Vorlesung

Methodische Grundlagen des Software-Engineering

im Sommersemester 2014

Prof. Dr. Jan Jürjens

TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

Teil 1.4: Workflow-Automatisierung

v. 30.04.2014

1.4 Workflow-Automatisierung

**[inkl. Beiträge von
Prof. Dr. Frank Leyman (Universität Stuttgart)]**

Literatur:

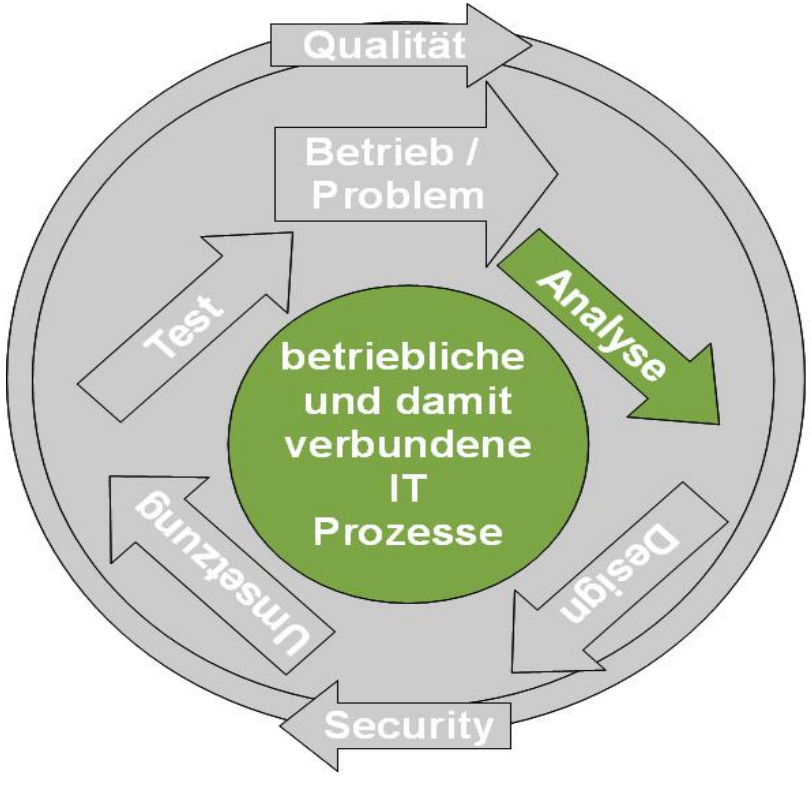
[LLN11] T. van Lessen, D. Lübke, J. Nitsche: Geschäftsprozesse automatisieren mit BPEL. dpunkt.verlag, 2011, 1. Auflage. Unibibliothek (6 Exemplare): http://www.ub.tu-dortmund.de/katalog/titel/1372568

- Kapitel 5

Bei Engpässen in der Ausleihe kann **Kopiervorlage** der relevanten Ausschnitte zur Verfügung gestellt werden.

Workflow-Automatisierung: Einordnung

- Geschäftsprozessmodellierung
  - Grundlagen Geschäftsprozesse
  - Ereignisgesteuerte Prozessketten (EPKs)
  - Einführung in die BPMN 2.0
  - Workflow-Management-Systeme
  - Workflow-Automatisierung
- Process Mining
- Modellbasierte Entwicklung sicherer Software



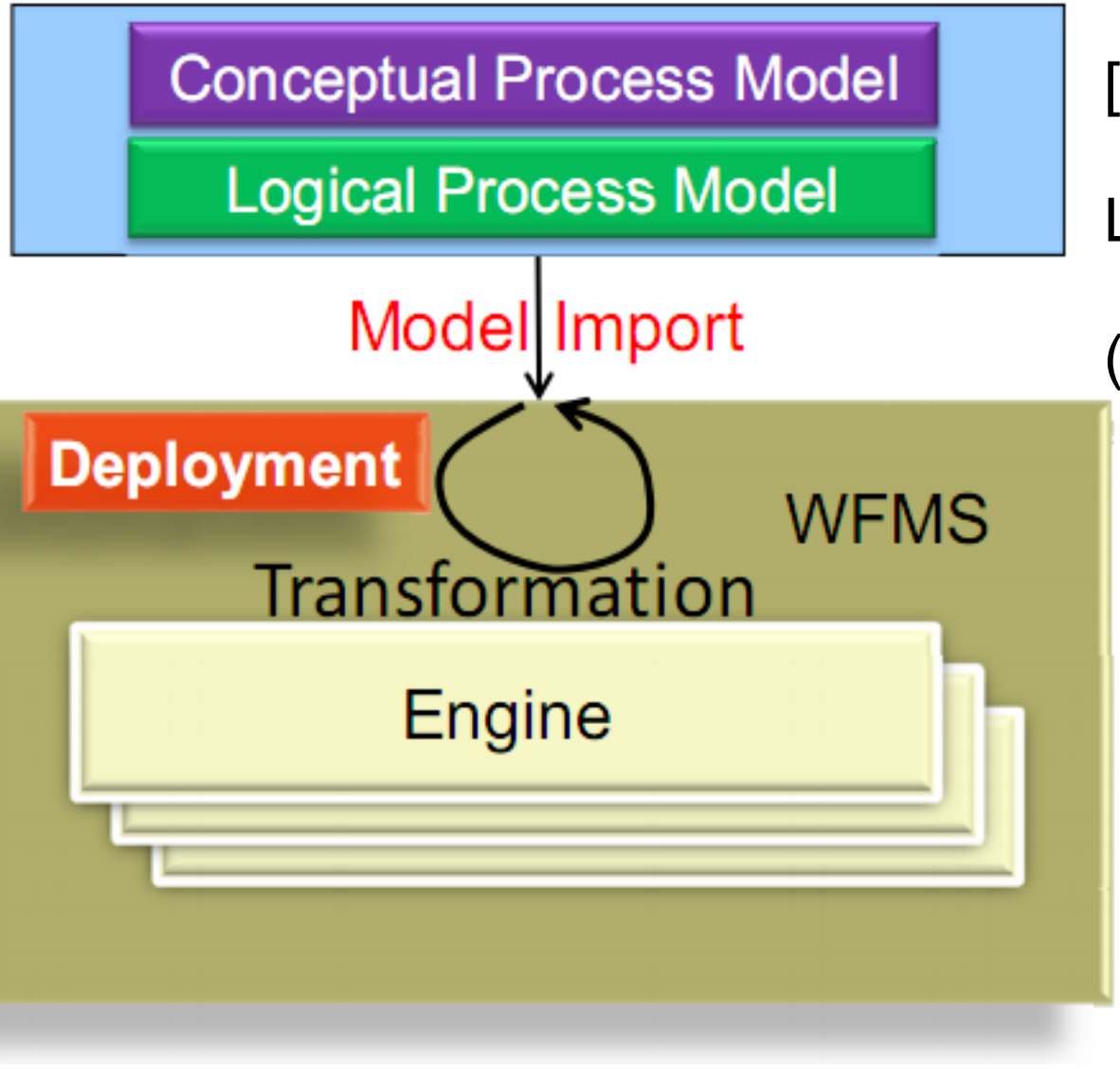
Einleitung
Workflow-Automatisierung

- **Letzter Abschnitt**: Workflow-Management-Systeme:
  Unterstützen Ausführung von Workflows.
  - Insbesondere **Workflow-Engines**.
- **Dieser Abschnitt**: „Workflow-Automatisierung“:
  Ausführung der Workflows.
  - Allgemeine Grundlagen
  - Konkretes Beispiel: Übersetzung von BPMN-Modellen in Business Process Execution Language (BPEL) zur Ausführung.

Workflow-Automatisierung: Überblick

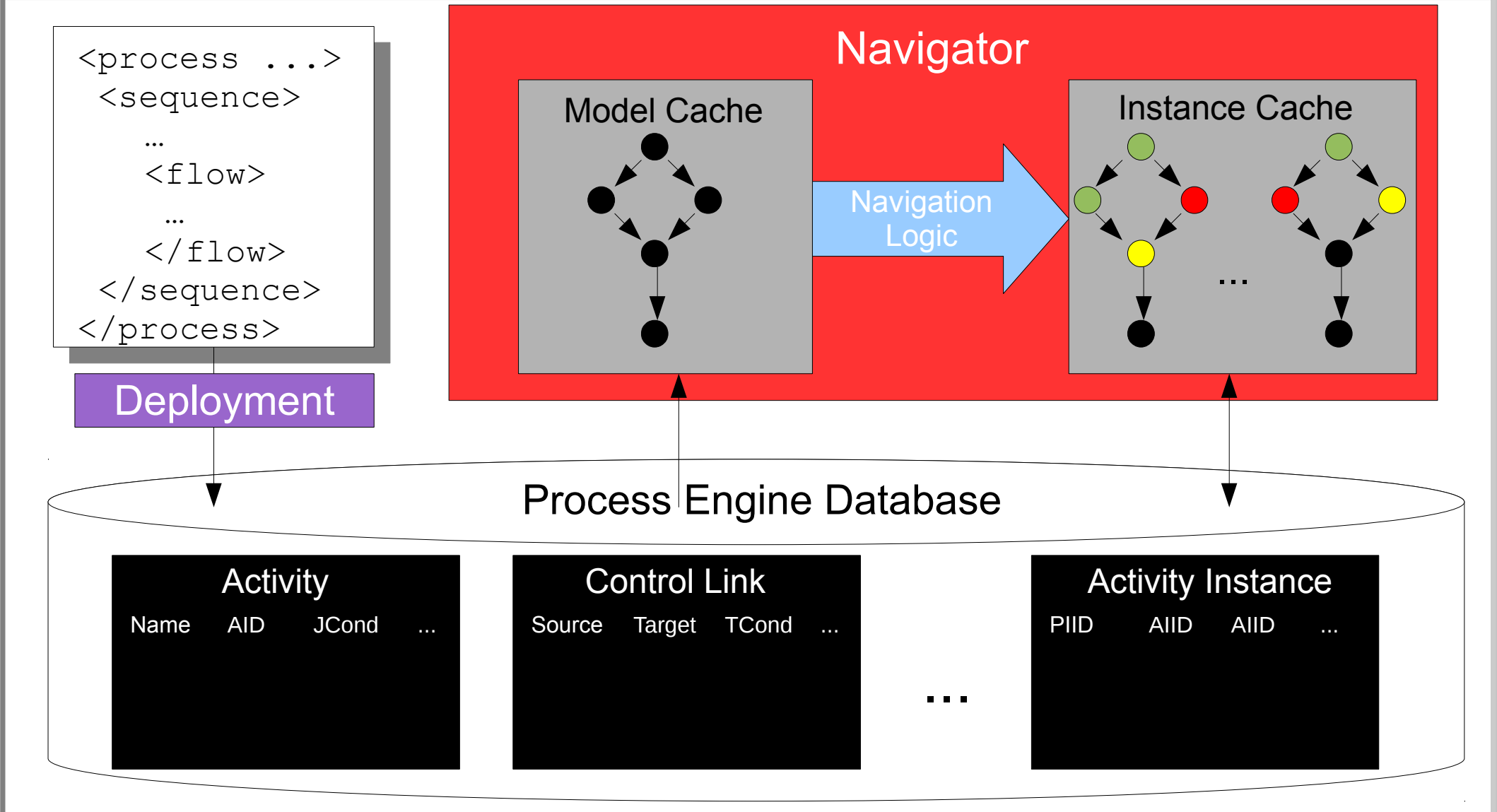
- Grundlagen **Modell-Deployment**
  - Natives Metamodell einer Workflow-Engine und Modelltransformation
- **BPEL** und **Transformation**: BPMN 2 nach BPEL 2
  - Kurz-Einführung BPEL, Aktivitäten
  - Ereignisse
  - Strukturierte Aktivitäten

Vom Prozess-Modell zur Workflow-Engine



[Conceptual:
z.B. BPMN
Logical:
z.B. BPEL
(vgl. T1.2 F18)]

Modell-Deployment: Modell zur Ausführung bringen



Deployment vs. Metamodell

Deployment: Prozessmodell produktiv schalten.

- z.B. bereit für die Ausführung machen.

Zentrale Rolle dabei: Speicherformat für die Modelle.

Definiert als **Metamodell**: Definition einer Modellierungssprache, die selber als Modell gegeben wird.

- Mehr Informationen: Vorlesung „Softwarekonstruktion“, Teil 1.2: „Modellbasierte Softwareentwicklung“:

http://www-secse.cs.tu-dortmund.de/secse/pages/teaching/ws13-14/swk

Natives Meta-Modell einer Workflow-Engine

- **BPEL-** (bzw. **BPMN-**) **Engine:** Workflow-Engine, die BPEL- (bzw. BPMN-) Prozessmodelle importieren kann.

„**Natives Metamodell**“: Metamodell zur Definition des internen Modell-Speicherformats einer Workflow-Engine.

- **„Native“ BPEL- (bzw. BPMN-) Engines**: BPEL (bzw. BPMN) ist internes („natives“) Metamodell.

Natives Metamodell…

… ist unterstützt in **Datenbank des WFMS:**

- Datenbankschema enthält Instanzen des Metamodellkonstrukts.

… ist unterstützt in **Zustandsmodell des WFMS:**

- Alle Metamodellkonstrukte haben Menge von Zuständen und Transitionen.
- Zustandsmodell: im Monitoringmodell und Protokoll reflektiert.

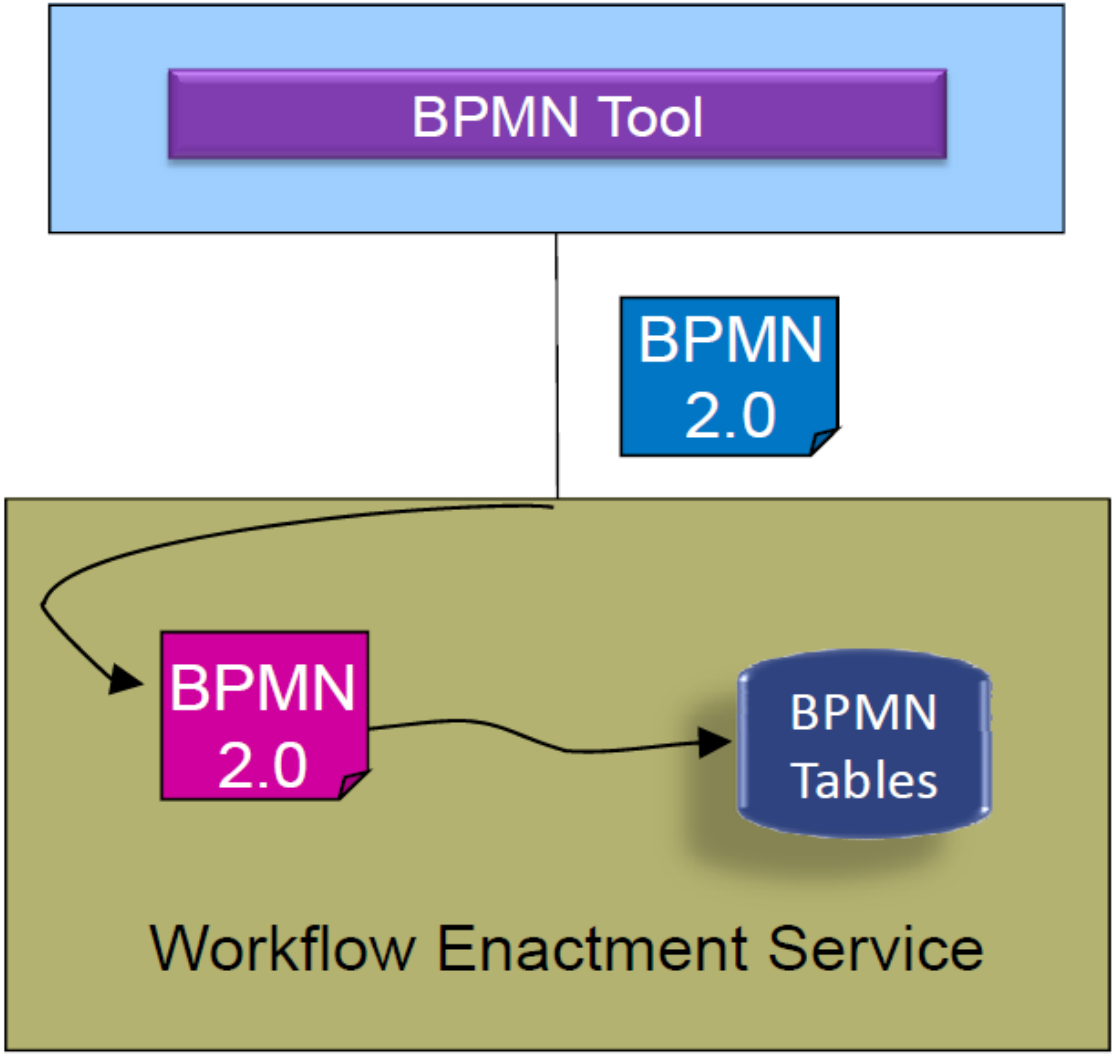
… ist im **Navigator des WFMS** implementiert:

- Navigator versteht direkt jedes Metamodellkonstrukt, dessen Zustände, dessen gültige Transitionen und die Relation zwischen den Zuständen verschiedener Artefakte.

Native BPMN 2.0 Engines

Native Unterstützung von **BPMN 2.0** (d.h. ohne Transformation nach BPEL).

BPMN Tables: Interne Speicherung des BPMN-Modells.



Speicherung von BPMN-2.0-Modellen

- **XML-Schema** für BPMN 2.0:
  Speichern und Weiterverarbeiten der Modelle.
- Enthält für **Ausführung** relevante Informationen.
- Zugehörige syntaktische Details in BPMN 2.0 Spezifikation durch **UML-Klassendiagramme** oder **XML-Schema-Definitionen** definiert.

Natives Metamodell vs. Modell-Transformation

Für Deployment importiertes Modell kann in **anderem Metamodell** spezifiziert sein, als natives Metamodell des WFMS.

- Muss dann in dieses umgewandelt werden (**Transformation** während des Modell-Deployments).

(Beispiele siehe folgende Folien.)

Motivation dafür: Wichtiger als **natives Metamodell** einer Prozessengine ist ihre **Stabilität, Effizient, Skalierbarkeit**:

- Anbieter von aktuellen BPEL Engines haben in nicht-funktionale Eigenschaften ihrer Engines investiert.
- Können BPMN 2.0 (Business Process Model and Notation) Engines mit ähnlichen nicht-funktionalen Eigenschaften anbieten.

Nativ vs. Transformation: DBMS-Analogie

DBMS-Analogie:

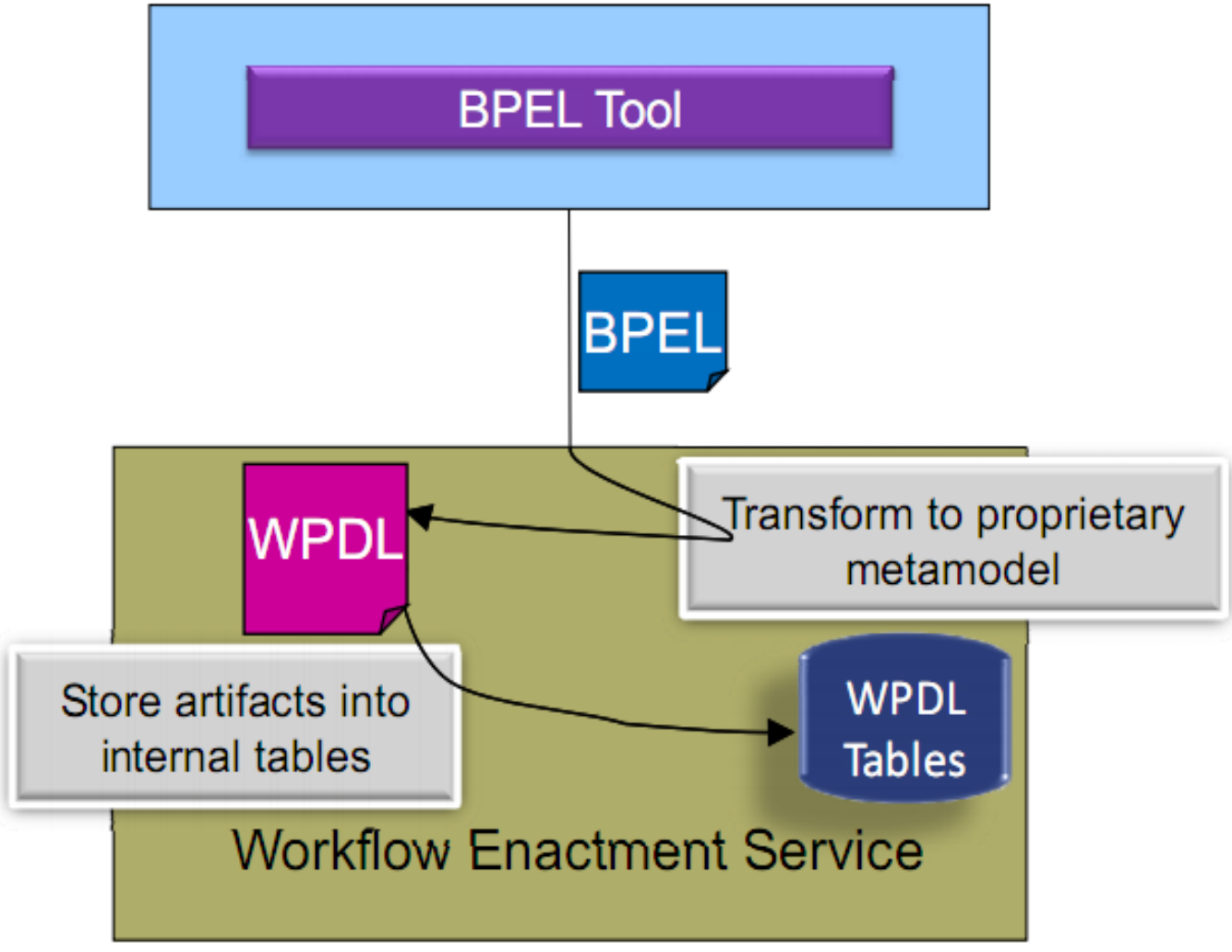
- Objekt-orientierte Datenbanksysteme (OODBMS): Natives Objektmodell, aber Mängel bei Stabilität, Effizienz, Skalierbarkeit.
- Etablierte Relationale DBMS unterstützten auch Schlüsselkonstrukte des Objektparadigmas bei höherer Stabilität, Effizienz, Skalierbarkeit.

→ OODBMS **kein „Mainstream“** mehr.

Beispiel-Transformation: BPEL => WPDL

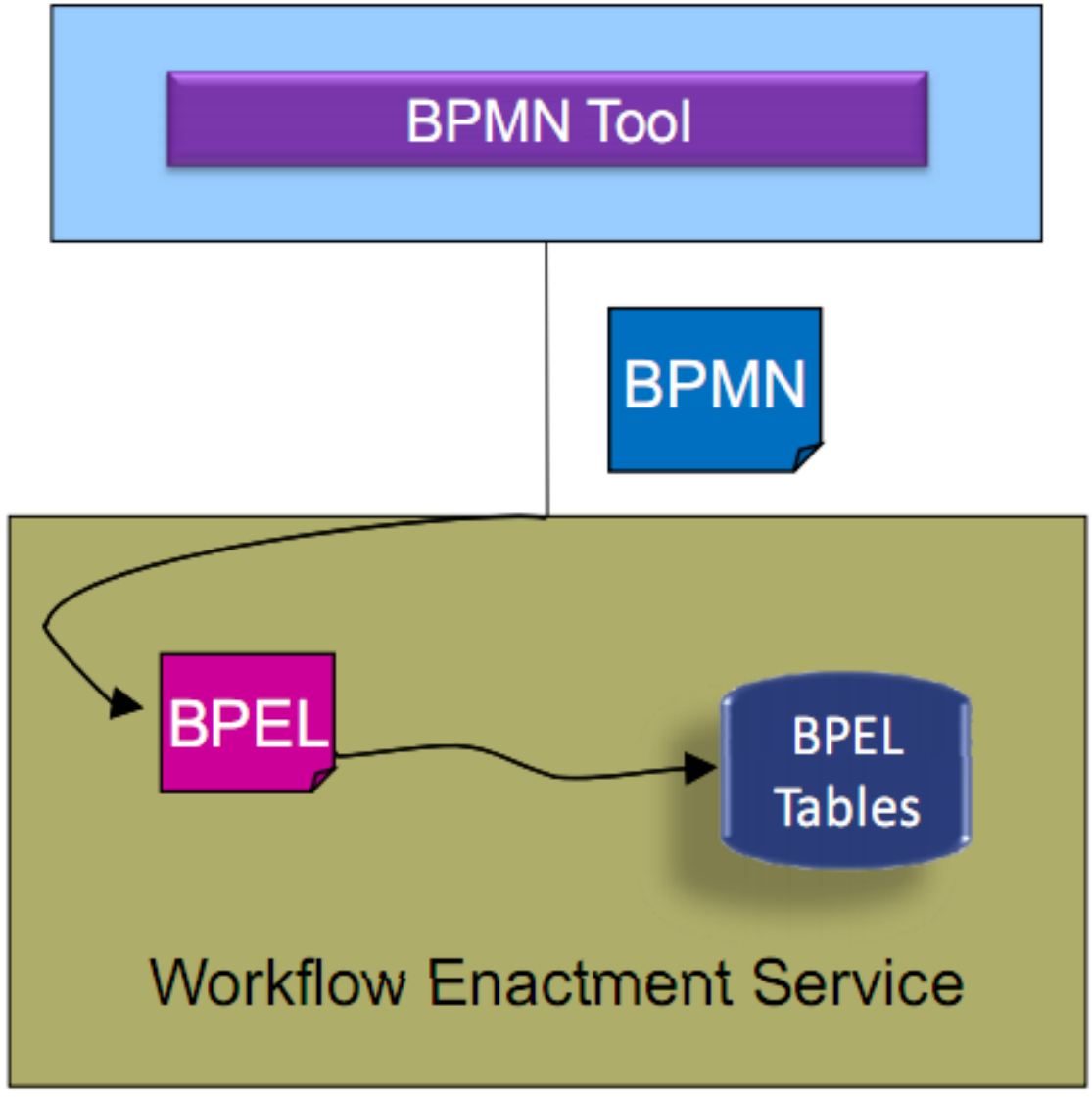
Native **WPDL**[1]-Engine kann **BPEL**-Modell nach zugehöriger Transformation ausführen.

[1] **Workflow Process Definition Language** (WPDL): Vorläufer der XML Process Definition Language (XPDL)



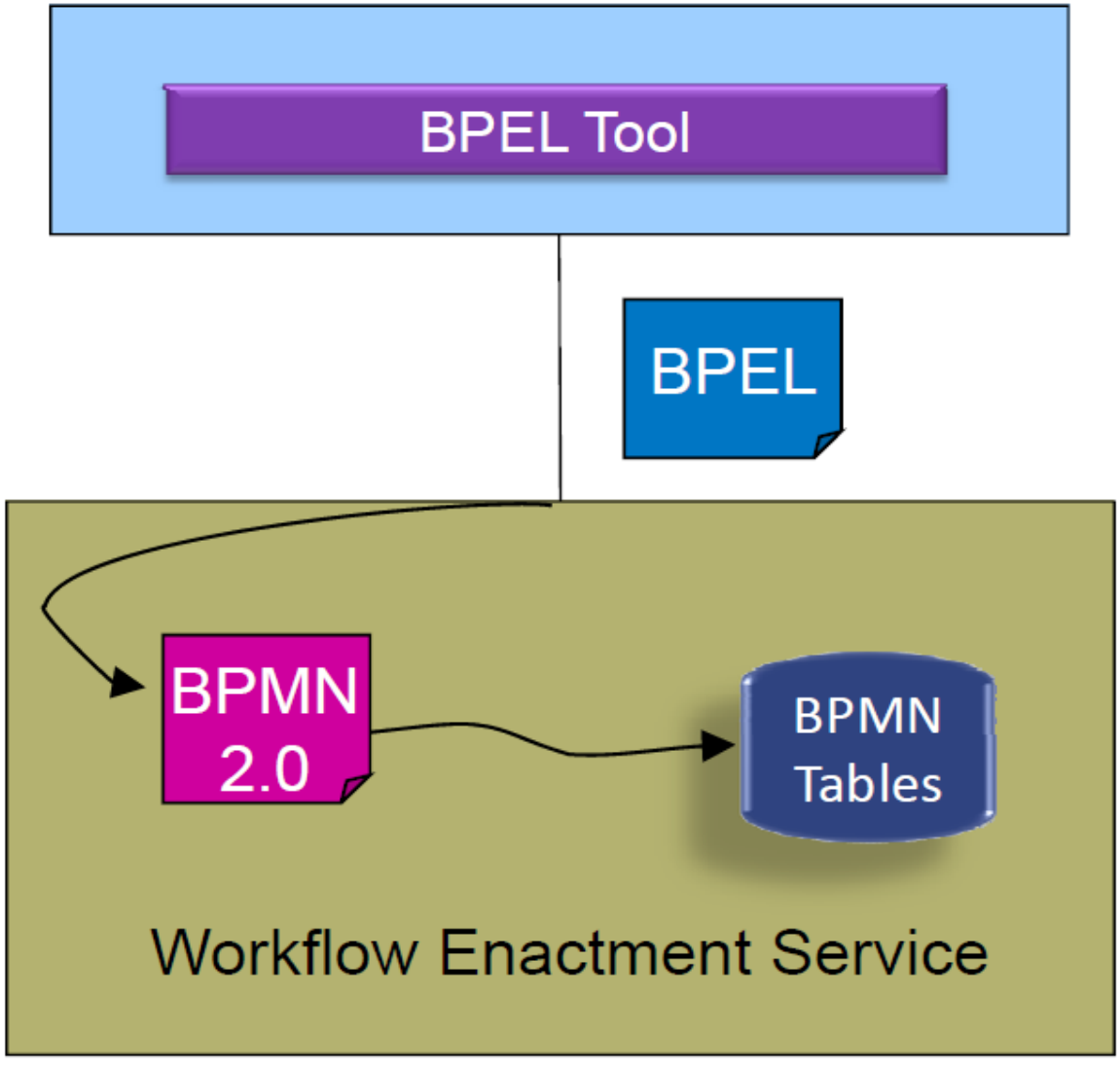
Beispiel-Transformation: BPMN => BPEL

Native **BPEL**-Engine kann **BPMN**-Modell nach zugehöriger Transformation ausführen.



Beispiel-Transformation: BPEL => BPMN

Native **BPMN 2.0**-Engine kann nach Transformation auch **BPEL** unterstützen.



Diskussion: Probleme bei Modelltransformation

Herausforderungen bei **Modelltransformation** zwischen GP-Modellierungsnotationen ? (Vgl. z.B. BPMN, EPKs.)

Diskussion: Probleme bei Modelltransformation

Herausforderungen bei **Modelltransformation** zwischen GP-Modellierungsnotationen ? (Vgl. z.B. BPMN, EPKs.)

Antwort: Transformationen nicht immer **verhaltensbewahrend**:

Semantisches Verhalten:

Quell-/Ziel-Modellelemente nicht mit derselben Semantik.
Muss Quell-Modellelemente in Zielmodell „nachbauen“.
Oft nicht perfekt möglich bzw. praktikabel.

- z.B.: BPEL's Exception-Verhalten schwer zu emulieren

Operatives Verhalten:

Durch diese Emulation wird transformiertes Modell weniger effizient ausgeführt.

- z.B.: Unterstützung eines FDL-Datenflusses in BPEL schwerfällig.

Diskussion: Probleme bei Austausch des WfMS

Welche Probleme können sich aus diesen Problemen bzgl. Modelltransformationen bei Austausch des WfMS ergeben ?

Diskussion: Probleme bei Austausch des WfMS

Welche Probleme können sich aus diesen Problemen bzgl. Modelltransformationen bei Austausch des WfMS ergeben ?

Antwort:

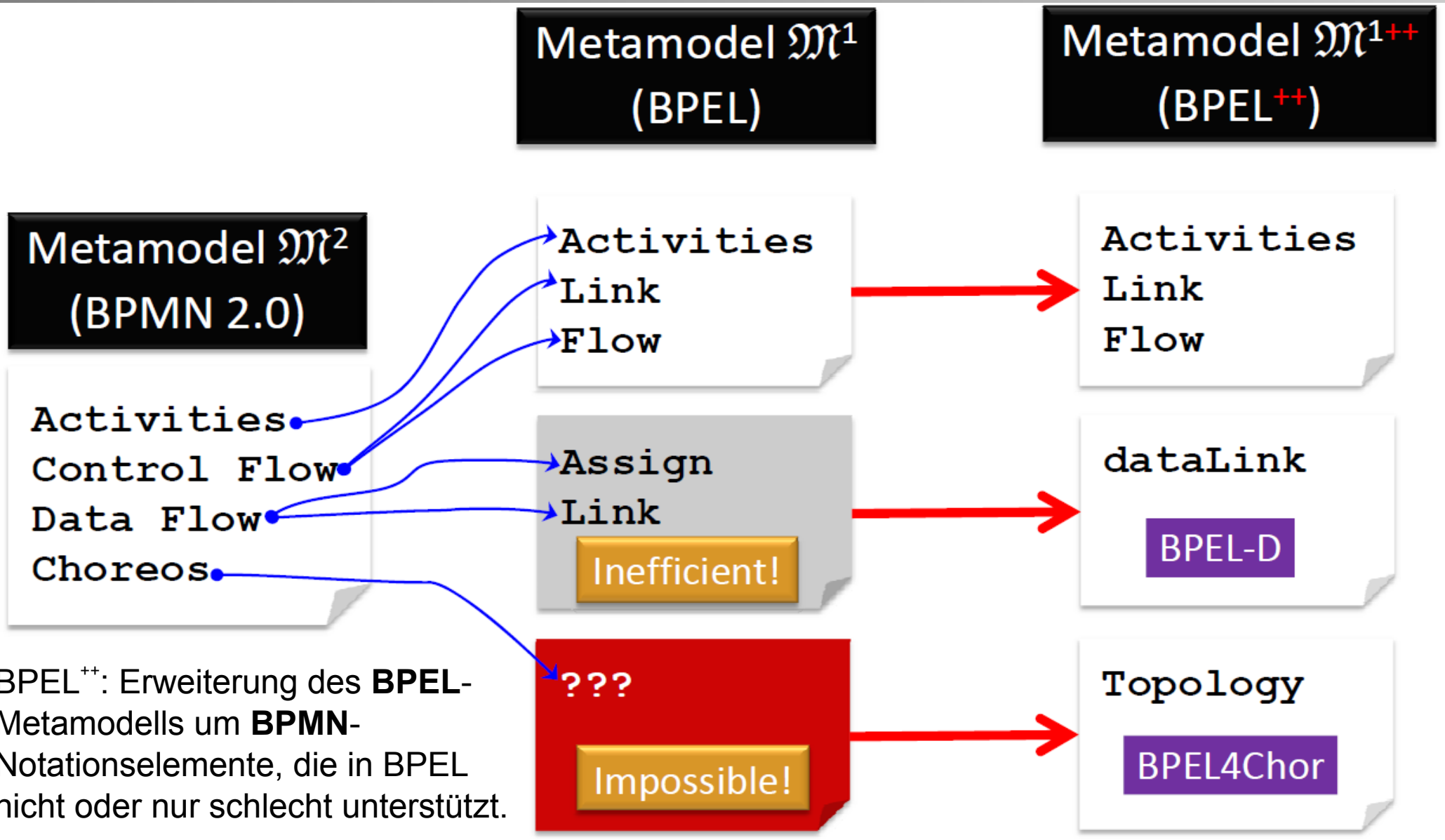
- WfMS mit verschiedenen nativen Metamodellen setzen verschiedene Modelltransformationen ein.
- Sind i.A. nicht perfekt verhaltenserhaltend (vgl. F. 19).
- Wenn Austausch des WfMS zu Austausch der Modelltransformation führt, kann dies zu Verhaltensänderung bei Modellausführung führen.

Da Neumodellieren aufwendig ist, führt dies in Konsequenz zu Lock-in bzgl. WfMS (Kunde vom Produkt abhängig).

Lösung: Erweiterung des Ziel-Metamodells

- Modelle, die in Metamodell M^2 spezifiziert wurden, müssen in Engine mit anderem Metamodell M^1 perfekt unterstützt werden.
- **Lösungsansatz**: Konstrukte aus M^2, die schwer in M^1 zu emulieren sind, in M^1 neu **hinzufügen**.
- BPEL **erweiterbar** entworfen:
  Kann neue Konstrukte hinzufügen.
  → Optimale Zuordnung in verschiedenen Metamodellen.
- Erweiterte Variante einer M^1-Engine („M^{1++}-Engine“) kann z.B. Prozessmodelle von Metamodellen M^2, M^3, ..., M^n unterstützen.

Erweiterung des Ziel-Metamodells: Beispiel



BPEL^{++}: Erweiterung des **BPEL**-Metamodells um **BPMN**-Notationselemente, die in BPEL nicht oder nur schlecht unterstützt.

Diskussion: Erweiterung des Ziel-Metamodells

Was sind **Nachteile** dieses Ansatzes ?

Diskussion: Erweiterung des Ziel-Metamodells

Was sind **Nachteile** dieses Ansatzes ?

Antwort:

- Durch Erweiterung Ziel-Metamodell immer **komplexer**.
- „Schwer im Zielmodell emulieren“ heisst nicht immer „unmöglich zu emulieren“.
  => Erweiterung des Ziel-Metamodelles führt zu (zumindest partieller) **Redundanz.**
  => Redundanz erhöht wie üblich Wartungsaufwand und Fehleranfälligkeit (wegen Notwendigkeit der Konsistenz zwischen redundanten Teilen).

Weitere Entwicklung im Bereich Process-Engines

- BPMN 2.0 (Business Process Model and Notation) signifikant **komplexer** als BPEL.
- Werkzeuge unterstützen **Teile** von BPMN 2.0, basierend auf Kunden-Anforderungen.
- Kein Werkzeug wird alle Aspekte der BPMN 2.0 unterstützen.
- **BPEL-Engines:** Um fehlende Schlüsseleigenschaften von BPMN 2.0 erweitert.
- **BPEL:** Um fehlende BPMN 2.0-Eigenschaften erweitert.

Workflow-Automatisierung: Überblick

- Grundlagen **Modell-Deployment**
  - Natives Metamodell einer Workflow-Engine und Modelltransformation
- **BPEL** und **Transformation**: BPMN 2 nach BPEL 2
  - Kurz-Einführung BPEL, Aktivitäten
  - Ereignisse
  - Strukturierte Aktivitäten

Beispiel für Modell-Transformation: BPMN nach BPEL

Betrachten als Beispiel für **Modelltransformation**:
Transformation von **BPMN**-Modellen zu **BPEL**-Modellen.

Als Teil des **BPMN 2.0-Standards** definiert.

→ BPMN 2.0-Notation enthält **Teilnotation isomorph** zu **BPEL**.

Alternative Sichtweise:
Erhalte **Visualisierung** der **BPEL** als Teil von BPMN 2.0.

BPEL

- Kurz für: **Web Services Business Process Execution Language (WS-BPEL)**
- **XML-basierte** textuelle Notation.
- Ziel: Zusammenfügen von Services in Prozessfluss.
- 2003: OASIS-Standardisierung von **BPEL4WS 1.1**
  (BEA Systems, IBM, Microsoft, SAP, Siebel Systems)
  - OASIS: Organization for the Advancement of Structured Information Standards (Globales Konsortium für Standardisierung im Bereich e-Business und Web-Service)
- 2007: OASIS-Standard **WS-BPEL 2.0**
- 2007: menschliche Interaktion in BPEL:
  BPEL4People, **WS-HumanTask**
  (Active Endpoints, Adobe Systems, BEA, IBM, Oracle, SAP)

WS-BPEL: Notation

Hier: **vereinfachter Ausschnitt** der Notation.

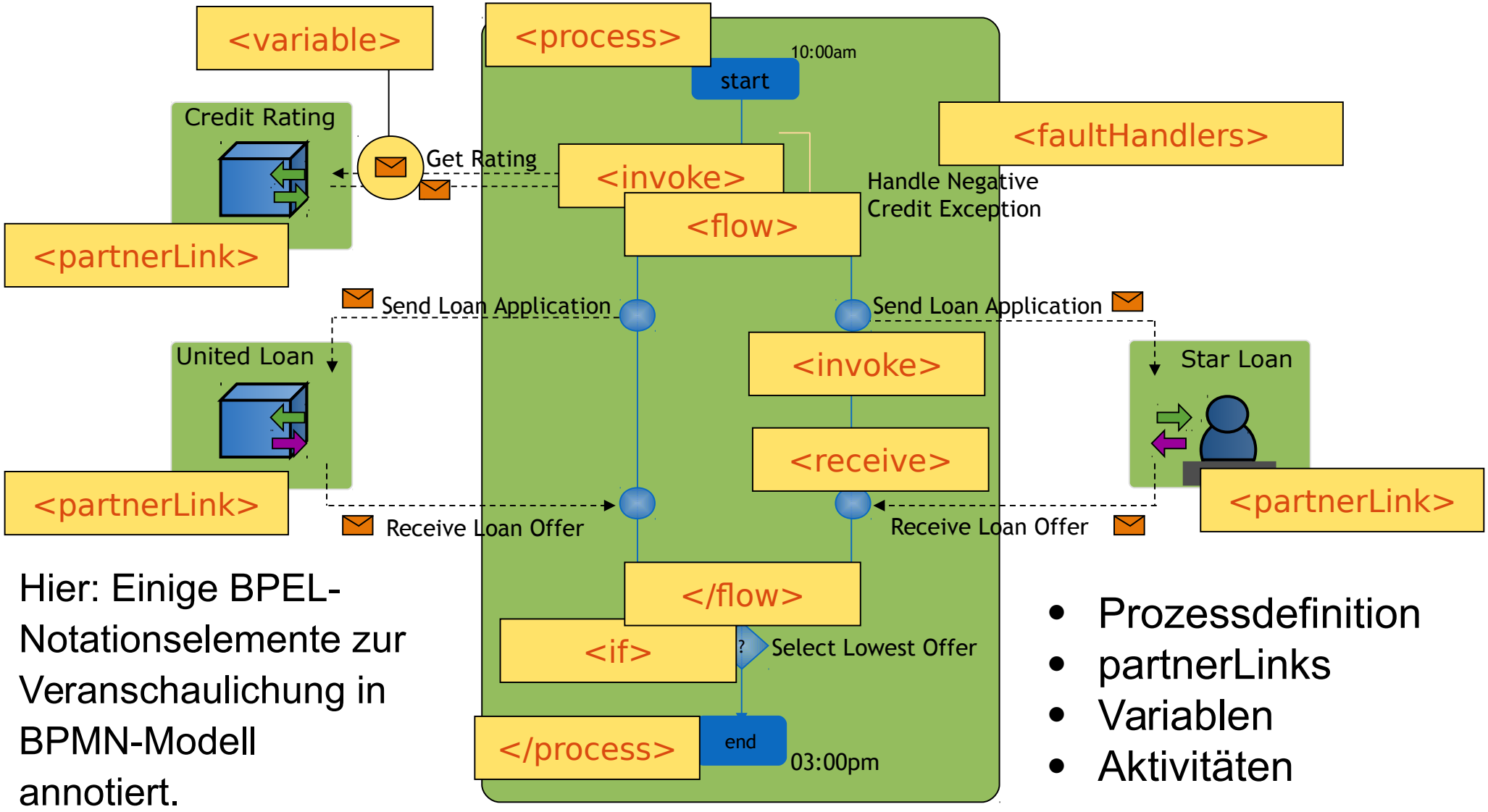
- **Definition der Notation** (**top-down**: 1.Prozessdefinition, 2. elementare Notationselemente)
- **Transformation BPMN → BPEL.**

Vgl. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html (insbes. “5.2. The Structure of a Business Process“).

Vorsicht: Im Netz noch viele Infos zu **BPEL 1.x** (**veraltet**, nicht immer erkennbar).

- **Änderungen BPEL 1.1 → 2.0**: http://wiki.open-esb.java.net/attach/BpelMigration/MigrationBP1.1ToBP2.0.odt
- **Änderungen BPEL 1.0 → 1.1**: http://msdn.microsoft.com/en-us/library/ee251594%28v=bts.10%29.aspx#bpel1-1_topic4

BPEL: Überblick, veranschaulicht mit BPMN



Hier: Einige BPEL-Notationselemente zur Veranschaulichung in BPMN-Modell annotiert.

- Prozessdefinition
- partnerLinks
- Variablen
- Aktivitäten

Struktur eines WS-BPEL-Prozesses


```
<process ...>   <!-- Prozessdefinition -->
<!-- Web-Services, mit denen der Prozess interagiert: -->
   <partnerLinks> ... </partnerLinks>
<!-- Daten, die von Prozess benutzt werden: -->
   <variables> ... </variables>
<!-- Wird für asynchrone Interaktionen verwendet: -->
   <correlationSets> ... </correlationSets>
<!-- Alternativer Ausführungspfad bei fehlerhafter Bedingung: -->
   <faultHandlers> ... </faultHandlers>
<!-- Code für Verarbeitung eines Ereignisses: -->
   <eventHandlers> ... </eventHandlers>
<!-- Was der Prozess eigentlich tut: -->
   (activities)*
</process>
```

Grundprinzip: Transformation BPMN => BPEL

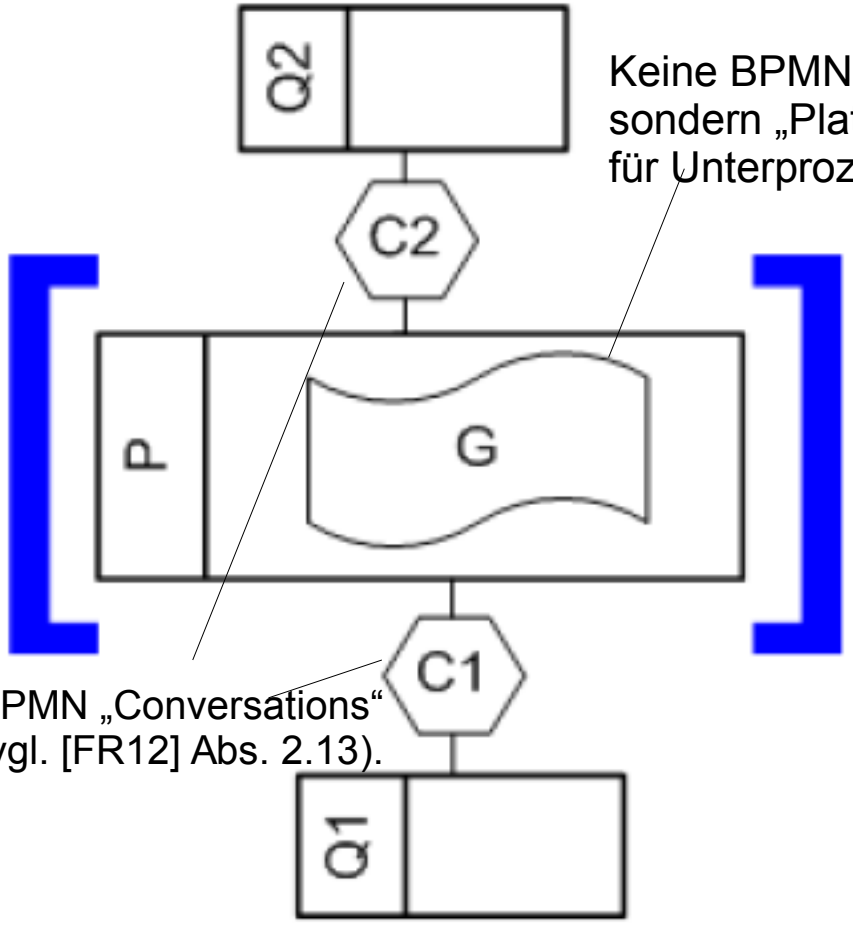
Funktion […]: bildet Untermenge von BPMN auf BPEL ab.

Rekursiv definiert.

- **Induktionsanfang**:
  - Für **elementare BPMN-Aufgabe** t, definiere BPEL-Abbild [t].
  - Für **elementares BPMN-Ereignis** e, definiere BPEL-Abbild [e].
- **Induktionsschritt:**
  - Für **BPMN-Struktur** s, definiere BPEL-Abbild [s].

Definiert konstruktiv die Menge der BPMN-Modelle, die in BPEL abgebildet werden kann, sowie zugehörige Transformation [...].

Transformation BPMN => BPEL: Prozess

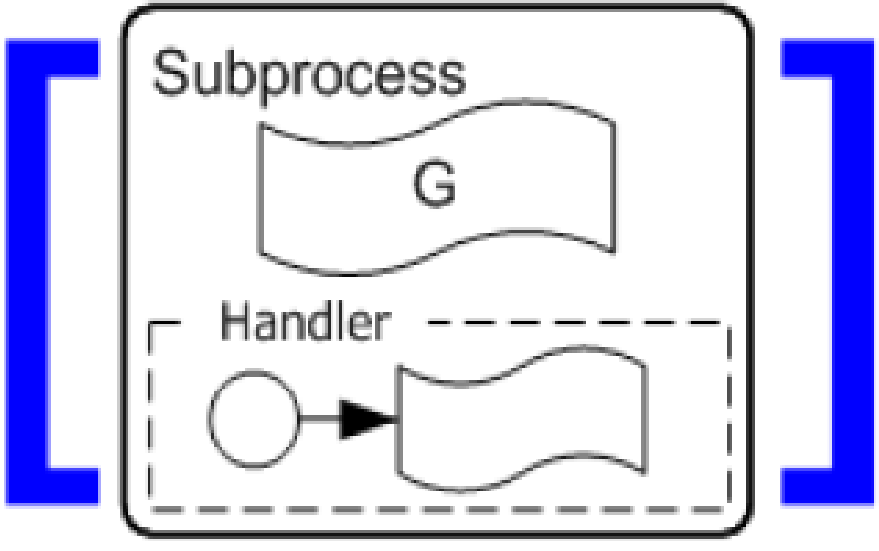


Keine BPMN-Syntax, sondern „Platzhalter“ für Unterprozess

BPMN „Conversations“ (vgl. [FR12] Abs. 2.13).

```
<process name="[P-name]"
...
=
<partnerLinks>
  [ {P-Interfaces} UNION {Qi-Interfaces} ]
</partnerLinks>
<variables>
  [ {dataObjects} UNION {properties} ]
</variables>
<correlationSets>
  [ {Ci-CorrelationKeys} ]
</correlationSets>
[G]
</process>
```

Transformation: Unterprozess



=

```
<scope name="[Subprocess-name]">
  <partnerLinks>
    [ {serviceRefs} ]
  </partnerLinks>
  <variables>
    [ {dataObjects} UNION {properties} ]
  </variables>
  <correlationSets>
    [ {correlations} ]
  </correlationSets>
  [Handler]
  [G]
</scope>
```

Partnerdefinitionen und Links

Partner-Service: über Web-Service-“Channel” abrufbar, definiert durch PartnerLinkTyp:

```
<partnerLink name="..."
     partnerLinkType="..."
     partnerRole="..." myRole="..."    />
```

partnerLinkType definiert zwei **Rollen** (die Endpunkte der Kommunikationsverbindung) und die **Porttypen**, die jede Rolle unterstützen muss:

```
<plnk:partnerLinkType name="...">
  <plnk:role name="..." portType="..."   />
  <plnk:role name="..."> portType="..." />
</plnk:partnerLinkType>
```

Einfache Aktivitäten: Operation aufrufen / erhalten

Prozess **aktiviert Operation** bei Partner:

```
<invoke name="..."
     partnerLink="..."
     portType="..."
     operation="..."
     inputVariable="..."
     outputVariable="..."  />
```

Prozess **erhält Aufruf** des Partners:

```
<receive name="..."
     [createInstance="..."]
     partnerLink="..."
     portType="..."
     operation="..."
     variable="..."  />
```

Antwortnachricht, Variablenzuweisung

Process sendet **Antwortnachricht** in Partneraufruf:

```
<reply name="..."
     partnerLink="..."
     portType="..."
     operation="..."
     variable="..." />
```

Datenbelegung zwischen **Variablen**:

```
<assign>
   <copy>
     <from variable="..."/> <to variable="..."/>
   </copy>
</assign>
```

Transformation: invoke / receive

[ ]

```
<invoke name="[Task-name]"
  partnerLink="[Task-serviceRef]"
  portType="[Task-operation-interface]"
  operation="[Task-operation]">
</invoke>
```

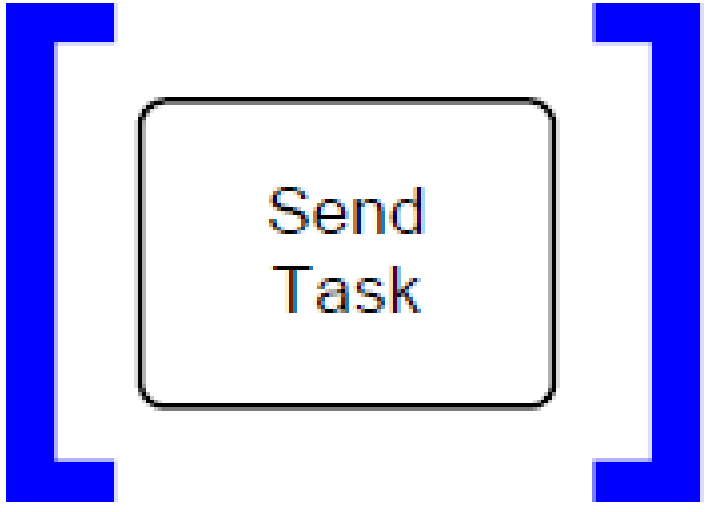
=

[ ]

```
<receive name="[Task-name]"
  createInstance="[instantiate? 'yes':'no']"
  partnerLink="[Task-serviceRef]"
  portType="[Task-operation-interface]"
  operation="[Task-operation]">
</receive>
```

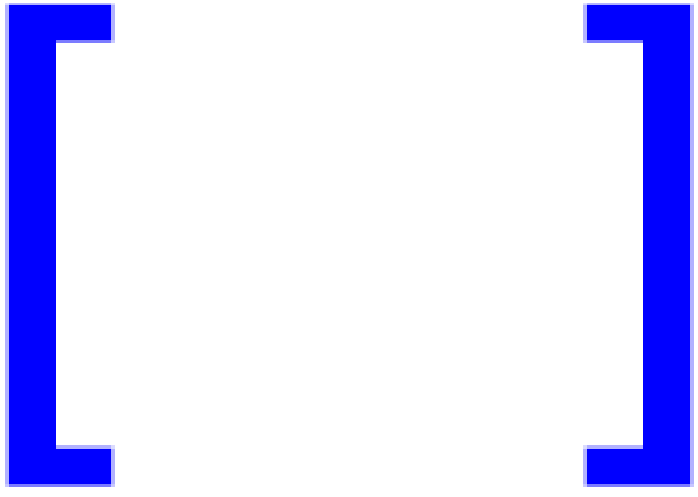
Wie würde dies in BPMN modelliert ?

Transformation: invoke / receive




```
<invoke name="[Task-name]"
  partnerLink="[Task-serviceRef]"
  portType="[Task-operation-interface]"
  operation="[Task-operation]">
</invoke>
```

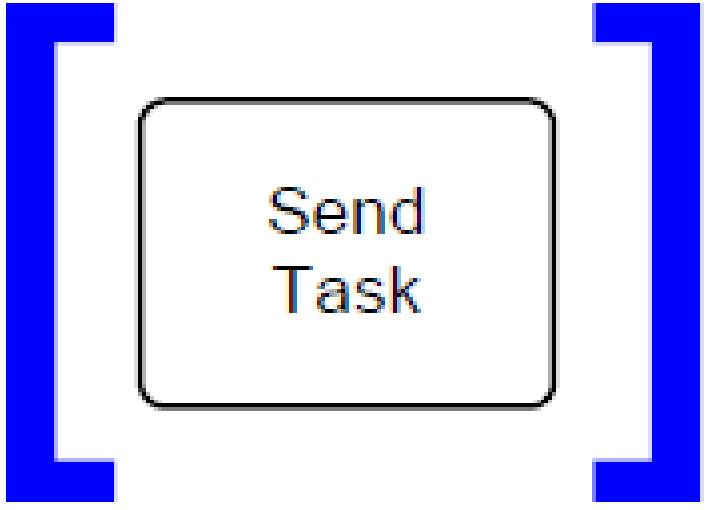
=



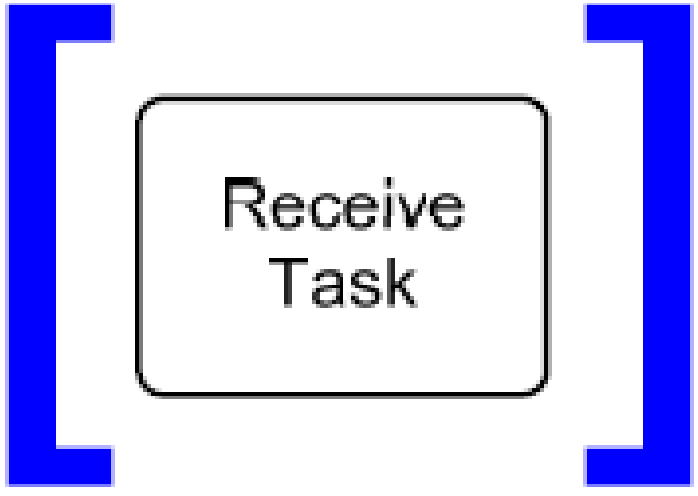
```
<receive name="[Task-name]"
  createInstance="[instantiate? 'yes':'no']"
  partnerLink="[Task-serviceRef]"
  portType="[Task-operation-interface]"
  operation="[Task-operation]">
</receive>
```

Wie würde dies in BPMN modelliert ?

Transformation: invoke / receive




```
<invoke name="[Task-name]"
  partnerLink="[Task-serviceRef]"
  portType="[Task-operation-interface]"
  operation="[Task-operation]">
</invoke>
```




```
<receive name="[Task-name]"
  createInstance="[instantiate? 'yes':'no']"
  partnerLink="[Task-serviceRef]"
  portType="[Task-operation-interface]"
  operation="[Task-operation]">
</receive>
```

Zustandshafte Interaktion

Gibt verschiedene Arten von **Interaktion** in verteilten Systemen:

- **Einfache, zustandslose** Interaktionen.
- **Zustandshafte**, **lang laufende**, **asynchrone** Interaktionen.

Wie könnte man sie jeweils in BPMN bzw. BPEL realisieren ?

Zustandshafte Interaktion: Korrelationsmengen

Gibt verschiedene Arten von **Interaktion** in verteilten Systemen:

- **Einfache, zustandslose** Interaktionen. => Send/receive tasks (s.o.).
- **Zustandshafte**, **lang laufende**, **asynchrone** Interaktionen.

Für Letzteres:

- Benötige Daten, um **Zustand der Interaktion** aufrechtzuerhalten.
  => Ankommende Nachrichten richtigen Prozessinstanzen zuordnen.

Lösung: **Korrelationsmengen** (Correlation Sets, CSs):

- **Menge von Geschäftsdatenfelder** für Interaktions-Zustand.
  Z.B.: “Bestellnummer”, “Benutzer ID”, etc.
- Jede Menge einmal initialisiert.
- Werte der Menge ändern nicht den Interaktions-Ablauf.

Definition von Korrelationsmengen

Korrelationsmenge: **benannte Menge an Eigenschaften**:

```
<correlationSet name="..." properties="..."/>
```

Eigenschaft hat globalen **Namen** und einfachen **Typ**:

```
<bpws:property name="..." type="..."/>
```

Eigenschaft ist auf **Feld** (part) in **Nachrichtentyp** (messageType) abgebildet und kann entsprechend abgefragt werden:

```
<bpws:propertyAlias propertyName="..."
        messageType="..." part="..." query="..."/>
```

Benutzung der Korrelationsmengen

Input- / Output-Operation ordnet Korrelationsmenge zur gesendeten / empfangenen **Nachricht** zu.

Korrelationsmenge stellt sicher, dass Nachricht zur zugehörigen **zustandshaften Interaktion** gehört.

```
<receive partner="..."
    operation="..." portType="..." variable="..." >
    <correlations>
```

<!-- CS **einmal initialisiert** innerhalb Interaktion: -->

```
        <correlation set="..."  initiate="..."/>
    </correlations>
</receive>
```

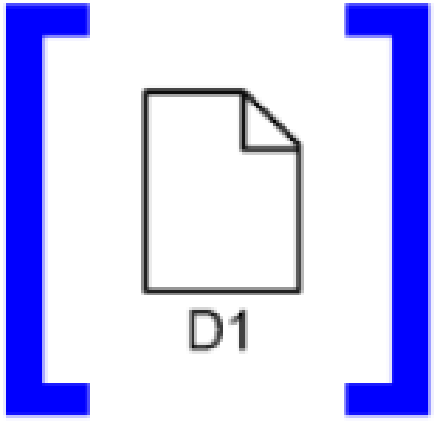
<!-- Analog für invoke statt receive ! -->

Transformation: Operation aufrufen / empfangen mittels Correlations

=

```
<invoke name="[Task-name]"
  partnerLink="[Q, Task-operation-interface]"
  portType="[Task-operation-interface]"
  operation="[Task-operation]">
  <correlations>
    <correlation set="[Task-messageFlow-conversation-correlationKey]"
        initiate="[initialInConversation? 'join':'no']"/>
  </correlations>
</invoke>
```

Transformation: Dokumente / Variablen



= <variable name="[D1-name]" type="[D1-structureDefinition]"/>

Transformation: Eingabe, Ausgabe

Ausgabe aus Aktivität A erhalten:

=

```
<receive>
  <fromParts>
    <fromPart part="[dataOutput1-name]"
      toVariable="[D3-name]"/>
    <fromPart part="[dataOutput2-name]"
      toVariable="[D4-name]"/>
  </fromParts>
</receive>
```

Wie würde dies in BPMN modelliert ?

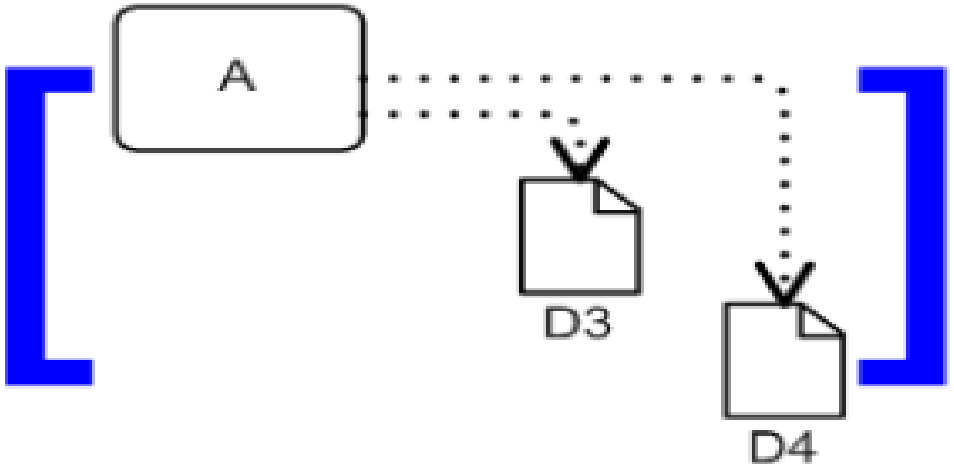
Aktivität A mit Eingabe aufrufen:

=

```
<invoke>
  <toParts>
    <toPart part="[dataInput1-name]"
      fromVariable="[D1-name]"/>
    <toPart part="[dataInput2-name]"
      fromVariable="[D2-name]"/>
  </toParts>
</invoke>
```

Transformation: Eingabe, Ausgabe

Ausgabe aus Aktivität erhalten:

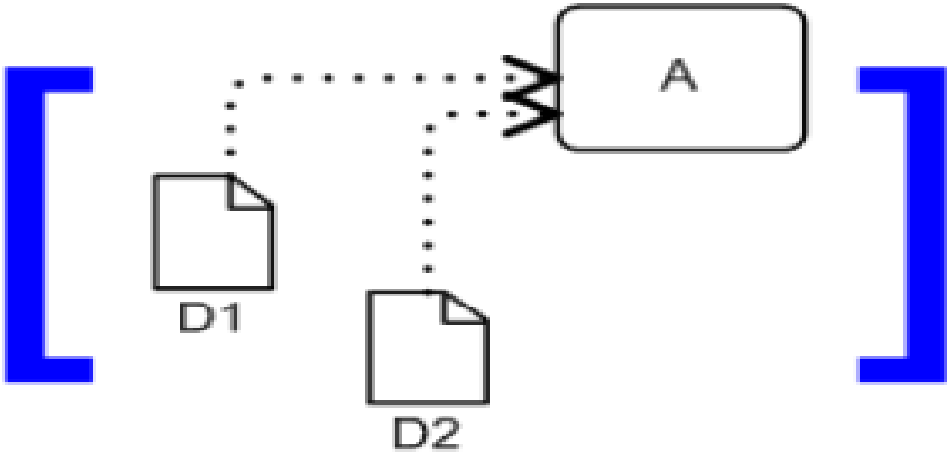


=

```
<receive>
  <fromParts>
    <fromPart part="[dataOutput1-name]"
      toVariable="[D3-name]"/>
    <fromPart part="[dataOutput2-name]"
      toVariable="[D4-name]"/>
  </fromParts>
</receive>
```

[Datenobjekte mit Assoziationen zuweisen, vgl. T. 1.2 F. 99.]

Aktivität mit Eingabe aufrufen:



=

```
<invoke>
  <toParts>
    <toPart part="[dataInput1-name]"
      fromVariable="[D1-name]"/>
    <toPart part="[dataInput2-name]"
      fromVariable="[D2-name]"/>
  </toParts>
</invoke>
```

Transformation: Kombinierte Ein-/Ausgabe

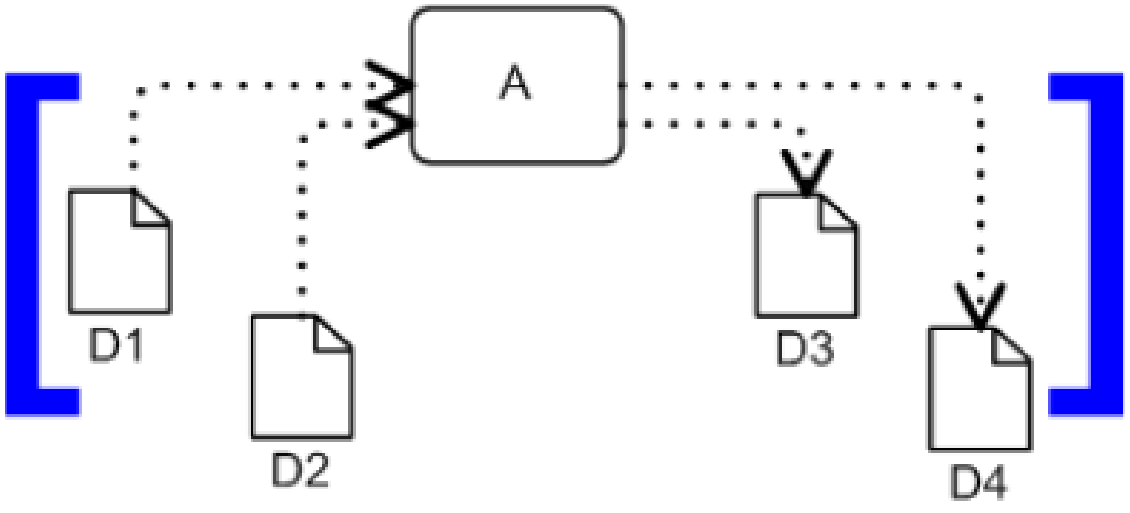
Ein-/Ausgabe an/von Aktivität A:

=

```
<invoke ...>
  <toParts>
    <toPart part="[dataInput1-name]"
      fromVariable="[D1-name]"/>
    <toPart part="[dataInput2-name]"
      fromVariable="[D2-name]"/>
  </toParts>
  <fromParts>
    <fromPart part="[dataOutput1-name]"
      toVariable="[D3-name]"/>
    <fromPart part="[dataOutput2-name]"
      toVariable="[D4-name]"/>
  </fromParts>
</invoke>
```

Transformation: Kombinierte Ein-/Ausgabe

Ein-/Ausgabe an/von Aktivität A:



=

```
<invoke ...>
  <toParts>
    <toPart part="[dataInput1-name]"
      fromVariable="[D1-name]"/>
    <toPart part="[dataInput2-name]"
      fromVariable="[D2-name]"/>
  </toParts>
  <fromParts>
    <fromPart part="[dataOutput1-name]"
      toVariable="[D3-name]"/>
    <fromPart part="[dataOutput2-name]"
      toVariable="[D4-name]"/>
  </fromParts>
</invoke>
```

Fehlerbehandlung, Ende, Pause, Nichts

Prozess entdeckt Ausführungsfehler und wechselt in **Fehlerausführungsbetrieb**:

```
<throw faultName="..." faultVariable="..."/>
```

Prozess **beenden** (inkl. alle aktiven Kontrollflüsse):

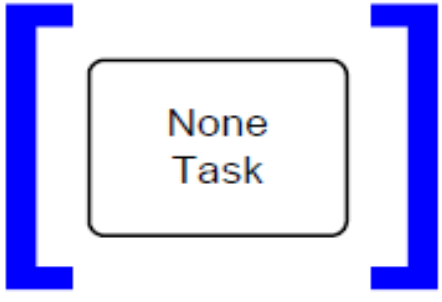
```
<exit></exit>
```

Prozess **stoppt** für bestimmte Zeit:

```
<wait name="..."> <for>"..."</for>
</wait>
```

Nichts tun:

```
<empty name="...">
</empty>
```

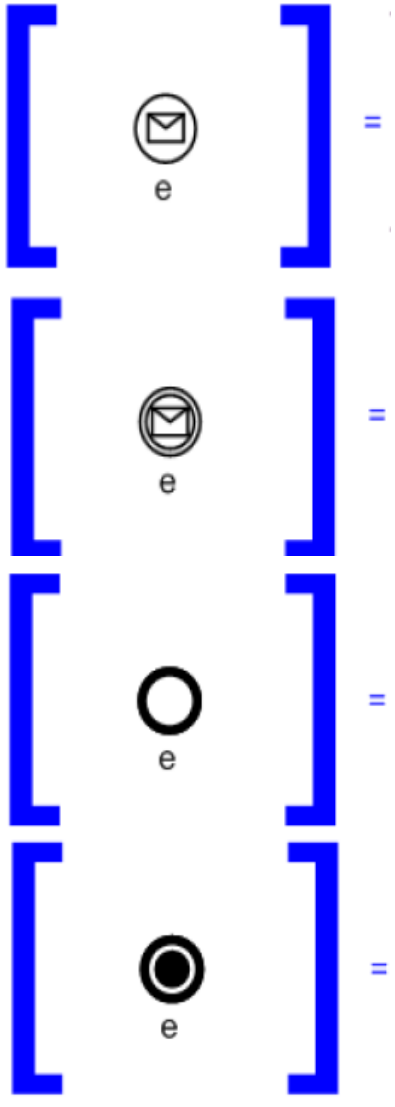


= <empty name="[Task-name]">
</empty>

Workflow-Automatisierung: Überblick

- **Grundlagen**
  - Natives Meta-Modell einer Workflow-Engine und Modell-Transformation
- **BPEL** und **Transformation**: BPMN 2 nach BPEL 2
  - Kurz-Einführung BPEL, Aktivitäten
  - Ereignisse
  - Strukturierte Aktivitäten

Transformation: Ereignisse



Prozess aktiviert Operation bei Partner:

```
<invoke name="..." partnerLink="..."
portType="..." operation="..."
inputVariable="..." outputVariable="..."/>
```

Prozess erhält Aufruf des Partners:

```
<receive name="..." [createInstance="..."]
partnerLink="..." portType="..."
operation="..." variable="..."/>
```

Prozess sendet Antwortnachricht in Partneraufruf:

```
<reply name="..." partnerLink="..."
portType="..." operation="..."
variable="..."/>
```

Datenbelegung zwischen Variablen:

```
<assign> <copy> <from variable="..."/>
<to variable="..."/> </copy>+ </assign>
```

Prozess wechselt in Fehlerausführungsbetrieb: `<throw faultName="..." faultVariable="..."/>`

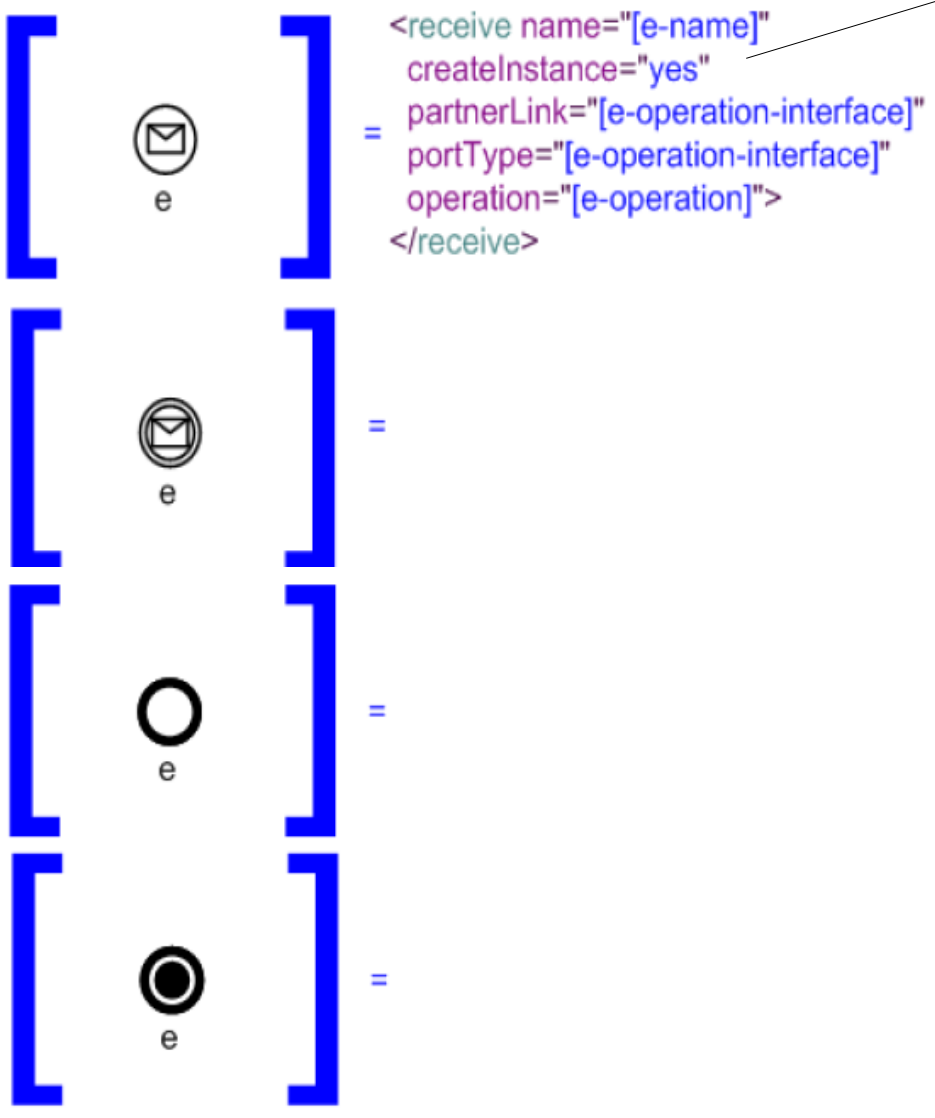
Prozess beenden: `<exit></exit>`

Prozess stoppt für bestimmte Zeit: `<wait name="..."> <for>"..."</for></wait>`

Nichts tun: `<empty name="..."> </empty>`

Transformation: Ereignisse

Startereignis !



Prozess aktiviert Operation bei Partner:

```
<invoke name="..." partnerLink="..."
portType="..." operation="..."
inputVariable="..." outputVariable="..."/>
```

Prozess erhält Aufruf des Partners:

```
<receive name="..." [createInstance="..."]
partnerLink="..." portType="..."
operation="..." variable="..."/>
```

Prozess sendet Antwortnachricht in Partneraufruf:

```
<reply name="..." partnerLink="..."
portType="..." operation="..."
variable="..."/>
```

Datenbelegung zwischen Variablen:

```
<assign> <copy> <from variable="..."/>
<to variable="..."/> </copy>+ </assign>
```

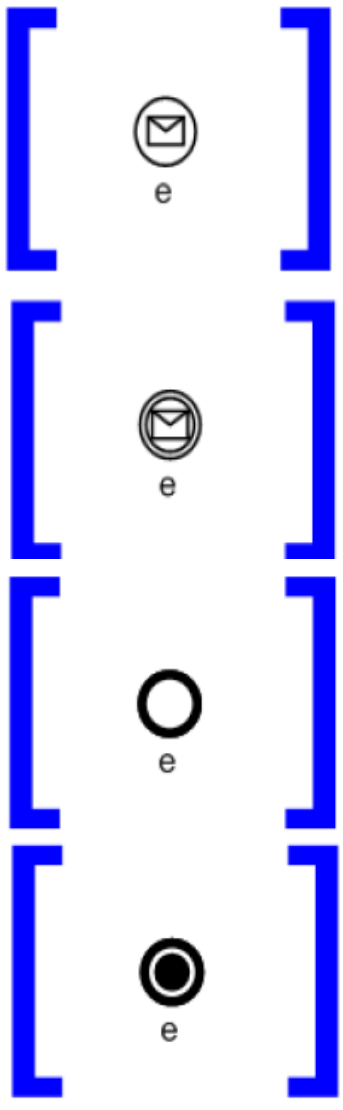
Prozess wechselt in Fehlerausführungsbetrieb: `<throw faultName="..." faultVariable="..."/>`

Prozess beenden: `<exit></exit>`

Prozess stoppt für bestimmte Zeit: `<wait name="..."> <for>"..."</for></wait>`

Nichts tun: `<empty name="..."> </empty>`

Transformation: Ereignisse



=

```
<receive name="[e-name]"
 createInstance="yes"
 partnerLink="[e-operation-interface]"
 portType="[e-operation-interface]"
 operation="[e-operation]">
</receive>
```

=

```
<receive name="[e-name]"
 createInstance="no"
 partnerLink="[e-operation-interface]"
 portType="[e-operation-interface]"
 operation="[e-operation]">
</receive>
```

Zwischenereignis !

=

=

Prozess aktiviert Operation bei Partner:

```
<invoke name="..." partnerLink="..."
portType="..." operation="..."
inputVariable="..." outputVariable="..."/>
```

Prozess erhält Aufruf des Partners:

```
<receive name="..." [createInstance="..."]
partnerLink="..." portType="..."
operation="..." variable="..."/>
```

Prozess sendet Antwortnachricht in Partneraufruf:

```
<reply name="..." partnerLink="..."
portType="..." operation="..."
variable="..."/>
```

Datenbelegung zwischen Variablen:

```
<assign> <copy> <from variable="..."/>
<to variable="..."/> </copy>+ </assign>
```

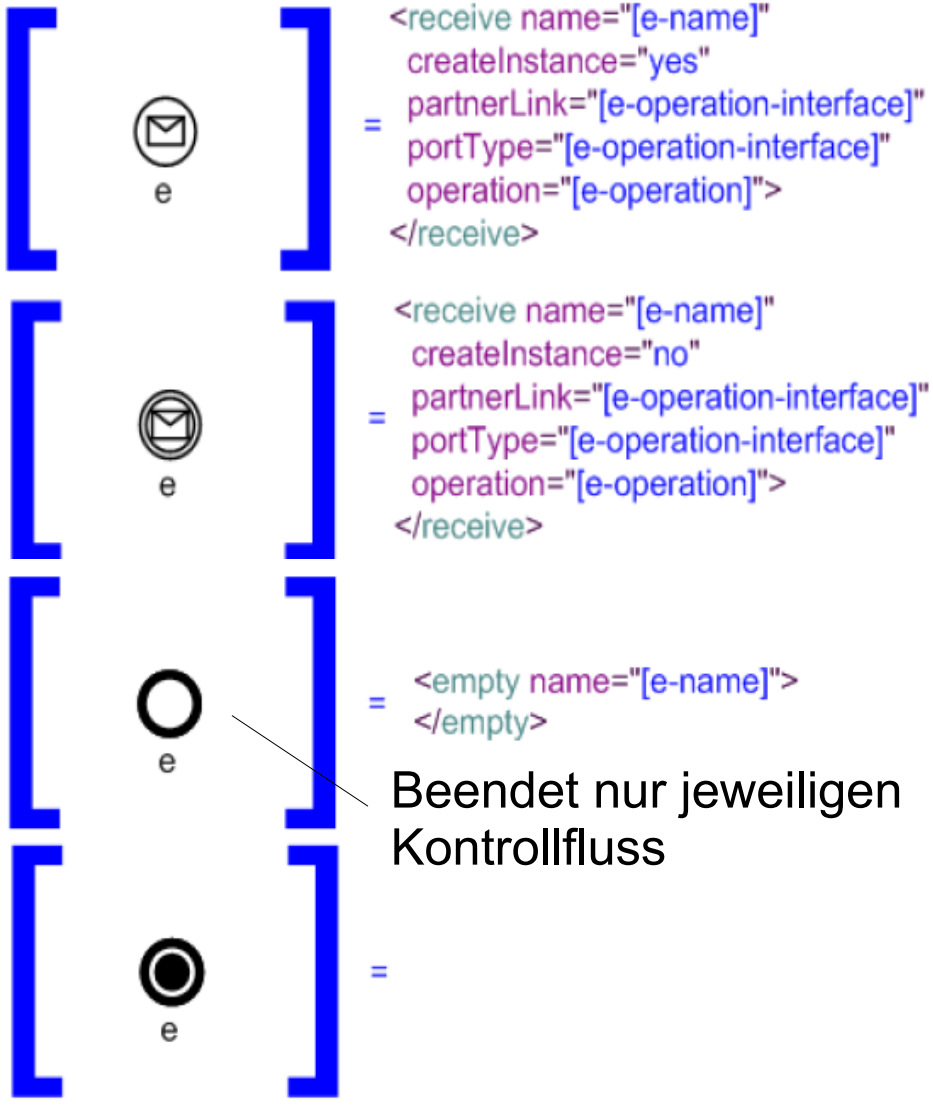
Prozess wechselt in Fehlerausführungsbetrieb: `<throw faultName="..." faultVariable="..."/>`

Prozess beenden: `<exit></exit>`

Prozess stoppt für bestimmte Zeit: `<wait name="..."> <for>"..."</for></wait>`

Nichts tun: `<empty name="..."> </empty>`

Transformation: Ereignisse



Prozess aktiviert Operation bei Partner:

```
<invoke name="..." partnerLink="..."
portType="..." operation="..."
inputVariable="..." outputVariable="..."/>
```

Prozess erhält Aufruf des Partners:

```
<receive name="..." [createInstance="..."]
partnerLink="..." portType="..."
operation="..." variable="..."/>
```

Prozess sendet Antwortnachricht in Partneraufruf:

```
<reply name="..." partnerLink="..."
portType="..." operation="..."
variable="..."/>
```

Datenbelegung zwischen Variablen:

```
<assign> <copy> <from variable="..."/>
<to variable="..."/> </copy>+ </assign>
```

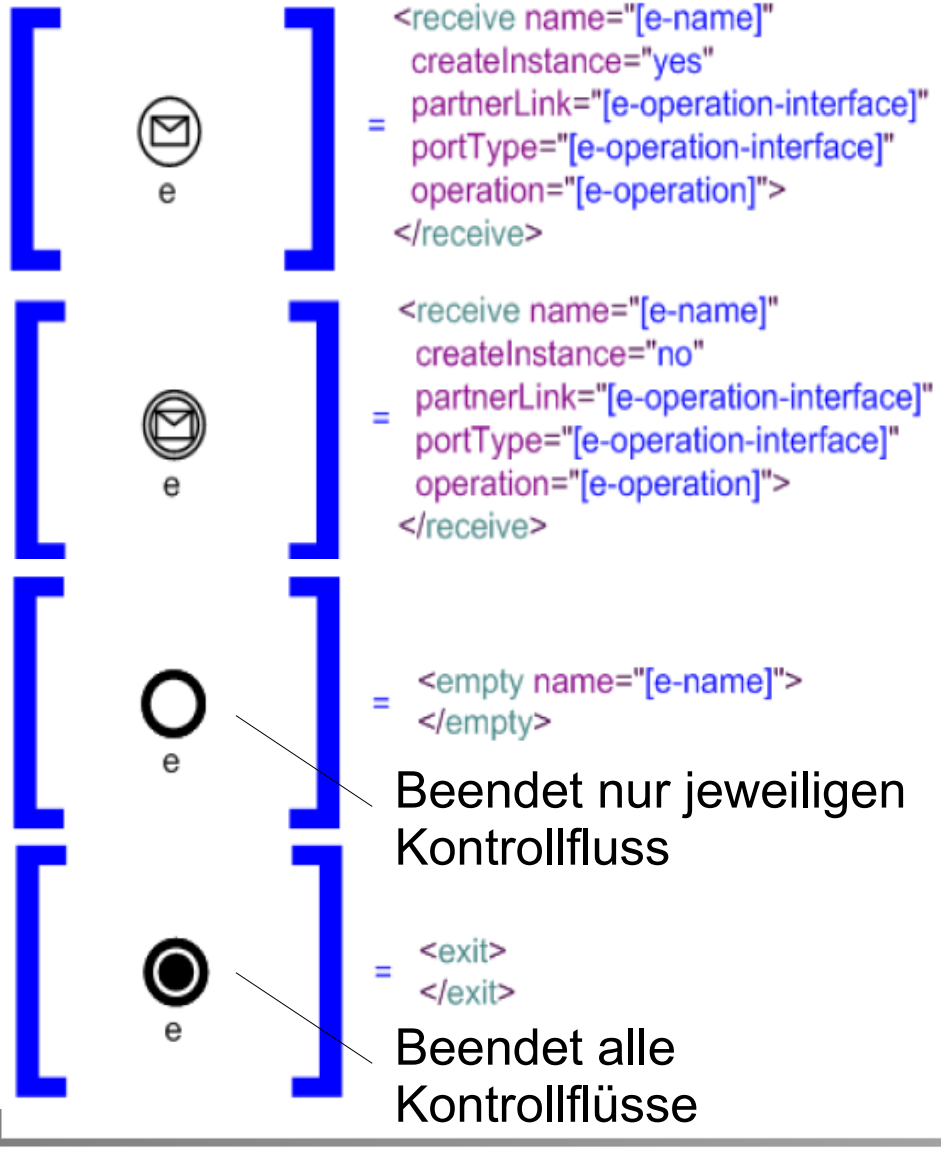
Prozess wechselt in Fehlerausführungsbetrieb: `<throw faultName="..." faultVariable="..."/>`

Prozess beenden: `<exit></exit>`

Prozess stoppt für bestimmte Zeit: `<wait name="..."> <for>"..."</for></wait>`

Nichts tun: `<empty name="..."> </empty>`

Transformation: Ereignisse



Prozess aktiviert Operation bei Partner:

```
<invoke name="..." partnerLink="..."
portType="..." operation="..."
inputVariable="..." outputVariable="..."/>
```

Prozess erhält Aufruf des Partners:

```
<receive name="..." [createInstance="..."]
partnerLink="..." portType="..."
operation="..." variable="..."/>
```

Prozess sendet Antwortnachricht in Partneraufruf:

```
<reply name="..." partnerLink="..."
portType="..." operation="..."
variable="..."/>
```

Datenbelegung zwischen Variablen:

```
<assign> <copy> <from variable="..."/>
<to variable="..."/> </copy>+ </assign>
```

Prozess wechselt in Fehlerausführungsbetrieb: `<throw faultName="..." faultVariable="..."/>`

Prozess beenden: `<exit></exit>`

Prozess stoppt für bestimmte Zeit: `<wait name="..."> <for>"..."</for></wait>`

Nichts tun: `<empty name="..."> </empty>`

Transformation: Ereignisse

Prozess aktiviert Operation bei Partner:

```
<invoke name="..." partnerLink="..."
portType="..." operation="..."
inputVariable="..." outputVariable="..."/>
```

Prozess erhält Aufruf des Partners:

```
<receive name="..." [createInstance="..."]
partnerLink="..." portType="..."
operation="..." variable="..."/>
```

Prozess sendet Antwortnachricht in Partneraufruf:

```
<reply name="..." partnerLink="..."
portType="..." operation="..."
variable="..."/>
```

Datenbelegung zwischen Variablen:

```
<assign> <copy> <from variable="..."/>
<to variable="..."/> </copy>+ </assign>
```

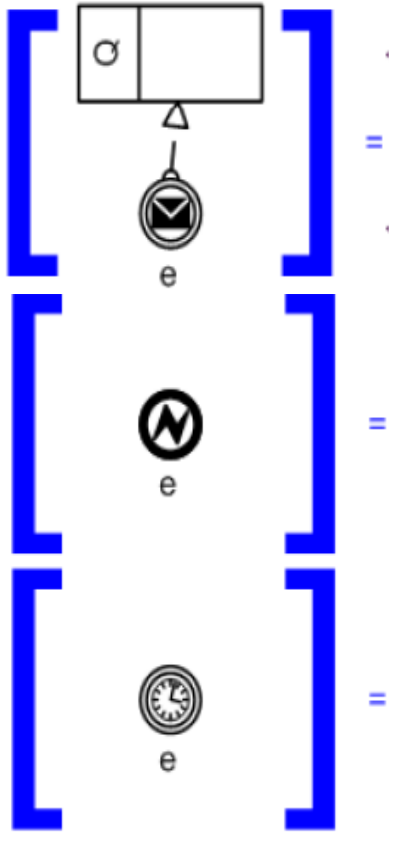
Prozess wechselt in Fehlerausführungsbetrieb: `<throw faultName="..." faultVariable="..."/>`

Prozess beenden: `<exit></exit>`

Prozess stoppt für bestimmte Zeit: `<wait name="...">`

```
<for>"..."</for></wait>
```

Nichts tun: `<empty name="...">  </empty>`



Transformation: Ereignisse

Prozess aktiviert Operation bei Partner:

```
<invoke name="..." partnerLink="..."
portType="..." operation="..."
inputVariable="..." outputVariable="..."/>
```

Prozess erhält Aufruf des Partners:

```
<receive name="..." [createInstance="..."]
partnerLink="..." portType="..."
operation="..." variable="..."/>
```

Prozess sendet Antwortnachricht in Partneraufruf:

```
<reply name="..." partnerLink="..."
portType="..." operation="..."
variable="..."/>
```

Datenbelegung zwischen Variablen:

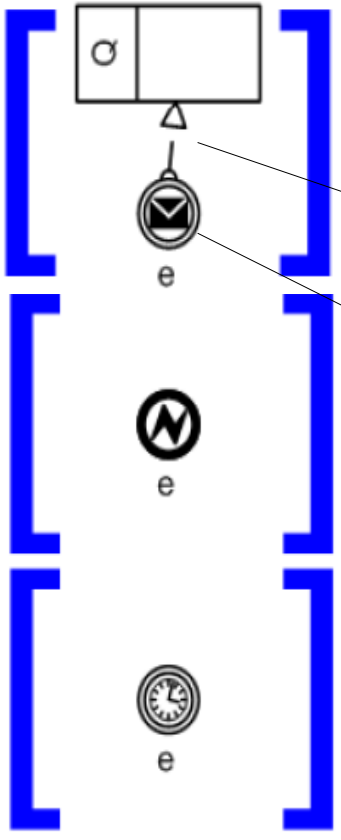
```
<assign> <copy> <from variable="..."/>
<to variable="..."/> </copy>+ </assign>
```

Prozess wechselt in Fehlerausführungsbetrieb: `<throw faultName="..." faultVariable="..."/>`

Prozess beenden: `<exit></exit>`

Prozess stoppt für bestimmte Zeit: `<wait name="..."> <for>"..."</for></wait>`

Nichts tun: `<empty name="..."> </empty>`



=

=

```
<invoke name="[e-name]"
  partnerLink="[Q, e-operation-interface]"
= portType="[e-operation-interface]"
  operation="[e-operation]">
</invoke>
```

[Nachrichtenfluss, vgl. T. 1.2 F. 95]]

[Verschicken der Nachrichten **ausgelöst** (statt **eingetreten**), vgl. T. 1.2 F. 49/50]

Transformation: Ereignisse

Prozess aktiviert Operation bei Partner:

```
<invoke name="..." partnerLink="..."
portType="..." operation="..."
inputVariable="..." outputVariable="..."/>
```

Prozess erhält Aufruf des Partners:

```
<receive name="..." [createInstance="..."]
partnerLink="..." portType="..."
operation="..." variable="..."/>
```

Prozess sendet Antwortnachricht in Partneraufruf:

```
<reply name="..." partnerLink="..."
portType="..." operation="..."
variable="..."/>
```

Datenbelegung zwischen Variablen:

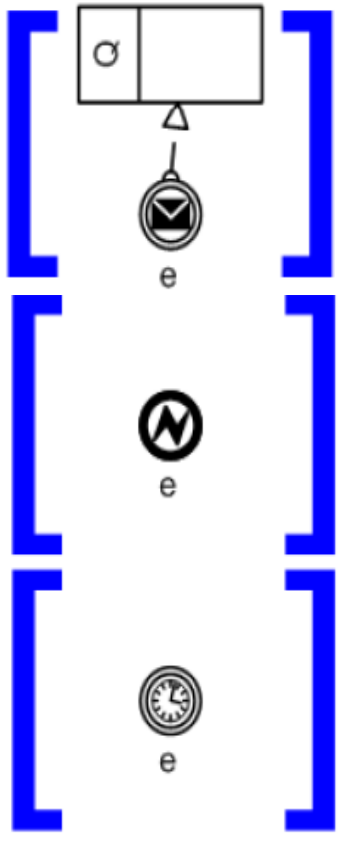
```
<assign> <copy> <from variable="..."/>
<to variable="..."/> </copy>+ </assign>
```

Prozess wechselt in Fehlerausführungsbetrieb: `<throw faultName="..." faultVariable="..."/>`

Prozess beenden: `<exit></exit>`

Prozess stoppt für bestimmte Zeit: `<wait name="..."> <for>"..."</for></wait>`

Nichts tun: `<empty name="..."> </empty>`




```
= <invoke name="[e-name]"
    partnerLink="[Q, e-operation-interface]"
    portType="[e-operation-interface]"
    operation="[e-operation]">
  </invoke>

= <throw faultName="[e-name]">
  </throw>

=
```

Transformation: Ereignisse

Prozess aktiviert Operation bei Partner:

```
<invoke name="..." partnerLink="..."
portType="..." operation="..."
inputVariable="..." outputVariable="..."/>
```

Prozess erhält Aufruf des Partners:

```
<receive name="..." [createInstance="..."]
partnerLink="..." portType="..."
operation="..." variable="..."/>
```

Prozess sendet Antwortnachricht in Partneraufruf:

```
<reply name="..." partnerLink="..."
portType="..." operation="..."
variable="..."/>
```

Datenbelegung zwischen Variablen:

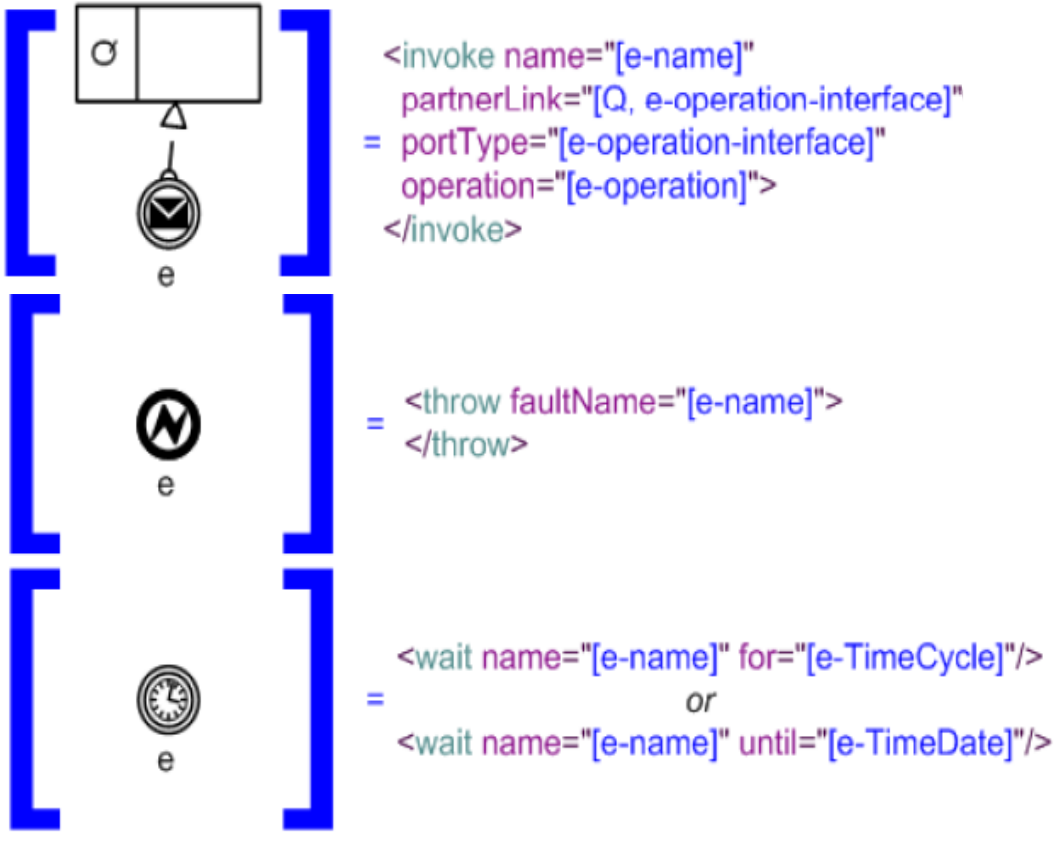
```
<assign> <copy> <from variable="..."/>
<to variable="..."/> </copy>+ </assign>
```

Prozess wechselt in Fehlerausführungsbetrieb: `<throw faultName="..." faultVariable="..."/>`

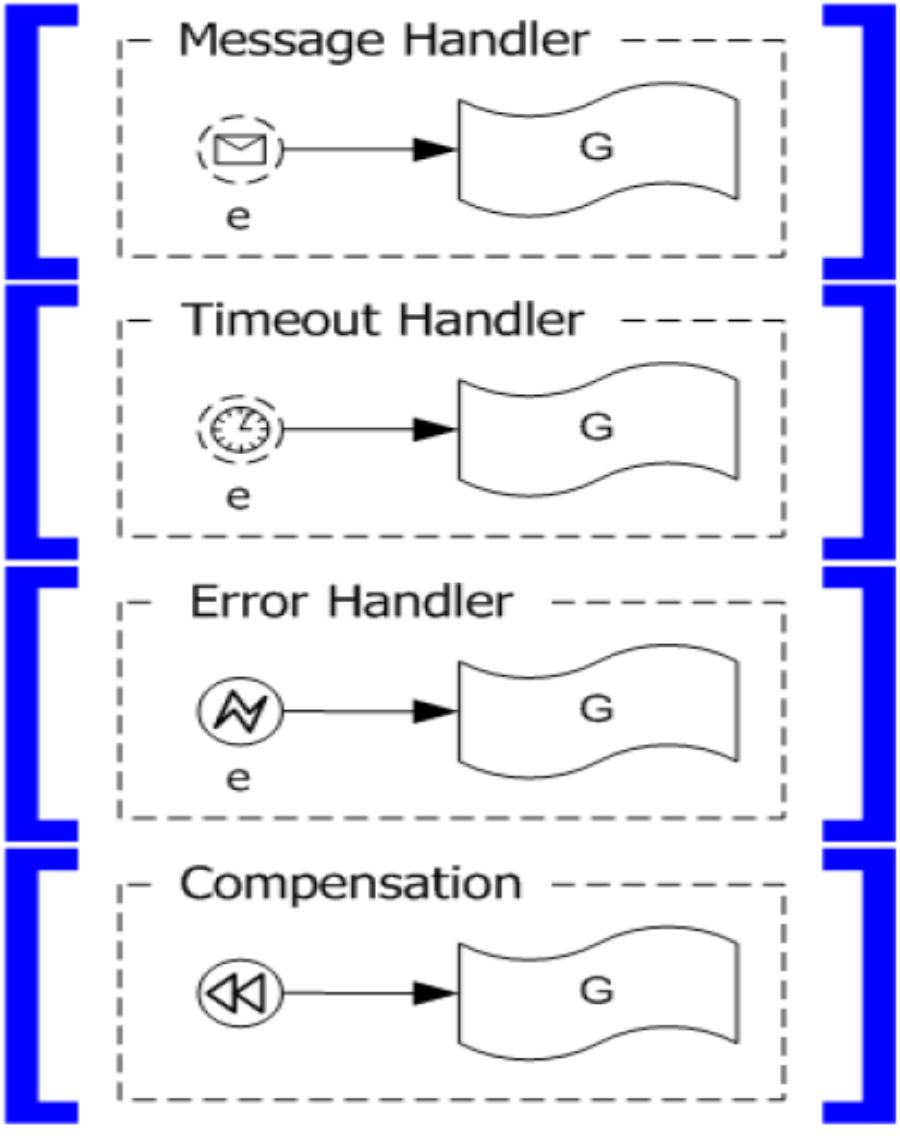
Prozess beenden: `<exit></exit>`

Prozess stoppt für bestimmte Zeit: `<wait name="..."><for>"..."</for></wait>`

Nichts tun: `<empty name="..."> </empty>`



Ereignis-Verarbeitung (Event Handling)



=
```
<eventHandlers>
  <onEvent partnerLink="[e-operation-interface]"
           operation="[e-operation]">
    <scope>[G]</scope>
  </onEvent>
</eventHandlers>
```

=
```
<eventHandlers>
  <onAlarm>[timer-spec]
    <scope>[G]</scope>
  </onAlarm>
</eventHandlers>
```

=
```
<faultHandlers>
  <catch faultName="[e-error]">
    [G]
  </catch>
</faultHandlers>
```

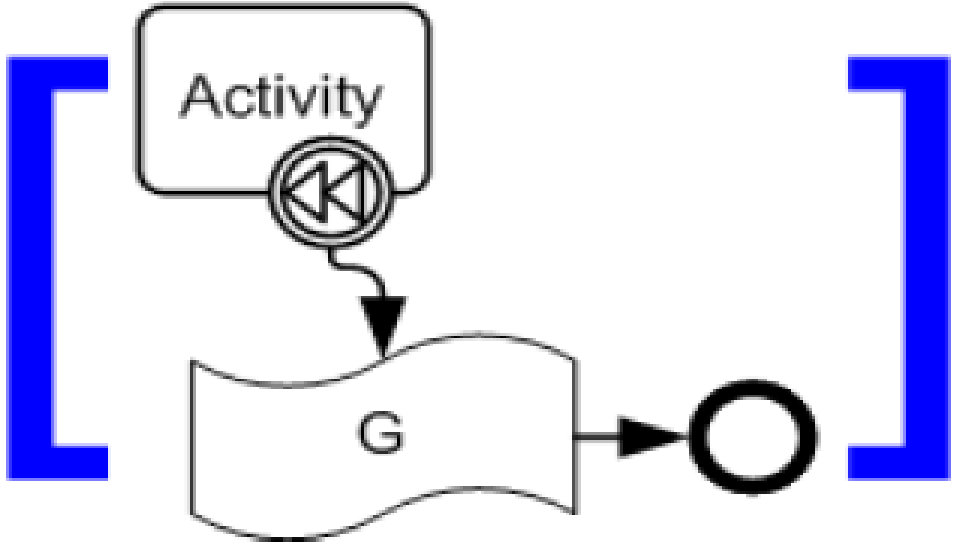
=
```
<compensationHandler>
  [G]
</compensationHandler>
```

Rand-Ereignis (1/4): Kompensation

[ ] =

```
<scope name="[Activity-name]">
  <compensationHandler>
    [G]
  </compensationHandler>
  [Activity]
</scope>
```

Rand-Ereignis (1/4): Kompensation



=

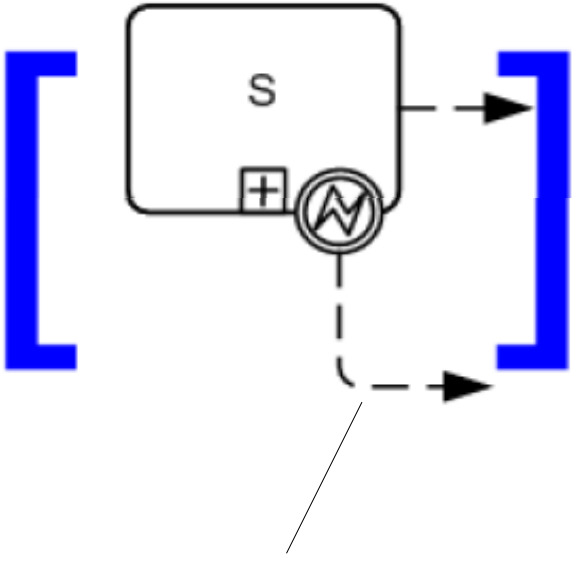
```
<scope name="[Activity-name]">
  <compensationHandler>
    [G]
  </compensationHandler>
  [Activity]
</scope>
```

Rand-Ereignis (2/4): Fehlerbehandlung

=

```
<scope>
 <faultHandlers>
  <catch faultName="...">
   <empty>
    <sources><source linkName="faultLink"/></sources>
   </empty>
  </catch>
 </faultHandlers>
 [S]
</scope>
```

Rand-Ereignis (2/4): Fehlerbehandlung



=

```
<scope>
 <faultHandlers>
  <catch faultName="...">
   <empty>
    <sources><source linkName="faultLink"/></sources>
   </empty>
  </catch>
 </faultHandlers>
 [S]
</scope>
```

[Nachrichtenfluss, vgl. T. 1.2 F. 95.]

Rand-Ereignis (3/4): Fehler-Behandlung vs. Normaler Fluss

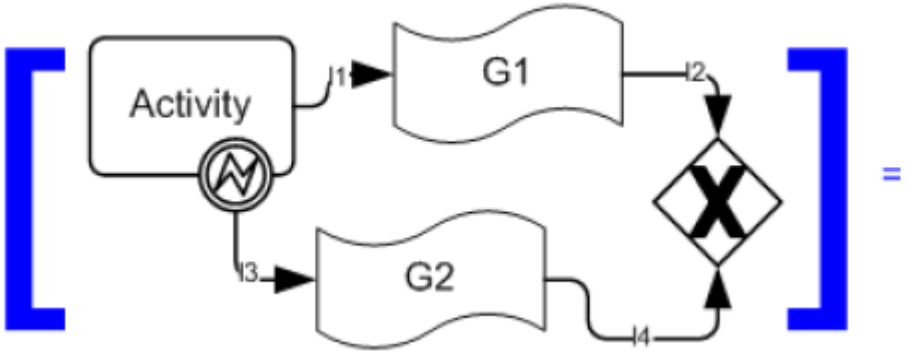
[ ] =

```
<flow>
 <links>
   <link name="[l1]"/>
   ...
   <link name="[l4]"/>
 </links>
 <scope>
   <sources><source linkName="[l1]"/></sources>
   <faultHandlers>
     <catch faultName="[e-error]">
       <empty>
         <sources><source linkName="[l3]"/></sources>
       </empty>
     </catch>
   </faultHandlers>
   [Activity]
 </scope>
 ...
```

Definiert Kontrollfluss

```
...
<flow>
  <targets><target linkName="[l1]"/></targets>
  <sources><source linkName="[l2]"/></sources>
  [G1]
</flow>
<flow>
  <targets><target linkName="[l3]"/></targets>
  <sources><source linkName="[l4]"/></sources>
  [G2]
</flow>
<empty>
  <targets><target linkName="[l2]"/>
 <target linkName="[l4]"/></targets>
</empty>
</flow>
```

Rand-Ereignis (3/4): Fehler-Behandlung vs. Normaler Fluss



=

```
<flow>
 <links>
   <link name="[l1]"/>
   ...
   <link name="[l4]"/>
 </links>
 <scope>
   <sources><source linkName="[l1]"/></sources>
   <faultHandlers>
     <catch faultName="[e-error]">
       <empty>
         <sources><source linkName="[l3]"/></sources>
       </empty>
     </catch>
   </faultHandlers>
   [Activity]
 </scope>
 ...
```

```
...
<flow>
  <targets><target linkName="[l1]"/></targets>
  <sources><source linkName="[l2]"/></sources>
  [G1]
</flow>
<flow>
  <targets><target linkName="[l3]"/></targets>
  <sources><source linkName="[l4]"/></sources>
  [G2]
</flow>
<empty>
  <targets><target linkName="[l2]"/>
 <target linkName="[l4]"/></targets>
</empty>
</flow>
```

Rand-Ereignis (4/4): Fehlerbehandlung, Kompensation, Schleife und normaler Fluss

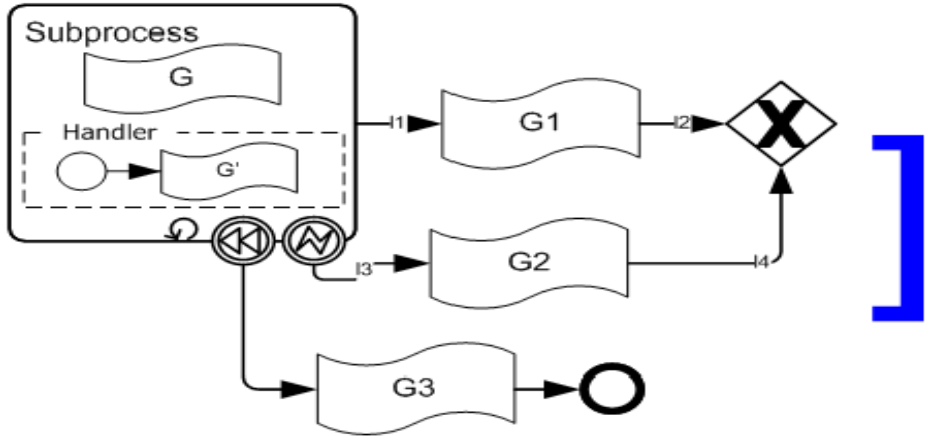

```
<flow>
  <links>
    <link name="[l1]"/>
    ...
    <link name="[l4]"/>
  </links>
  <scope>
    <sources><source linkName="[l1]"/></sources>
    <faultHandlers>
      <catch faultName="[e-error]">
        <empty>
          <sources><source linkName="[l3]"/></sources>
        </empty>
      </catch>
    </faultHandlers>
    <compensationHandler>
      [G3]
    </compensationHandler>
    <while>
      <condition>[p]</condition>
      <scope>
        [Handler]
        [G]
      </scope>
    </while>
  </scope>
  ...
```

[...] =

```
...
<flow>
  <targets><target linkName="[l1]"/></targets>
  <sources><source linkName="[l2]"/></sources>
  [G1]
</flow>
<flow>
  <targets><target linkName="[l3]"/></targets>
  <sources><source linkName="[l4]"/></sources>
  [G2]
</flow>
<empty>
  <<targets><target linkName="[l2]"/>
  <target linkName="[l4]"/></targets>
</empty>
</flow>
```

Rand-Ereignis (4/4): Fehlerbehandlung, Kompensation, Schleife und normaler Fluss


```
<flow>
  <links>
    <link name="[l1]"/>
    ...
    <link name="[l4]"/>
  </links>
  <scope>
    <sources><source linkName="[l1]"/></sources>
    <faultHandlers>
      <catch faultName="[e-error]">
        <empty>
          <sources><source linkName="[l3]"/></sources>
        </empty>
      </catch>
    </faultHandlers>
    <compensationHandler>
      [G3]
    </compensationHandler>
    <while>
      <condition>[p]</condition>
      <scope>
        [Handler]
        [G]
      </scope>
    </while>
  </scope>
  ...
```

[

] =

```
...
<flow>
  <targets><target linkName="[l1]"/></targets>
  <sources><source linkName="[l2]"/></sources>
  [G1]
</flow>
<flow>
  <targets><target linkName="[l3]"/></targets>
  <sources><source linkName="[l4]"/></sources>
  [G2]
</flow>
<empty>
  <<targets><target linkName="[l2]"/>
  <target linkName="[l4]"/></targets>
</empty>
</flow>
```

Workflow-Automatisierung: Überblick

- **Grundlagen**
  - Natives Meta-Modell einer Workflow-Engine und Modell-Transformation
- **BPEL** und **Transformation**: BPMN 2 nach BPEL 2
  - Kurz-Einführung BPEL, Aktivitäten
  - Ereignisse
  - Strukturierte Aktivitäten

Sequenzielles Ausführen von Aktivitäten:

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

Paralleles Ausführen von Aktivitäten:

```
<flow>...</flow>
```

Iterieren der Ausführung von Aktivitäten **solange** Bedingung erfüllt:

```
<while><condition>...</condition>
    ...
</while>
```

Iterieren der Ausführung von Aktivitäten **bis** Bedingung erfüllt:

```
<repeatUntil><condition>...</condition>
    ...
</repeatUntil>
```

Ereignisgesteuerte Auswahl:

Mehrere Event-Aktivitäten (z.B. Annehmen von Nachrichten) angesetzt für parallele Ausführung.

Erste eintretende auswählen und ausführen:

```
<pick createinstance="...">
    <onMessage partnerLink="..." operation="...">
    ...
    </onMessage>
    <onAlarm>
    ...
    </onAlarm>
</pick>
```

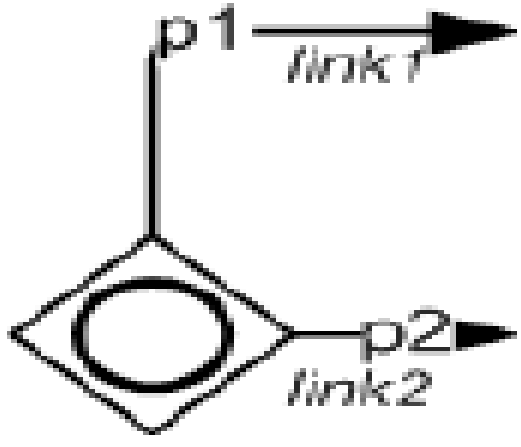
Strukturierte Aktivitäten: Bedingte Verzweigung


```
<if name="...">
    <condition> ... </condition>
    ...
    <elseif>
        <condition> ... </condition> …
    </elseif>
    <else> ... </else>
</if>
```

Inklusives Oder:

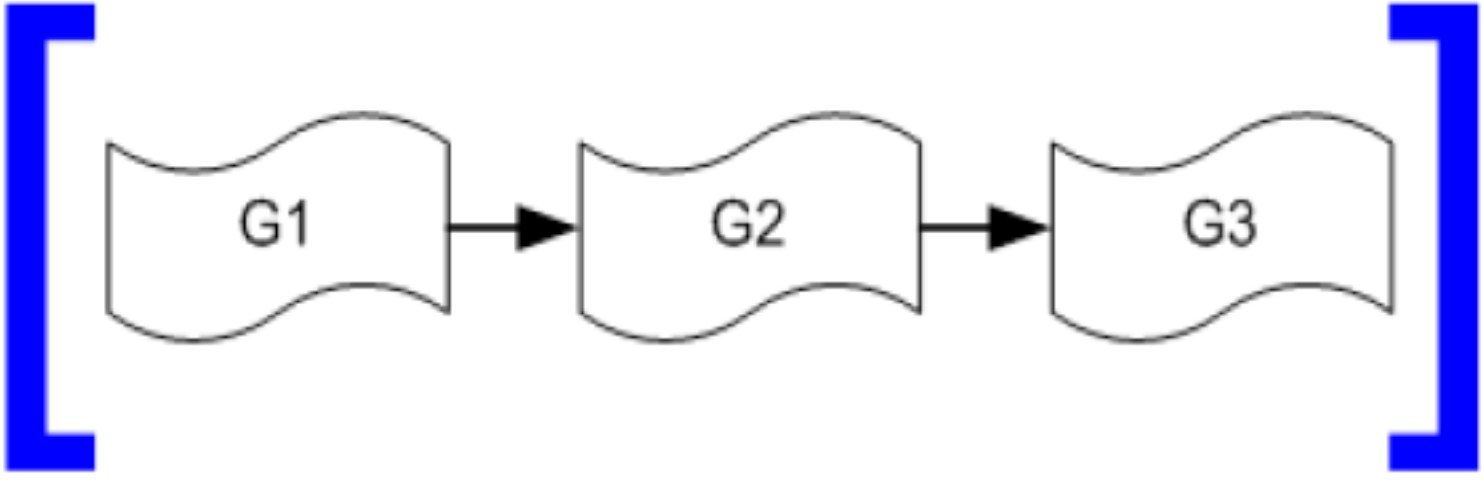
Kann Bedingungen an Sequenzverbindungen spezifizieren:

```
<source linkName="[link1]">
   <transitionCondition>[p1]</transitionCondition>
</source>
```



Transformation (Strukturierte Aktivitäten 1/8): Sequenzen

Transformation (Strukturierte Aktivitäten 1/8): Sequenzen



=

```
<sequence>
  [G1]
  [G2]
  [G3]
</sequence>
```

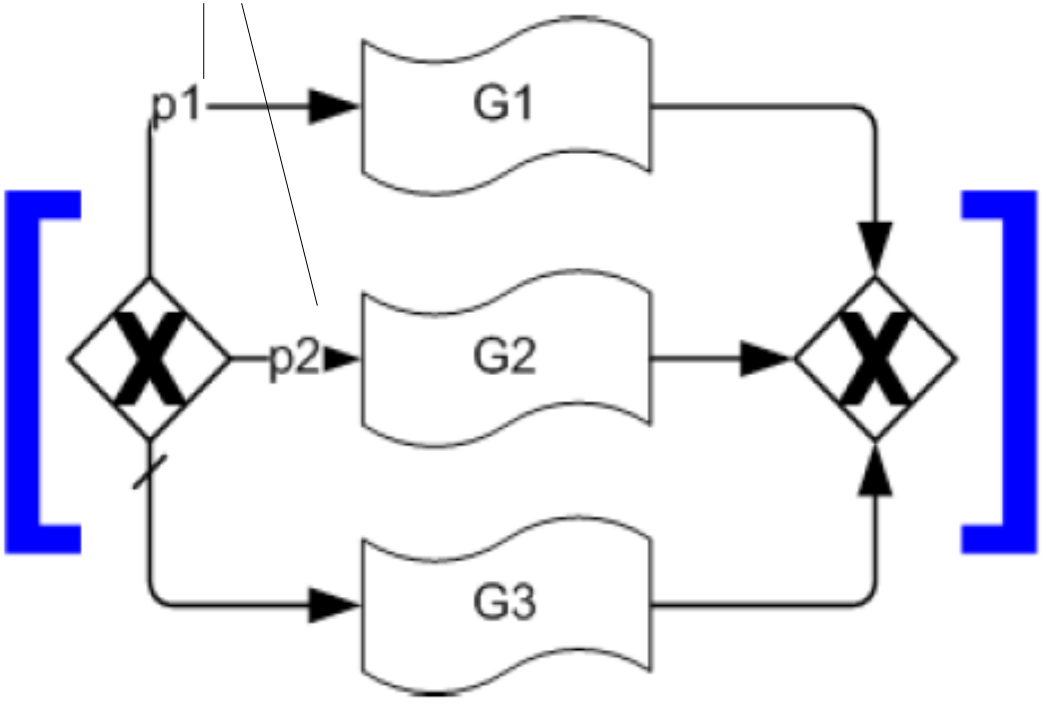
Transformation (Strukturierte Aktivitäten 2/8): If-Then-Else

=

```
<if><condition>[p1]</condition>
  [G1]
<elseif><condition>[p2]</condition>
  [G2]
</elseif>
<else>
  [G3]
</else>
</if>
```

Transformation (Strukturierte Aktivitäten 2/8): If-Then-Else

p1, p2 müssen wechselseitig exklusiv sein !



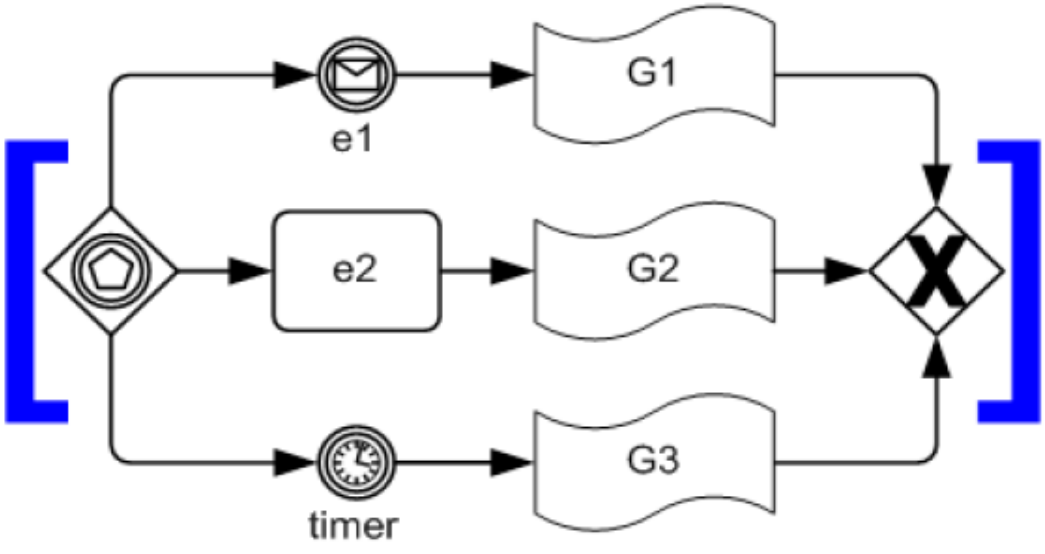
=

p1, p2 nicht notwendig wechselseitig exklusiv !

```
<if><condition>[p1]</condition>
 [G1]
<elseif><condition>[p2]</condition>
 [G2]
</elseif>
<else>
 [G3]
</else>
</if>
```

[< >] =

```
<pick createInstance="[instantiate? 'yes':'no']">
  <onMessage partnerLink="[e1-operation-interface]"
       operation="[e1-operation]">
    [G1]
  </onMessage>
  <onMessage partnerLink="[e2-operation-interface]"
       operation="[e2-operation]">
    [G2]
  </onMessage>
  <onAlarm>
    [timer-spec]
    [G3]
  </onAlarm>
</pick>
```

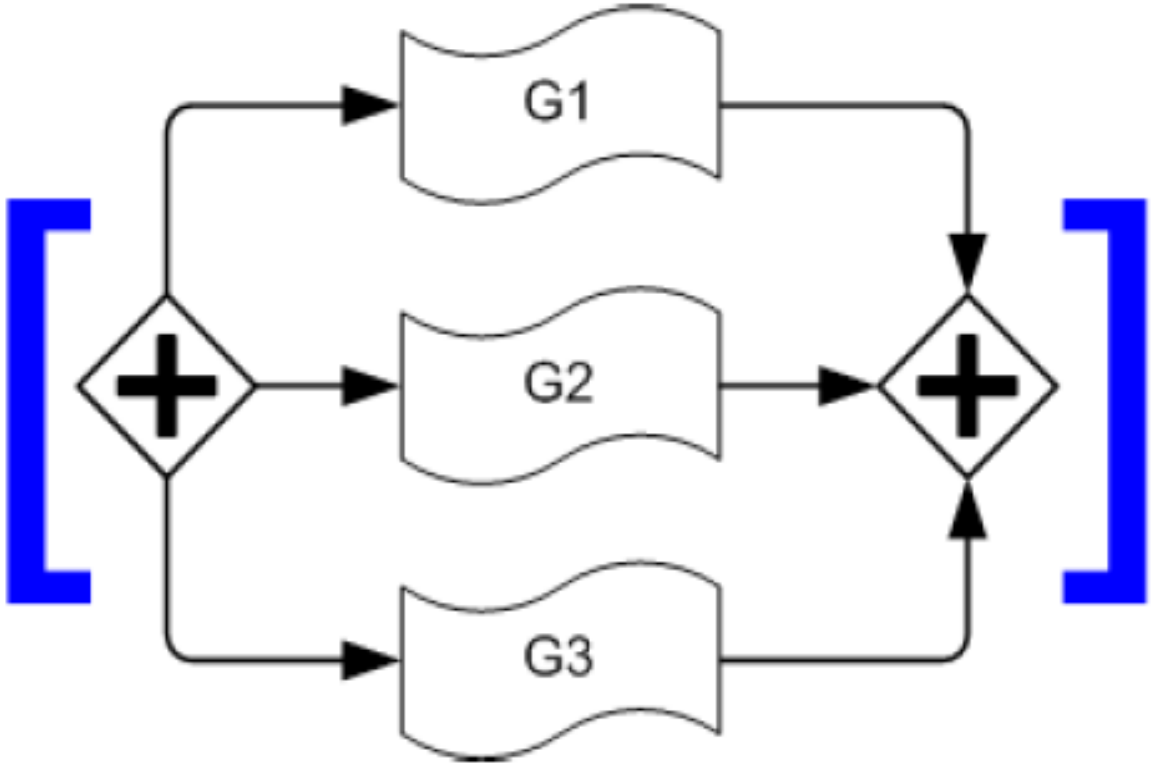


=

```
<pick createInstance="[instantiate? 'yes':'no']">
  <onMessage partnerLink="[e1-operation-interface]"
       operation="[e1-operation]">
    [G1]
  </onMessage>
  <onMessage partnerLink="[e2-operation-interface]"
       operation="[e2-operation]">
    [G2]
  </onMessage>
  <onAlarm>
    [timer-spec]
    [G3]
  </onAlarm>
</pick>
```

Transformation (Strukturierte Aktivitäten 4/8): Parallele Ausführung

Transformation (Strukturierte Aktivitäten 4/8): Parallele Ausführung



=

```
<flow>
  [G1]
  [G2]
  [G3]
</flow>
```

Transformation (Strukturierte Aktivitäten 5/8): Inklusives Oder-Gateway

[‹ ›] =

```
<flow>
 <links>
   <link name="[link1]"/>
   ...
   <link name="[link6]"/>
 </links>

 <empty>
   <sources>
     <source linkName="[link1]">
       <transitionCondition>[p1])</transitionCondition>
     </source>
     <source linkName="[link2]">
       <transitionCondition>[p2])</transitionCondition>
     </source>
     <source linkName="[link3]">
       <transitionCondition>[p3])</transitionCondition>
     </source>
   </sources>
 </empty>
```

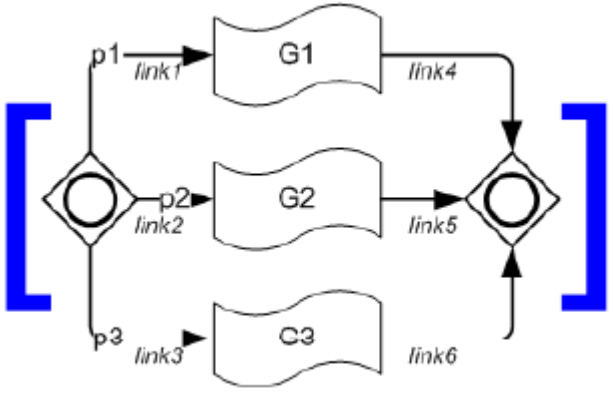
```
<flow>
  <targets><target linkName="[link1]"/></targets>
  <sources><source linkName="[link4]"/></sources>
  [G1]
</flow>

<flow>
  <targets><target linkName="[link2]"/></targets>
  <sources><source linkName="[link5]"/></sources>
  [G2]
</flow>

<flow>
  <targets><target linkName="[link3]"/></targets>
  <sources><source linkName="[link6]"/></sources>
  [G3]
</flow>

<empty>
  <targets>
    <target linkName="[link4]"/>
    <target linkName="[link5]"/>
    <target linkName="[link6]"/>
  </targets>
</empty>
</flow>
```

Transformation (Strukturierte Aktivitäten 5/8): Inklusives Oder-Gateway



=

```
<flow>
 <links>
   <link name="[link1]"/>
   ...
   <link name="[link6]"/>
 </links>

 <empty>
   <sources>
     <source linkName="[link1]">
       <transitionCondition>[p1])</transitionCondition>
     </source>
     <source linkName="[link2]">
       <transitionCondition>[p2])</transitionCondition>
     </source>
     <source linkName="[link3]">
       <transitionCondition>[p3])</transitionCondition>
     </source>
   </sources>
 </empty>
```

```
<flow>
  <targets><target linkName="[link1]"/></targets>
  <sources><source linkName="[link4]"/></sources>
  [G1]
</flow>

<flow>
  <targets><target linkName="[link2]"/></targets>
  <sources><source linkName="[link5]"/></sources>
  [G2]
</flow>

<flow>
  <targets><target linkName="[link3]"/></targets>
  <sources><source linkName="[link6]"/></sources>
  [G3]
</flow>

<empty>
  <targets>
    <target linkName="[link4]"/>
    <target linkName="[link5]"/>
    <target linkName="[link6]"/>
  </targets>
</empty>
</flow>
```

Transformation (Strukturierte Aktivitäten 6/8): While-Schleifen

=

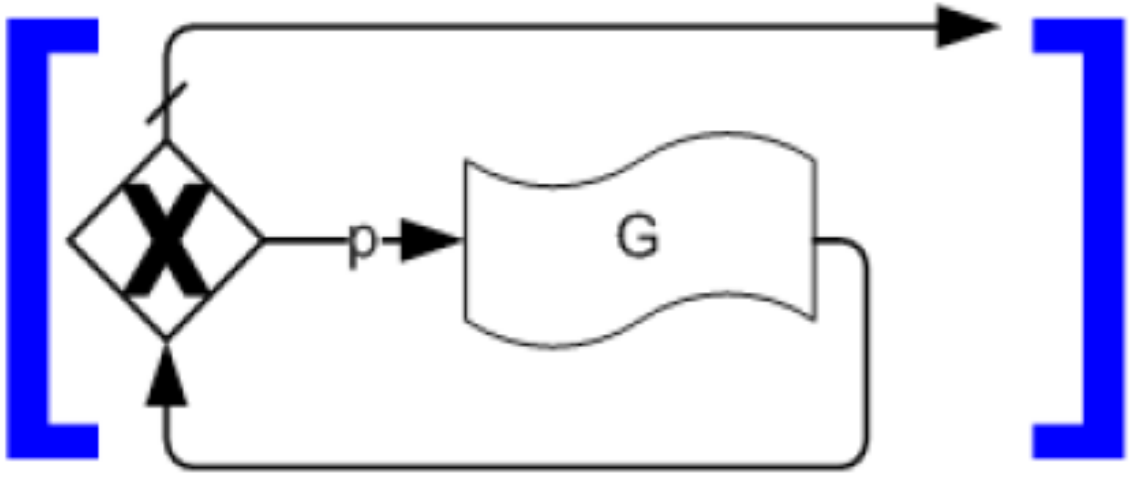
```
<while>
  <condition>[p]</condition>
  [G]
</while>
```

Analog:

=

```
<while>
  <condition>[p]</condition>
  [Task]
</while>
```

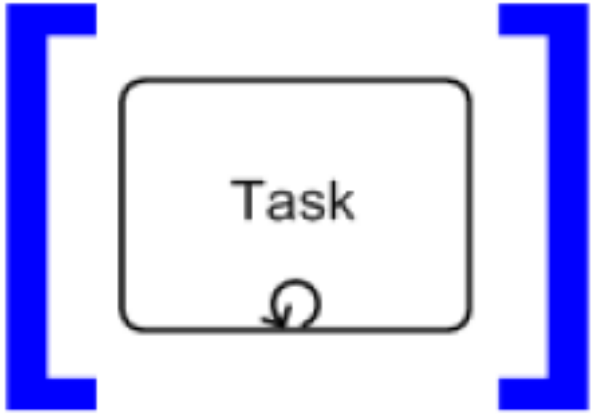
Transformation (Strukturierte Aktivitäten 6/8): While-Schleifen



=

```
<while>
  <condition>[p]</condition>
  [G]
</while>
```

Analog:



=

```
<while>
  <condition>[p]</condition>
  [Task]
</while>
```

Transformation (Strukturierte Aktivitäten 7/8): Until-Schleifen

=

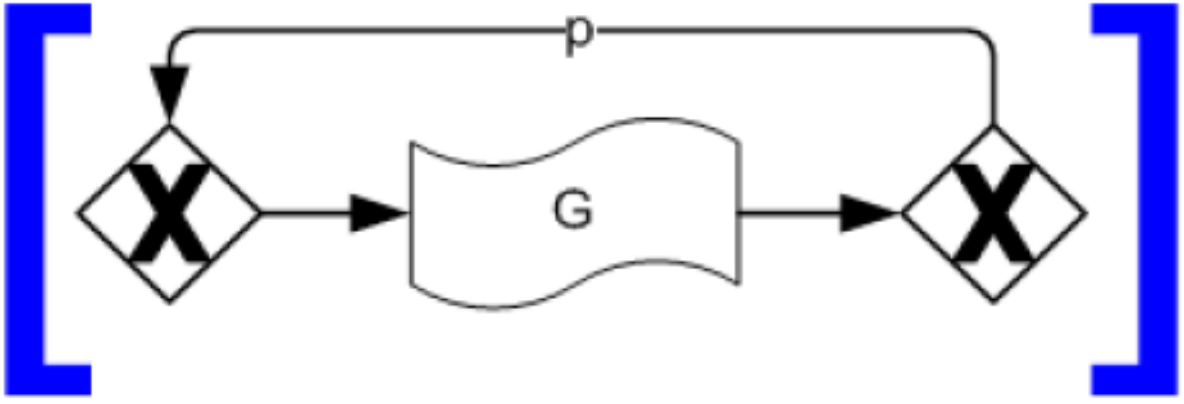
```
<repeatUntil>
  [G]
  <condition>[not p]</condition>
</repeatUntil>
```

Analog:

=

```
<repeatUntil>
  [Task]
  <condition>[not p]</condition>
</repeatUntil>
```

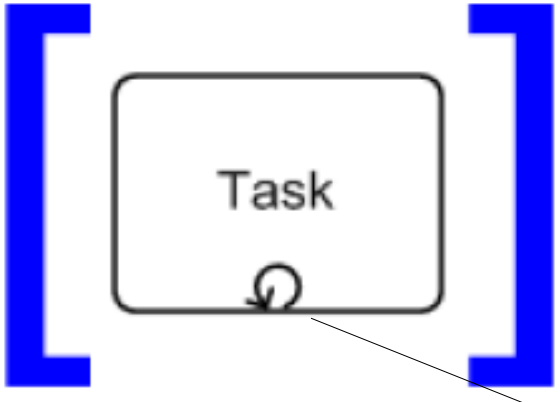
Transformation (Strukturierte Aktivitäten 7/8): Until-Schleifen



=

```
<repeatUntil>
  [G]
  <condition>[not p]</condition>
</repeatUntil>
```

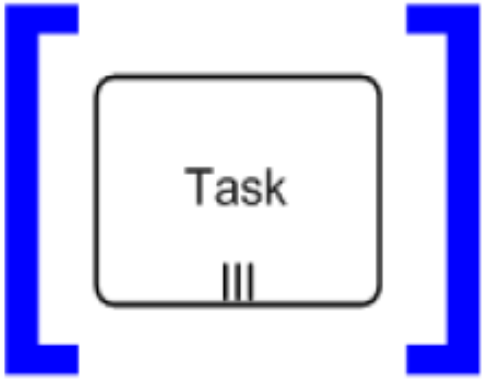
Analog:



=

```
<repeatUntil>
  [Task]
  <condition>[not p]</condition>
</repeatUntil>
```

BPMN2-Standard S. 190: „The Activity will loop as long as the boolean condition is true. The condition is evaluated for every loop iteration, and MAY be evaluated **at the beginning or at the end** of the iteration.“



=

```
<variable name="[counter]" type="xsd:integer"/>
...
<forEach counterName="[counter]" parallel="[isSequential? 'no':'yes']">
  <startCounterValue>1</startCounterValue>
  <finalCounterValue>[condition]</finalCounterValue>
  <scope>
    [Task]
  </scope>
</forEach>
```

BPEL Beispiel: PartnerLinks


```
<process name="insuranceSelectionProcess"
 targetNamespace="http://packtpub.com/bpel/example/"
 xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:ins="http://packtpub.com/bpel/insurance/"
  xmlns:com="http://packtpub.com/bpel/company/" >
    <partnerLinks>
       <partnerLink name="client"
                    partnerLinkType="com:selectionLT"
                    myRole="insuranceSelectionService"/>
       <partnerLink name="insuranceA"
                    partnerLinkType="ins:insuranceLT"
                    myRole="insuranceRequester"
                    partnerRole="insuranceService"/>
 …
 </partnerLinks>
```

partnerRole nicht festgelegt

BPEL Beispiel: Variables


```
<variables>
  <!-- input for BPEL process -->
  <variable name="InsuranceRequest"
            messageType="ins:InsuranceRequestMessage"/>
  <!-- output from insurance A -->
  <variable name="InsuranceAResponse"
            messageType="ins:InsuranceResponseMessage"/>
 …
  <!-- output from BPEL process -->
  <variable name="InsuranceSelectionResponse"
            messageType="ins:InsuranceResponseMessage"/>
 </variables>
```

Message Name von Partner Process Definition

BPEL Beispiel: Prozess (1)


```
<sequence>
      <!-- Receive the initial request from client -->
      <receive partnerLink="client"
               portType="com:InsuranceSelectionPT"
               operation="SelectInsurance"
               variable="InsuranceRequest"
               createInstance="yes" />
    <!-- Make concurrent invocations to Insurance A and B -->
      <flow>
         <!-- Invoke Insurance A web service -->
         <invoke partnerLink="insuranceA"
                 portType="ins:ComputeInsurancePremiumPT"
                 operation="ComputeInsurancePremium"
                 inputVariable="InsuranceRequest"
                 outputVariable="InsuranceAResponse" />
         ...
       </flow>
```

BPEL Beispiel: Prozess (2)


```
<!-- Select the best offer and construct the response -->
      <switch>
        <case condition="bpws:getVariableData('InsuranceAResponse',
                        'confirmationData','/confirmationData/Amount')
                     <= bpws:getVariableData('InsuranceBResponse',
                        'confirmationData','/confirmationData/Amount')">
          <!-- Select Insurance A -->
          <assign>
            <copy>
              <from variable="InsuranceAResponse" />
              <to variable="InsuranceSelectionResponse" />
            </copy>
          </assign>
        </case>
        <otherwise> … </otherwise>
      </switch>
      <!-- Send a response to the client -->
 <reply partnerLink="client" portType="com:InsuranceSelectionPT"
          operation="SelectInsurance" variable="InsuranceSelectionResponse"/>
   </sequence>
 </process>
```

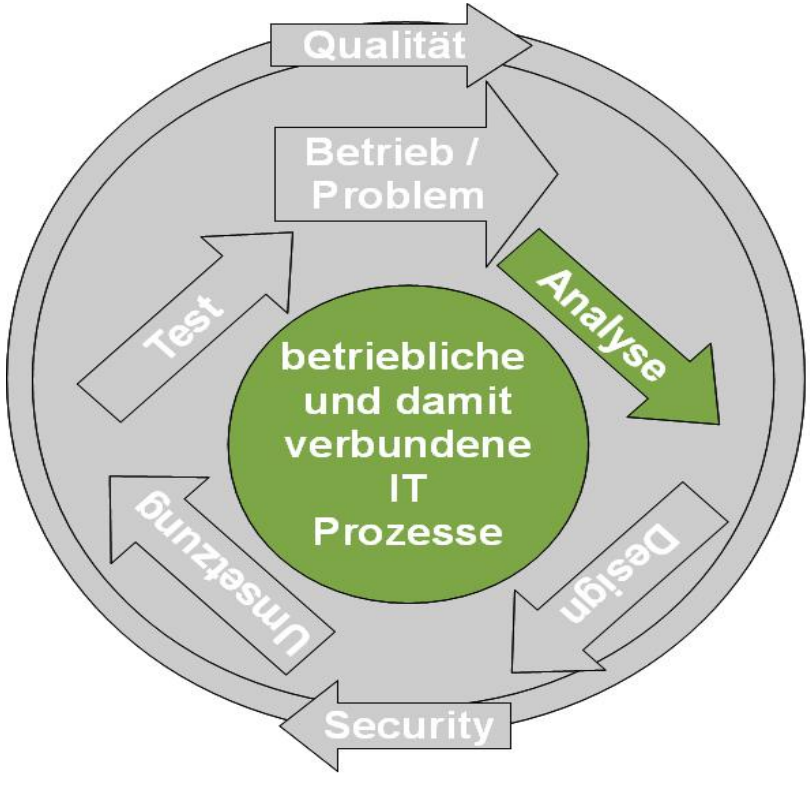
Workflow-Automatisierung: Zusammenfassung

In diesem Abschnitt:

- **Grundlagen:**
  - Natives Meta-Modell einer Workflow-Engine und Modell-Transformation.
- **BPEL** und **Transformation**: BPMN 2 nach BPEL 2:
  - Kurz-Einführung BPEL, Aktivitäten.
  - Ereignisse.
  - Strukturierte Aktivitäten.

Zusammenfassung und Ausblick: Kapitel 1: Geschäftsprozessmodellierung

- Geschäftsprozessmodellierung
  - Grundlagen Geschäftsprozesse
  - Ereignisgesteuerte Prozessketten (EPKs)
  - Einführung in die BPMN 2.0
  - Workflow-Management-Systeme
  - Workflow-Automatisierung
- Process Mining
- Modellbasierte Entwicklung sicherer Software



Anhang (Weitere Informationen zum selbständigen Nacharbeiten)

Was bestimmt das Native Metamodel einer Engine?

- Ziel: **Stabilität, Effizienz, Skalierbarkeit**.
- Modellierungssprachen: Ab **erstem Release** der Prozess-Engine unterstützen.
- Wünschenswert für natives Metamodell:
  - **Allgemeingültigkeit** und **Erweiterbarkeit**,
  - möglichst einfache Unterstützung von **Erweiterungen** existierender Modelierungssprachen.
- Natives Metamodell: Gegenüber Nutzer normalerweise **transparent**.

Konsequenzen für BPMN 2.0 Engines

- **Natives Metamodell** einer BPMN 2.0 Engine nicht notwendigerweise BPMN 2.0 Metamodell (vgl. F. 9).
  - Könnte z.B. auch BPEL sein.
- BPEL Engine, die BPMN 2.0 Modelle importieren kann, wird zu BPMN 2.0 Engine.
- Ggf. vor Import BPEL-Modell aus BPMN 2.0-Modell generieren.

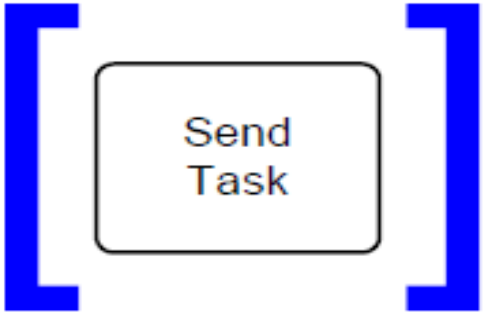
WS-BPEL: Sprachkonstrukte

- Prozessdefinition
- Rekursiver Aufbau und partnerLinks
- Variablen
- Correlation Sets
- Einfache und strukturierte Aktivitäten
- Anwendungsbereiche
- Compensation Handling

Transformation: Zusammenfassung Send / Receive

=

```
<invoke name="[Task-name]"
 partnerLink="[Q, Task-operation-interface]"
 portType="[Task-operation-interface]"
 operation="[Task-operation]">
 <correlations>
  <correlation set="[Task-messageFlow-conversation-correlationKey]"
      initiate="[initialInConversation? 'join':'no']"/>
 </correlations>
</invoke>
```



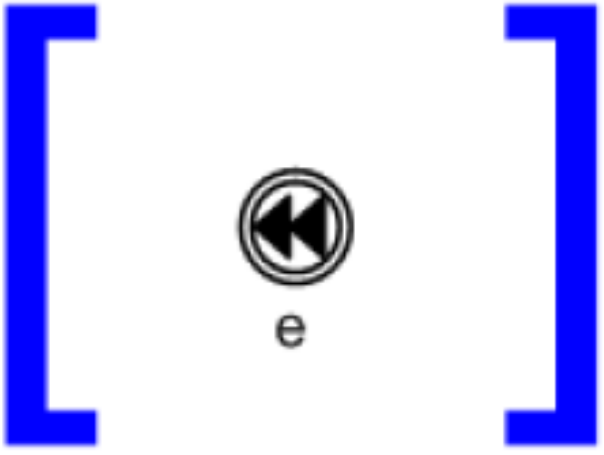
=

```
<invoke name="[Task-name]"
 partnerLink="[Task-serviceRef]"
 portType="[Task-operation-interface]"
 operation="[Task-operation]">
</invoke>
```

=

```
<receive name="[Task-name]"
 createInstance="[instantiate? 'yes':'no']"
 partnerLink="[Task-serviceRef]"
 portType="[Task-operation-interface]"
 operation="[Task-operation]">
</receive>
```

Transformation: Kompensationsereignis

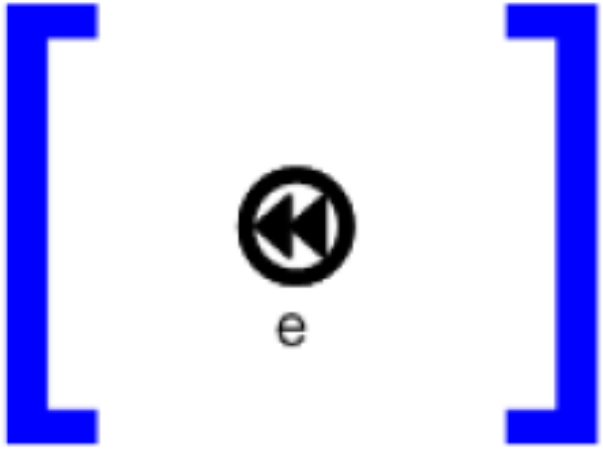


=

<compensate/>

or

<compensateScope target="[referencedActivity]"/>



=

<compensate/>

or

<compensateScope target="[referencedActivity]"/>

Transformation: Überblick Ereignisse