LEHRSTUHL 14
SOFTWARE ENGINEERING

Vorlesung

Methodische Grundlagen des Software-Engineering

im Sommersemester 2013

Prof. Dr. Jan Jürjens

TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

Teil 1.4: Workflow-Automatisierung

v. 30.04.2013

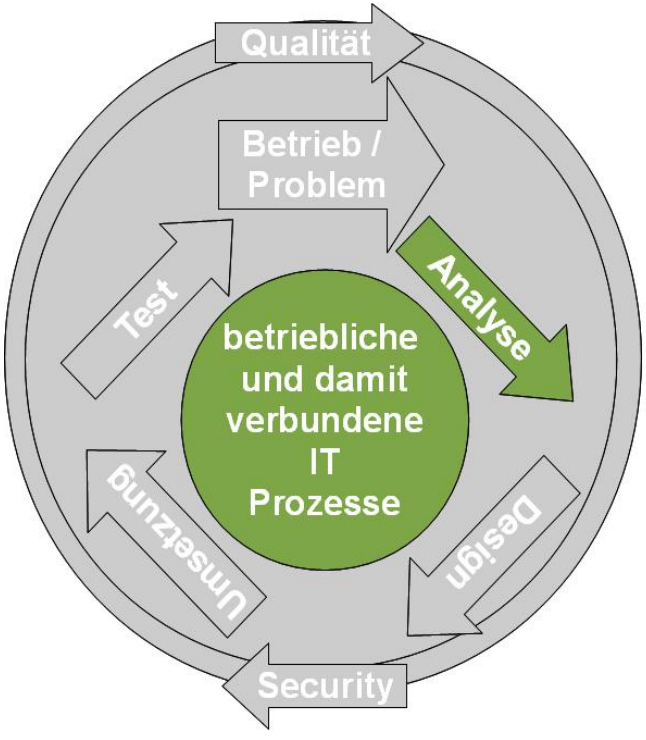
1.4 Workflow-Automatisierung

[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 Softwareentwicklung
- Modellbasierte Entwicklung sicherer Software



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

Einleitung
Workflow-Automatisierung

Letzter Abschnitt: Ausführung von Workflows („Workflow Management“) und Systeme, die dies unterstützen (Workflow-Management-Systeme).

Insbesondere **Workflow-Engines** im Kontext von Workflow-Management-Systemen.

Dieser Abschnitt: „Workflow-Automatisierung“, insbesondere Übersetzung von BPMN-Modellen in Business Perocess Execution Language (BPEL).

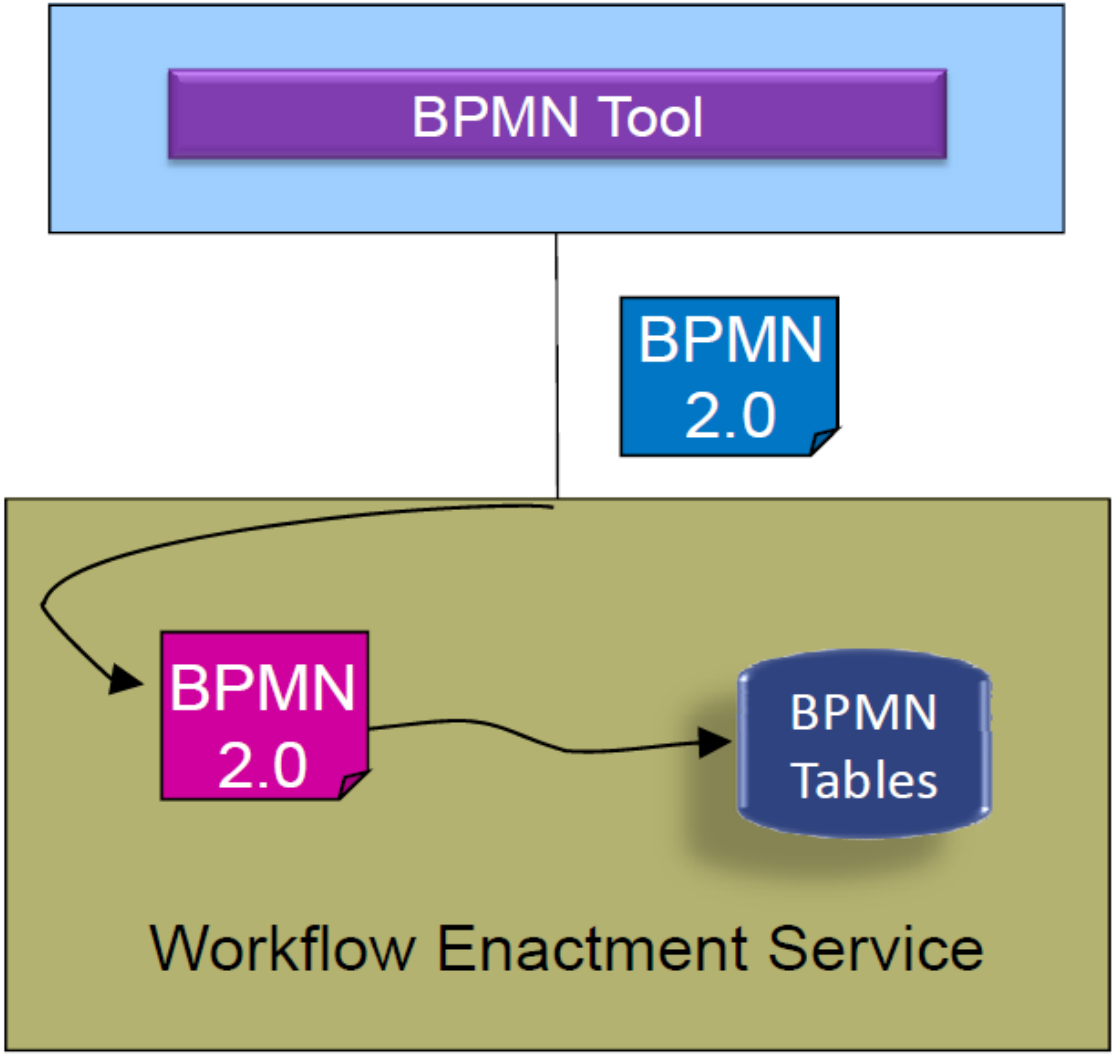
Insbesondere BPMN- / BPEL-Engines.

Workflow-Engines und Meta-Modelle

- **Metamodell**: Definition einer Modellierungssprache, die selber als Modell gegeben wird.
- Insbesondere verwendet zur Definition des internen Modell-Speicherformats einer Workflow-Engine (sogenanntes „**Natives Metamodell**“).
- **BPEL-** (bzw. **BPMN-**) **Engine:** Workflow-Engine, die BPEL- (bzw. BPMN-) Prozessmodelle importieren kann.
- **„Native“ BPEL- (bzw. BPMN-) Engines**: BPEL (bzw. BPMN) ist internes („natives“) Metamodell.

Native BPMN 2.0 Engines

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



Native Unterstützung eines Metamodells

Üblicherweise wird **natives Metamodell**...

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

- Datenbankschema enthält sofort Instanzen des Metamodellkonstrukts
- Datenbankschema ist zur Unterstützung des Navigators angepasst.

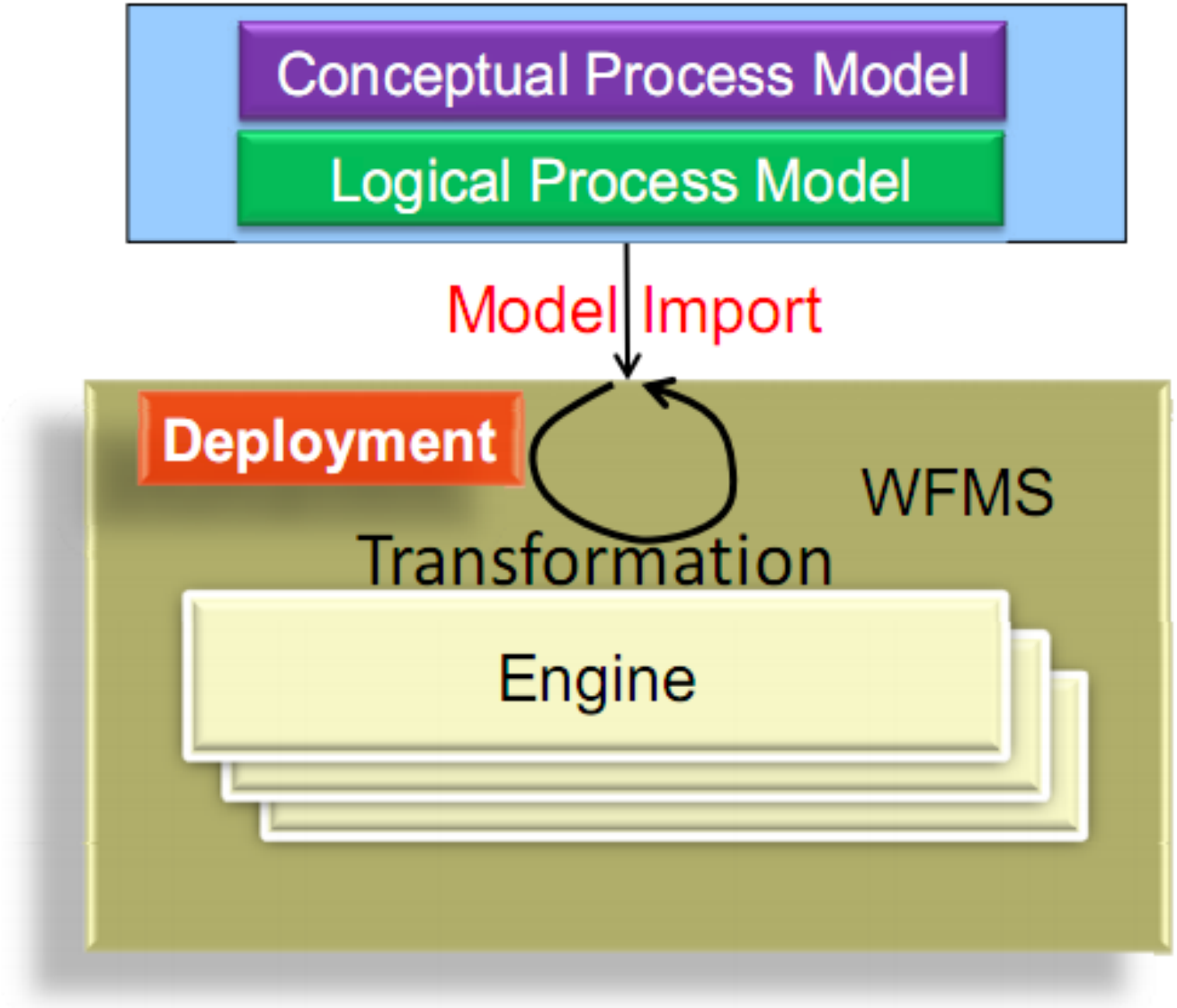
… direkt unterstützt in **Zustandsmodell des WFMS:**

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

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

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

Vom Prozess-Modell zur Workflow-Engine



Anwendung des Modells: Deployment

Deployment: Prozessmodell produktiv schalten.

- z.B. bereit für die Ausführung machen

Anwendung übersetzt normalerweise Modell in **unterschiedliche Formate**.

- Importiertes Modell kann in **anderem Metamodell** spezifiziert sein, als nativem Metamodell des WFMS.
- Muss dann in dieses umgewandelt werden (**Transformation** während des Modell-Deployments, Beispiele siehe folgende Folien).

Modell-Deployment: Überblick


```
<process ...>
 <sequence>
   ...
   <flow>
    ...
   </flow>
 </sequence>
</process>
```

Deployment

Navigator

Model Cache

Navigation Logic

Instance Cache

...

Process Engine Database

Activity			
Name	AID	JCond	...

Control Link			
Source	Target	TCond	...

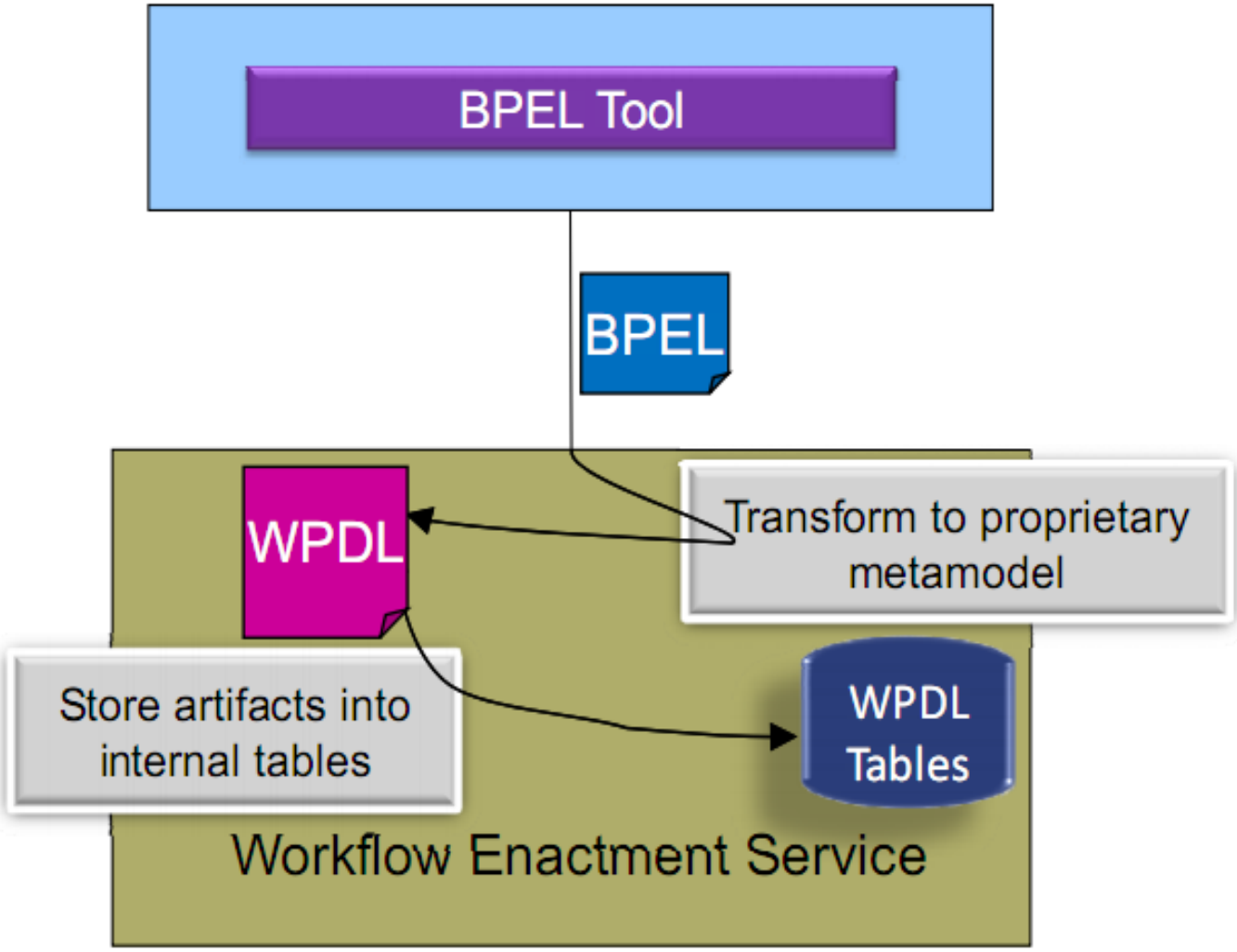
...

Activity Instance			
PIID	AIID	AIID	...

Transformation während Modell-Deployment: BPEL => WPDL

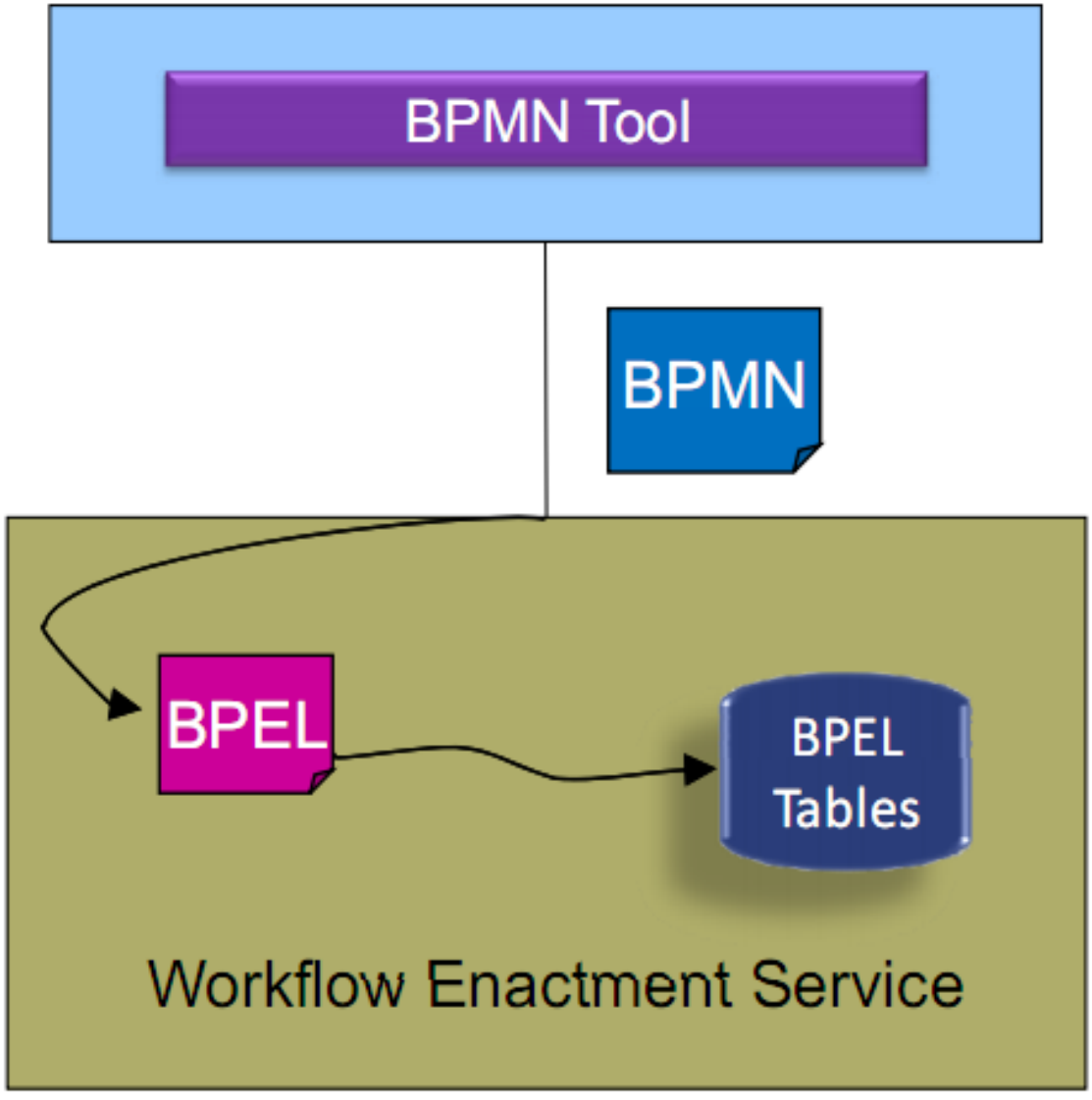
WFMS, das **WPDL**[1] nativ unterstützt, kann **BPEL**-Modell nach zugehöriger Transformation ausführen.

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



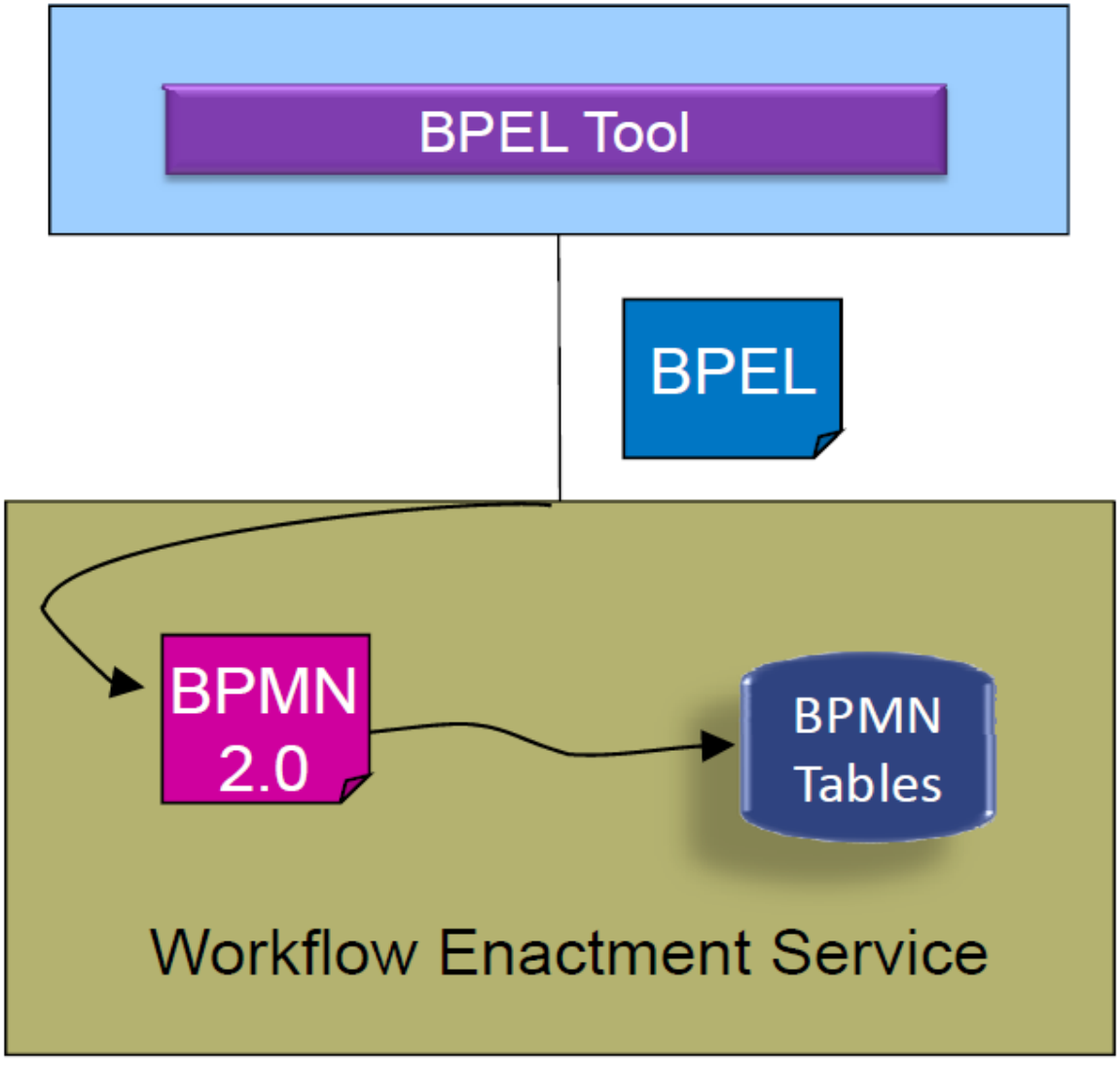
Transformation während Modell-Deployment: BPMN => BPEL

WFMS, das **BPEL** nativ unterstützt, kann **BPMN**-Modell nach zugehöriger Transformation ausführen.



Transformation während Modell-Deployment: BPEL => BPMN

Native **BPMN 2.0**-Engines können nach Transformation auch **BPEL** unterstützen.



Ausführbarkeit von BPMN-2.0-Modellen

- **XML-Schema** für BPMN 2.0:
  Zum Speichern und Weiterverarbeiten der Modelle.
- Enthält auch für **Ausführung** relevante Informationen.
- Grafische Notation muss für Ausführung um diese Informationen angereichert werden.
- Zugehörige syntaktische Details in BPMN 2.0 Spezifikation durch UML-Klassendiagramme oder XML-Schema-Definitionen definiert.

Diskussion: Probleme der Modell-Transformationen

Welche **Herausforderungen** könnten sich bei der **Modell-Transformation** zwischen verschiedenen GP-Modellierungs-Notationen (wie BPMN, WPDL, BPEL) ergeben ?

Diskussion: Probleme der Modell-Transformationen

Welche **Herausforderungen** könnten sich bei der **Modell-Transformation** zwischen verschiedenen GP-Modellierungs-Notationen (wie BPMN, WPDL, BPEL) ergeben ?

Antwort: Transformationen nicht immer **verhaltensbewahrend**:

Semantisches Verhalten: Verhalten des Ausgangsmodells muss in Zielmodell emuliert werden, soweit möglich.

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

Operatives Verhalten: Navigator führt transformiertes Modell oft weniger effizient aus, als WFMS mit Metamodell des Originalmodells.

- z.B.: Unterstützung eines FDL-Datenflusses in BPEL schwerfällig (FDL = Flow Definition Language (vgl. IBM Websphere)).

Diskussion: Transformation vs. Lock-in

Kann die Wahl der **Workflow-Engine** (und damit der enthaltenen **Transformation**) zu einem **Lock-in-Effekt** führen ?

Diskussion: Transformation vs. Lock-in

Kann die Wahl der **Workflow-Engine** (und damit der enthaltenen **Transformation**) zu einem **Lock-in-Effekt** führen ?

Antwort:

Bei Transformation oft **Änderung** des Prozesses (vgl. F. 18).

=> Prozess-Modellierung teilweise auf Transformation abgestimmt, um gewünschtes Ergebnis zu erhalten.

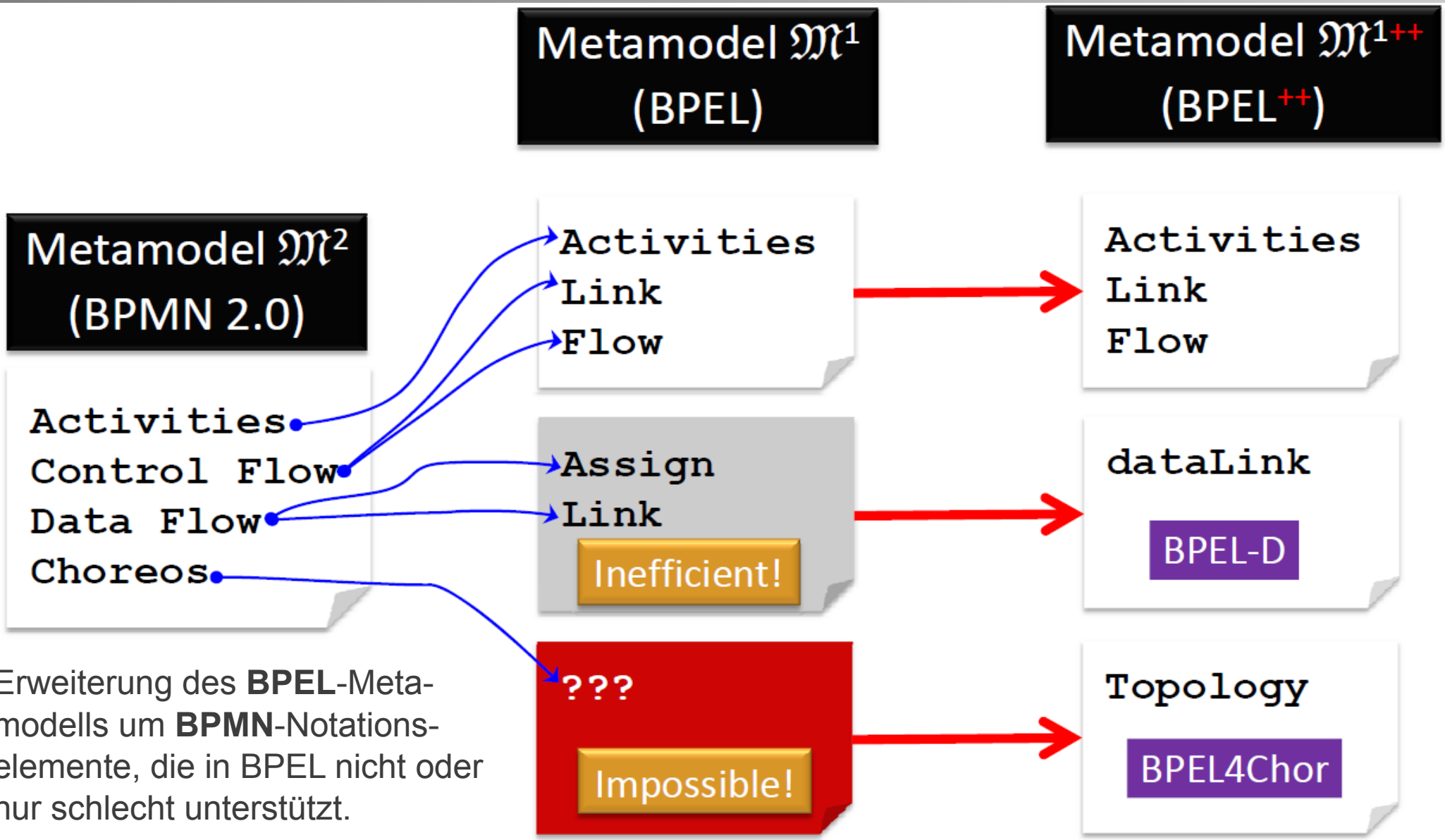
Damit können Prozessmodelle auf anderen Workflow-Systemen ein anderes, nicht gewünschtes Verhalten zeigen.

Neumodellieren aufwendig => Lock-in (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**.
- Dafür BPEL **erweiterbar** entworfen:
  Kann neue Konstrukte hinzufügen für 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



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 die Erweiterung wird Ziel-Metamodell immer **komplexer**.
- „Schwer im Zielmodell emulieren“ heisst nicht immer „unmöglich zu emulieren“. Erweiterung des Ziel-Metamodelles führt daher zu (zumindest partieller) **Redundanz**, die eigentlich zu vermeiden ist.

Was bestimmt das Native Metamodel einer Engine?

- Ziel: **Stabilität, Effizienz, Skalierbarkeit**.
- Modellierungssprachen müssen ab **erstem Release** der Prozess-Engine unterstützt werden.
- Wünschenswert für natives Metamodell:
  - **Allgemeingültigkeit** und **Erweiterbarkeit**,
  - möglichst einfache Unterstützung von **Erweiterungen** existierender (und sogar von neuen) Modelierungssprachen.
- Natives Metamodell (und Unterschiede zu BPMN/BPEL-Metamodell) ist gegenüber Nutzer normalerweise **transparent** (= unsichtbar).

Konsequenzen für BPMN 2.0 Engines

- **Natives Metamodell** einer BPMN 2.0 Engine oft nicht BPMN 2.0 Metamodell (wegen der Trade-offs seitens der Hersteller, 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.

Natives Metamodell vs. Stabilität, Effizienz, Skalierbarkeit

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

- Anbieter von aktuellen BPEL Engines haben viel in nicht-funktionale Eigenschaften ihrer Engines investiert.
- Können BPMN 2.0 Engines mit ähnlichen nicht-funktionalen Eigenschaften anbieten.

DBMS-Analogie:

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

=> OODBMS sind **kein „Mainstream“** mehr.

Weitere Entwicklung im Bereich Process-Engines

- BPMN 2.0 signifikant **komplexer** als BPEL.
- Werkzeuge unterstützen nur **Teile** von BPMN 2.0, basierend auf Kunden-Anforderungen.
- Vermutlich wird kein Werkzeug alle Aspekte der BPMN 2.0 unterstützen.
- **BPEL-Engines** werden erweitert um fehlende Schlüsseleigenschaften von BPMN 2.0.
- Mittelfristig wird **BPEL** erweitert um fehlende BPMN 2.0-Eigenschaften.

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

Beispiel für Modell-Transformation: BPMN nach BPEL

Hier 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**.

Oder: **Visualisierung** der **BPEL** als Teil von BPMN 2.0.

BPEL

- Kurz für: **Web Services Business Process Execution Language (WS-BPEL)**
- **XML-basierte** textuelle Notation („Markup language“), um Services in Prozess-Fluss zusammenzufügen.
- 2003: BEA Systems, IBM, Microsoft, SAP, Siebel Systems schlagen BPEL4WS 1.1 für Standardisierung durch OASIS vor.
  - OASIS: Organization for the Advancement of Structured Information Standards. Globales Konsortium für Standardisierung im Bereich e-Business und Web-Service
- 2007: **WS-BPEL 2.0 Standard** (OASIS)
- 2007: Active Endpoints, Adobe Systems, BEA, IBM, Oracle, SAP: BPEL4People, WS-HumanTask (menschliche Interaktion in BPEL)

WS-BPEL Einführung

Hier: **vereinfachter Ausschnitt** der Notation.

- **Definition der Notation** (**top-down**: erst Prozessdefinition, dann elementare Notationselemente)
- **Transformation BPMN => BPEL**
  (Gleichzeitig mit Notationsdefinition: dadurch implizit Semantik.)

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**
(also **veraltet**, nicht immer klar zu erkennen).

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

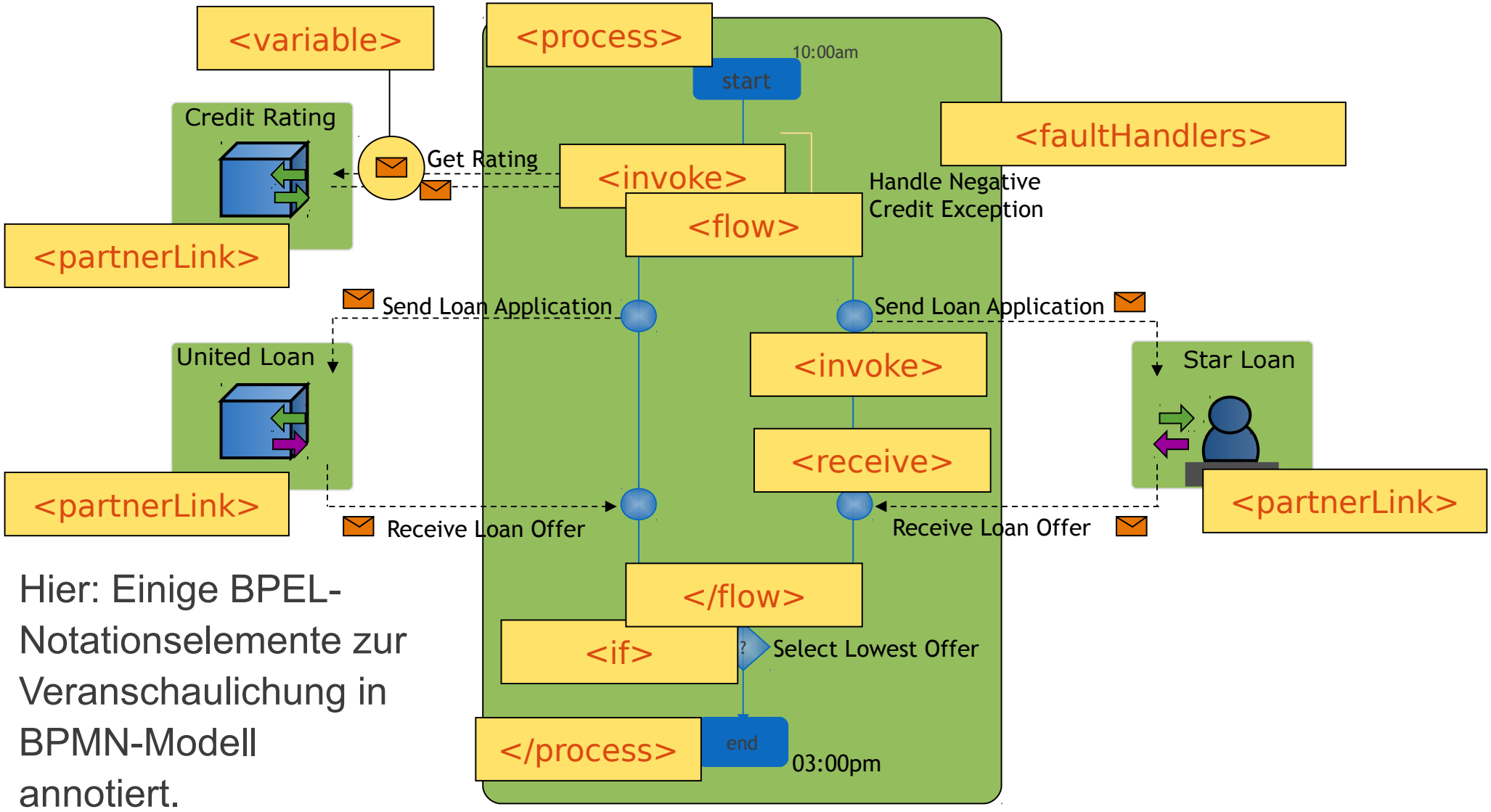
Grundprinzip: Transformation BPMN => BPEL

- **Funktion [...]**: bildet Untermenge von BPMN auf BPEL ab.
- **Rekursiv** definiert.
- **Induktionsanfang**:
  - [t] für **elementare BPMN-Aufgaben** t, die auf BPEL abgebildet werden.
  - [e] für **elementare BPMN-Ereignisse** e, die auf BPEL abgebildet werden.
- **Induktionsschritt:**
  - [s] für **BPMN-Strukturen** s, die direkte Abbildung auf BPEL haben.
- Definiert konstruktiv BPMN-Untermenge, die abgebildet werden kann, sowie Abbildung [...].

WS-BPEL: Sprachkonstrukte

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

BPEL: Überblick, veranschaulicht mit BPMN

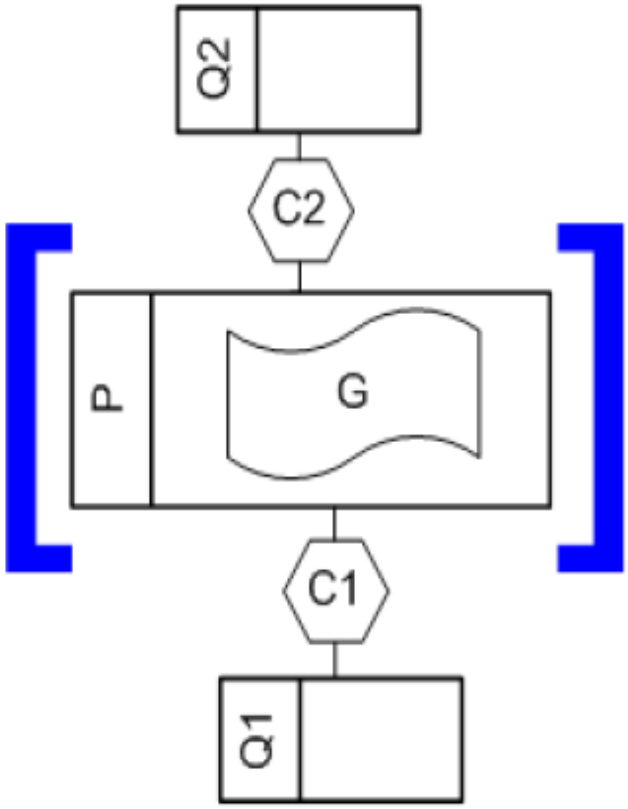


Hier: Einige BPEL-Notationselemente zur Veranschaulichung in BPMN-Modell annotiert.

Struktur eines WS-BPEL-Prozesses


```
<process ...>
<!-- 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>
```

Transformation BPMN => BPEL: Prozess

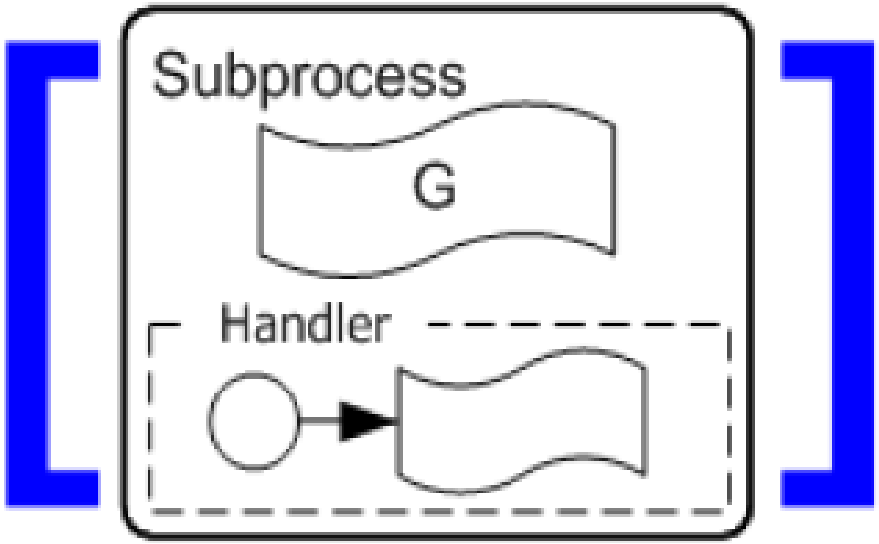


=

```
<process name="[P-name]"
   targetNamespace="[targetNamespace]"
   expressionLanguage="[expressionLanguage]"
   suppressJoinFailure="yes"
   xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable">
  <partnerLinks>
    [ {P-Interfaces} UNION {Qi-Interfaces} ]
  </partnerLinks>
  <variables>
    [ {dataObjects} UNION {properties} ]
  </variables>
  <correlationSets>
    [ {Ci-CorrelationKeys} ]
  </correlationSets>
  [G]
</process>
```

Anm.: C1, C2 sind BPMN „Conversations“ (vgl. [FR12] Abs. 2.13).

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

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

partnerLinkType definiert zwei **Rollen** 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

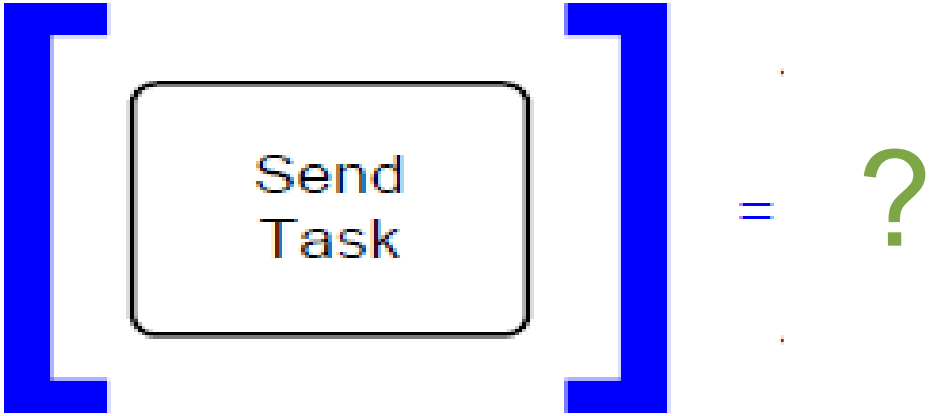
Process sendet **Antwortnachricht** in Partneraufruf:

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

Datenbelegung zwischen **Variablen**:

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

Transformation: Operation aufrufen



= ?

Prozess aktiviert Operation bei Partner:

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

Prozess erhält Aufruf des Partners: `<receive name="..." [createInstance="..."]`

```
partnerLink="..." portType="..." operation="..." variable="..."/>
```

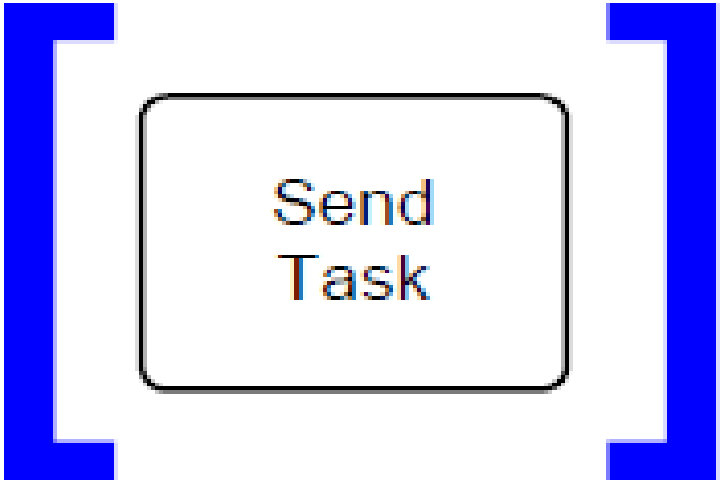
Prozess sendet Antwortnachricht in Partneraufruf: `<reply name="..."`

```
partnerLink="..." portType="..." operation="..." variable="..."/>
```

Datenbelegung zwischen Variablen: `<assign> <copy> <from variable="..."/>`

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

Transformation: Operation aufrufen




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

Transformation: Operation aufrufen / empfangen

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>
```

Transformation: Operation aufrufen / empfangen

=

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

Korrelationsmengen

Kann mehrere **Interaktions-Arten** modellieren:

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

Für Letzteres: **Korrelationsmengen** (Correlation Sets, CSs):

- Repräsentieren Daten, um **Zustand der Interaktion** (eine "**Konversation**") aufrechtzuerhalten.
- Ermöglichen ankommenden Nachrichten, richtige Prozessinstanzen erreichen.

Was genau ist **Korrelationsmenge** ?

- **Menge von Geschäftsdatenfelder** für Interaktions-Zustand ("correlating business data"). Zum Beispiel: "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

= ?

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>`

Input- / Output-Operation ordnet Korrelationsmenge zur gesendeten / empfangenen **Nachricht** zu **(analog für invoke statt receive !)**:

```
<receive partner="..." operation="..." portType="..." variable="..." >
 <correlations> <correlation set="..." initiate="..."/>
 </correlations> </receive>
```

Transformation: Operation aufrufen / empfangen

=

```
<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: Zusammenfassung Send / Receive

[Service Task] =

```
<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>
```

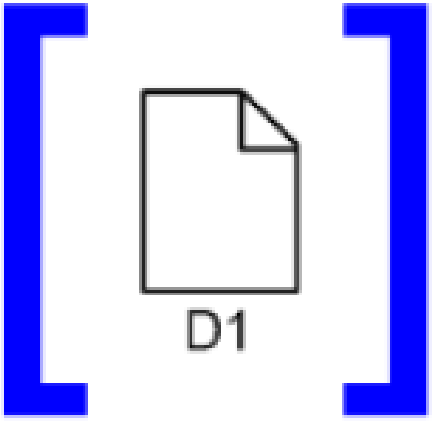
[Send Task] =

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

[Receive Task] =

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

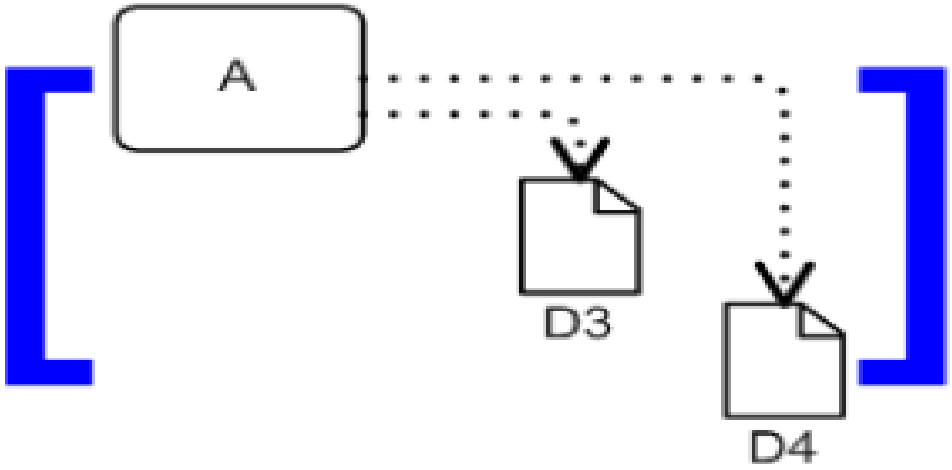
Transformation: Dokumente / Variablen



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

Transformation: Eingabe, Ausgabe

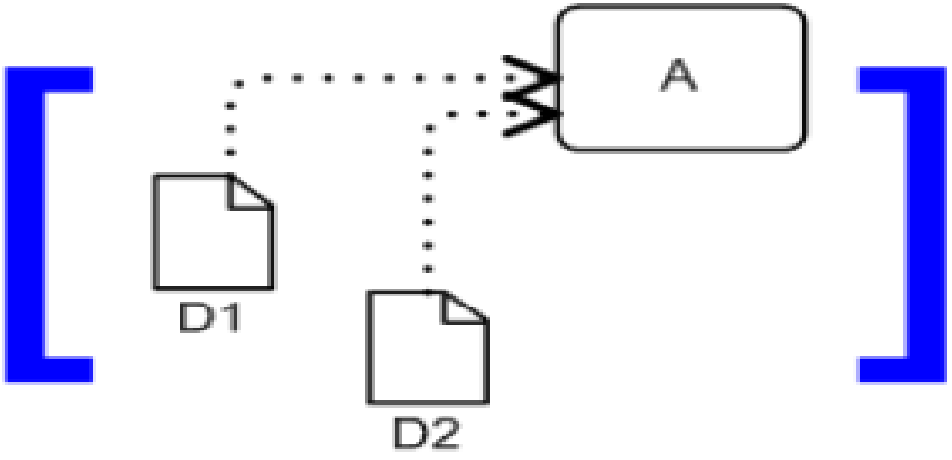
Ausgabe aus Aktivität erhalten:



=

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

Aktivität mit Eingabe aufrufen:

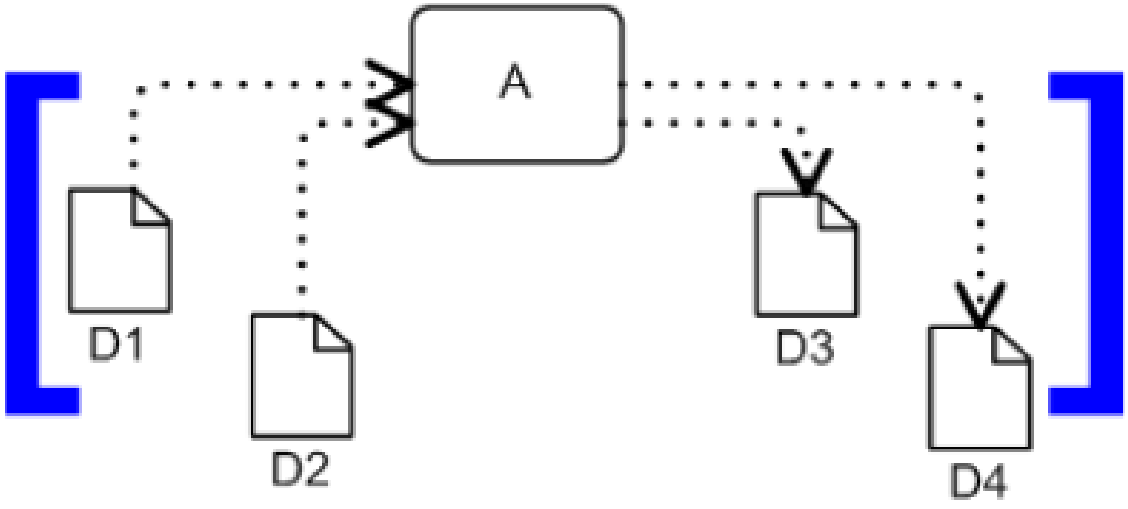


=

```
<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:



=

```
<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**:

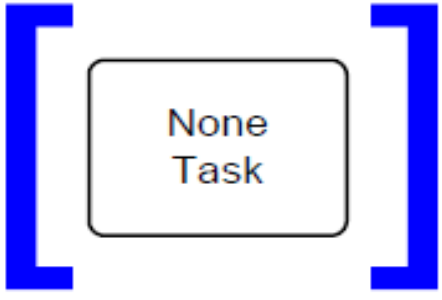
```
<exit></exit>
```

Prozess **stoppt** für bestimmte Zeit:

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

Nichts tun:

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



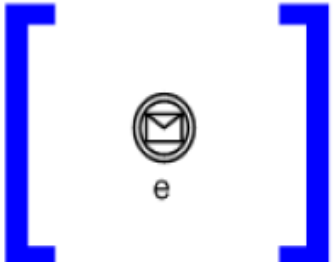
= <empty name="[Task-name]">
</empty>

Workflow-Automatisierung: Überblick

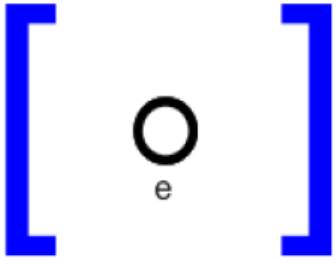
- Grundlagen
  - Natives Meta-Modell einer Workflow-Engine und Modell-Transformation
- BPEL und Transformation: BPMN 2 nach BPEL 2
  - Kurz-Einführung BPEL, Aktivitäten
  - Ereignisse
  - Strukturierte Aktivitäten

Transformation: Ereignisse

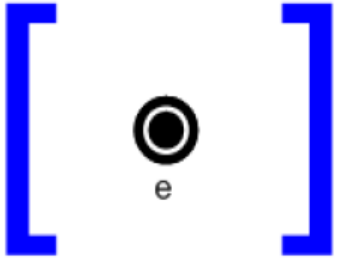
= ?



= ?



= ?



= ?

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>`

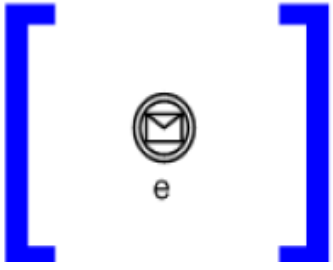
Prozess stoppt für bestimmte Zeit: `<wait name="..."> <for>"..."</for></wait>`

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

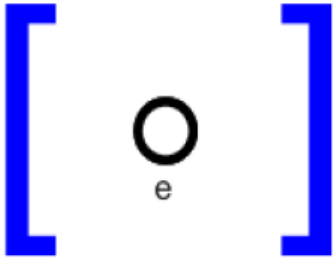
Transformation: Ereignisse

=

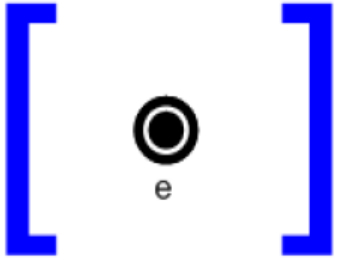
```
<receive name="[e-name]"
  createInstance="yes"
  partnerLink="[e-operation-interface]"
  portType="[e-operation-interface]"
  operation="[e-operation]">
</receive>
```



= ?



= ?



= ?

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>`

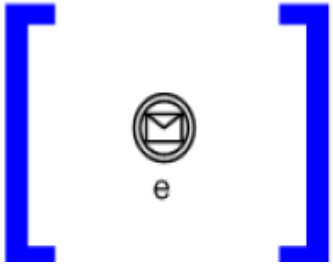
Prozess stoppt für bestimmte Zeit: `<wait name="..."> <for>"..."</for></wait>`

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

Transformation: Ereignisse

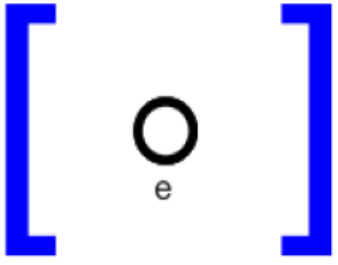
=

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

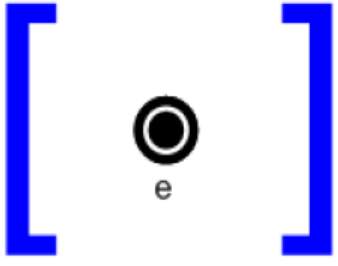


=

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



= ?



= ?

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="..."/>`

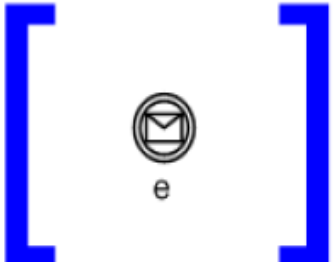
Prozess beenden: `<exit></exit>`

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

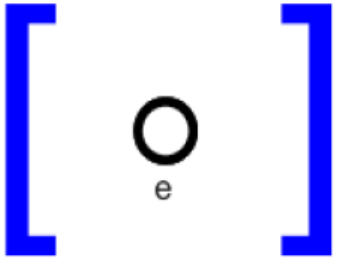
Nichts tun: `<empty name="..."> </empty>`

Transformation: Ereignisse

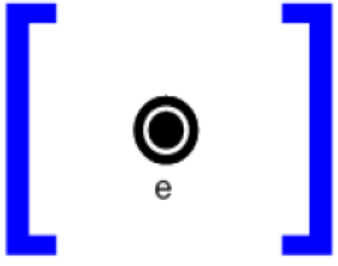

```
<receive name="[e-name]"
  createInstance="yes"
  partnerLink="[e-operation-interface]"
  portType="[e-operation-interface]"
  operation="[e-operation]">
</receive>
```




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




```
<empty name="[e-name]">
</empty>
```



?

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="..."/>`

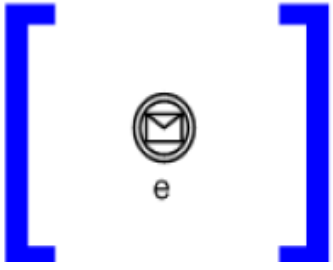
Prozess beenden: `<exit></exit>`

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

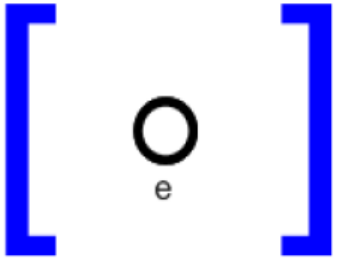
Nichts tun: `<empty name="..."> </empty>`

Transformation: Ereignisse

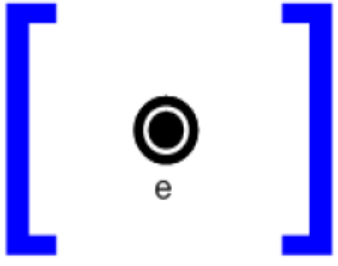

```
<receive name="[e-name]"
  createInstance="yes"
  partnerLink="[e-operation-interface]"
  portType="[e-operation-interface]"
  operation="[e-operation]">
</receive>
```




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




```
<empty name="[e-name]">
</empty>
```




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

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:

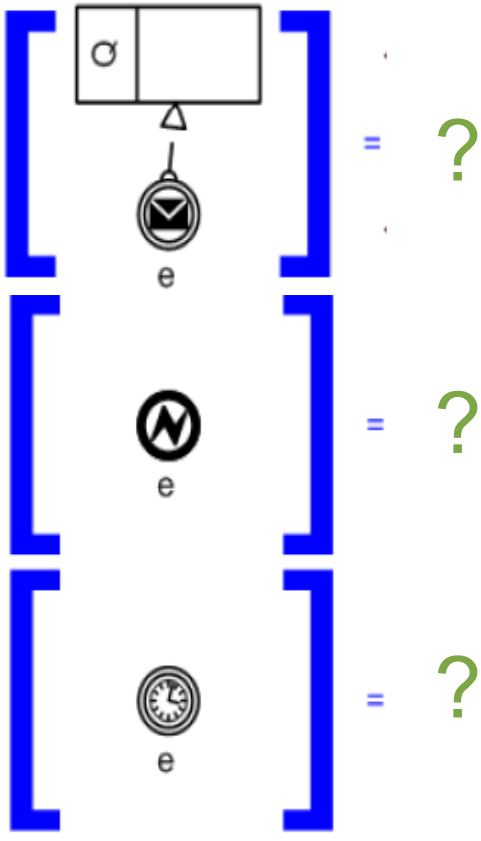
```
<assign> <copy> <from variable="..."/>
<to variable="..."/> </copy>+ </assign>
```

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:

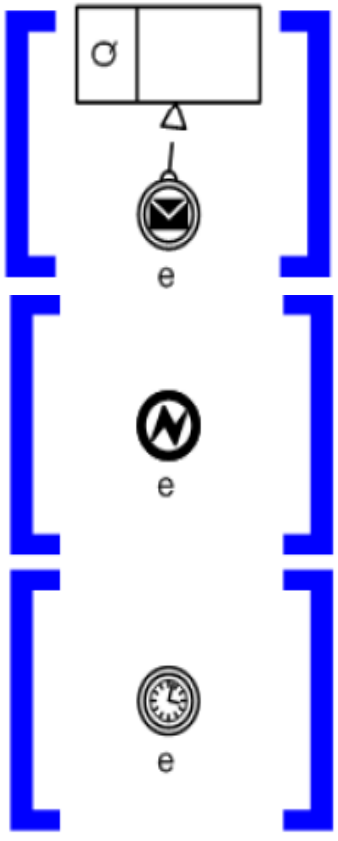
```
<assign> <copy> <from variable="..."/>
<to variable="..."/> </copy>+ </assign>
```

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>
```

= ?

= ?

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:

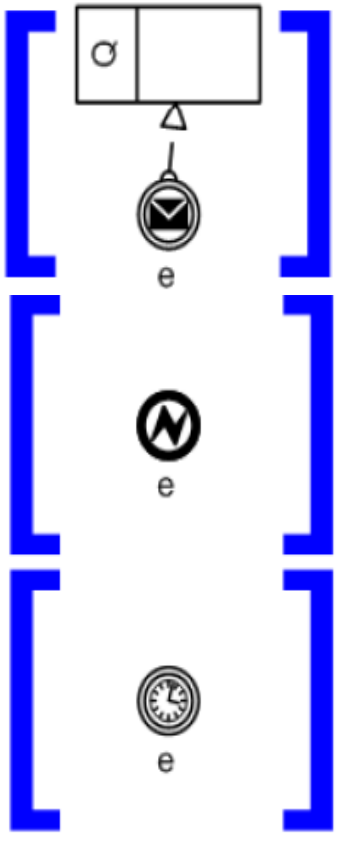
```
<assign> <copy> <from variable="..."/>
<to variable="..."/> </copy>+ </assign>
```

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:

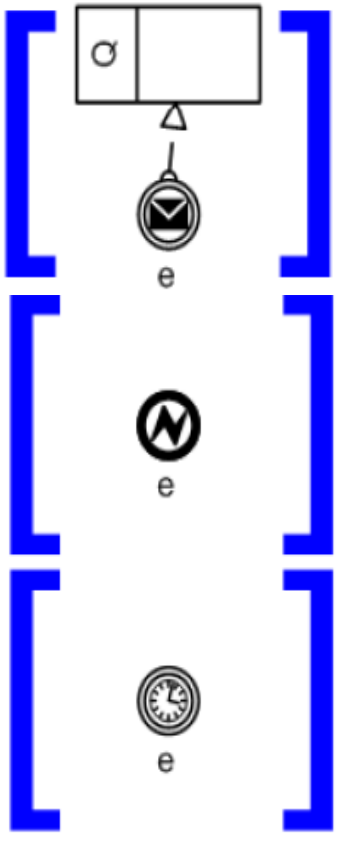
```
<assign> <copy> <from variable="..."/>
<to variable="..."/> </copy>+ </assign>
```

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>`




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

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

```
  <wait name="[e-name]" for="[e-TimeCycle]"/>
=                    or
  <wait name="[e-name]" until="[e-TimeDate]"/>
```

Transformation: Kompensationsereignis

[e] = <compensate/> *or* <compensateScope target="[referencedActivity]"/>

[e] = <compensate/> *or* <compensateScope target="[referencedActivity]"/>

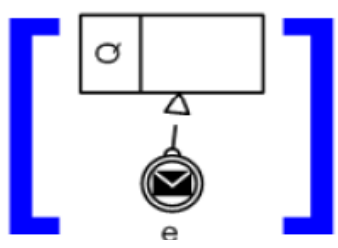
Transformation: Überblick Ereignisse


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

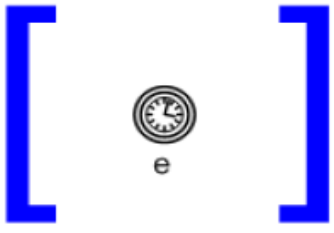

```
<invoke name="[e-name]"
  partnerLink="[Q, e-operation-interface]"
  portType="[e-operation-interface]"
  operation="[e-operation]">
</invoke>
```


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


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




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




```
<wait name="[e-name]" for="[e-TimeCycle]"/>
                    or
<wait name="[e-name]" until="[e-TimeDate]"/>
```

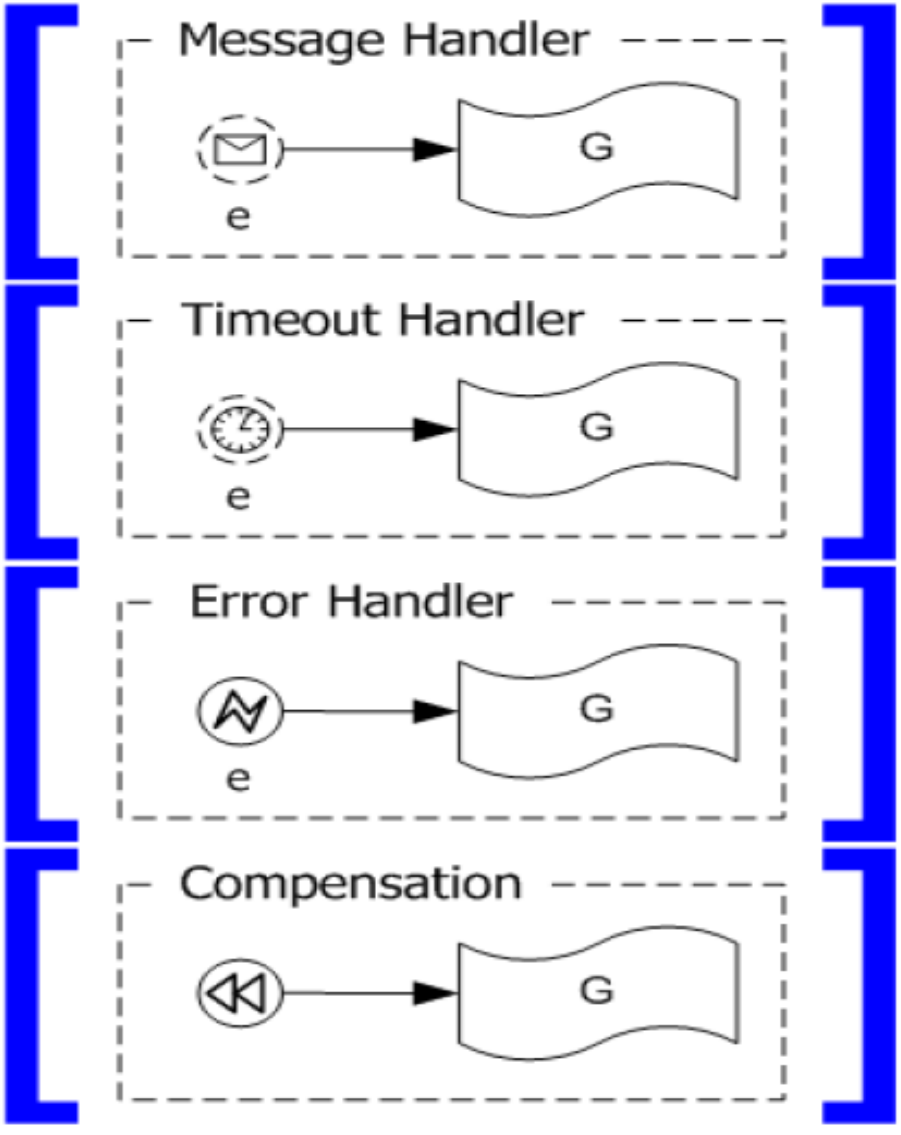

```
<empty name="[e-name]">
</empty>
```


```
<compensate/>
                    or
<compensateScope target="[referencedActivity]"/>
```


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


```
<compensate/>
                    or
<compensateScope target="[referencedActivity]"/>
```

Ereignisverarbeitung (Event Handler)




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

=

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

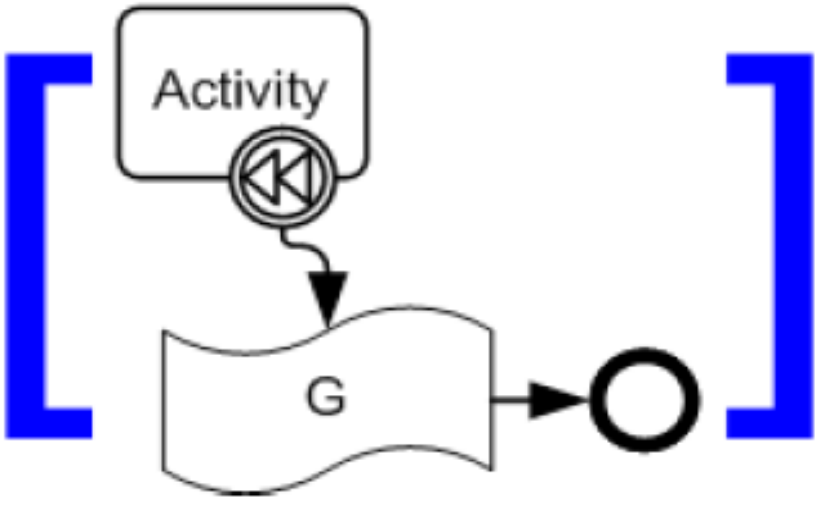
=

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

=

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

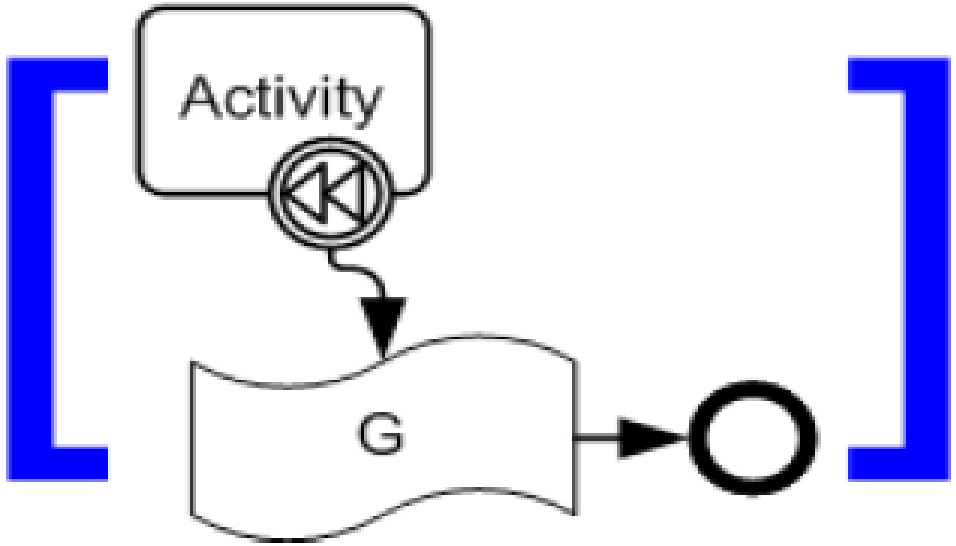
Rand-Ereignis (1/4)



= ?

```
<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>
```

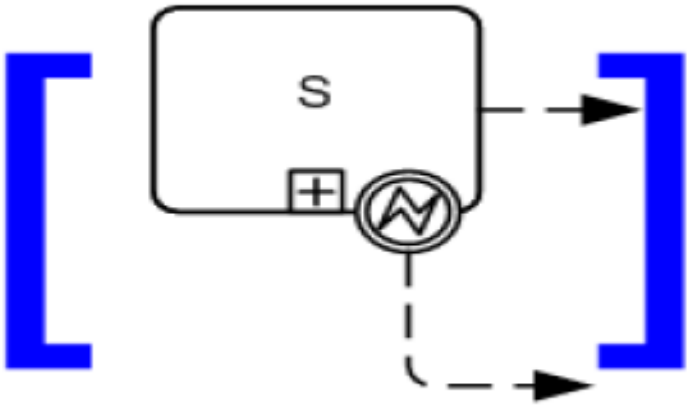
Rand-Ereignis (1/4): Kompensation



=

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

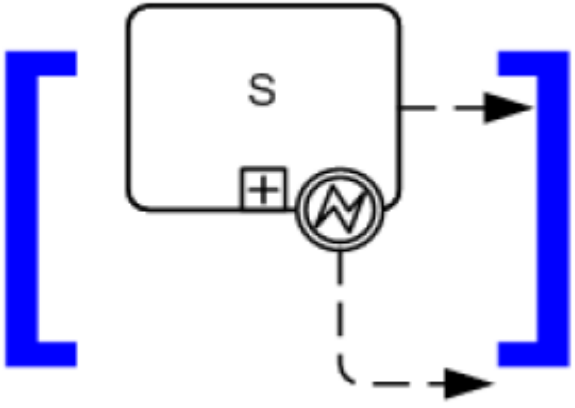
Rand-Ereignis (2/4)



= ?

```
<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>
```

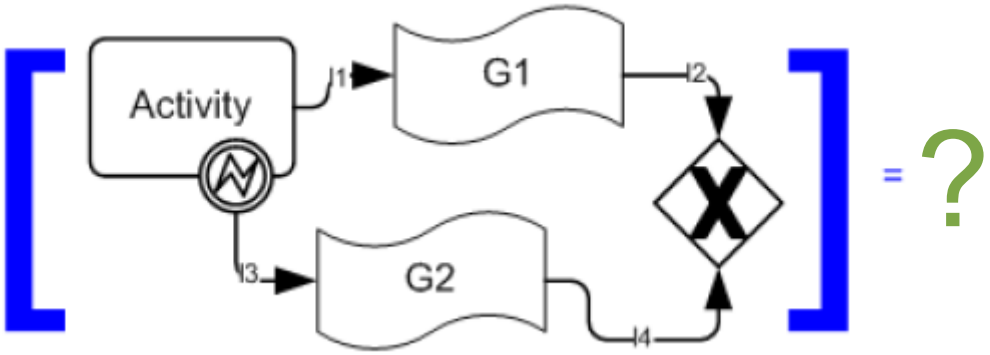
Rand-Ereignis (2/4): Fehlerbehandlung



=

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

Rand-Ereignis (3/4)



Definiert Kontrollzusammenhang zwischen Aktivitäten:

```
<links>
  <link name="[l1]"/>
  ...
  <link name="[l4]"/>
</links>
```

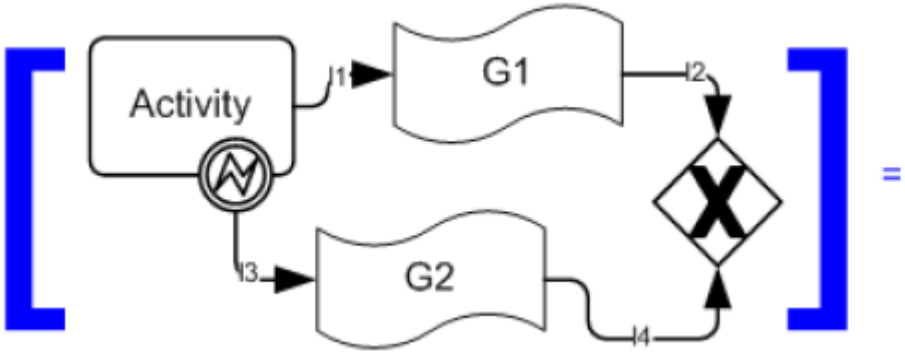
```
<eventHandlers>
  <onEvent partnerLink="[e-operation-interface]
      operation="[e-operation]">
    <scope>[G]</scope>
  </onEvent>
</eventHandlers>
```

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

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

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

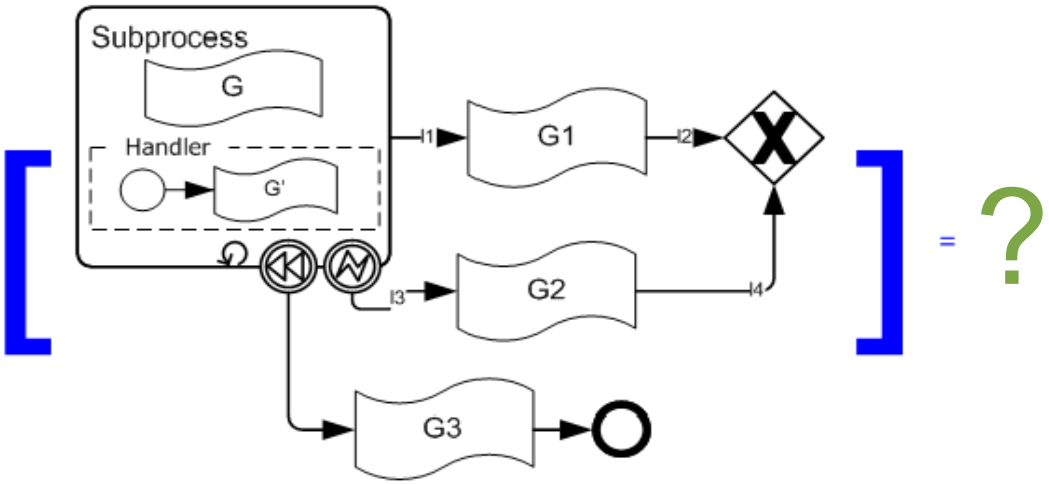
Rand-Ereignis (3/4): Fehler-Behandlung vs. Normaler Fluss




```
<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>
```

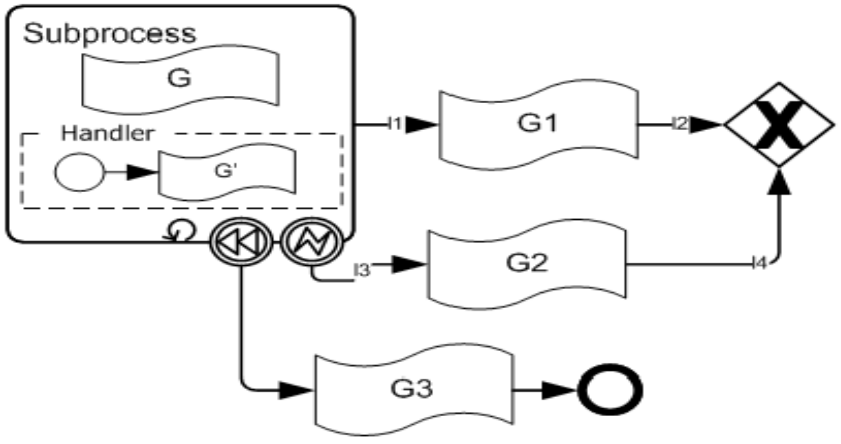
Rand-Ereignis (4/4)




```
<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 (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

Strukturierte Aktivitäten: Sequenzielles / Paralleles Ausführen, Iterieren

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>
```

Strukturierte Aktivitäten: Ereignisgesteuerte Auswahl

Ereignisgesteuerte Auswahl:

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

Erste eintretende wird ausgewählt und ausgeführt:

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

Strukturierte Aktivitäten: Bedingte Verzweigung

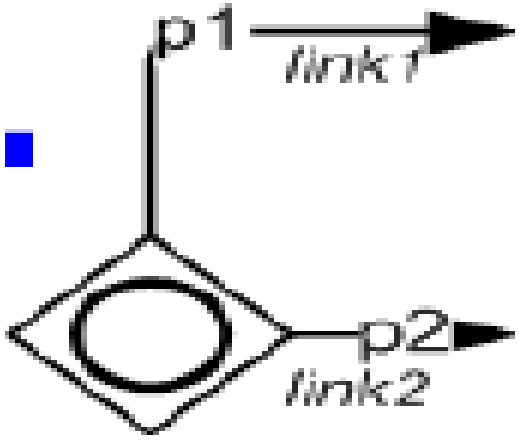

```
<if name="...">
   <condition> ... </condition>
   ...
   <elseif>
      <condition> ... </condition> …
   </elseif>
   <else> ... </else>
</if>
```

Strukturierte Aktivitäten: Inklusives Oder

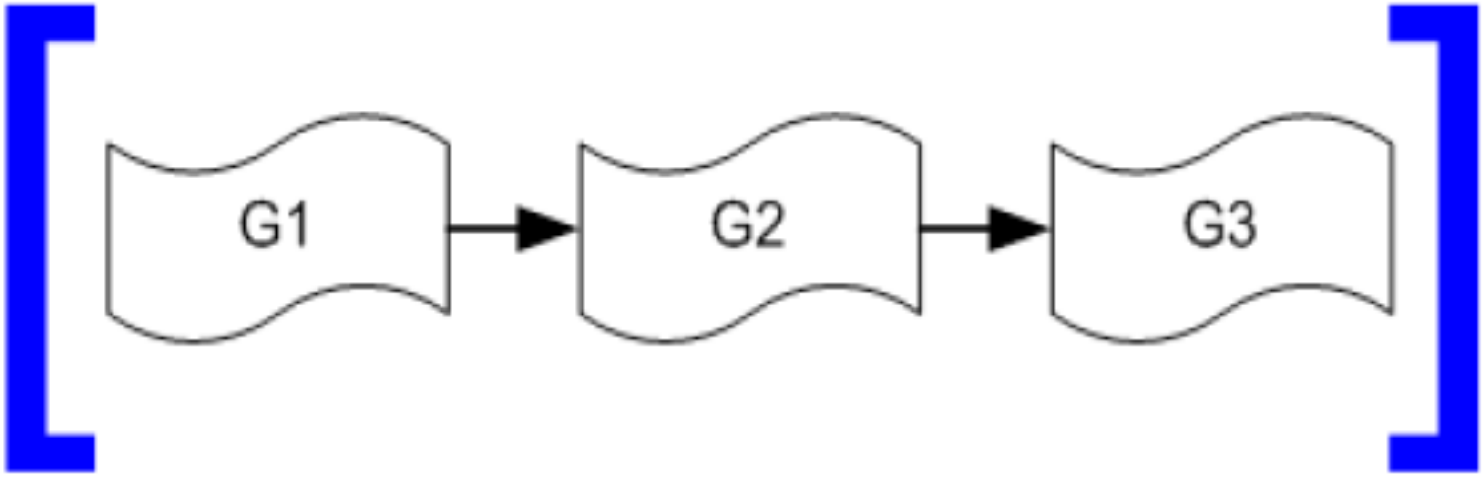
Inklusives Oder:

Kann Bedingungen an Sequenzverbindungen spezifizieren:

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



Transformation: Strukturierte Aktivitäten (1/8)



= ?

Sequenzielles Ausführen von Aktivitäten: `<sequence> </sequence>`
Paralleles Ausführen von Aktivitäten: `<flow> </flow>`
Iterieren der Ausführung von Aktivitäten bis Bedingung nicht erfüllt ist:

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

Ereignisgesteuerte Auswahl: `<pick createinstance="...">`

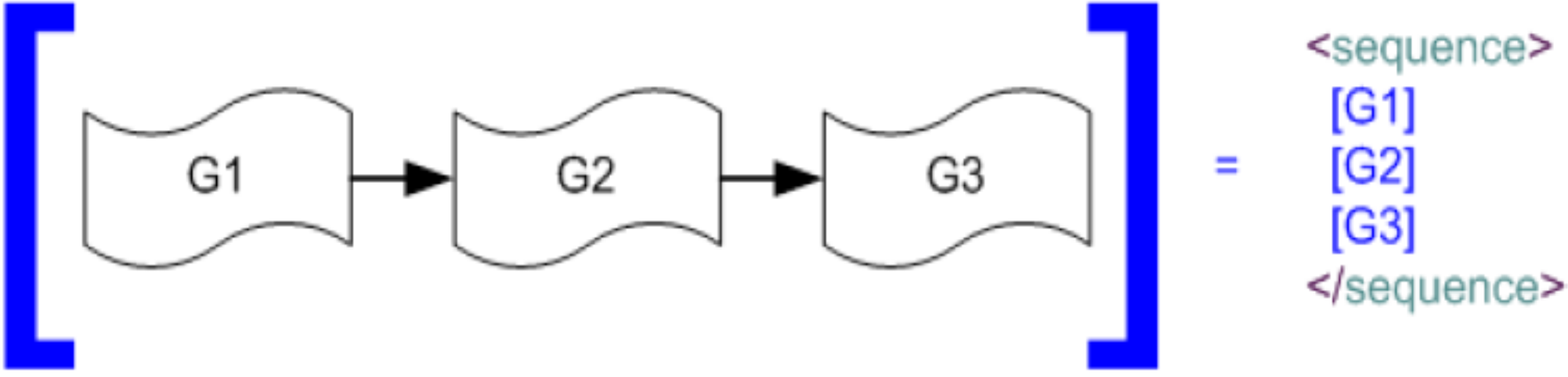
```
    <onMessage partnerLink="..." operation="...">... </onMessage>
    <onAlarm> ... </onAlarm> </pick>
```

Bedingte Verzweigung: `<if name="..."> <condition> ... </condition> ...`

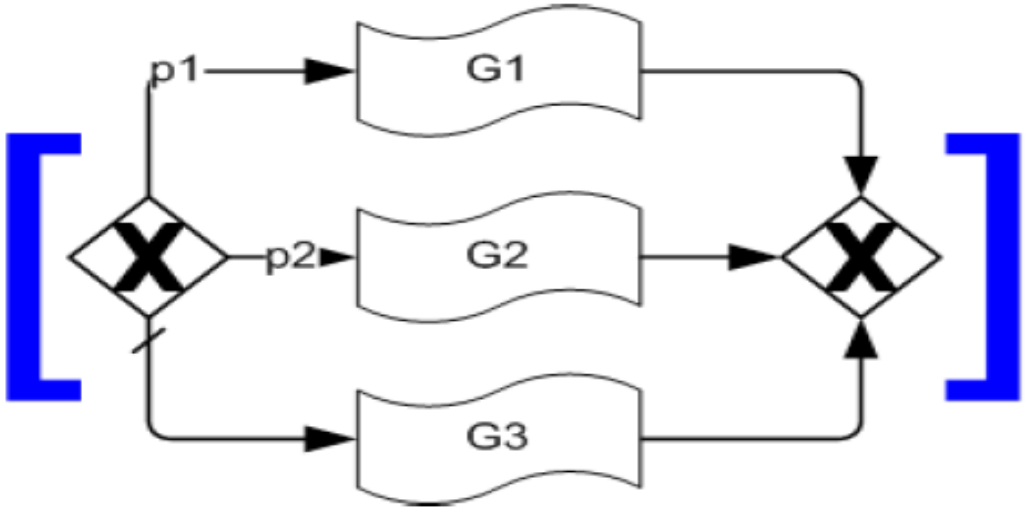
```
<elseif> <condition> … </condition> ... </elseif>
<else> ... </else> </if>
```

Inklusives Oder: `<transitionCondition>...</transitionCondition>`

Transformation (Strukturierte Aktivitäten 1/8): Sequenzen



Transformation: Strukturierte Aktivitäten (2/8)



= ?

Sequenzielles Ausführen von Aktivitäten: `<sequence> </sequence>`

Paralleles Ausführen von Aktivitäten: `<flow> </flow>`

Iterieren der Ausführung von Aktivitäten bis Bedingung nicht erfüllt ist:

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

Ereignisgesteuerte Auswahl:

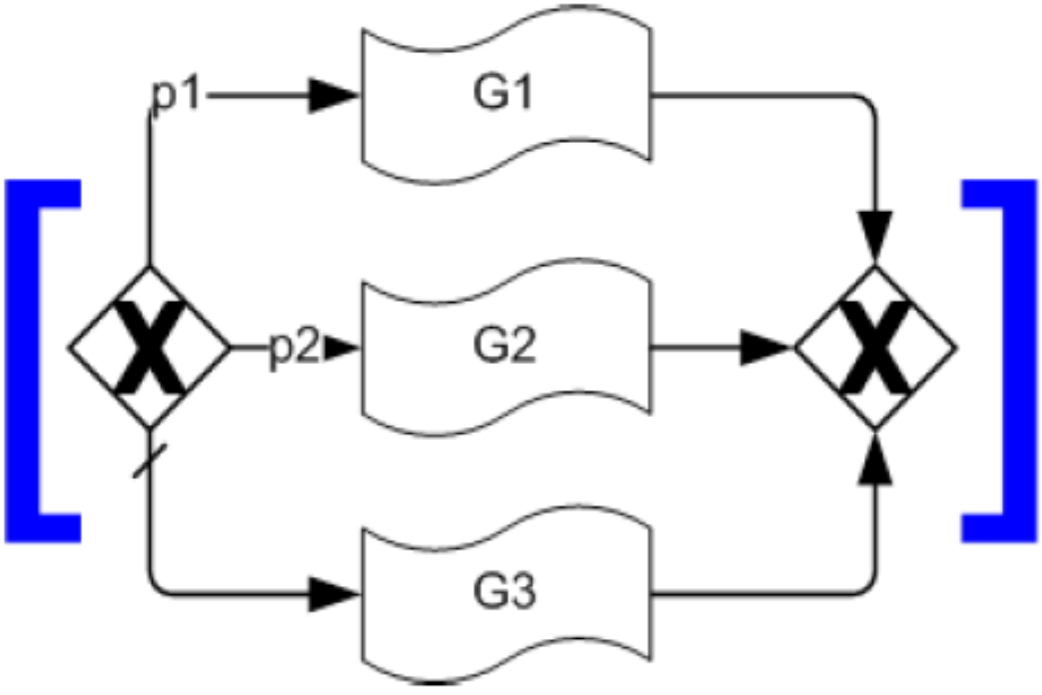
```
<pick createinstance="...">
    <onMessage partnerLink="..." operation="...">... </onMessage>
    <onAlarm> ... </onAlarm> </pick>
```

Bedingte Verzweigung:

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

Inklusives Oder: `<transitionCondition>...</transitionCondition>`

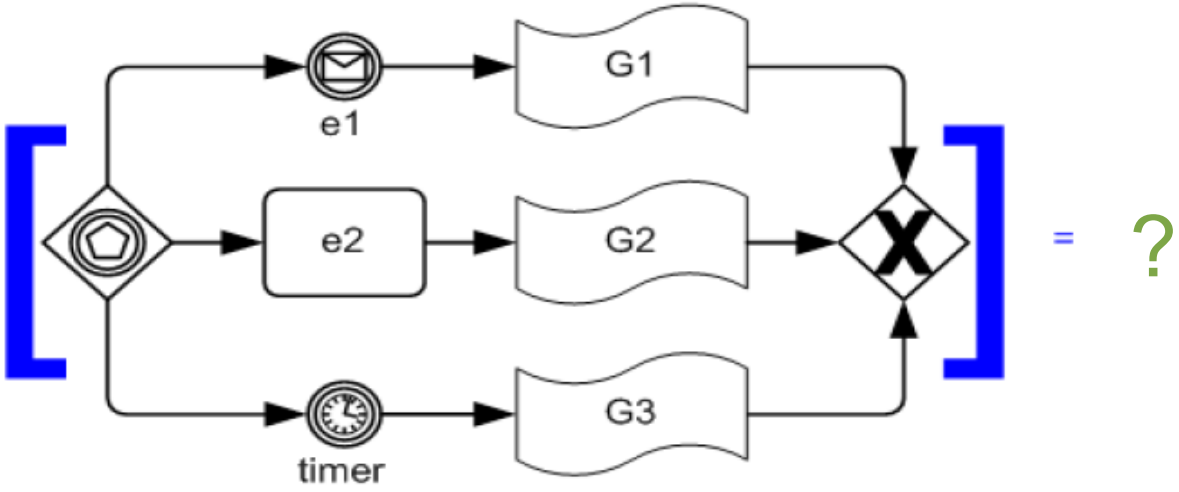
Transformation (Strukturierte Aktivitäten 2/8): If-Then-Else



=

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

Transformation: Strukturierte Aktivitäten (3/8)



Sequenzielles Ausführen von Aktivitäten: `<sequence> </sequence>`

Paralleles Ausführen von Aktivitäten: `<flow> </flow>`

Iterieren der Ausführung von Aktivitäten bis Bedingung nicht erfüllt ist:

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

Ereignisgesteuerte Auswahl: `<pick createinstance="...">`

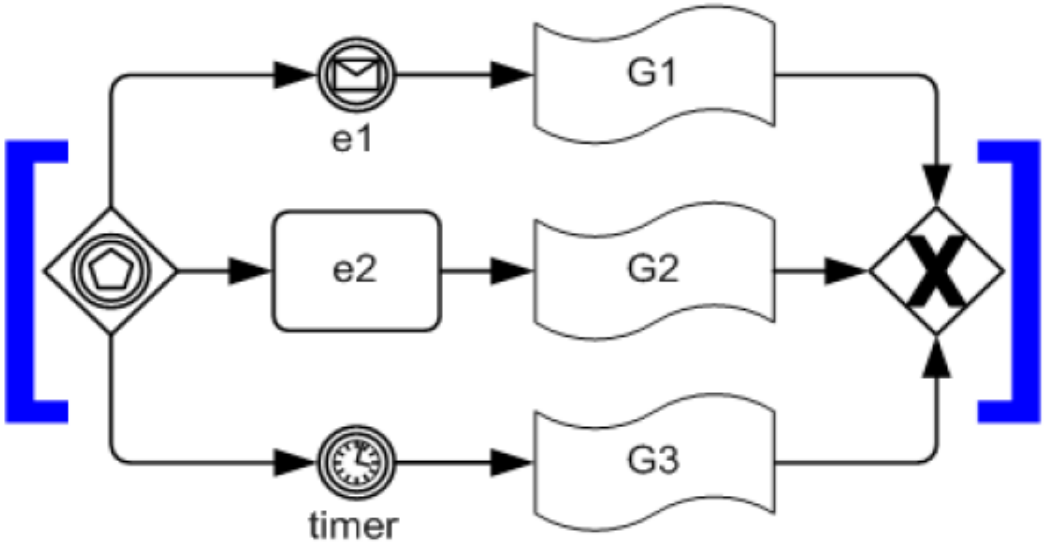
```
    <onMessage partnerLink="..." operation="...">... </onMessage>
    <onAlarm> ... </onAlarm> </pick>
```

Bedingte Verzweigung: `<if name="..."> <condition> ... </condition> ...`

```
<elseif> <condition> … </condition> ... </elseif>
<else> ... </else> </if>
```

Inklusives Oder: `<transitionCondition>...</transitionCondition>`

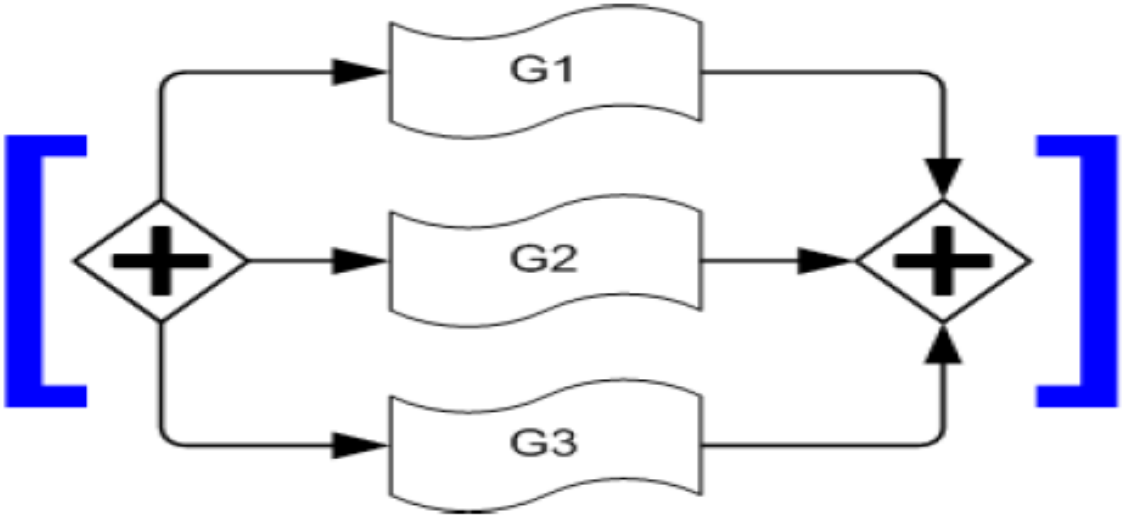
Transformation (Strukturierte Aktivitäten 3/8): Ereignisgesteuerte Auswahl



=

```
<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)



= ?

Sequenzielles Ausführen von Aktivitäten: `<sequence> </sequence>`

Paralleles Ausführen von Aktivitäten: `<flow> </flow>`

Iterieren der Ausführung von Aktivitäten bis Bedingung nicht erfüllt ist:

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

Ereignisgesteuerte Auswahl:

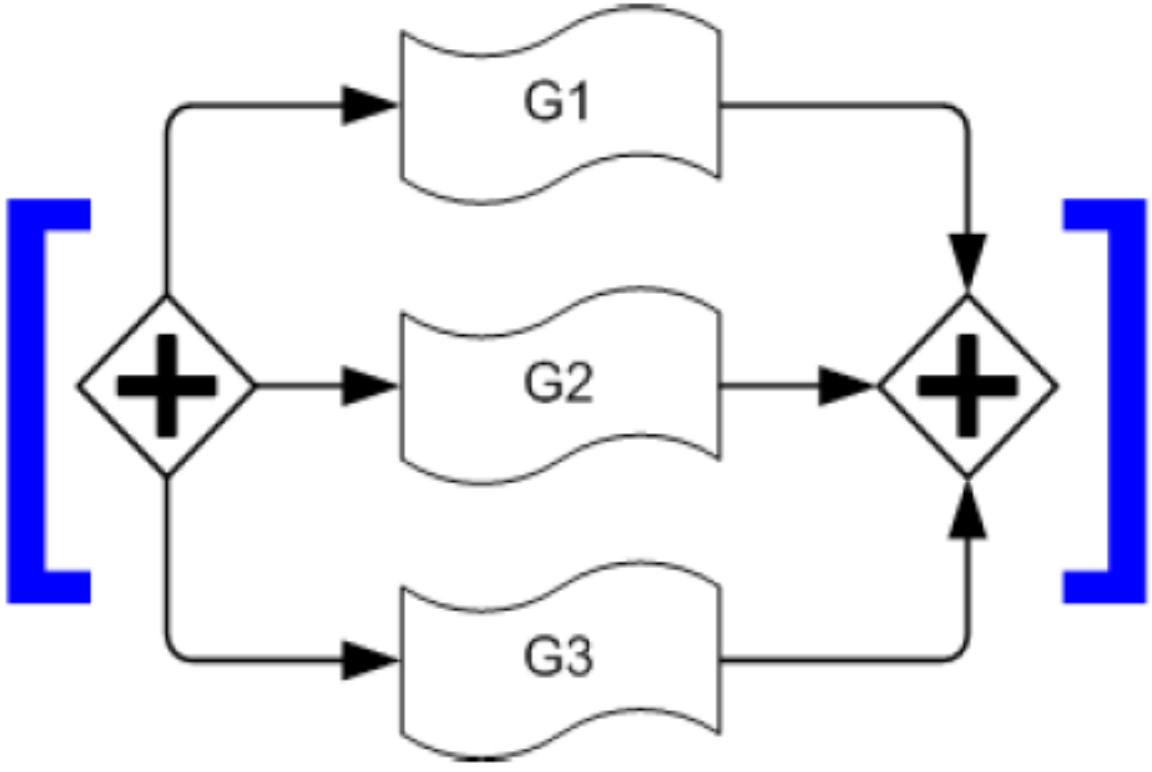
```
<pick createinstance="...">
    <onMessage partnerLink="..." operation="...">... </onMessage>
    <onAlarm> ... </onAlarm> </pick>
```

Bedingte Verzweigung:

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

Inklusives Oder: `<transitionCondition>...</transitionCondition>`

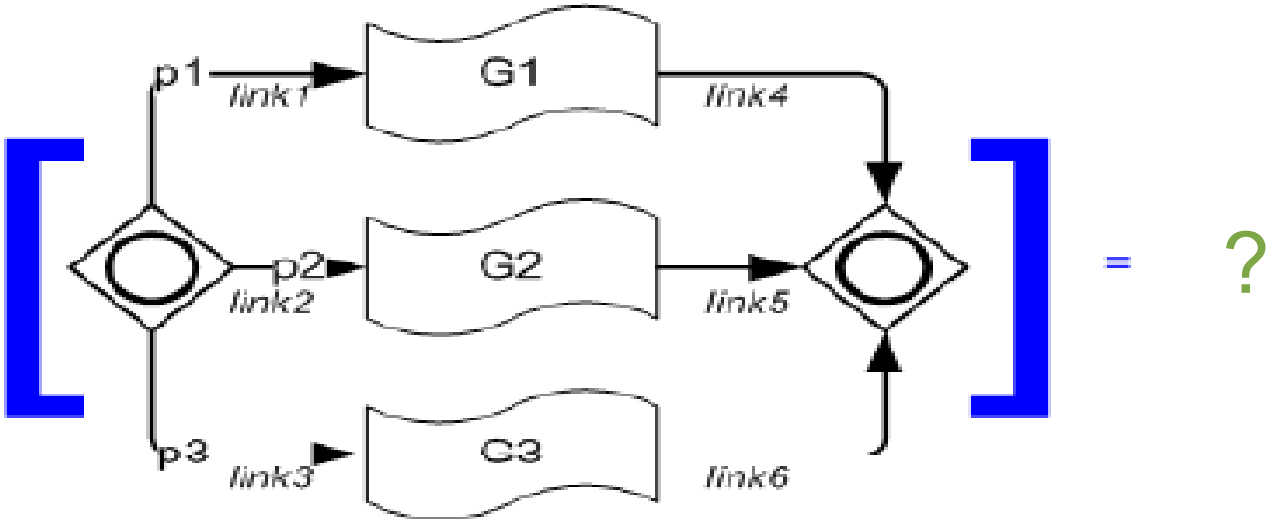
Transformation (Strukturierte Aktivitäten 4/8): Parallele Ausführung



=

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

Transformation: Strukturierte Aktivitäten (5/8)



Sequenzielles Ausführen von Aktivitäten: `<sequence> </sequence>`

Paralleles Ausführen von Aktivitäten: `<flow> </flow>`

Iterieren der Ausführung von Aktivitäten bis Bedingung nicht erfüllt ist:

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

Ereignisgesteuerte Auswahl: `<pick createinstance="...">`

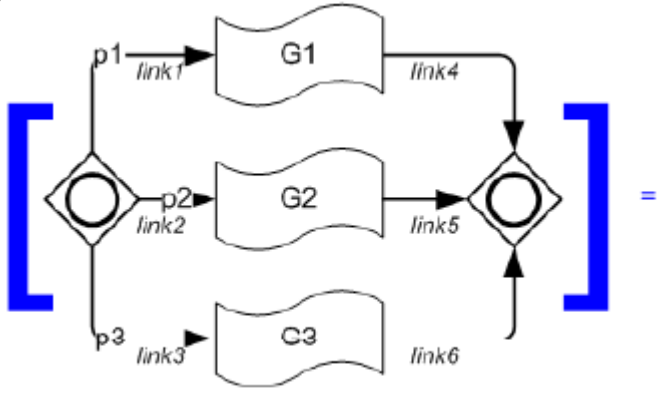
```
    <onMessage partnerLink="..." operation="...">... </onMessage>
    <onAlarm> ... </onAlarm> </pick>
```

Bedingte Verzweigung: `<if name="..."> <condition> ... </condition> ...`

```
<elseif> <condition> … </condition> ... </elseif>
<else> ... </else> </if>
```

Inklusives Oder: `<transitionCondition>...</transitionCondition>`

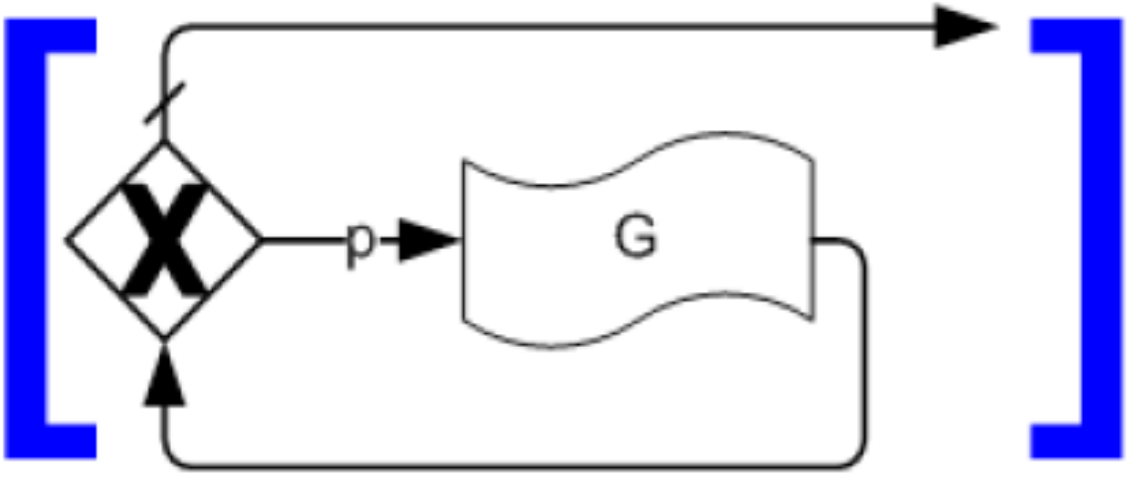
Transformation (Strukturierte Aktivitäten 5/8): Inklusives Oder-Gateway




```
<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)



= ?

Sequenzielles Ausführen von Aktivitäten: `<sequence> </sequence>`

Paralleles Ausführen von Aktivitäten: `<flow> </flow>`

Iterieren der Ausführung von Aktivitäten bis Bedingung nicht erfüllt ist:

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

Ereignisgesteuerte Auswahl: `<pick createinstance="...">`

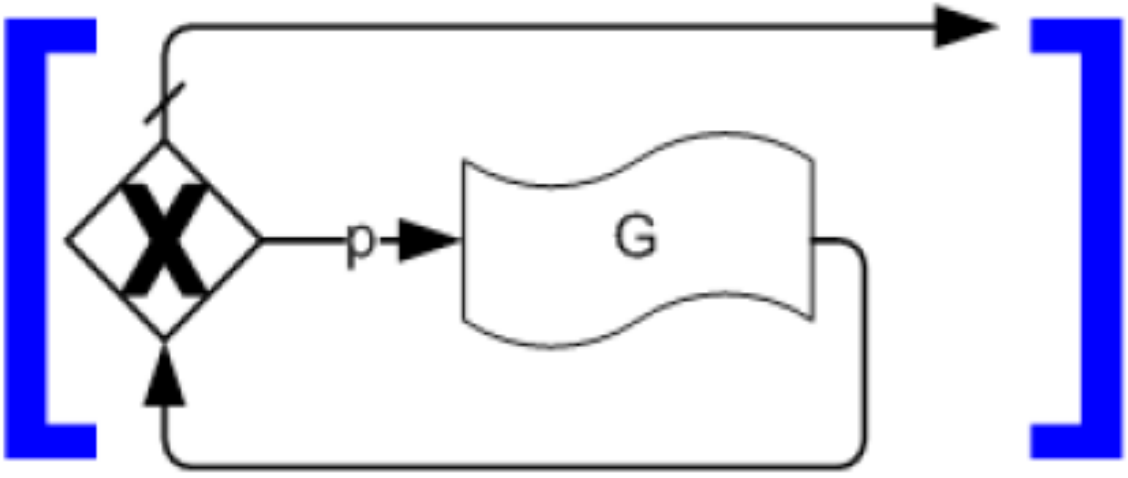
```
    <onMessage partnerLink="..." operation="...">... </onMessage>
    <onAlarm> ... </onAlarm> </pick>
```

Bedingte Verzweigung: `<if name="..."> <condition> ... </condition> ...`

```
<elseif> <condition> … </condition> ... </elseif>
<else> ... </else> </if>
```

Inklusives Oder: `<transitionCondition>...</transitionCondition>`

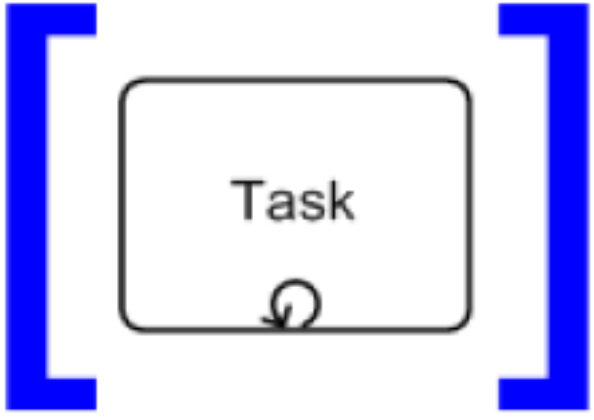
Transformation (Strukturierte Aktivitäten 6/8): While-Schleifen



=

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

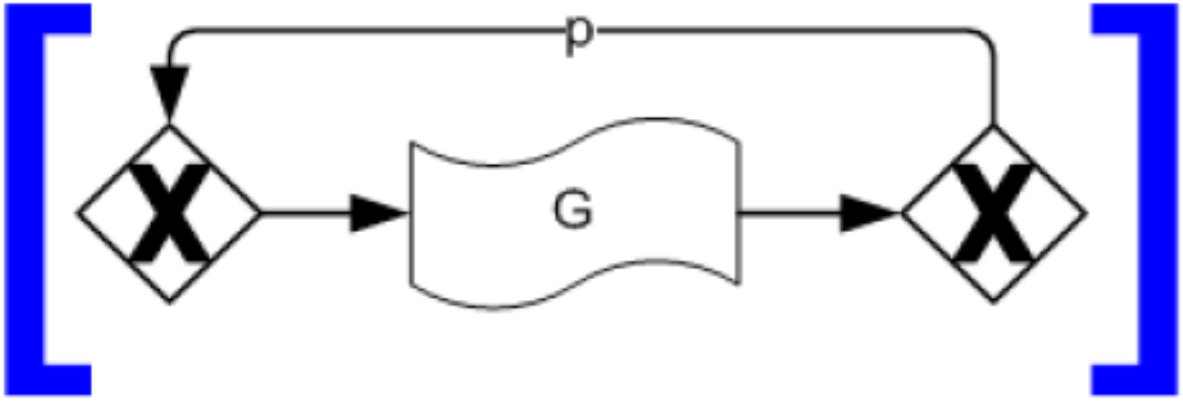
Analog:



=

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

Transformation: Strukturierte Aktivitäten (7/8)



= ?

Sequenzielles Ausführen von Aktivitäten: `<sequence> </sequence>`
Paralleles Ausführen von Aktivitäten: `<flow> </flow>`
Iterieren der Ausführung von Aktivitäten bis Bedingung nicht erfüllt ist:

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

Ereignisgesteuerte Auswahl: `<pick createinstance="...">`

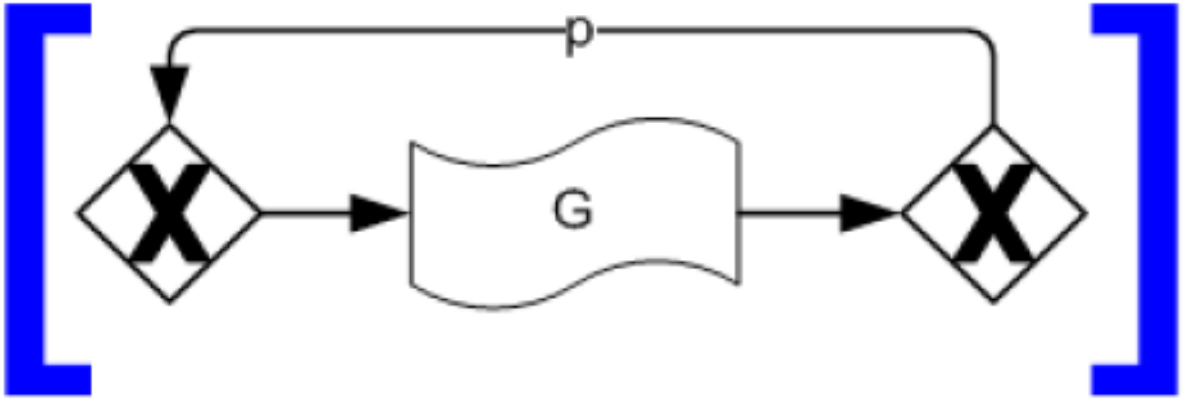
```
    <onMessage partnerLink="..." operation="...">... </onMessage>
    <onAlarm> ... </onAlarm> </pick>
```

Bedingte Verzweigung: `<if name="..."> <condition> ... </condition> ...`

```
<elseif> <condition> … </condition> ... </elseif>
<else> ... </else> </if>
```

Inklusives Oder: `<transitionCondition>...</transitionCondition>`

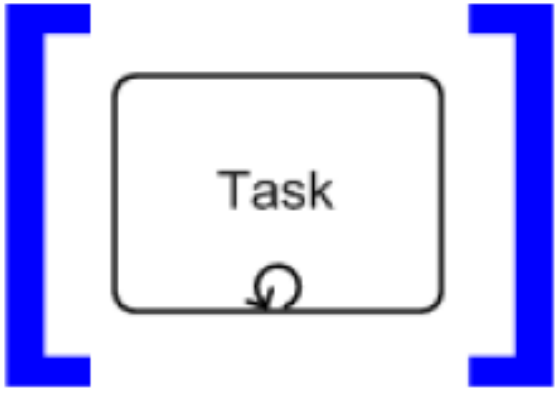
Transformation (Strukturierte Aktivitäten 7/8): Until-Schleifen



=

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

Analog:



=

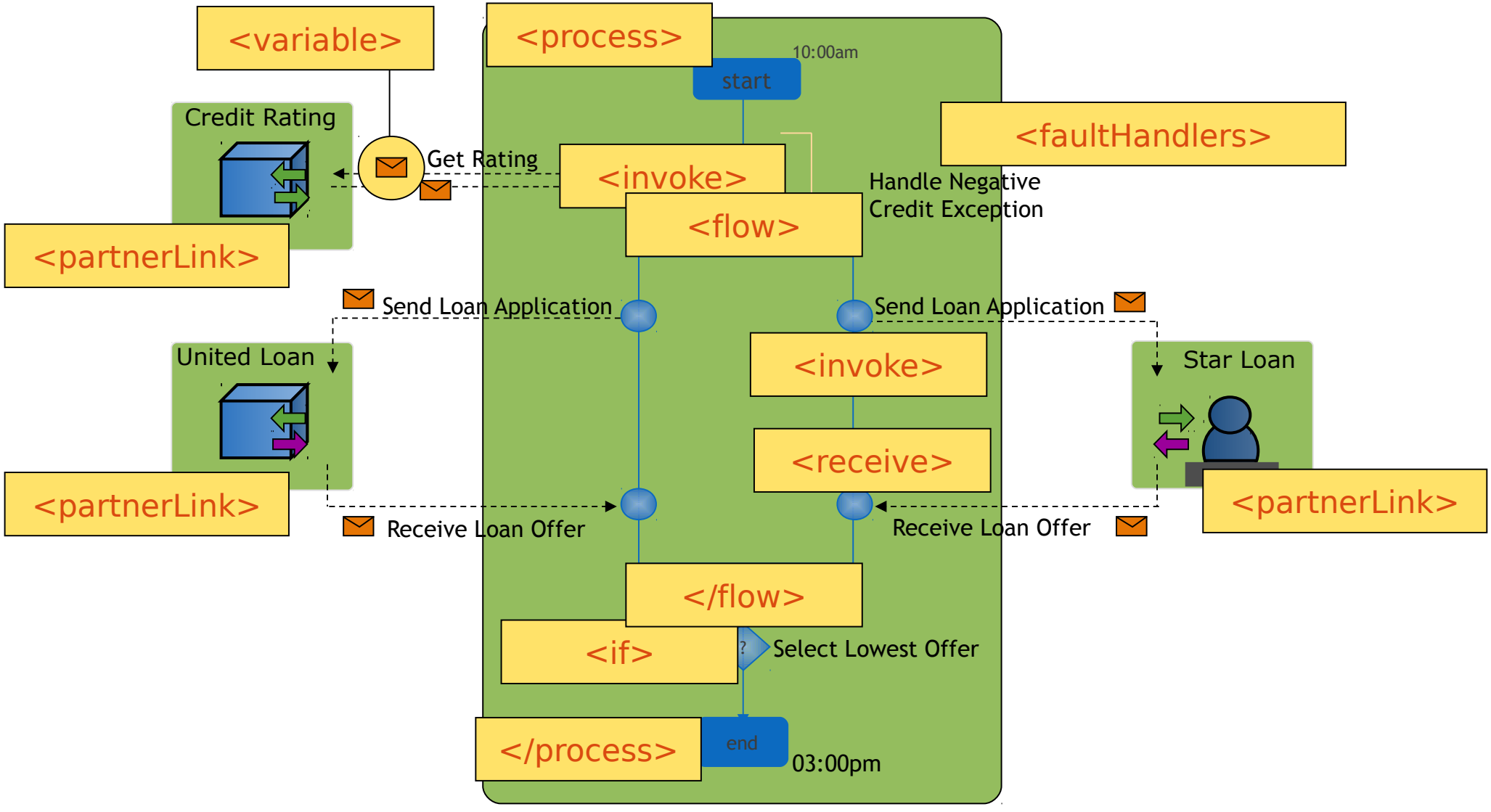
```
<repeatUntil>
  [Task]
  <condition>[not p]</condition>
</repeatUntil>
```

Transformation (Strukturierte Aktivitäten 8/8): Multi-Instanz-Schleifen

=

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

Transformation BPMN => BPEL: Überblick



Workflow-Automatisierung: Zusammenfassung

In diesem Abschnitt haben wir folgendes behandelt:

- 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 Softwareentwicklung
- Modellbasierte Entwicklung sicherer Software