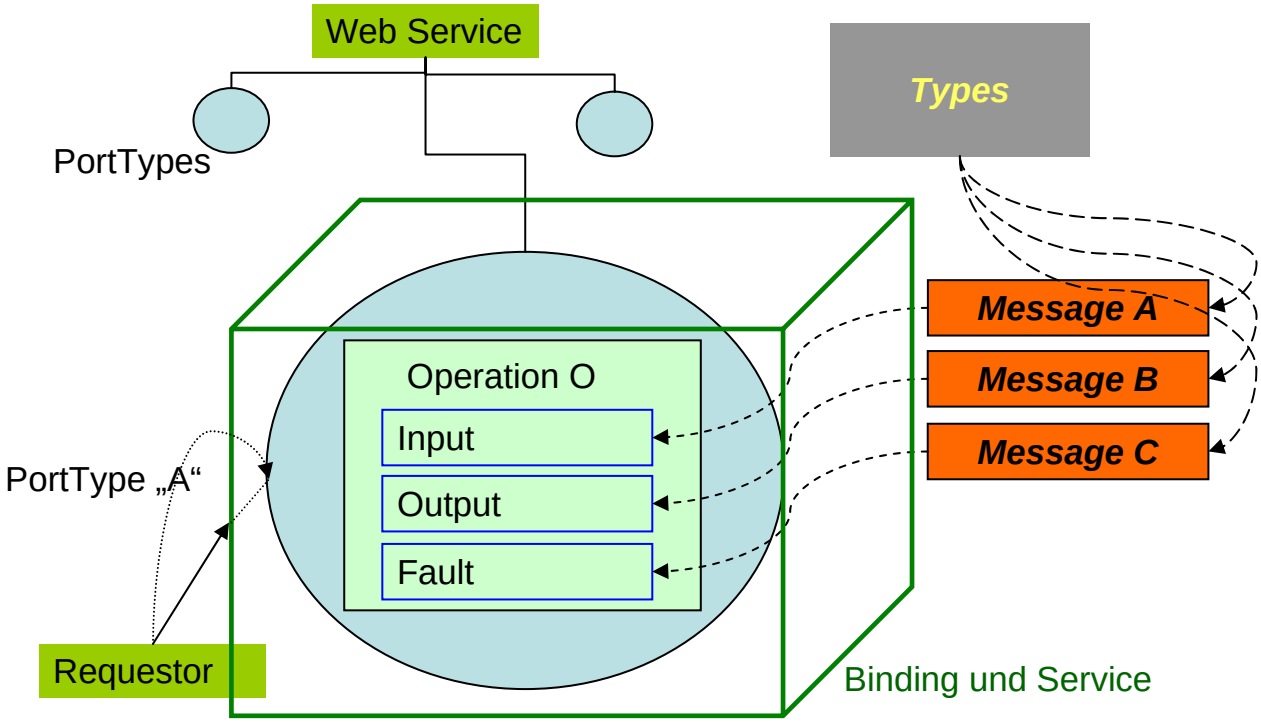

Abbildung 5: Schematische Darstellung des Inhalts eines WSDL-Dokuments

5.2. Types

ist ein optionales Element, welches von den Nachrichten verwendete komplexe Datentypen beschreibt. In klassischen Programmiersprachen entspricht dies der Definition eigener Typen.

5.3. Message

beschreibt, welche Datentypen bei welcher Nachricht verwendet werden. Eine Nachricht darf dabei mehrere Parameter bzw. Rückgabewerte („parts“) beinhalten.
Es handelt sich quasi um die Deklaration von „Nachrichten-Blaupausen“, Muster, wie zu verschickende Nachrichten aussehen können.
Es können als Parametertypen die zuvor definierten eigenen Typen oder einfache Datentypen verwendet werden.

5.4. portType

beschreibt eine Kommunikationsendpunkt als eine Bündlung konkret aufrufbarer Funktionen („operation“).
Eine Operation kann mehrere Nachrichten enthalten, wie sie zuvor definiert wurden.
Die Nachricht kann als Input oder Output bzw. als Fault verwendet werden.

Folgende Aufrufverfahren können beschrieben werden:

- Request-Response (input, output)
- Solicit-Response (output, output)
- One-Way (input)
- Notification (output)

5.5. Binding

Beim Binding werden den abstrakt definierten Operationen konkrete Informationen bezüglich Nachrichtenformat, Codierung und Transport zugeordnet.

5.6. Service

Bezüglich eines deklarierten Bindings werden konkrete Verbindungs-Informationen hinzugefügt. Genau kann hier die exakte URI und der zu verwendende Port angegeben werden.
Auch die Angabe mehrerer Ports als Alternativen für einen Web Service ist möglich.
Somit ist eine konkrete Verwendung durch einen Requestor möglich.

6. Die Sprache BPEL4WS

6.1. Überblick

Technisch gesehen ist die Sprache BPEL4WS eine XML-Sprache, die einem speziellen XML-Schema folgt.
Logisch handelt es sich dabei aber – genau genommen - um eine imperative Programmiersprache.
Die konkrete Umsetzung erfolgt dabei durch Prozess-Manager oder Workflow-Engines. Wie die konkrete Umsetzung durch jene aussieht oder auf welcher Basis sie selbst programmiert sind, ist für BPEL4WS irrelevant.
Die Sprache arbeitet auf einer höheren Abstraktionsebene („orchestrastion service layer“).
Grundlegend sieht ein BPEL-Dokument folgendermaßen aus:

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

      <!—some activity -->
</process>
```

Codebeispiel 1: Grundlegender Aufbau eines BPEL4WS-Dokuments

Das umschließende Element ist immer „process“, wobei über das Attribut „name“ die Bezeichnung des Geschäftsprozesses angegeben wird.
Im Folgenden soll die Dokumentstruktur näher erläutert und weiter ergänzt werden.

6.2. Partnerbeziehungen und Kommunikation

Partnerbeziehung / Rolle (PartnerLinkTypes, PartnerLinks)

BPEL4WS ermöglicht die Definition der Art der Kommunikation.

Mit der WSDL-Erweiterung „PartnerLinkTypes“ (Namespace: „plink“) werden Kommunikations-Klassifikationen erzeugt, indem Rollen generiert werden, denen Porttypen (entsprechend der WSDL portTypes) zugewiesen werden. Diese eigenen portTypes finden entsprechend ihres Services und ihres Bindings eine Korrespondenz bei den portTypes eines fremden Web Services.

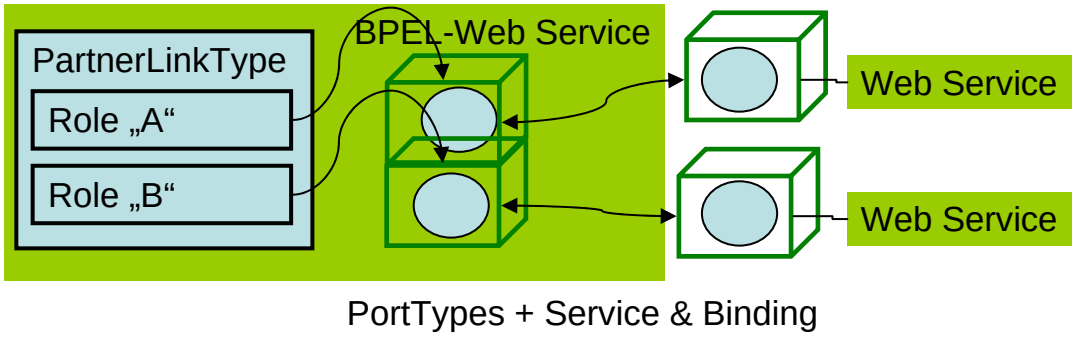

Abbildung 6: Schematische Darstellung einer PartnerLinkType-Definition

```
<plink:partnerLinkType name=“EmployeeServiceType>
      <plink:role name=”EmployeeServiceProvider”>
            <portType name=”EmployeeServiceInterface”>
      </plink:role>
      <plink:role name=”EmployeeServiceRequester”>
            <portType name=”EmployeeRequestInterface”>
      </plink:role>
</plink:partnerLinkType>
```

Codebeispiel 2: Definition von PartnerLinkTypes im WSDL-Dokument

Mit „PartnerLinks“ werden konkrete Kommunikationsmuster entsprechend der definierten „PartnerLinkTypes“ definiert. Dabei wird festgelegt, welche Rolle selbst und welche vom Kommunikationspartner übernommen werden soll.

Es lassen sich mehrere PartnerLinks, an denen ein Dienst beteiligt ist, als „partners“ zusammenfassen.

```
<partnerLinks>
      <partnerLink name=“Employee“
      partnerLinkType=“EmployeeServiceType“
```

```
        partnerRole="EmployeeServiceProvider" myRole="
        EmployeeServiceRequester />
</partnerLinks>
```

Codebeispiel 3: partnerLink-Deklaration

Invoke

Ein Web Service wird mit <Invoke> aufgerufen. Dies kann synchron (request/response) oder asynchron geschehen. Mit <Invoke> werden die partnerLinks, portTypes und Operationen (entsprechend des WSDL-Dokuments) konkret verwendet. Es können In- und Output-Variablen angegeben werden.

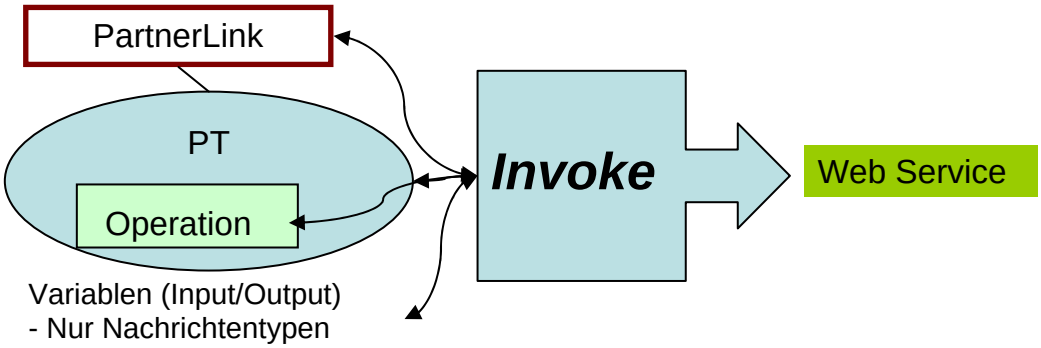

Abbildung 7: Schematische Darstellung eines Invoke

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

Codebeispiel 4: Aufruf der Funktion „buy“ eines anderen Web Services über partnerLink „buying“ und portType „SellerPT“

Receive / Reply

Mit <Receive> und <Reply> lassen sich eingehende Webdienstaufrufe realisieren.

<Receive> wartet auf Partner-Web Services, die den BPEL-Webservice aufrufen. Bis dies geschehen ist, wird der weitere Programmablauf geblockt.

Erwartet der Partner-Dienst eine Rückgabe oder Fehlermeldung, so wird <Reply> verwendet. Dieses arbeitet ähnlich dem „return“ aus klassischen Programmiersprachen.

Es werden Variablen beim <Receive> als Input und beim <Reply> als Output angegeben.

Per Attribut „createInstance“ kann kontrolliert werden, ob eine neue Instanz des Prozesses gestartet werden soll.

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

Codebeispiel 5: receive: Eingehender Webdienstaufruf, bei dem der Partner den Preis eines Produktes bezüglich einer angegebenen Produktnummer wissen möchte und so die weiteren Aktionen des BPEL-Dienstes in Gang setzt.

```
<reply partnerLink=“selling“ portType=“SellerPT“ operation=“getamount“
variable=“price“/>
```

Codebeispiel 6: Reply: Als Ergebnis der Operation „getamount“ wird der Preis an den Aufrufer zurückgeliefert.

Ereignisbehandlung

Bei bestimmten Ereignissen können unmittelbar und unabhängig Aktionen ausgeführt werden. Diese Ereignisse können Operationsaufrufe oder auch abgelaufene Zeitlimits sein. Sinnvoll ist dies z.B., um eine Abbruchmeldung oder eine TimeOut-Länge zu realisieren.

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

Codebeispiel 7: Ereignisbehandlung - Terminierung bei Abbruchsignal

6.3. Strukturanweisungen

Analog zu „klassischen“ Programmiersprachen verfügt auch WS-BPEL über eine Reihe von strukturierten Anweisungen.

Selektion

Mit dem <Switch>-Element ist die allgemeinste Form der Selektion, die Mehrfachauswahl, möglich. Dabei handelt es sich um eine bedingte Ausführung von Aktionen. <Switch> arbeitet wie auch in anderen Sprachen mit <case> für

beliebig viele zu definierende Fälle (in Abhängigkeit von bestimmten Bedingungen) und <otherwise> zum Abfangen nicht definierter Fälle. Wird nur eine Bedingung definiert, wird eine bedingte Aktion bzw. Alternativen-Semantik („if-then-else“) realisiert.

```
<switch>
      <case condition=“amount>0“>
            <!-- some activity -->
      </case>
      <otherwise>
            <!-- some other activity -->
      </otherwise>
</switch>
```

Codebeispiel 8: Auswahl abhängig von Bedingungen. Hier Umsetzung einer simplen if-then-else-Semantik

Iteration

Mit dem <while>-Element wird eine klassische Schleife umgesetzt: Aktivitäten werden so lange ausgeführt, wie eine boolesche Bedingung erfüllt ist.

```
<while condition=“bpws:getVariableData(‚counter’)<10“>
      <!-- some activities -->
      <!-- change ‚counter’ -->
</while>
```

Codebeispiel 9: Eine einfache Schleife

Sequenz

Klassische Ausführungsreihenfolge: Die Operationen werden hintereinander abgearbeitet.

Insbesondere wird diese explizite Deklaration einer sequentiellen Ausführung interessant, wenn es sich um einen Ausführungsblock innerhalb einer nebenläufigen Umgebung handelt.

```
<sequence>
      ...
</sequence>
```

Codebeispiel 10: Sequenz-Deklaration

Nebenläufigkeiten

Auch BPEL4WS unterstützt Nebenläufigkeiten (Threads). Abhängigkeiten zur Synchronisation (Vorbedingungen, Effekte) können über <Links>, welche als <target> oder <source> eingesetzt werden, realisiert werden.

Per „transitionCondition“ kann festgelegt werden, ob ein <Link> zur Anwendung kommt oder nicht, wodurch auch eine Selektions-Semantik realisiert werden kann.

Auch die Definition mehrerer Kausalitätsbedingungen und die Zusammenführung dieser per join-Condition-Attribut ist möglich.

```
<flow>
    <links>
        <link name="AtoB">
    </links>
    <!-- invoke operation A -->
    <invoke ..>
        <source linkName="AtoB"/>
    </invoke>

    <sequence>
        ... <!-- some activity -->

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

Codebeispiel 11: Nebenläufigkeit; Operation B (innerhalb der Sequenz) darf erst dann starten, wenn Operation A abgeschlossen ist.

Pick

realisiert eine Kombination von Switch- und receive-Semantik.

Die Reihenfolge von außen kommender Prozessaufrufe kann nicht durch das Programm selber bestimmt werden. Es ist nicht deterministisch.

Soll aber nur genau eines der Ereignisse Aktivitäten nach sich ziehen, so müssen mit Eintreten des ersten Ereignisses alle anderen Ereignis-Behandlungen gesperrt werden.

<pick> ermöglicht dieses gegenseitige Ausschließen, wie es auch <switch> für deterministische Fälle macht.

```
<pick>
      <onMessage partnerLink=“selling“ portType=“SellerPT“
      operation=“getPrice“ variable=“itemid“>
            <!-- some activities -->
      </onMessage>
      <onMessage partnerLink=“selling“ portType=“SellerPT“
      operation=“buy“ variable=“itemid“>
            <!-- some other activities -->
      </onMessage>
 </pick>
```

Codebeispiel 12: Auswahl einer Aktivität je nach Nachricht; onMessage arbeitet dabei analog zu receive

Gültigkeitsraum (Scope)

Mit einem <scope> lässt sich ein Gültigkeitsbereich und Namensraum für Variablen und Fehlerbehandlungen deklarieren. Es ist möglich, <scopes> in einander zu verschachteln und so eine Hierarchie von Gültigkeitsräumen aufzubauen.

Das <scope>-Element darf alle Elemente außer dem <partnerLinks>-Element enthalten, welches sich nur auf den gesamten Prozess beziehen darf.

```
<scope name="InnerScope" variableAccessSerializable="no">
      ...
</scope>
```

Codebeispiel 13: Deklaration eines Gültigkeitsraums

6.4. Zustände / Sitzungen

Variable

Daten können referenziert werden. Dies gilt auch für Prozessinstanzen.

```
<variables>
      <variable name=“anzahl“ type=“xsd:int“/>
</variables>
```

Codebeispiel 14: Variable „anzahl“ wird als Typ Integer definiert. Alternative Typdeklaration wäre über „messageType“ oder „element“ möglich.

Mit Assign werden Variablen explizit mit Werten belegt.

```
<assign>
      <copy>
            <from>70</from>
            <to variable=“anzahl“>
      </copy>
```

```
</assign>
```

Codebeispiel 15: Beispiel-Wertezuweisung auf Variable.

Korrelation

In WS-BPEL versteht man unter Korrelationsmengen Mengen von Eigenschaften, die zur Identifikation verwendet werden. Dies wird wichtig, wenn das Dokument zur gleichen Zeit von unterschiedlichen Aufrufern genutzt wird. Diese Eigenschaften werden einmalig initialisiert und finden Verwendung bei ein- und ausgehenden Nachrichten sowie Ereignissen und übernehmen damit die Funktion von „Session-Informationen“.

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

Codebeispiel 16: Deklaration von zu verwendenden Korrelationen

```
<receive createInstance="yes" name="BuyerReceive"
operation="submit" partnerLink="buyer" portType="tns:buyerPT"
variable="buyerInfo">
     <correlations>
          <correlation initiate="yes" set="userid"/>
     </correlations>
</receive>
```

Codebeispiel 17: Verwendung des Korrelations-Sets bei einer Receive-Operation

Eigenschaften

Eigenschaften erlauben den Zugriff auf interne Werte von Nachrichten, deren Inhalte ansonsten normalerweise unsichtbar sind.

Diese Transparenz ist zwar im Sinne der losen Kopplung sehr sinnvoll. Jedoch ist in manchen Situationen dennoch eine Kenntnis dieser Inhalte wichtig, da Entscheidung davon abhängen.

Mit <property> lässt sich ein Alias auf den Inhalt einer Nachricht definieren, auf den öffentlicher Zugriff möglich ist.

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

<propertyAlias propertyName=”userID” messageType=”orderDetails”
part=”identification” query=”/credentials/”/>
```

Codebeispiel 18: Die Eigenschaft userID erlaubt einen öffentlichen Zugriff auf die Identifikationsnummer in der Bestellnachricht

Zum Auslesen der nun öffentlich gemachten Informationen wird die Funktion *getVariableProperty* (’variableName’,’*propertyName*’) verwendet.

6.5. Ausnahme- und Fehlerbehandlung

FaultHandling und CompensationHandling

In komplexen Abläufen (vor allem bei verteilten Systemen) kann es immer wieder zu Situationen kommen, die vom geplanten Programmablauf abweichen. Es bestehen daher Mechanismen zum Abfangen von Fehlern und Ausnahmen im Programmablauf. Der „Fault“ ist bereits Bestandteil von WSDL und wird in WS-BPEL dafür verwendet.
Mit der Anweisung <Catch> werden Fehler abgefangen. Wird ein Fehler nicht aufgefangen und erreicht den globalen Scope, so terminiert der Prozess.
Mit <CatchAll> werden unspezifiziert alle Fehler abgefangen.
Man gibt mit dem <FaultHandler> an, wie die Behandlung eines konkreten Fehlerfalls aussehen soll.

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

Codebeispiel 19: Abfangen möglicher Fehler: Kompensieren (siehe unten) bei “NoDatabaseConnectionFault“, sonst terminieren.

Da Fehler abgefangen werden können, können auch eigene Ausnahmen hervorgerufen werden. Dies geht sowohl als Fehlerattribut einer ausgehenden Kommunikation als auch durch explizites Hervorrufen des Fehlers (<throw>).

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

Codebeispiel 20: explizites Werfen eins Fehlers

Business-Aktivitäten werden in der Regel durch verschiedene Teilaktivitäten realisiert. Unter Umständen laufen sie auch über längere Zeiträume (z.B. weil auf die Informationen von Benutzerschnittstellen und damit auf Benutzereingaben gewartet werden muss), weswegen ein Sperren von Ressourcen nicht sinnvoll ist.

Wird innerhalb einer Reihe semantisch zusammen gehörender (sich im selben Scope befindender) Aktionen ein Fehler festgestellt, so sollten auch die eigentlich in sich erfolgreichen Teilschritte zurückgenommen werden, da sie alleine inkonsistente Zustände hinterlassen können.

Mit <Compensate> kann dieses Rückgängigmachen realisiert werden.

Dabei werden die Aktivitäten eines zuvor definierten <CompensationHandlers> ausgeführt.

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

Codebeispiel 21: Kompensation durch Rücknahme des Auktionsgebots.

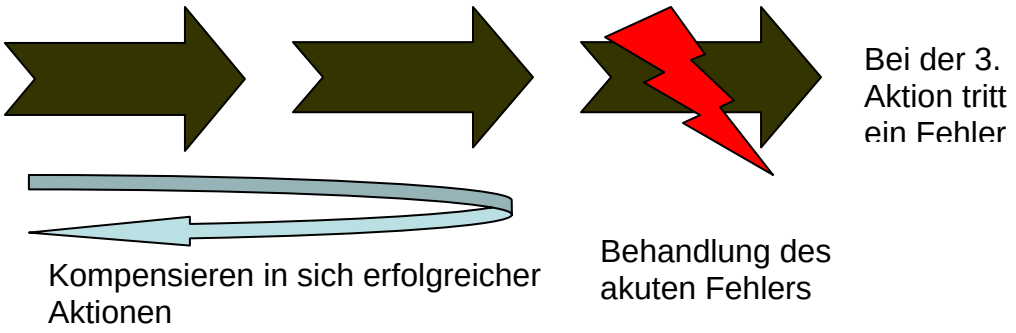

Abbildung 8: Schematische Darstellung des Fault- und CompensationHandlings

6.6. Zusätzliche Sprachelemente

Wait

Warten auf einen Zeitpunkt oder für eine Zeitspanne.

```
<wait for="'PT1H2M20S'"/>
```

Codebeispiel 22: Warten: 1 Stunde, 2 Minuten, 20 Sekunden

Empty

Nichts tun, z.B. um in einer Fehlerbehandlung nichts zu tun und den Fehler so zu unterdrücken. (NOOP: No Operation)

```
<empty/>
```

Codebeispiel 23: Leere Aktivität

Terminate

Der Prozess wird explizit beendet. Dies entspricht einem „Exit“ „klassischer“ Programmiersprachen.

```
<terminate/>
```

Codebeispiel 24: Beendigung des Prozesses

7. BPEL4WS in der Anwendung

7.1. Tools / Engines

Zum konkreten Ausführen von BPEL-Prozessen werden BPEL-Workflow-Engines oder Prozess-Manager verwendet. Dafür muss der Prozess eingebracht (deployed) werden.

Zum Erstellen von BPEL-Prozessen stehen unterschiedliche Design-Tools zur Verfügung, mit deren Hilfe ein Design auf Basis graphischer Elemente möglich ist.

Der eigentliche Code wird automatisch im Hintergrund erstellt, kann aber bei Bedarf betrachtet und verändert werden.

Folgende Tools sind zur Zeit auf dem Markt:

- *Oracle BPEL Process Manager*
  - Implementierung des BPEL-Standards 1.1
  - grafisches Modellierungs-/Orchestrierungs-Tool
- *Twister*
  - Open Source Implementierung des BPEL-Standards
  - Twister unterstützt das SOA-Pattern wie auch direkte Interaktion mit Menschen (durch Arbeitslisten)
  - Wird jetzt unter dem Namen Agila weitergeführt
- *ActiveBPEL*

  - Open Source Implementierung von BPEL4WS 1.1 durch ActiveEndpoints
- *INTALIO*
  - Open Source BPEL Server
- *BPWS4J*
  - BPEL-Implementierung von IBM
- *WebSphere Integration Developer*
  - Workflow-Modellierungstool von IBM
- *Microsoft BizTalk Server*
  - BizTalk Server 2006 kann Prozessmodelle, die mit Modellierungswerkzeugen wie Microsoft Visio erstellt worden sind, ausführen.

Das Open Source Tool ActiveBPEL Designer von der Firma Active Endpoints soll hier kurz näher vorgestellt werden.
Die Entwicklung erfolgt über eine auf Eclipse basierenden Oberfläche.
Dabei kann sowohl über graphische Elemente modelliert als auch der Code direkt manipuliert werden.
Ebenfalls unterstützt sind Debug und Test.
Ein Tomcat-Server kann in Verbindung mit einem generischen SOAP-Client zum Testen der Prozesse verwendet werden.

7.2. Beispiel

Nachdem die Sprache theoretisch sowie mit vielen Einzelcodebeispielen erläutert wurde, ist hier noch einmal ein komplettes Beispiel zusammengefasst.
Das Beispiel gehört zur Beispiel-Sammlung des ActiveBPEL-Designers und basiert auf den Beispielen zur offiziellen Spezifikation BPEL4WS 1.1.
Selbstverständlich deckt es nur ein paar Konzepte der Sprache ab. Da es jedoch ein vollständig lauffähiger Prozess ist, soll er hier einmal betrachtet werden.

Alle relevanten Codes und Dateiinhalte befinden sich im Anhang dieses Dokuments.

Der Prozess wird graphisch im ActiveBPEL-Designer folgendermaßen dargestellt:

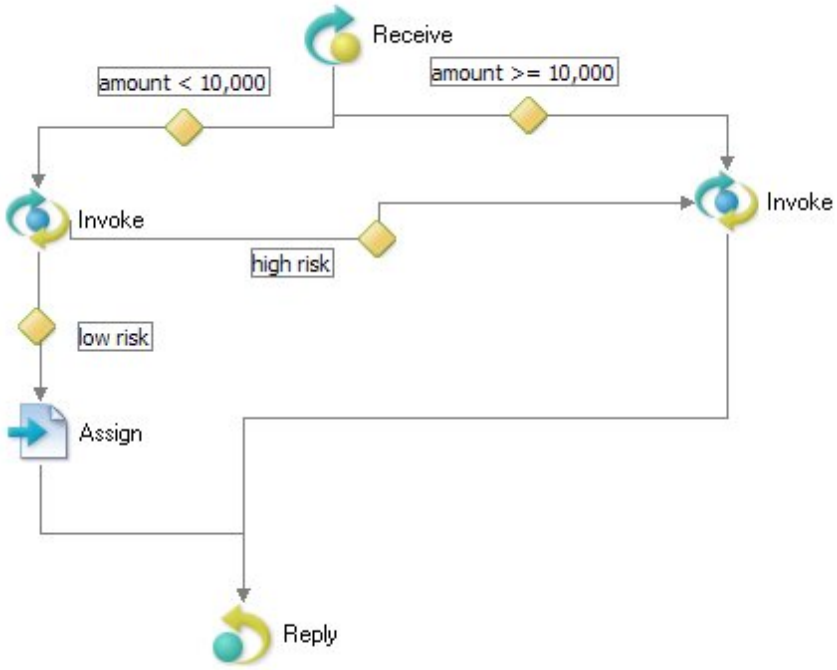

Abbildung 9: LoanApproval-Service in der graphischen Darstellung

Der LoanApproval-Service ist eine simple Umsetzung einer Kreditwürdigkeitsprüfung wie sie im eingangs vorgestellten Beispiel[1] Verwendung finden könnte.
Der Prozess erhält eine Anfrage mit Kundeninformationen, die neben Vorname und Name auch einen Wert (die Höhe des gewünschten Kredits) enthält.

Liegt dieser Wert unter 10000, so wird der WebService „AssessorWebService“ mit der Operation „check“ aufgerufen, d.h. es wird von der Prozesslogik die Aufgabe der Risikoeinschätzung an die konkrete Umsetzungsinstanz weitergeleitet.

Dieser Bewertungs-Service ist in diesem Beispiel sehr einfach gehalten.
Er gibt einen risk-level zurück, der je nach Einstellung der Konfigurationsdatei „loan_approval_config.xml“ „low“ oder „high“ ist. Es kann auch ein Fehler zurückkommen.

[1] siehe EPK im Abschnitt. 2

Bei niedrigem Risiko wird die Antwort „yes“ generiert (Variable „accept“).

Liegt der Wert über 10000 oder ist gleich diesem bzw. gibt der Assessor ein hohes Risiko zurück, so wird der WebService „ApproverWebService“ mit der Operation „approve“ („billigen“) aufgerufen.

Dieser „Billiger“ ist ebenfalls sehr simpel konstruiert.
Die zu generierende Antwort hängt auch hier einzig von den Einstellungen der Konfigurationsdatei „loan_approval_config.xml“ ab. Die Antwort (die „approvalMessage“ in der „approveResponse“) kann entweder „yes“ oder „no“ lauten.

Mit dem Ergebnis dieser Antwort wird nun die Gesamt-Antwort für den Client generiert.

Zum Schluss wird die Antwort zurückgesendet und der übergeordnete Prozess erfährt damit, ob der Kredit gewährt wird oder nicht.

8. Ausblick: Erweiterungen zu BPEL4WS

8.1. BPELJ

BPELJ ist eine Kombination aus den Programmiersprachen WS-BPEL und Java. Es erlaubt, beide Sprachen gemeinsam zu nutzen, um Geschäftsprozess-Applikationen zu erstellen.
Dabei werden zum einen kleinere Änderungen an der Grundsprache BPEL durchgeführt sowie Erweiterungen mit aufgenommen.
So können auch Java-Code-Elemente mit integriert werden, um neben Web Services auch direkt lang laufende Interaktionen mit J2EE-Komponenten realisieren zu können. Die bereits vorhandenen Java-Komponenten für Geschäftslogik werden direkt zugreifbar gemacht.

8.2. BPEL4people

WS-BPEL ist als Sprache entwickelt worden, die ausschließlich Maschine-zu-Maschine-Kommunikation ermöglicht. Automatisch sollen Geschäftsprozesse basierend auf Web Services abgebildet werden.
Benutzerinteraktion ist nicht Teil der Sprache. Diese findet in vorgelagerten Interface-Programmen statt, welche die Web Services und somit die WS-BPEL-Dokumente aufrufen.
Da viele Prozesse in der Realität auf Benutzereingaben oder gar Benutzerinteraktion angewiesen sind, ist jeder, der WS-BPEL verwenden möchte, darauf angewiesen, eigene Lösungen für diese Interaktion zu finden.
BPEL4people soll als Erweiterungsschicht zur Kernsprache WS-BPEL standardisierte Konzepte zur Benutzerinteraktion zur Verfügung stellen.

9. Fazit

In dieser Seminararbeit wurden Sprache WS-BPEL und damit ihre Vorzüge zur Geschäftslogik-Programmierung vorgestellt.
Trotz jedoch aller zum Teil begründeter Euphorie bezüglich Web Services und BPEL4WS sind die Konzepte der Webservices und von Geschäftsprozess-Applikationen nicht neu.
Middleware-Lösungen wie CORBA oder RPC sind schon länger im Einsatz.
Applikationsserver stellen Unternehmen Funktionen zur Verfügung, die auch – trotz ursprünglich anderer Konzeption - für Geschäftslogik zuständig sein können.
Die beiden konkurrierenden Frameworks, das DotNet von Microsoft und Java z.B. unter J2EE von Sun Microsystems, stellen für spezialisierte Software die bedeutendsten Produkte dar.
Für Standardgeschäftsabläufe wie Materialplanung, Finanzwirtschaft, Produktionswirtschaft und Controlling hat sich SAP R/3 als Marktführer etabliert.

Die Frage, die sich Unternehmen stellen müssen, ist, inwiefern es sich lohnt, in eine neue IT-Infrastruktur zu investieren. Neben einer Bindung von Kapital und Personal ist abzuwarten, inwieweit sich Web Services und BPEL4WS einerseits durchsetzen werden, andererseits Vorgänge wirklich optimieren können.

Als positiv ist zu sehen, dass viele wichtige IT-Unternehmen an der Spezifikation beteiligt waren, was eine hohe Marktakzeptanz vermuten lässt.

Ein sinnvoller Weg scheint zu sein, bestehende Systeme insoweit zu belassen, wie sie geschlossene Funktionalitäten darstellen, und diese als Ganzes als Web Service anzubieten. Systeme mit schwächerer logischer Kopplung sollten umgeschrieben werden in einzelnen Services.
Für den betriebswirtschaftlichen Überbau (der in dieser Form zum Teil noch gar nicht besteht) stellt WS-BPEL eine sehr sinnvolle Möglichkeit dar durchaus.
Die Vorteile mit zentraler Geschäftslogik, hoher Anpassungsgeschwindigkeit und erleichterter B2B-Kommunikation sind klar vorhanden.
Die Frage, für die sich eine gründliche Analyse lohnt, ist der Grad der Granulierung und der Möglichkeiten, (funktionierende, wertvolle) Altsysteme in die neue Architektur zu integrieren.
Selbstverständlich ist die Entscheidung, die IT-Infrastruktur diesbezüglich umzustellen, gerade für größere Unternehmen keine leichte, da die Konsequenzen teuer sein können.

Noch teurer kann es jedoch sein, wenn man den Anschluss an eine neue Technologie und damit Marktpotential verliert, denn gerade kleine, junge Unternehmen werden diese Chancen wahrnehmen.

Anhang

1. Literaturverzeichnis

1.1. Bücher

Gustavo Alonso / Fabio Casati / Harumi Kuno / Vijaj Machiraju:
Web Services: Concepts, Architectures, and Applications
Springer 2004, ISBN 3-540-44008-9

Thomas Erl:
Service-Oriented Architecture, Concepts, Technology, and Design
Prentice Hall 2005, ISBN 0-13-185858-0

Thorsten Langner:
Web-basierte Abwendungsentwicklung - Die wichtigsten Technologien für Webapplikationen im Überblick
Elsevier (Spektrum) 2004, ISBN 3-874-1501-2

1.2. Wissenschaftliche Veröffentlichungen

Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, Sanjiva Weerawarana:
Business Process Execution Language for Web Services
Version 1.1
05.05.2003
u.a.:
http://www-128.ibm.com/developerworks/library/specification/ws-bpel/
ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf

Frank Leymann, Dieter Roller, and Satish Thatte
Goals of the BPEL4WS Specification
u.a.:
http://xml.coverpages.org/BPEL4WS-DesignGoals.pdf

Andreas Schmietendorf
Grundlagen und Einführung im Kontext SOA
Vorlesung - SOA
IVS – Arbeitsgruppe Softwaretechnik
Wintersemester 2005
Otto-von-Guericke-Universität Magdeburg

Riad Djemili
BPEL4WS – Business Process Execution Language for Web Services
Seminararbeit Advanced Topics in Networking
Wintersemester 2003/2004
Freie Universität Berlin
Fachbereich Mathematik und Informatik
http://riad.de/edu/

http://riad.de/edu/BPEL4WS.pdf
http://riad.de/edu/BPEL4WS.ppt

1.3. Internetquellen

WS-BPEL Extension for People – BPEL4People
http://www-128.ibm.com/developerworks/webservices/library/specification/ws-bpel4people/

BPELJ: BPEL for Java
http://www-128.ibm.com/developerworks/library/specification/ws-bpelj/)

Web Services Choreography Description Language Version 1.0
W3C Candidate Recommendation
09.11.2005
http://www.w3.org/TR/ws-cdl-10/

Introduction to the ActiveBPEL Engine
http://www.activebpel.org/info/intro.html

Infrasoft: Aufbau von Web Services
http://www.infrasoft.at/service/Aufbau_von_Web_Services.pdf

Wikipedia – die freie Enzyklopädie
http://de.wikipedia.org/wiki/BPEL4WS
http://de.wikipedia.org/wiki/Gesch%C3%A4ftsprozess
http://de.wikipedia.org/wiki/XML
http://de.wikipedia.org/wiki/OASIS

Dipl.-Ing. Torsten Winterberg, Rolf Scheuch
Mit BPEL in eine neue Entwicklungs-Ära
http://www.bpm-guide.de/articles/41

OIO (Oriented in Objects):
Einführung in BPEL4WS
http://www.oio.de/public/xml/einfuehrung-in-bpel4ws.htm

Die WS-* Spezfikationen
http://www.oio.de/public/xml/web-service-specifications.htm

Jboss.com – Wiki - WSBPEL
http://wiki.jboss.org/wiki/Wiki.jsp?page=WSBPEL

Pierre Feldbusch
http://www.cachaca.de
Web Services
http://www.cachaca.de/index.php?section=137

SOA - Service-Oriented-Architecture
http://www.cachaca.de/index.php?section=129

2. Inhalt der verwendeten Beispieldateien

2.1. Beispiel-BPEL-Prozess

```
<?xml version="1.0" encoding="UTF-8"?>
<process name="loanApprovalProcess" suppressJoinFailure="yes"
targetNamespace="urn:active-endpoints.samples.bpel1_1.2005-10.loan_approval"
xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
xmlns:lns="urn:active-endpoints.resources.wsdl.2005-10.loan_approval"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<partnerLinks>
        <partnerLink myRole="loanService" name="customer"
        partnerLinkType="lns:loanPartnerLinkType"/>
        <partnerLink name="approver"
        partnerLinkType="lns:loanApprovalLinkType" partnerRole="approver"/>
        <partnerLink name="assessor"
        partnerLinkType="lns:riskAssessmentLinkType" partnerRole="assessor"/>
  </partnerLinks>
  <variables>
    <variable messageType="lns:creditInformationMessage" name="request"/>
    <variable messageType="lns:riskAssessmentMessage" name="risk"/>
    <variable messageType="lns:approvalMessage" name="approval"/>
    <variable messageType="lns:errorMessage" name="error"/>
  </variables>
  <faultHandlers>
    <catch faultName="lns:loanProcessFault" faultVariable="error">
        <reply faultName="lns:unableToHandleRequest" operation="request"
        partnerLink="customer" portType="lns:loanServicePT" variable="error"/>
    </catch>
  </faultHandlers>

<flow>
    <links>
      <link name="receive-to-assess"/>
      <link name="receive-to-approval"/>
      <link name="assess-to-setMessage"/>
      <link name="assess-to-approval"/>
      <link name="setMessage-to-reply"/>
      <link name="approval-to-reply"/>
    </links>

 <receive createInstance="yes" operation="request" partnerLink="customer"
 portType="lns:loanServicePT" variable="request">
        <source linkName="receive-to-assess"
        transitionCondition="bpws:getVariableData('request','amount') &lt;
        10000"/>
        <source linkName="receive-to-approval"
        transitionCondition="bpws:getVariableData('request','amount') &gt;=
        10000"/>
 </receive>
 <invoke inputVariable="request" operation="check" outputVariable="risk"
partnerLink="assessor" portType="lns:riskAssessmentPT">
        <target linkName="receive-to-assess"/>
        <source linkName="assess-to-setMessage"
        transitionCondition="bpws:getVariableData('risk','level')='low'"/>
        <source linkName="assess-to-approval"
        transitionCondition="bpws:getVariableData('risk','level')!='low'"/>
```

```
 </invoke>
 <assign>
       <target linkName="assess-to-setMessage"/>
       <source linkName="setMessage-to-reply"/>
       <copy>
         <from expression="'yes'"/>
         <to part="accept" variable="approval"/>
       </copy>
 </assign>
  <invoke inputVariable="request" operation="approve" outputVariable="approval"
partnerLink="approver" portType="lns:loanApprovalPT">
       <target linkName="receive-to-approval"/>
       <target linkName="assess-to-approval"/>
       <source linkName="approval-to-reply"/>
 </invoke>
 <reply operation="request" partnerLink="customer" portType="lns:loanServicePT"
variable="approval">
       <target linkName="setMessage-to-reply"/>
       <target linkName="approval-to-reply"/>
 </reply>
 </flow>
</process>
```

2.2. WSDL-Dokument zur Beispiel-BPEL-Datei

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace="urn:active-endpoints.resources.wsdl.2005-
10.loan_approval" xmlns:tns="urn:active-endpoints.resources.wsdl.2005-
10.loan_approval" xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-
link/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:lns="urn:active-
endpoints.resources.wsdl.2005-10.loan_approval"
xmlns="http://schemas.xmlsoap.org/wsdl/">
  <message name="approvalMessage">
    <part name="accept" type="xsd:string"/>
  </message>
  <message name="creditInformationMessage">
    <part name="firstName" type="xsd:string"/>
    <part name="name" type="xsd:string"/>
    <part name="amount" type="xsd:integer"/>
  </message>
  <message name="riskAssessmentMessage">
    <part name="level" type="xsd:string"/>
  </message>
  <message name="errorMessage">
    <part name="errorCode" type="xsd:integer"/>
  </message>
  <portType name="loanApprovalPT">
    <operation name="approve">
      <input message="tns:creditInformationMessage"/>
      <output message="tns:approvalMessage"/>
      <fault name="loanProcessFault" message="tns:errorMessage"/>
    </operation>
  </portType>
  <portType name="riskAssessmentPT">
    <operation name="check">
      <input message="tns:creditInformationMessage"/>
      <output message="tns:riskAssessmentMessage"/>
      <fault name="loanProcessFault" message="tns:errorMessage"/>
```

```
    </operation>
  </portType>
  <portType name="loanServicePT">
    <operation name="request">
      <input message="tns:creditInformationMessage"/>
      <output message="tns:approvalMessage"/>
      <fault name="unableToHandleRequest" message="tns:errorMessage"/>
    </operation>
  </portType>
  <binding name="SOAPBinding1" type="tns:riskAssessmentPT">
        <soap:binding xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="check">
        <soap:operation xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap/
        soapAction="" style="rpc"/>
      <input>
        <soap:body xmlns:soap="http//schemas.xmlsoap.org/wsdl/soap/"
        encdingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="urn:loanassessor" use="encoded"/>
      </input>
      <output>
        <soap:body xmlns:soap="http//schemas.xmlsoap.org/wsdl/soap/"
        encdingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="urn:loanassessor" use="encoded"/>
      </output>
    </operation>
  </binding>
  <binding name="SOAPBinding" type="tns:loanApprovalPT">
        <soap:binding xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        tyle="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="approve">
        <soap:operation xmlns:soap=http://schemas.xmlsoap.org/wsdl/soap/
        soapAction="" style="rpc"/>
          <input>
              <soap:body xmlns:soap="http:/schemas.xmlsoap.org/wsdl/soap/"
              encodngStyle="http://schemas.xmlsoap.org/soap/encoding/"
              namespace="urn:loanapprover" use="encoded"/>
          </input>
          <output>
              <soap:body xmlns:soap="http:/schemas.xmlsoap.org/wsdl/soap/"
              encoingStyle="http://schemas.xmlsoap.org/soap/encoding/"
              namespace="urn:loanapprover" use="encoded"/>
          </output>
    </operation>
  </binding>
  <service name="LoanAssessor">
<documentation>Loan Assessor Service</documentation>
    <port name="SOAPPort1" binding="tns:SOAPBinding1">
    <soap:address xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
location="http://localhost:8080/active-bpel/services/AssessorWebService"/>
    </port>
  </service>
  <service name="LoanApprover">
<documentation>Loan Approver Service</documentation>
    <port name="SOAPPort" binding="tns:SOAPBinding">
    <soap:address xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
location="http://localhost:8080/active-bpel/services/ApproverWebService"/>
    </port>
  </service>
<plnk:partnerLinkType name="loanPartnerLinkType"
xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
```

```
  <plnk:role name="loanService">
    <plnk:portType name="tns:loanServicePT"/>
  </plnk:role>
</plnk:partnerLinkType>
<plnk:partnerLinkType name="loanApprovalLinkType"
xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
  <plnk:role name="approver">
    <plnk:portType name="tns:loanApprovalPT"/>
  </plnk:role>
</plnk:partnerLinkType>
<plnk:partnerLinkType name="riskAssessmentLinkType"
xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
  <plnk:role name="assessor">
    <plnk:portType name="tns:riskAssessmentPT"/>
  </plnk:role>
</plnk:partnerLinkType>
</definitions>
```

2.3. WSDL-Datei des aufgerufenen Services AssessorWebService

```
<wsdl:definitions targetNamespace="http://tempuri.org/services/loanassessor">

<!--
WSDL created by Apache Axis version: 1.2.1
Built on Apr 17, 2006 (04:00:53 EDT)
-->

<wsdl:types>

  <schema targetNamespace="http://loan_approval.samples.activebpel.org">
        <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>

        <complexType name="loanProcessFault">
                <sequence/>
        </complexType>
  </schema>
</wsdl:types>

<wsdl:message name="checkRequest">
        <wsdl:part name="firstName" type="xsd:string"/>
        <wsdl:part name="name" type="xsd:string"/>
        <wsdl:part name="amount" type="xsd:integer"/>
</wsdl:message>

<wsdl:message name="checkResponse">
        <wsdl:part name="riskAssessmentMessage" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="loanProcessFault">
        <wsdl:part name="fault" type="tns1:loanProcessFault"/>
</wsdl:message>

<wsdl:portType name="AssessorWebService">

        <wsdl:operation name="check" parameterOrder="firstName name
        amount">
```

```
                <wsdl:input message="impl:checkRequest"
                name="checkRequest"/>
                <wsdl:output message="impl:checkResponse"
                name="checkResponse"/>
                <wsdl:fault message="impl:loanProcessFault"
                name="loanProcessFault"/>
        </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="AssessorWebServiceSoapBinding"
type="impl:AssessorWebService">
<wsdlsoap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="check">
        <wsdlsoap:operation soapAction=""/>
                <wsdl:input name="checkRequest">
                        <wsdlsoap:body
                        encodingStyle="http://schemas.xmlsoap.org/soap/encoding
                        /" namespace="http://tempuri.org/services/loanassessor"
                        use="encoded"/>
                </wsdl:input>
                <wsdl:output name="checkResponse">
                        <wsdlsoap:body
                        encodingStyle="http://schemas.xmlsoap.org/soap/encoding
                        /" namespace="http://tempuri.org/services/loanassessor"
                        use="encoded"/>
                </wsdl:output>
                <wsdl:fault name="loanProcessFault">
                        <wsdlsoap:fault
                        encodingStyle="http://schemas.xmlsoap.org/soap/encoding
                        /" name="loanProcessFault"
                        namespace="http://tempuri.org/services/loanassessor"
                        use="encoded"/>
                </wsdl:fault>
        </wsdl:operation>
</wsdl:binding>

<wsdl:service name="AssessorWebServiceService">
        <wsdl:port binding="impl:AssessorWebServiceSoapBinding"
        name="AssessorWebService">
                <wsdlsoap:address location="http://localhost:8080/active-
                bpel/services/AssessorWebService"/>
        </wsdl:port>
</wsdl:service>

</wsdl:definitions>
```

2.5. WSDL-Datei des aufgerufenen Services ApproverWebService

```
<wsdl:definitions targetNamespace="http://tempuri.org/services/loanapprover">

<!--
WSDL created by Apache Axis version: 1.2.1
Built on Apr 17, 2006 (04:00:53 EDT)
-->

<wsdl:types>
```

```
        <schema
        targetNamespace="http://loan_approval.samples.activebpel.org">
                <import
                namespace="http://schemas.xmlsoap.org/soap/encoding/"/>

                <complexType name="loanProcessFault">
                        <sequence/>
                </complexType>
        </schema>
</wsdl:types>

<wsdl:message name="approveResponse">
        <wsdl:part name="approvalMessage" type="xsd:string"/>
</wsdl:message>

<wsdl:message name="approveRequest">
        <wsdl:part name="firstName" type="xsd:string"/>
        <wsdl:part name="name" type="xsd:string"/>
        <wsdl:part name="amount" type="xsd:integer"/>
</wsdl:message>

<wsdl:message name="loanProcessFault">
        <wsdl:part name="fault" type="tns1:loanProcessFault"/>
</wsdl:message>

<wsdl:portType name="ApproverWebService">
        <wsdl:operation name="approve" parameterOrder="firstName name
        amount">
                <wsdl:input message="impl:approveRequest"
                name="approveRequest"/>
                <wsdl:output message="impl:approveResponse"
                name="approveResponse"/>
                <wsdl:fault message="impl:loanProcessFault"
                name="loanProcessFault"/>
        </wsdl:operation>
</wsdl:portType>

<wsdl:binding name="ApproverWebServiceSoapBinding"
type="impl:ApproverWebService">
        <wsdlsoap:binding style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http"/>

        <wsdl:operation name="approve">
                <wsdlsoap:operation soapAction=""/>
                <wsdl:input name="approveRequest">
                        <wsdlsoap:body
                        encodingStyle="http://schemas.xmlsoap.org/soap/encoding
                        /" namespace="http://tempuri.org/services/loanapprover"
                        use="encoded"/>
                </wsdl:input>
                <wsdl:output name="approveResponse">
                        <wsdlsoap:body
                        encodingStyle="http://schemas.xmlsoap.org/soap/encoding
                        /" namespace="http://tempuri.org/services/loanapprover"
                        use="encoded"/>
                </wsdl:output>
                <wsdl:fault name="loanProcessFault">
                        <wsdlsoap:fault
                        encodingStyle="http://schemas.xmlsoap.org/soap/encoding
                        /" name="loanProcessFault"
```

```
                    namespace="http://tempuri.org/services/loanapprover"
                    use="encoded"/>
                </wsdl:fault>
        </wsdl:operation>
</wsdl:binding>

<wsdl:service name="ApproverWebServiceService">
        <wsdl:port binding="impl:ApproverWebServiceSoapBinding"
        name="ApproverWebService">
                <wsdlsoap:address location="http://localhost:8080/active-
                bpel/services/ApproverWebService"/>
        </wsdl:port>
</wsdl:service>

</wsdl:definitions>
```

2.6. Konfigurationsdatei loan_approval_config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<rundata>

  <!-- Assessor answer: high, low, or FAULT
(Constants.MAGIC_FAULT_STRING) -->
  <!-- The BPEL logic flow skips the assesor completely if the request -->
  <!-- amount is greater than or equal to 10,000 -->
  <assessor>
    <risk-level>low</risk-level>
  </assessor>

  <!-- Approver answer: yes, no, or FAULT -->
  <approver>
    <accept>no</accept>
  </approver>

  <!-- The expected BPEL process answer depends upon risk-level and accept -
->
  <!-- Here, amount 0 represents any value less than 10,000 and -->
  <!-- 20000 represents any value greater than or equal to 10,000 -->

  <!-- amount   risk-level     accept      expected response -->
  <!-- ======   ==========     ======      ================= -->
  <!--      0   high           yes         yes -->
  <!--      0   high           no          no -->
  <!--      0   high           FAULT       FAULT -->
  <!--      0   low            (ignored)   approved -->
  <!--      0   FAULT          (ignored)   FAULT -->
  <!--  20000   (ignored)      yes         yes -->
  <!--  20000   (ignored)      no          no -->
  <!--  20000   (ignored)      FAULT       FAULT -->

</rundata>
```