„Komposition von Web-Services mit Hilfe von BPEL4WS“

Holger Rössig

Betreuer: Jochen Dinger

Tele-Seminar Web-Services

Universität Mannheim/Universität Karlsruhe

08.06.2004

Tele-Seminar Web-Services
- BPEL4WS -

Inhalt

1. Einleitung & Motivation
2. Komposition von Web-Services
3. BPEL4WS im Detail (Bsp. loanapprover)
   - 3.1 Elemente
     - WSDL Description
     - Partner
     - Activities
   - 3.2 Structure Activities & Links
   - 3.3 Scopes & Fehlerbehandlung
     - Scopes
     - Fehlerbehandlung & Kompensation
4. Ausblick

1. Einleitung & Motivation (1)

- Wieso noch ein Standard für Web-Services ?
- Sind WSDL, SOAP, etc. nicht ausreichend ?

➔ Das sind alles Standards um Web-Services zu beschreiben, zu erstellen, sie zu benutzen etc.

➔ BPEL4WS dagegen wird benutzt, um vorhandene Web-Services zu neuen umfangreicheren Web-Services zu verbinden.

➔ Es wird eingesetzt für :

- ➔ „Implementierung ausführbarer Business Prozesse“
- ➔ „Beschreibung abstrakter nicht-ausführbarer Prozesse“

➔ BPEL4WS benutzt WSDL, SOAP, UDDI, etc.

1. Einleitung & Motivation (2)

BSP:

Eine Fluggesellschaft bietet einen Web-Service zur Reservierung von Flügen, eine Hotelkette bietet einen Web-Service zur Buchung von Zimmern an.

Ein Reisebüro könnte nun mit BPEL4WS einen eigenen Web-Service anbieten, der eine Komposition der anderen beiden darstellt.

Formal :

"A process is viewed as a series of activities where an activity represents a well-defined business function. ...." *

* WEB SERVICE COMPONENTIZATION By Jian Yang

1. Einleitung & Motivation (3)

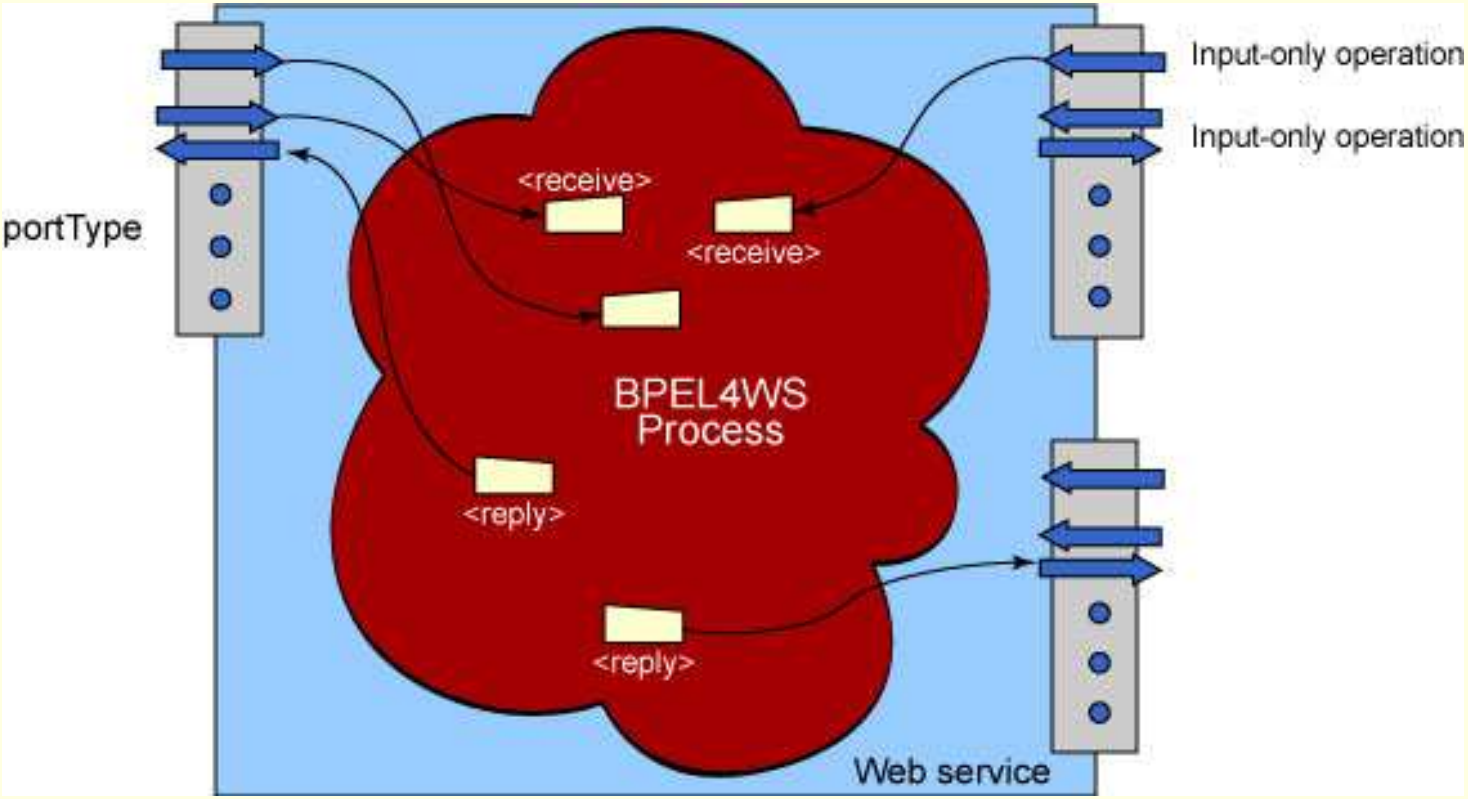
- BPEL4WS steht für „Business Process Execution Language for Web Services“
- Version 1.0 erschien im August 2002
- Version 1.1 (aktuell) erschien im März 2003
- Wurde von IBM, MS, und BEA initiiert und liegt inzwischen bei Oasis zur Standardisierung
- Aus 2 nicht kompatiblen Spezifikationen entstanden :
  - WSFL von IBM
  - XLANG von Microsoft
- Es kombiniert die Stärken von WSFL und XLANG :
  - WSFL : Graphen orientierte Prozesse
  - XLANG : strukturierte Konstrukte für Prozesse

2. Komposition von Web-Services (1)

- Die durch Web-Services angebotenen Online Dienste sind in ihren Möglichkeiten oft eingeschränkt bzw. nur für ein ganz spezielles Gebiet einsetzbar.
- ➔ Web-Services zu einem Prozess kombinieren.
- Mit BPEL4WS ist genau dieses möglich !
- Interface ist eine Kollektion von WSDL *portTypes.*
- ➔ Interface identisch mit dem „normaler“ Web-Services

2. Komposition von Web-Services (2)

Wie sieht ein BPEL4WS Prozess schematisch aus ?

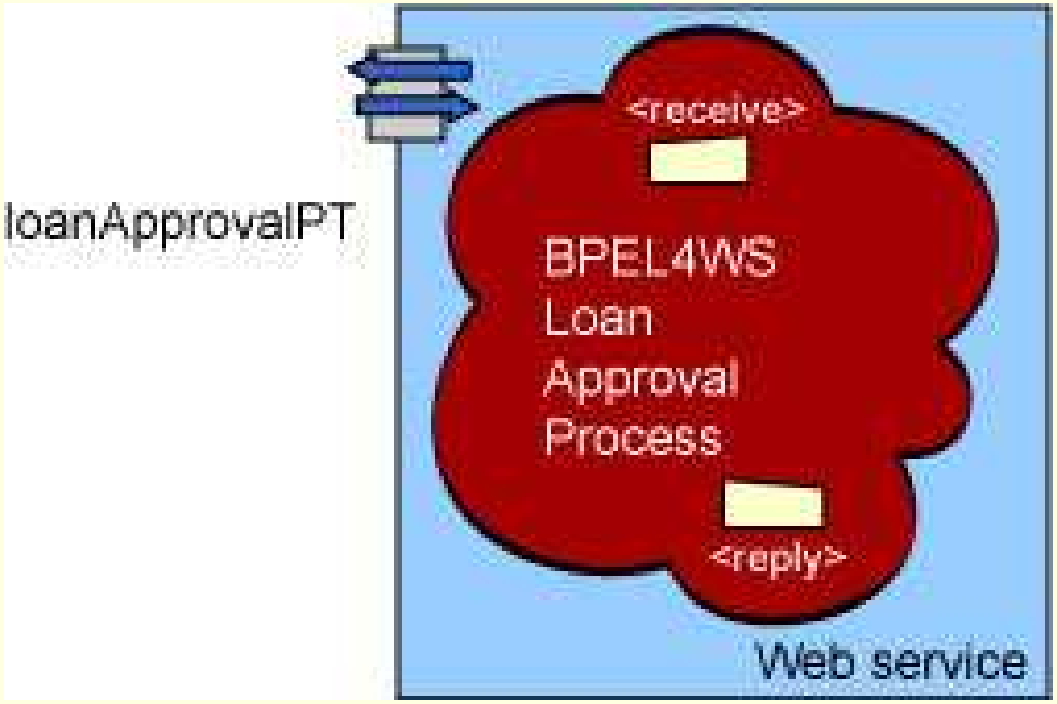


Quelle: IBM, „Understanding BPEL4WS, Part 1“

3.1 BPEL4WS im Detail

Einfaches Beispiel : „Loan Approval Process“

Ein Kunde kann mit diesem Web-Service überprüfen, ob sein gewünschter Kredit bei seiner Bank bewilligt wird.



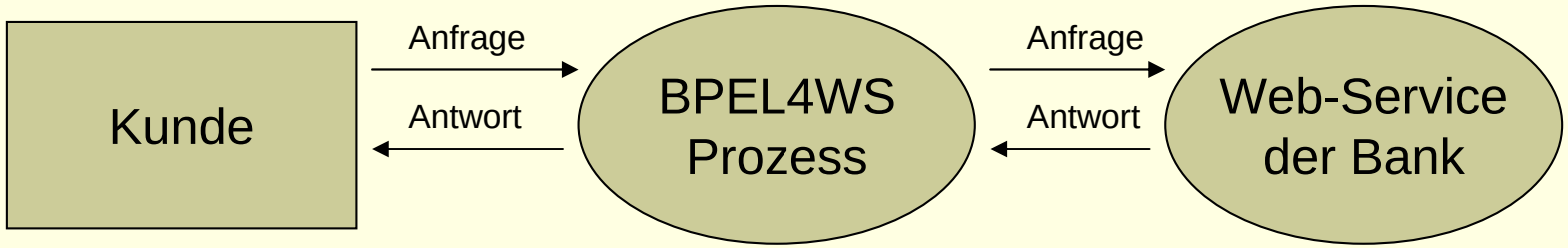
Quelle: IBM, „Learning BPEL4WS, Part 2“

3.1 BPEL4WS im Detail

Wir stellen uns folgende Situation vor :

- Kunde will die Bewilligung seines Kredits erfragen
- Web-Service bei der Bank kann diese Aufgabe übernehmen
- Wir schreiben einen BPEL4WS Prozess, der die Anfrage vom Kunden entgegen nimmt, diese an den Web-Service der Bank weiterleitet, die Antwort von diesem entgegen nimmt und an den Kunden zurücksendet.

3.1 BPEL4WS im Detail



3.1 BPEL4WS im Detail
WSDL Descriptions

- WSDL Definition für den Nachrichtenaustausch „loandefinition.wsdl“ :

```
<definitions targetNamespace="http://tempuri.org/services/loandefinitions"
                        xmlns:tns="http://tempuri.org/services/loandefinitions"
                        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                        xmlns="http://schemas.xmlsoap.org/wsdl/">
        <message name="creditInformationMessage">
                        <part name="firstName" type="xsd:string"/>
                        <part name="name" type="xsd:string"/>
                        <part name="amount" type="xsd:integer"/>
        </message>
        <message name="loanRequestErrorMessage">
                        <part name="errorCode" type="xsd:integer"/>
        </message>
</definitions>
```

3.1 BPEL4WS im Detail
WSDL Descriptions

- Interface des Web-Service der Bank „loanapprover.wsdl“ :

```
<definitions targetNamespace="http://tempuri.org/services/loanapprover"
                    xmlns:tns="http://tempuri.org/services/loanapprover"
                    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                    xmlns:loandef="http://tempuri.org/services/loandefinitions"
                    xmlns="http://schemas.xmlsoap.org/wsdl/">
        <import namespace="http://tempuri.org/services/loandefinitions"
                    location="http://localhost:8080/bpws-
                                samples/loanapproval/loandefinitions.wsdl"/>
        <message name="approvalMessage">
                    <part name="accept" type="xsd:string"/>
        </message>
```

3.1 BPEL4WS im Detail
WSDL Descriptions

```
<portType name="loanApprovalPT">
        <operation name="approve">
                <input message="loandef:creditInformationMessage"/>
                <output message="tns:approvalMessage"/>
                <fault name="loanProcessFault"
                        message="loandef:loanRequestErrorMessage"/>
        </operation>
</portType>
<binding ...> ... </binding>
<service name="LoanApprover">....</service>
</definitions>
```

3.1 BPEL4WS im Detail
WSDL Descriptions

- Jetzt müssen wir noch *serviceLinkType*s definieren, mit denen 2 Services verbunden werden.
- Diese referenzieren auf die jeweiligen *portType*s der 2 verbundenen Services.
- *portType*s sind die Schnittstelle über die Web-Services benutzt werden.
- Mit „role“ wird definiert, welche Rolle der Web-Service einnimmt.

3.1 BPEL4WS im Detail
WSDL Descriptions

- „loan-approval.wsdl“ :

```
<definitions targetNamespace="http://loans.org/wsdl/loan-approval"
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:slnk="http://schemas.xmlsoap.org/ws/2002/06/service-link/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:lns="http://loans.org/wsdl/loan-approval"
        xmlns:apns="http://tempuri.org/services/loanapprover">
    <import namespace="http://tempuri.org/services/loanapprover"
        location="http://localhost:8080/bpws-
                samples/loanapproval/loanapprover.wsdl"/>
    <import namespace="http://tempuri.org/services/loandefinitions"
        location="http://localhost:8080/bpws-
                samples/loanapproval/loandefinitions.wsdl"/>
    <slnk:serviceLinkType name="loanApprovalLinkType">
        <slnk:role name="approver">
                <portType name="apns:loanApprovalPT"/>
        </slnk:role>
    </slnk:serviceLinkType>

    <service name="loanapprovalServiceBP"/>
</definitions>
```

3.1 BPEL4WS im Detail

<process>-Tag

- Wir beginnen nun unseren BPEL4WS Prozess mit dem <process> Tag.
- In diesen binden wir *namespaces* ein, die es uns erlauben zu den zuvor erstellten WSDL Definitionen zu gelangen :

```
<process      name="loanApprovalProcess"
              targetNamespace="http://acme.com/simpleloanprocessing"
              xmlns="http://schemas.xmlsoap.org/ws/2002/07/business-process/"
              xmlns:lns="http://loans.org/wsdl/loan-approval"
              xmlns:loandef="http://tempuri.org/services/loandefinitions"

              xmlns:apns="http://tempuri.org/services/loanapprover">
```

3.1 BPEL4WS im Detail

Partner

- Als nächstes müssen wir definieren, welche Partner involviert sind. Dies sind in unserem Beispiel der Kunde und der Web-Service der Bank.

```
<partners>
        <partner name="customer"
                serviceLinkType="lns:loanApproveLinkType"
                myRole="approver"/>
        <partner name="approver"
                serviceLinkType="lns:loanApprovalLinkType"
                partnerRole="approver"/>
</partners>
```

- „myRole“ gibt an, welche Rolle ich selbst, also unser BPEL4WS Prozess spielt.
- „partnerRole“ gibt an, welche Rolle der Partner-Service, also der loanapprover bei der Bank spielt.

3.1 BPEL4WS im Detail

Containers

- Daten können nur in Containern gelesen und geschrieben werden
- Wir brauchen nun einen Container in dem wir die vom Kunden erhaltene Nachricht ablegen um sie dann weiterzuleiten.
- Und wir brauchen einen Container in den wir die von der Bank erhaltene Antwort ablegen um sie dann zum Kunden zu leiten.

```
<containers>
    <container   name="request"
                 messageType="loandef:CreditInformationMessage"/>
    <container   name="approvalInfo„
                 messageType="apns:approvalMessage"/>
</containers>
```

3.1 BPEL4WS im Detail

Activities

- Ein Prozess aus nur einer oder aus mehreren activities bestehen.
- In diesem Fall verwenden wir nur eine Haupt-activity nämlich <sequence>, welche dafür sorgt, dass alle dort hineingeschachtelten activities sequentiell nacheinander ausgeführt werden.
- Activities wie <sequence>, <flow>, <switch>, <while>, <pick> und <scope> werden als structure activities bezeichnet, da sie den Ablauf und die Struktur des Prozesses bestimmen.
- In diese structure activities hinein schachtelt man primitive activities wie <receive>, <invoke>, <reply>, <assign>, <terminate> oder <empty>.

3.1 BPEL4WS im Detail
Activities (<receive>)

```
<sequence>
    <receive      name="receive1"
                  partner="customer"
                  portType="apns:loanApprovalPT"
                  operation="approve"
                  container="request"
                  createInstance="yes">

    </receive>
```

- <receive> nimmt die Message vom Partner „customer“ entgegen und steckt sie in den container mit dem namen „request“.

3.1 BPEL4WS im Detail
Activities (<invoke>)

```
<invoke   name="invokeapprover"
          partner="approver"
          portType="apns:loanApprovalPT"
          operation="approve"
          inputContainer="request"
          outputContainer="approvalInfo">
</invoke>
```

- <invoke> nimmt die Message aus dem input container mit dem namen „request“, schickt sie an den Partner „approver“ und steckt die erhaltene Antwort in den output container.

3.1 BPEL4WS im Detail
Activities (<reply>)

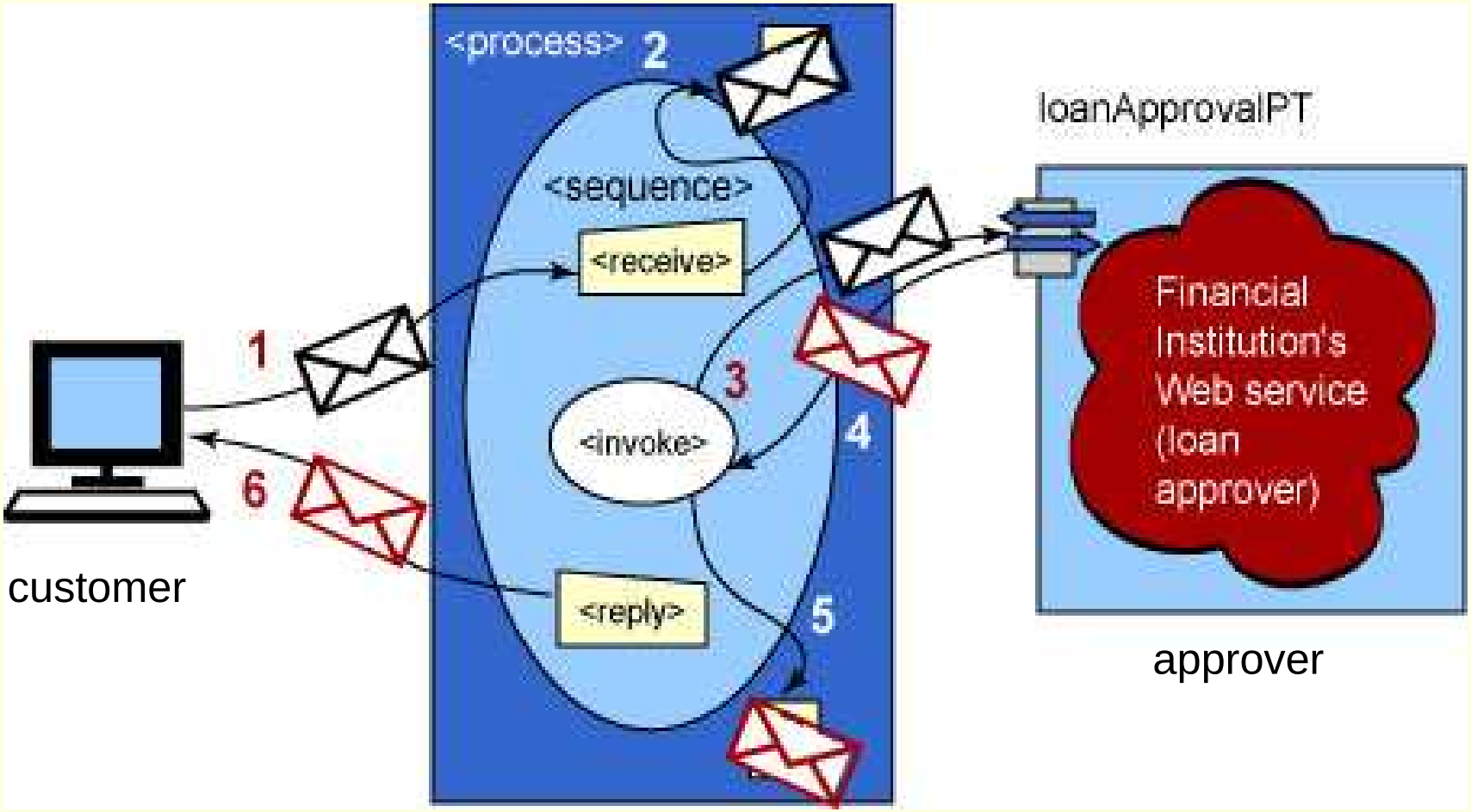
```
        <reply     name="reply"
                   partner="customer"
                   portType="apns:loanApprovalPT"
                   operation="approve"
                   container="approvalInfo">
        </reply>
    </sequence>
</process>
```

- <reply> nimmt die Message (die Antwort von der Bank) aus dem container „approvalInfo“ der bei <invoke> als output container diente, und schickt sie an den Partner „customer“

3.1 BPEL4WS im Detail

Übersicht

- Hier der ganze Ablauf noch einmal grafisch dargestellt :



3.2 Structure Activities & Links sequentielle Ausfürhrung

- Für die sequentielle Ausführung 2er activities gibt es 2 Möglichkeiten :

1. Mit <sequence> : (aus XLANG)

```
<sequence>
        activityA
        activityB
</sequence>
```

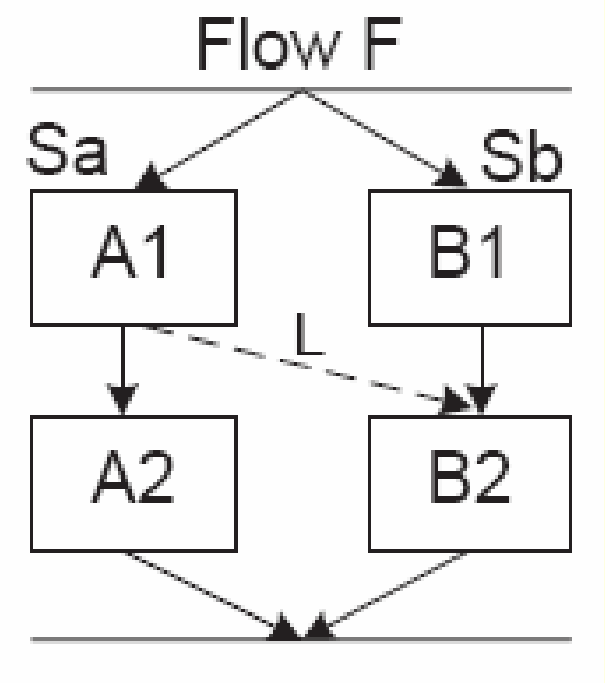
3.2 Structure Activities & Links sequentielle Ausführung

2. Mit control links : (aus WSFL)

```
<flow>
    <links>
        <link name="L"/>
    </links>
    activityA
        <source linkName="L"/> ...
    activityB
        <target linkName="L"/> ...
</flow>
```

3.2 Structure Activities & Links Synchronisation

- Folgende Situation :
  - A1 und B1 laufen parallel
  - A2 startet wenn A1 beendet hat
  - B2 startet wenn B1 beendet hat unter der Bedingung dass A1 auch schon beendet ist.



Quelle: Wohed, Aalst, Dumas, Hofstede: „Pattern Based Analysis of BPEL4WS

3.2 Structure Activities & Links Synchronisation

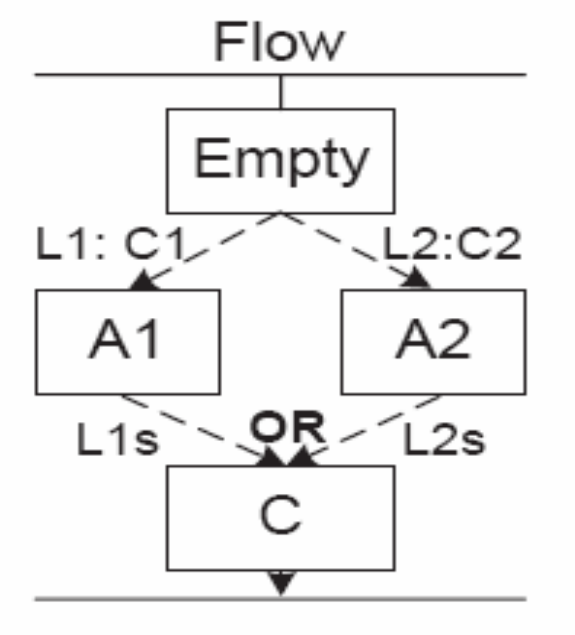
- Lösung : Durch Kombination der Techniken aus XLANG und WSFL

```
<flow name="F">
    <links>
        <link name="L"/>
    </links>
    <sequence name="Sa">
        activityA1
        <source linkName="L"/>
        activityA2
    </sequence>
    <sequence name="Sb">
        activityB1
        activityB2
        <target linkName="L"/>
    </sequence>
 </flow>
```

3.2 Structure Activities & Links

Switch

- Folgende Situation :
  - Prozess tut nichts bis Bedingung C1 oder Bedingung C2 eintritt
  - Bei Bedingung C1 wird Aktivität A1 ausgeführt, anschließend Aktivität C
  - Bei Bedingung C2 wird Aktivität A2 ausgeführt, anschließend Aktivität C



3.2 Structure Activities & Links

Switch

- Lösung mit <switch> (aus XLANG) :

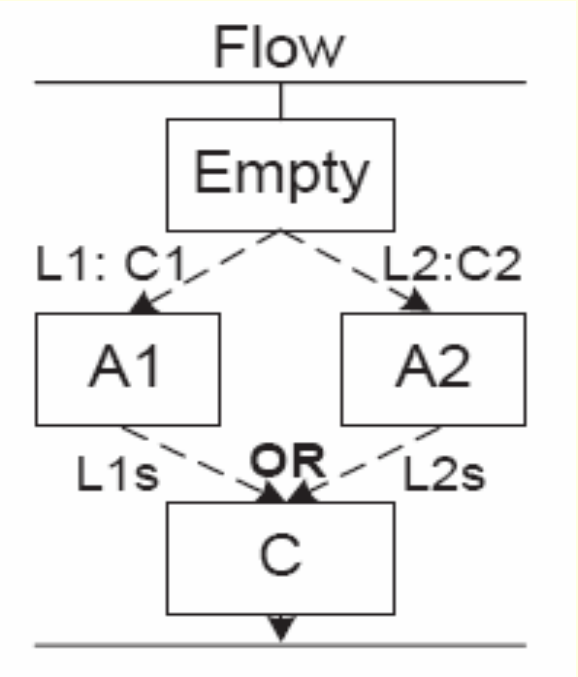
```
<switch>
        <case condition="C1">
                activityA1
        </case>
        <case condition="C2">
                activityA2
        </case>
</switch>
activityC
```

3.2 Structure Activities & Links

Switch

- Lösung über control links (WSFL) :

```
<flow>
    <links>
        <link name="L1"/>
        <link name="L2"/>
        <link name="L1s"/>
        <link name="L2s"/>
    </links>
    <empty>
        <source linkName="L1"
        transitionCondition="C1"/>
        <source linkName="L2"
        transitionCondition="C2"/>
    </empty>
    activityA1
        <target linkName="L1">
        <source linkName="L1s">
    activityA2
        <target linkName="L2">
        <source linkName="L2s">
    activityC
        joinCondition="L1s OR L2s"
        <target linkName="L1s">
        <target linkName="L2s"> ...
</flow>
```



Quelle: Wohed, Aalst, Dumas, Hofstede: „Pattern Based Analysis of BPEL4WS

3.3 Scopes & Fehlerbehandlung

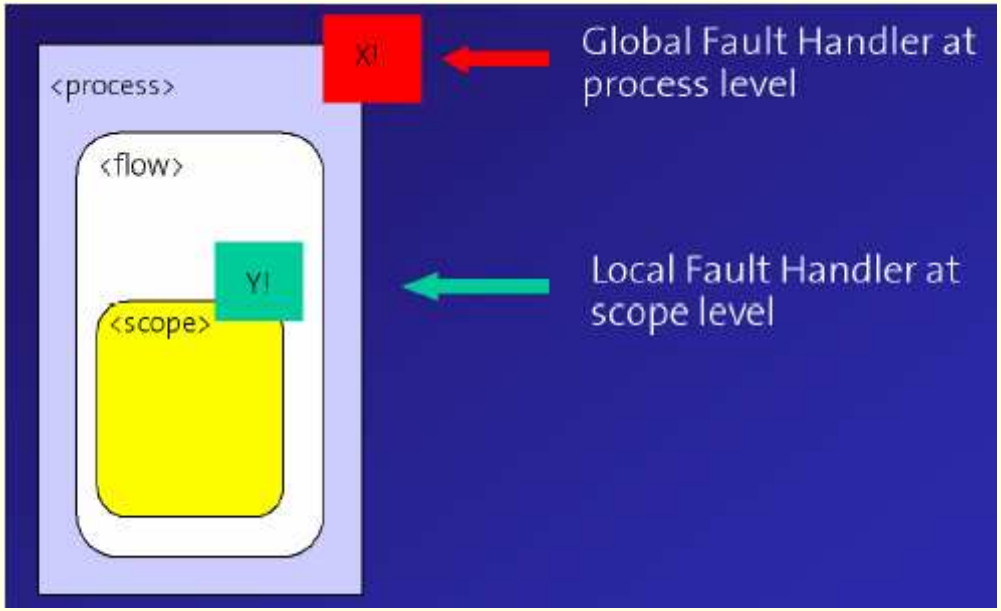
Scopes

- Mit einem Scope kann man mehrere Aktivitäten zusammenfassen, sie in einen gemeinsamen Kontext bringen.
- Variablen die innerhalb eines Scopes definiert sind, sind auch nur innerhalb dieses Scopes sichtbar.
- Man kann auch für alle Aktivitäten in einem Scope einen Faulthandler definieren, der dann evtl. entstehende Fehler in dem Scope abfängt.
- Jeder Scope hat eine Hauptaktivität, die dessen Verhalten bestimmt.
- Bsp. :

```
<scope>
        <flow> …. </flow>
</scope>
```

3.3 Scopes & Fehlerbehandlung
Faulthandler

- In BPEL4WS gibt es 3 Ursachen für Fehler :
  - Aufgerufener Web-Sevice verursacht Fehler
  - *throw*-Aktivität wurde aufgerufen
  - Interner Fehler tritt auf, z.B. typemissmatch
- Faulthandler können lokal für Scopes oder auch global definiert werden :



Quelle: Mikijevic, Maslic, Maier : „BPEL4WS“

3.3 Scopes & Fehlerbehandlung
Faulthandler

- Beispiel : Definition eines Faulthandlers (lokal)

```
<scope>
    <faultHandlers>
        <catch    faultName="lns:loanProcessFault"
                  faultVariable="error">
        <sequence name="fault-sequence">
                  <reply    partner="customer"
                            portType="lns:loanApprovalPT"
                            operation="obtain"
                            variable="approvalInfo"
                            faultName="lns:loanProcessFault"/>
        </sequence>
        </catch>
    </faultHandlers>
    <flow>
    -> insert some activities here
    </flow>
</scope>
```

3.3 Scopes & Fehlerbehandlung
Kompensation

- Warum ist Kompensation nötig ?
- Wenn ein BPEL4WS Prozess z.B. einen Flug und ein Hotelzimmer buchen soll, kann es passieren, dass der Flug zuerst erfolgreich gebucht wird, dann aber beim versuch ein Hotelzimmer zu buchen, der Web-Service des Hotels einen Fehler zurückgibst, da keine Zimmer mehr frei sind.
- ➔ Dann muss auch die Buchung des Flugs rückgängig gemacht werden.
- ➔ Die Schritte die zum Rückgängigmachen nötig sind können in so genannten „compensation handlern“ zusammengefasst und auf Scope-Ebene definiert werden.
- 2 Typen der Kompensierung werden unterschieden :
  - Explizite Kompensation
  - Implizite Kompensation

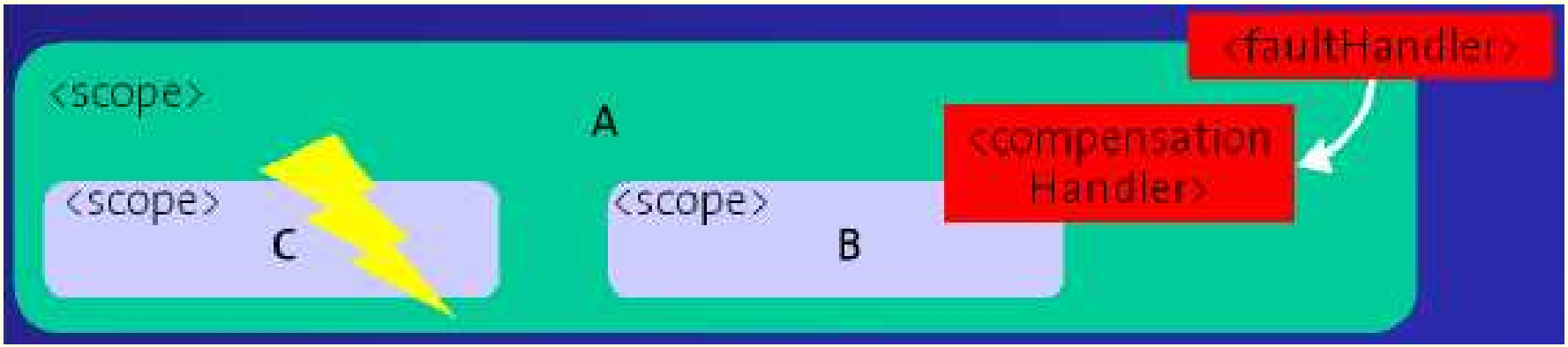
3.3 Scopes & Fehlerbehandlung
Explizite Kompensation

- Wenn die „compensate“ Aktivität aufgerufen wird, z.B. innerhalb eines Faulthandlers auf Scope-Ebene :

Quelle: Mikijevic, Maslic, Maier : „BPEL4WS“

3.3 Scopes & Fehlerbehandlung
Explizite Kompensation

- Wenn ein Fehler abgefangen und weitergeleitet wird :



Quelle: Mikijevic, Maslic, Maier : „BPEL4WS“

- B schließt erfolgreich ab, C wirft Fehler aus.
- C hat keinen Faulthandler, daher übernimmt der Faulthandler des umschließenden Scopes den Fehler.
- Dieser startet Kompensierung von B

4. Ausblick

- Gibt es noch andere Ansätze zur Komposition von Web-Services ?
- BPEL4WS wird von OASIS zum Standard entwickelt
- W3C hat als „Konkurrenz“ WSCI („whisky“) (Web Service Choreography Interface)
- "The two specifications may not be rivals. They could address different (but related) problem spaces." (Martin Chapman)

4. Ausblick

- „If by *orchestration* you mean there is a central controller and the business process is run by that controller, and by *choreography* you mean there is no controller and the business process is run in a distributed way by each of the participants, then I think I am right in saying that BPEL4WS does not cover choreography. In this case, the WS-Choreography working group will be working on a separate but related problem space.“

  (Martin Chapman)

4. Ausblick

- Zukunftsaussichten ?
- BPEL4WS ➔ gute Aussichten da große Unternehmen wie Microsoft, IBM, BEA und inzwischen auch SAP
- WSCI ➔ nicht so gute Aussichten, aber dennoch Daseinsberechtigung weil BPEL4WS choreography nicht beinhaltet
- ➔ Vielleicht irgendwann Kombination aus beiden !

4. Ausblick

- In welchen Produkten kann man BPEL4WS finden ?

- MS Biztalk Server 2004
- IBM WebSphere Business Integration

- Implementationen :

- IBM BPWS4J (BPEL Java Runtime)
- Collaxa Orchestrastion Server
- ChoreoServer von OpenStorm Software