Willkommen zur Vorlesung

Methodische Grundlagen des Software-Engineering

im Sommersemester 2012

Prof. Dr. Jan Jürjens

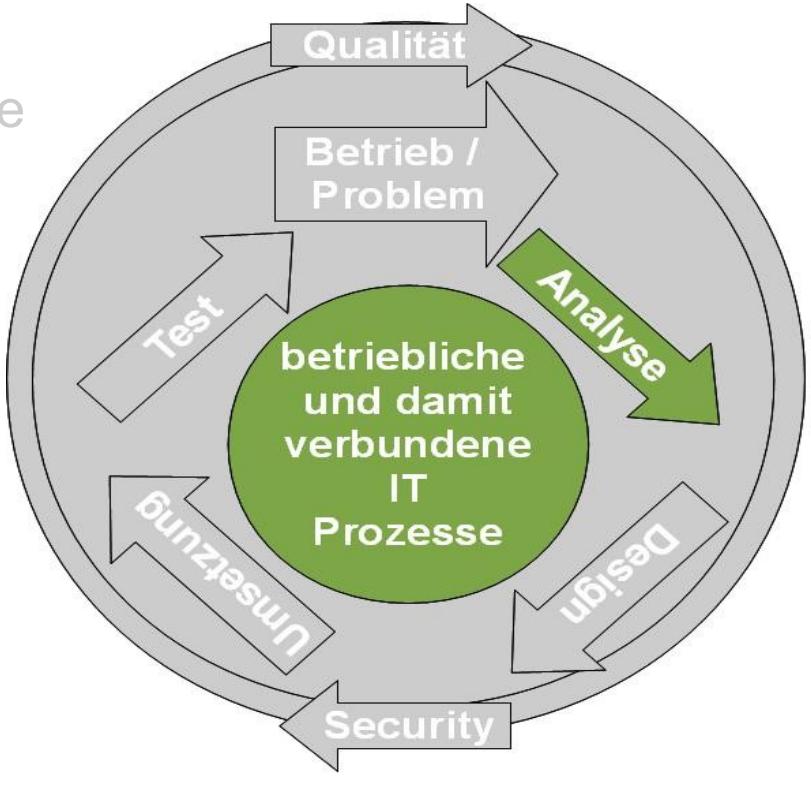
TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

2.6 Workflow-Automatisierung

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

Einordnung Workflow-Automatisierung

- Anwendungsbeispiel Finanz- und Versicherungsdomäne
- Geschäfts-Prozesse
  - Grundlagen Geschäfts-Prozesse
  - Einführung in die BPMN
  - Elektronische Prozessketten
  - Grundlagen der GP-Modellierung: Petri-Netze
  - Workflow-Management-Systeme
  - Workflow-Automatisierung
- Qualitätsmanagement
- Testen
- Sicherheit
- Sicheres Software Design



Überblick Workflow-Automatisierung

- Grundlagen
  - Natives Meta-Modell einer Workflow-Engine
  - Modell-Transformation
- Probleme mit Modell-Transformationen und Lösungen
- Kurz-Einführung BPEL
- Transformation: BPMN 2 nach BPEL 2

Im letzten Abschnitt ging es um die Ausführung von Workflows („Workflow Management“) und Systeme, die dies unterstützen (Workflow-Management-Systeme).

In diesem Abschnitt beschäftigen wir uns genauer mit dem Teilthema „Workflow-Automatisierung“, insbesondere mit der Übersetzung von BPMN-Modellen in die Business Perocess Execution Language (BPEL).

Workflow-Engines

Im letzten Abschnitt haben wir uns mit Workflow-Engines im Kontext von Workflow-Management-Systemen beschäftigt.

- Eine **BPEL-** (bzw. **BPMN-**) **Engine** ist eine Workflow-Engine, die Prozessmodelle importieren kann, die in BPEL (bzw. BPMN) spezifiziert sind, *entsprechend der operationalen Semantik von BPEL (bzw. BPMN).*
- Oft werden während des Imports die BPEL- (bzw. BPMN-) Artefakte bereits in interne Artefakte der Engine abgebildet. Anders ausgedrückt: Die meisten Engines implementieren ihr eigenes Prozess-Metamodell („**Native Metamodel**“).[1]
- „Native“ BPEL- (bzw. BPMN-) Engines sind Ausnahmen: Dort ist BPEL (bzw. BPMN) bereits das interne Metamodell.

[1] Metamodell: Definition einer Modellierungssprache, die selber als Modell gegeben wird.

Native Unterstützung eines Metamodells

Jede Workflow Engine unterstützt also **nativ** ein bestimmtes Metamodell.

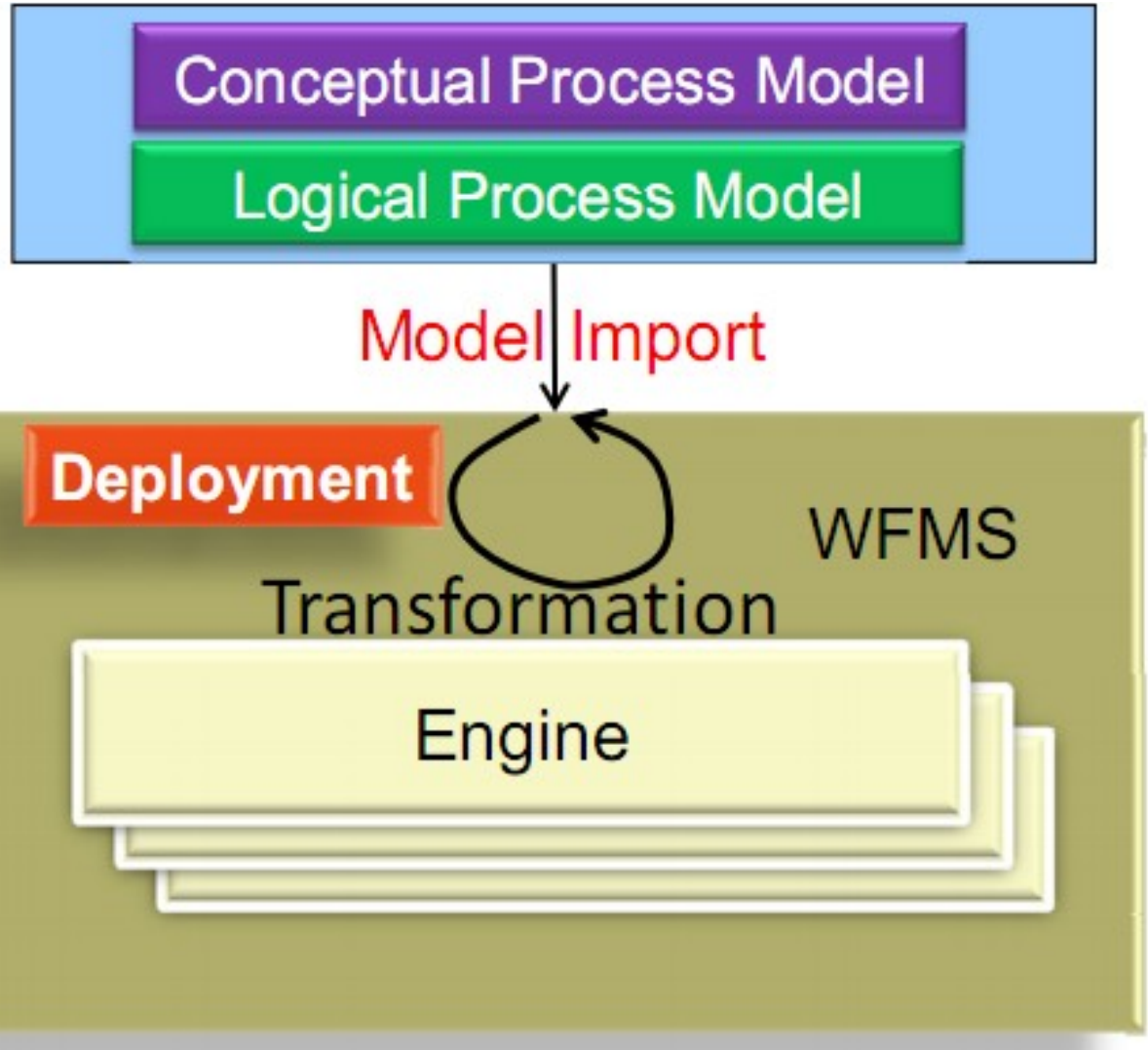
Üblicherweise wird das native Metamodell...

- direkt unterstützt in der Datenbank des WFMS.
  - Das Datenbankschema enthält sofort Instanzen des Metamodellkonstrukts
  - Das Datenbankschema ist zur Unterstützung des Navigators angepasst.
- direkt unterstützt in dem Zustandsmodell des WFMS.
  - Alle Metamodellkonstrukte haben eine 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“.

Was bestimmt ein Natives Metamodel einer Engine?

- Die Hersteller der Engines müssen sich dafür entscheiden, wie sie die Effizienz, Skalierbarkeit etc. der Anwendung sichern.
- Modellierungssprachen müssen ab dem ersten Release der Prozess-Engine unterstützt werden. Dies hat eine große Auswirkung auf die nativen Metamodelle.
- Gute Engine-Ersteller kümmern sich um die Allgemeingültigkeit und Erweiterbarkeit des nativen Metamodels, wobei eine möglichst einfache Unterstützung von Erweiterungen existierender (und sogar von neuen) Modelierungssprachen angestrebt wird.
- Normalerweise kennen die Benutzer der BPEL Engine nicht das native Metamodel und wissen nicht, wie unterschiedlich dieses zu BPEL als Metamodel selber ist.

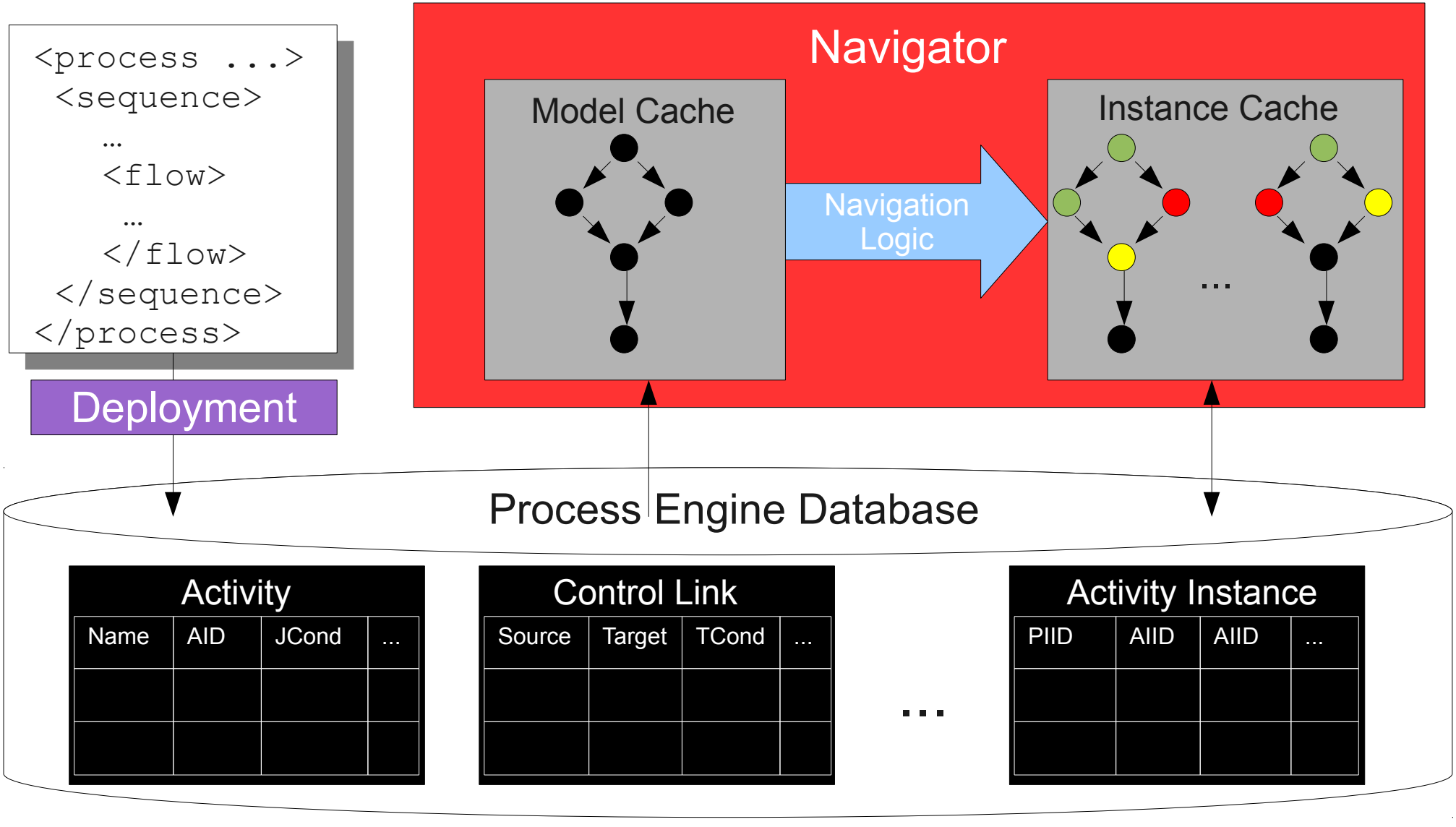
Vom Prozess-Modell zur Workflow-Engine



Anwendung des Modells: Deployment

- **Deployment** = Das Prozessmodell produktiv schalten.
  - z.B. bereit für die Ausführung machen
- Die Anwendung übersetzt normalerweise das Modell in unterschiedliche Formate.
  - Das importierte Modell kann in einem Metamodell spezifiziert sein, welches sich von dem, was das WFMS direkt unterstützt, unterscheidet.

Modell-Deployment: Überblick



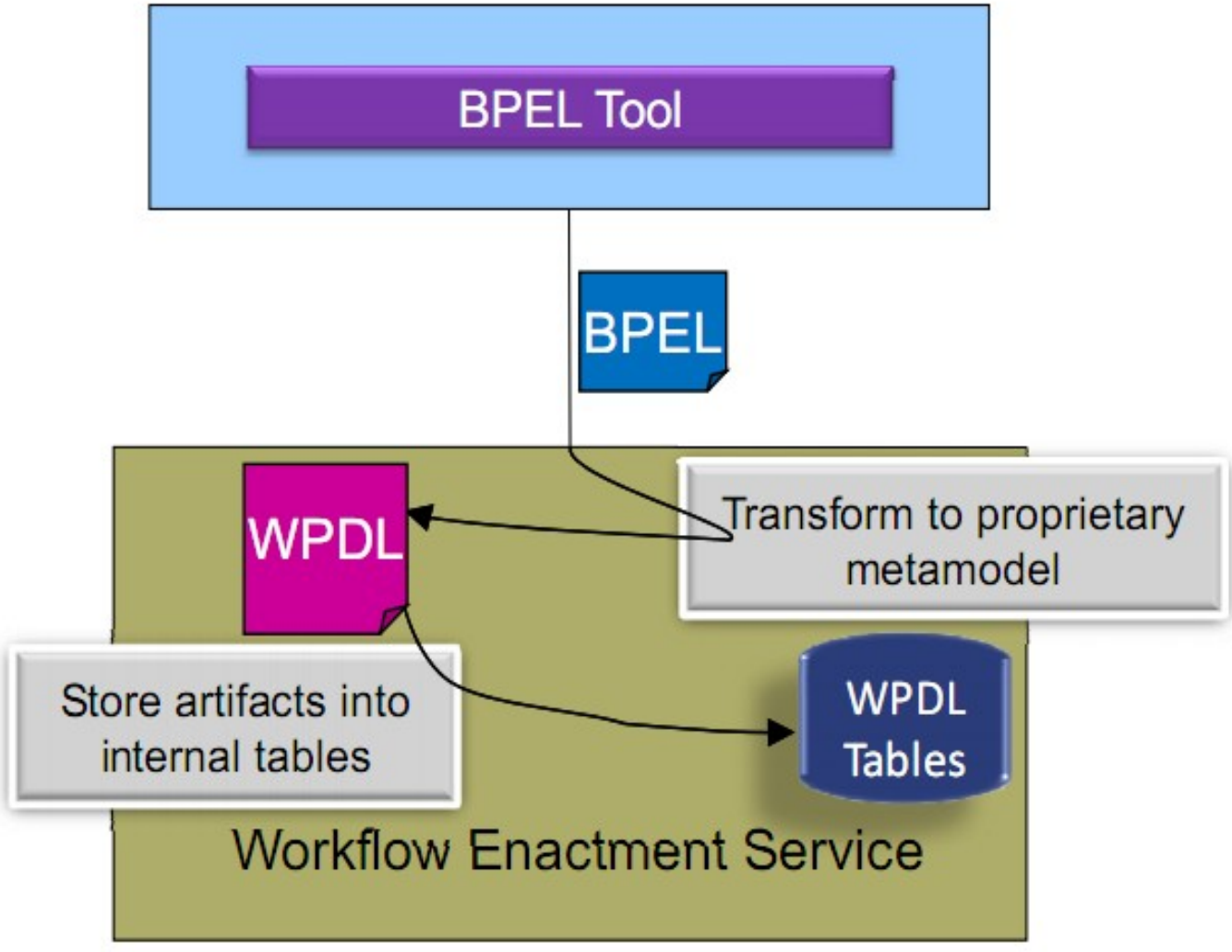
Transformation während des Modell-Deployments

Prozessmodelle, die in einem anderen Metamodell spezifiziert sind, als das der Engine, müssen in dieses native Metamodell, welches von der eigenen WFMS unterstützt wird, umgewandelt werden (Beispiele siehe folgende Folien).

Transformation während des Modell-Deployments: Beispiel

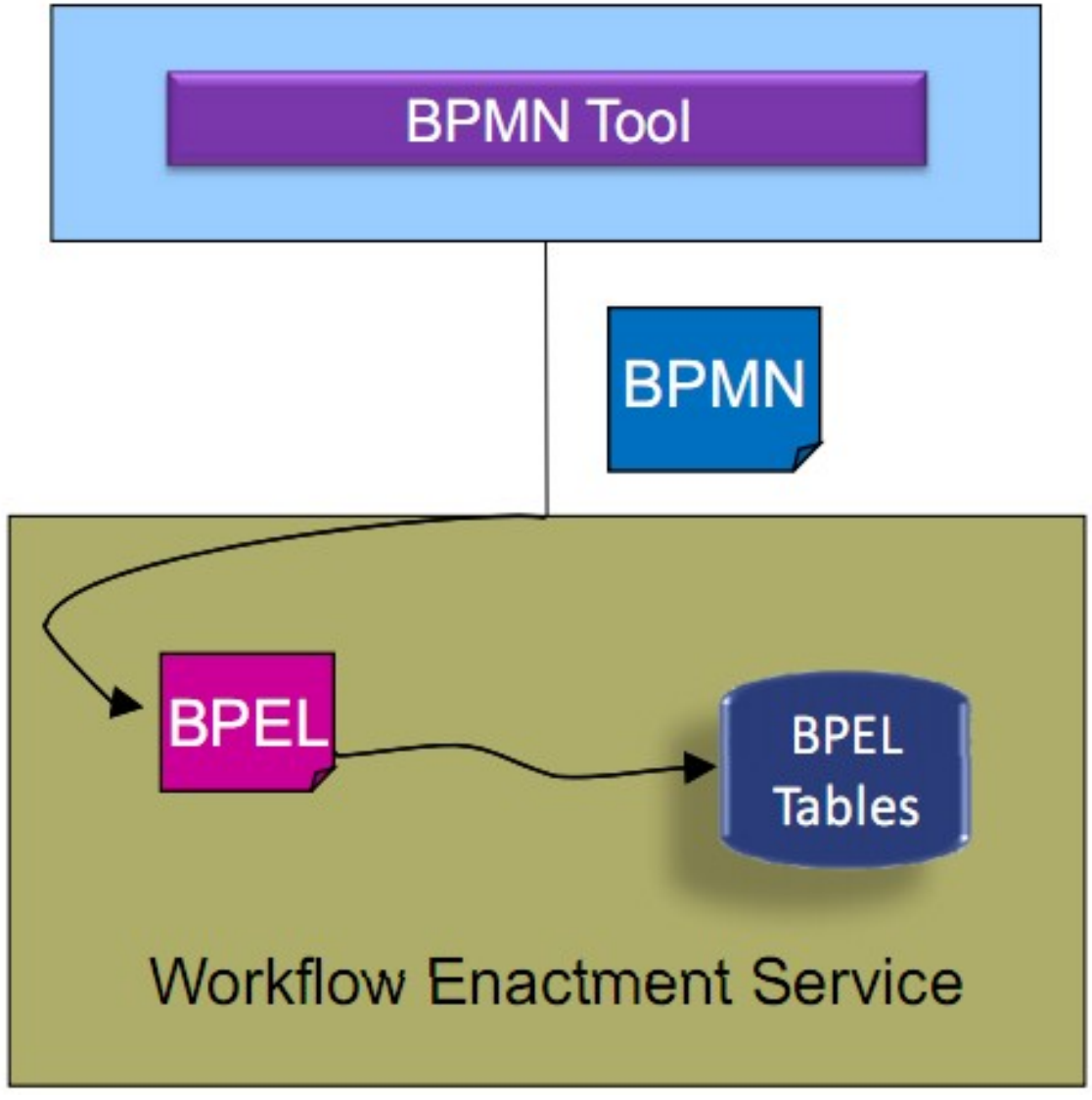
Ein WFMS, das WPDL[1] nativ unterstützt, kann ein BPEL-Modell nach der dazugehörigen Transformation ausführen.

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

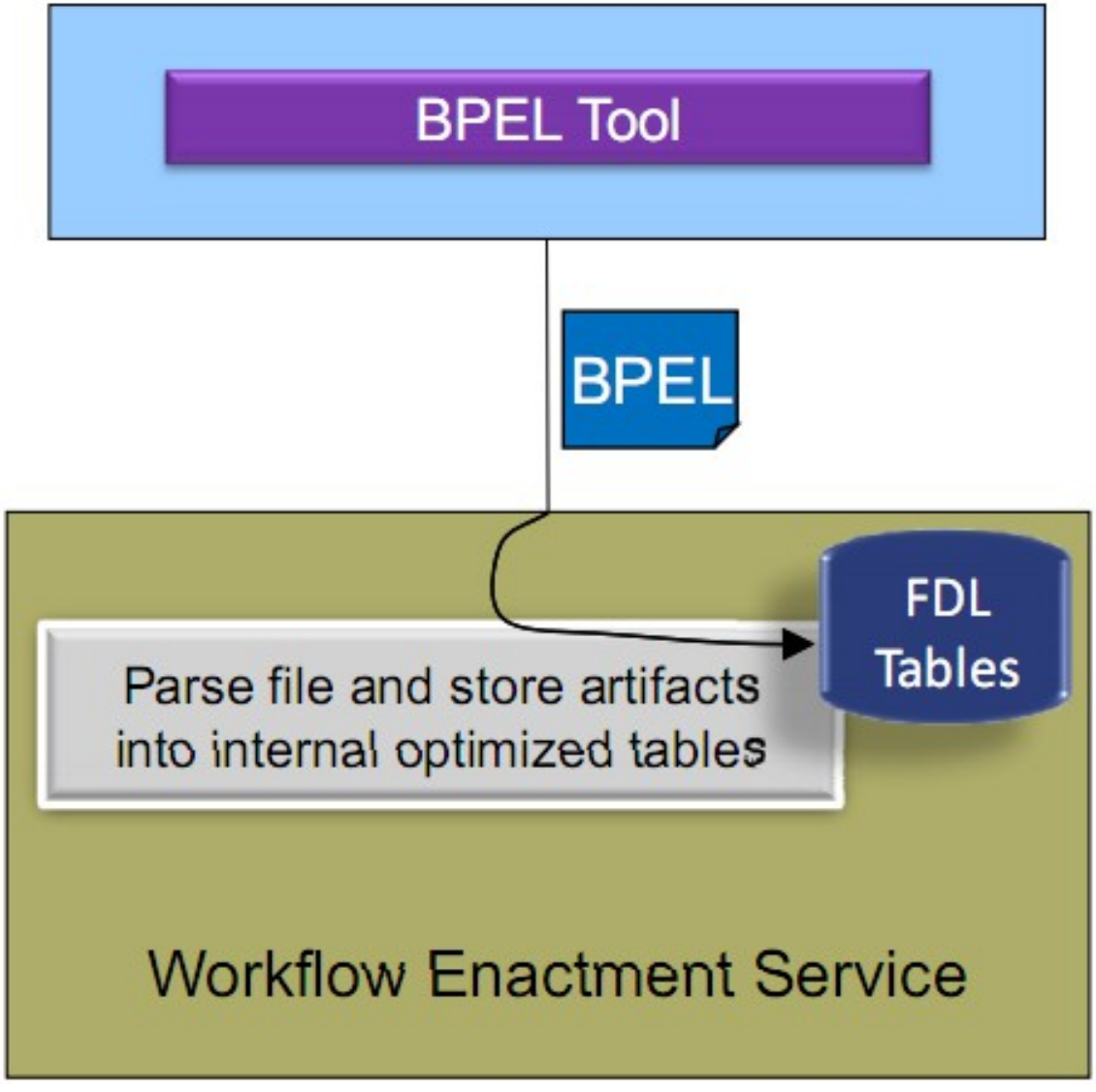


Transformation während des Modell-Deployments: Beispiel

Ein WFMS, das BPEL nativ unterstützt, kann ein BPMN-Modell nach der dazugehörigen Transformation ausführen.



Transformation während des Modell-Deployments: Beispiel



FDL = Flow Definition Language (vgl. IBM Websphere).

Überblick Workflow-Automatisierung

- Grundlagen
  - Natives Meta-Modell einer Workflow-Engine
  - Modell-Transformation
- Probleme mit Modell-Transformationen und Lösungen
- Kurz-Einführung BPEL
- Transformation: BPMN 2 nach BPEL 2

Probleme der Modell-Transformationen

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

Probleme der Modell-Transformationen

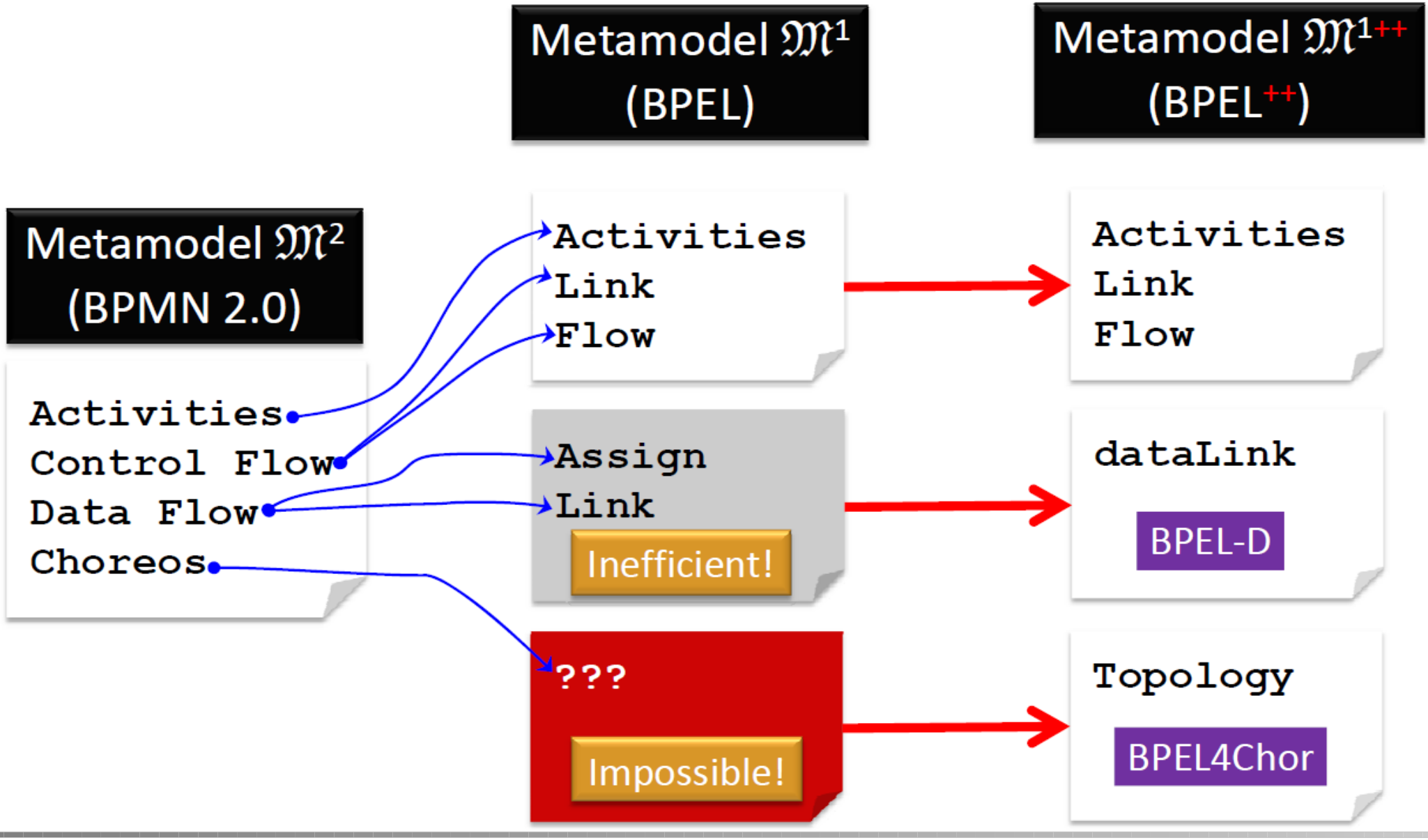
Solche Transformationen sind nicht immer genau, d.h. das transformierte Modell hat oft ein (etwas) anderes...

- … semantisches Verhalten als das Originalmodell
  - z.B.: Das Verhalten des Ausgangsmodell muss im Zielmodell emuliert werden – wenn möglich.
    - z.B.: BPEL's Exception-Verhalten ist schwer zu emulieren
- … operatives Verhalten als das Originalmodell
  - z.B.: Der Navigator führt das transformierte Modell weniger effizient aus als ein WFMS, das das Metamodell des Originalmodells unterstützt.
    - z.B.: Die Unterstützung eines FDL-Datenflusses in BPEL ist schwerfällig.

Lösung: Erweiterung des Ziel-Metamodells

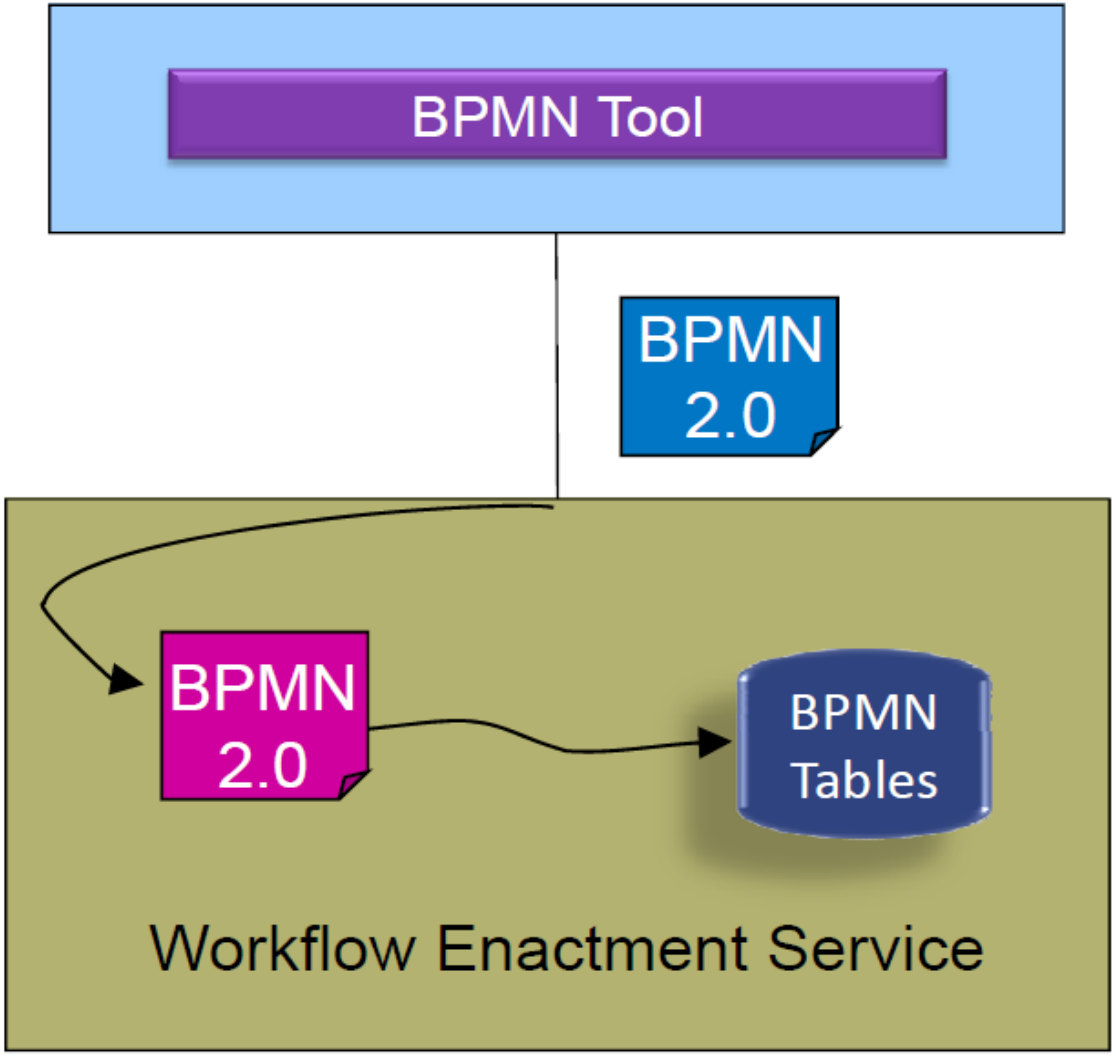
- Modelle, die in einem bestimmten Metamodell M^2 spezifiziert wurden, müssen in einer Engine mit einem anderen Metamodell M^1 perfekt unterstützt werden.
- Demnach müssen Konstrukte aus M^2, die schwer in M^1 zu emulieren sind, identifiziert werden, um entsprechende Konstrukte in M^1 hinzuzufügen.
- Das ist der Grund, warum BPEL erweiterbar entworfen wurde: Damit neue Konstrukte hinzugefügt werden können, um optimale Zuordnungen in verschiedenen Metamodellen zu gewährleisten.
- Eine erweiterte Variante einer gegebenen Engine (eine M^{1++} Engine) könnte so z.B. Prozessmodelle von anderen Metamodellen M^2, M^3, ..., M^n unterstützen.

Erweiterung des Ziel-Metamodells: Beispiel



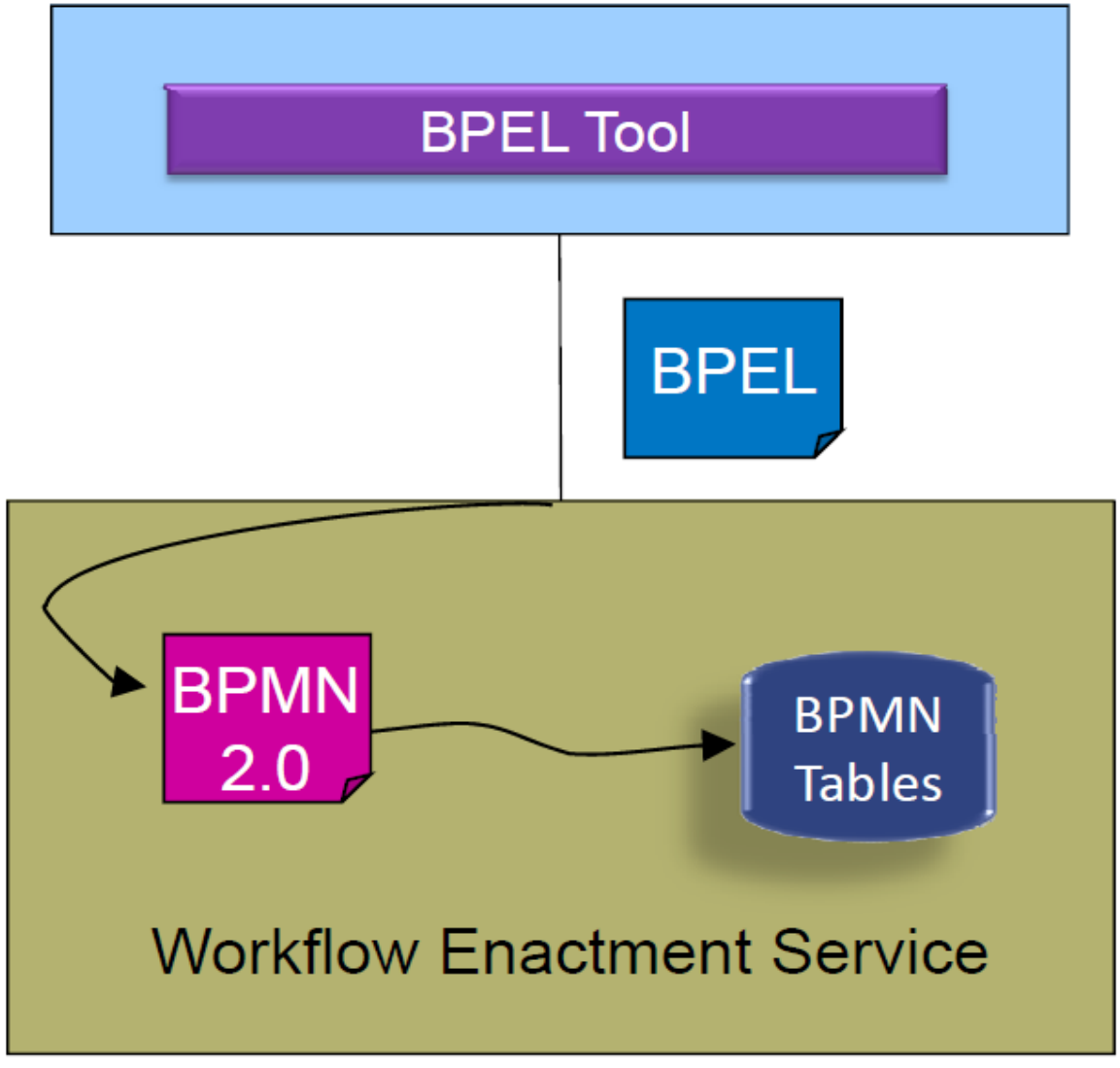
Native BPMN 2.0 Engines

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



Neues Anwendungsszenario: Transformation von BPEL nach BPMN

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



Konsequenzen für BPMN 2.0 Engines

- Ob ein BPMN 2.0 Prozessmodell durch eine BPEL Engine oder eine neue BPMN 2.0 Engine ausgeführt wird, ist nicht wichtig.
- Ein natives Metamodell einer BPMN 2.0 Engine sieht sehr wahrscheinlich anders aus, als BPMN 2.0 selber als Metamodell (wegen der Trade-offs, die durch die Hersteller gemacht werden müssen, vgl. F. 7).
- Ein natives Metamodell einer BPMN 2.0 Engine könnte auch durch BPEL unterstützt werden.
- D.h. ein Anbieter einer BPEL Engine, welche die Fähigkeit besitzt, ein BPMN 2.0 Prozessmodell zu importieren, kann auch zum Anbieter einer BPMN 2.0 Engine werden, basierend auf der gleichen 'execution engine'.
- Ggf. kann ein vorläufiges BPEL Prozessmodell aus einem BPMN 2.0 Prozessmodell generiert werden, bevor das Importieren stattfindet.
  - Die Benutzer müssen sich der Transformationsprozesse nicht einmal bewusst sein.

Was ist wichtig für eine Prozess Engine

Sehr viel wichtiger als das native Metamodell einer Prozessengine ist ihre Stabilität, Effizient, Skalierbarkeit, etc ...

- Anbieter von aktuellen BPEL Engines haben typischerweise viel in die nicht-funktionalen Eigenschaften ihrer Engines investiert.
- So haben diese Anbieter auch die Möglichkeit, BPMN 2.0 Engines mit sehr ähnlichen nicht-funktionalen Eigenschaften anzubieten.

Diskussion

„Oftmals wird von Entscheidern übersehen, dass die Wahl der Workflow-Engine und der darin enthaltenen Transformation zu einem irreversiblen Lock-in-Effekt führt“.
Wie interpretieren Sie diese Aussage ? Worin besteht die Gefahr, die durch die Transformation entsteht ?

Diskussion

„Oftmals wird von Entscheidern übersehen, dass die Wahl der Workflow-Engine und der darin enthaltenen Transformation zu einem irreversiblen Lock-in-Effekt führt“.
Wie interpretieren Sie diese Aussage ? Worin besteht die Gefahr, die durch die Transformation entsteht ?

Antwort: Bei einer Transformation findet oftmals ein Änderung des Prozesses statt (vgl. F. 16). Das führt dazu, dass die Modellierung der Prozesse sich ein Stück weit an der Transformation orientiert, um das Ergebnis zu erhalten, das gewünscht ist. Das bedeutet aber, dass die Prozessmodelle auf anderen Workflow-Systemen ein anderes, nicht gewünschtes, Verhalten zeigen können. Da das Neumodellieren aber oft sehr aufwendig ist, kommt es zu einem sogenannten Lock-in: das heißt, der Kunde wird vom Produkt abhängig.

Eine DBMS Analogie

- In der Vergangenheit haben ambitionierte Anbieter dediziert objekt-orientierte Datenbanksysteme (OODBMS) implementiert.
  - Das Objektmodell wurde nativ von diesen DBMS unterstützt.
- Aber an Stabilität, Effizienz, Skalierbarkeit, … mangelte es solchen Systemen.
- Anerkannte RDBMS Anbieter unterstützten dann auch Schlüsselkonstrukte dieses Objektparadigmas, während sie gleichzeitig Stabilität, Effizienz, Skalierbarkeit, … boten.

=> OODBMS sind kein „Mainstream“ mehr.

Entwicklung im Bereich Process-Engines

- BPMN 2.0 ist signifikant komplexer als BPEL.
- Also werden Anbieter Teile von BPMN 2.0 auswählen, basierend auf den Anforderungen ihrer Kunden.

Es ist zu erwarten, dass kein Anbieter alle Aspekte der BPMN 2.0 in seinem Produkt vorweisen kann.

- BPEL-Engines werden mit der Zeit erweitert, um Schlüsseleigenschaften von BPMN 2.0, die in BPEL und BPEL-Engines fehlen, anzubieten.
- BPEL-wird sich weiterentwickeln, um relevante BPMN 2.0-Eigenschaften (welche BPEL zur Zeit fehlen) aufzuweisen.
- Also sollte man warten, bis aktuelle Process-Engines die fehlenden BPMN-Eigenschaften aufweisen.

Wie die Ausführbarkeit ermöglicht wird

- Viele für die Ausführung relevante Informationen werden durch das XML-Schema für BPMN 2.0 zur Verfügung gestellt (das dafür da ist, Modelle zu speichern und weiterzuverarbeiten).
- D.h. die grafische Notation muss für die Ausführung um diese Informationen angereichert werden, die zur Instantiierung und Ausführung eines BPMN Prozessmodells nötig sind.
- Die dazugehörigen syntaktischen Details werden in der BPMN 2.0 Spezifikation durch UML-Klassendiagramme oder XML-Schema-Definitionen geliefert.

Überblick Workflow-Automatisierung

- Grundlagen
  - Natives Meta-Modell einer Workflow-Engine
  - Modell-Transformation
- Probleme mit Modell-Transformationen und Lösungen
- Kurz-Einführung BPEL
- Transformation: BPMN 2 nach BPEL 2

Beispiel für Modell-Transformation: BPMN nach BPEL

Als ein Beispiel für eine Modelltransformation betrachten wir auf den folgenden Folien die Transformation von BPMN-Modellen zu BPEL-Modellen.

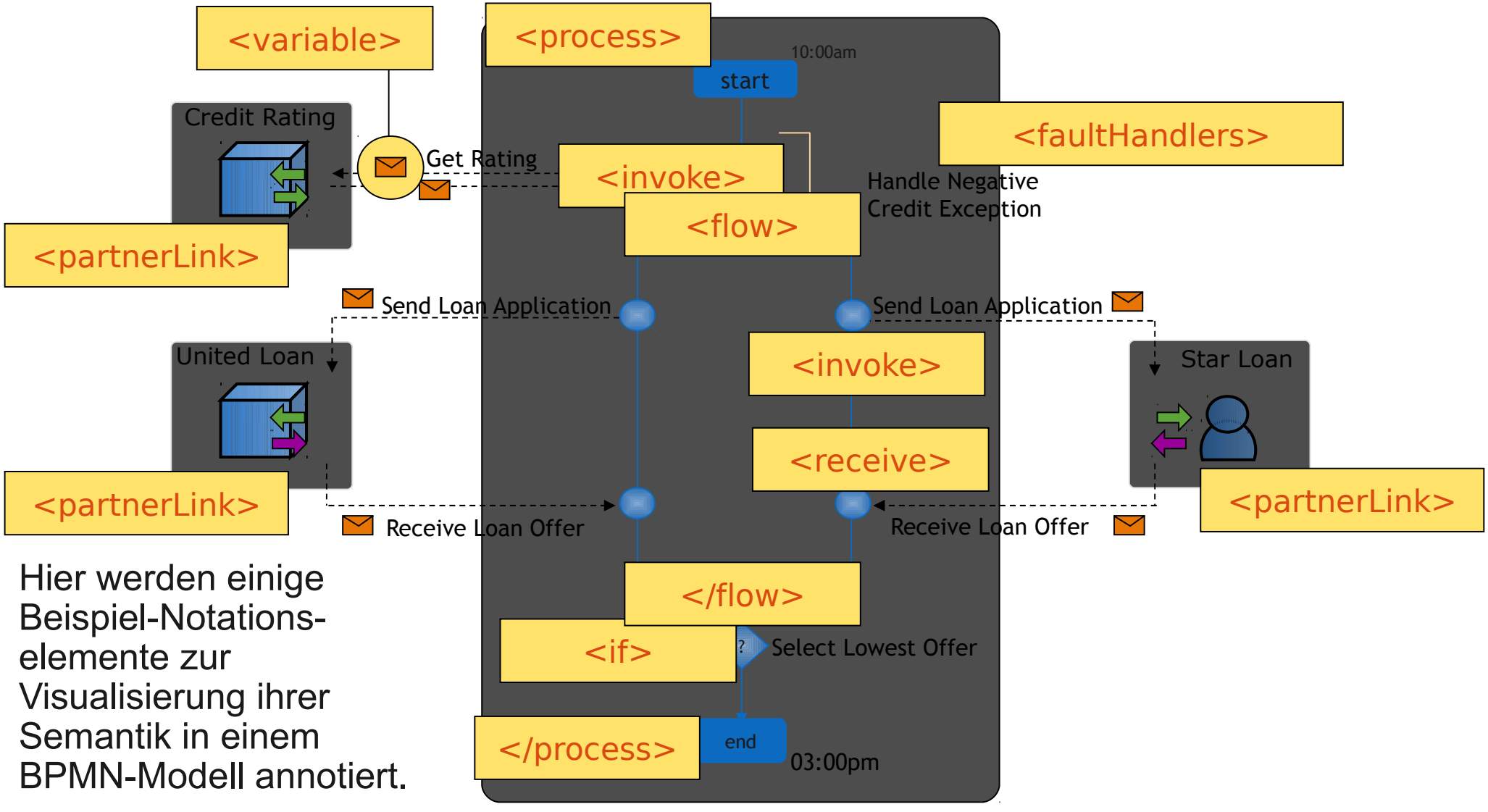
Diese Transformation wurde als Teil des BPMN 2.0-Standards definiert.

Mithilfe dieser Transformation enthält die BPMN 2.0-Notation eine Teilnotation, die isomorph zu BPEL ist. Anders ausgedrückt erhält man eine Visualisierung der BPEL als Teil von BPMN 2.0.

Dazu betrachten wir zunächst die BPEL-Notation noch etwas näher.

- BPEL: XML-basierte textuelle Notation („Markup language“), um Services in einen Prozess-Fluss zusammenzufügen.
- April 2007: WS-BPEL 2.0 Standard

BPEL Überblick



Hier werden einige Beispiel-Notations-elemente zur Visualisierung ihrer Semantik in einem BPMN-Modell annotiert.

WS-BPEL Sprachkonstrukte

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

Im Folgenden betrachten wir nur einen vereinfachten Ausschnitt der Notation.

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

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

Struktur eines BPEL4WS Projekts

```
<process ...>
<!-- Web Services, mit denen der Prozess interagiert: -->
   <partnerLinks> ... </partnerLinks>
<!-- Daten, die von dem Prozess benutzt werden: -->
   <variables> ... </variables>
<!-- Wird für asynchrone Interaktionen verwendet: -->
   <correlationSets> ... </correlationSets>
<!-- Alternativer Ausführungspfad bei fehlerhafter Bedingung: -->
   <faultHandlers> ... </faultHandlers>
<!-- Code, der ausgeführt wird, um ein Ereignis zu verarbeiten: -->
   <eventHandlers> ... </eventHandlers>
<!-- Was der Prozess eigentlich tut: -->
   (activities)*
</process>
```

Partnerdefinitionen und Links

<!-- Ein Partner ist über einen Web-Service-“Channel” abrufbar, definiert durch einen PartnerLinkTyp: -->

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

<!-- Ein 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>
```

BPEL Einfache Aktivitäten

```
<!-- Der Prozess aktiviert eine Operation beim Partner: -->
<invoke partnerLink="..." portType="..." operation="..."
            inputVariable="..." outputVariable="..."/>
<!-- Der Prozess erhält einen Aufruf des Partners:  -->
<receive partnerLink="..." portType="..."
  operation="..." variable="..."[createInstance="..."]/>
<!-- Der Process sendet eine Antwortnachricht in einem Partneraufruf: -->
<reply partnerLink="..." portType="..." operation="..."
                                          variable="..."/>
<!-- Datenbelegung zwischen Variablen: -->
<assign>
   <copy>
     <from variable="..."/> <to variable="..."/>
   </copy>+
</assign>
```

Weitere einfache Aktivitäten

<!-- Der Prozess entdeckt einen Ausführungsfehler und wechselt in den Fehlerausführungsbetrieb: -->

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

<!-- Den Prozess beenden: -->

```
<exit>
```

<!-- Der Prozess stoppt für eine bestimmte Zeit: -->

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

<!-- Nichts tun (syntaktischer Zucker): -->

```
<empty>
```

<!-- Bedingte Verzweigung: -->

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

BPEL Strukturierte Aktivitäten

<!-- Sequenzielles Ausführen von Aktivitäten: -->

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

<!-- Paralleles Ausführen von Aktivitäten: -->

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

<!-- Iterieren der Ausführung von Aktivitäten solange Bedingung erfüllt ist: -->

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

<!-- Iterieren der Ausführung von Aktivitäten bis Bedingung erfüllt ist: -->

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

<!-- Mehrere Event-Aktivitäten (z.B. Annehmen von Nachrichten, Zeit-Event) angesetzt für parallele Ausführung; erste eintretende wird ausgewählt und passender Code ausgeführt: -->

```
<pick>...</pick>
```

<!-- Definiert einen Kontrollzusammenhang zwischen einer Startaktivität und einem Ziel: -->

```
<link ...>
```

Korrelationsmengen

- BPEL4WS kann mehrere Arten von Interaktionen modellieren:
  - Einfache, zustandslose Interaktionen.
  - Zustandshafte, lang laufende, asynchrone Interaktionen.
- Correlation Sets (CSs) unterstützen Letzteres:
  - CSs repräsentieren die Daten, die benötigt werden, um den Zustand der Interaktion (eine “Konversation”) aufrecht zu erhalten.
  - Am Prozessende einer Interaktion erlauben CSs, dass ankommende Nachrichten die richtigen Prozessinstanzen erreichen.
- Was genau ist ein Correlation Set ?
  - Eine Menge von Geschäftsdatenfelder, die den Zustand der Interaktion erfassen (“correlating business data”).
    Zum Beispiel: eine “Bestellnummer”, eine “Benutzer ID”, etc.
  - Jede Menge wird einmal initialisiert.
  - Die Werte der Menge ändern nicht den Ablauf der Interaktion.

Benutzung der Korrelationsmengen

<!-- Eine Input- oder Output-Operation erkennt, welche Correlation Sets zu welcher gesendeten oder empfangenen Nachricht gehören. Dieses CS wird benötigt, um sicher zu stellen, dass die Nachricht zur dazugehörigen zustandshaften Interaktion gehört: -->

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

<!-- Ein CS wird einmal initialisiert innerhalb einer Interaktion, wobei die Menge mit dem "initiation" Attribut auf "yes" gesetzt wird. Dieser Wert darf danach nicht wieder veändert werden: -->

```
        <correlation set="PurchaseOrder"
                     initiation="yes"/>
    </correlations>
</receive>
```

Überblick Workflow-Automatisierung

- Grundlagen
  - Natives Meta-Modell einer Workflow-Engine
  - Modell-Transformation
- Probleme mit Modell-Transformationen und Lösungen
- Kurz-Einführung BPEL
- Transformation: BPMN 2 nach BPEL 2

Grundprinzip: BPMN2BPEL-Transformation

- Eine (rekursive) Funktion […] wird spezifiziert, die es erlaubt, eine Untermenge von BPMN auf BPEL abzubilden.
- Diese Abbildung wird spezifiziert durch folgende Beschreibungen:
  - [t] für alle elementaren BPMN Aufgaben t, die auf BPEL abgebildet werden können.
  - [e] für alle elementaren BPMN Events e, die auf BPEL abgebildet werden können.
  - [s] für alle BPMN Strukturen s, die eine direkte Abbildung auf BPEL haben.
- Das definiert konstruktiv eine BPMN Untermenge, welche abgebildet werden kann, sowie die assoziierte Abbildung [...] selber.

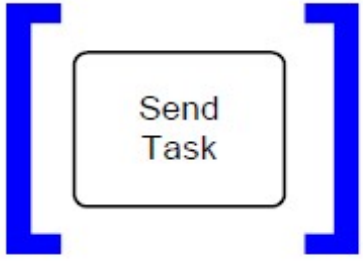
Transformation: Aktivitäten

=

```
<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>
```

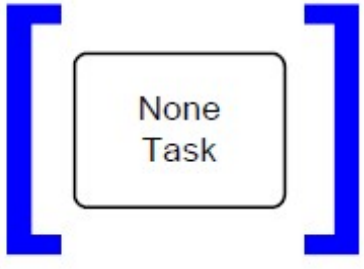
=

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



=

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



= ?

Transformation: Aktivitäten

<!-- Der Prozess entdeckt einen Ausführungsfehler und wechselt in den Fehlerausführungsbetrieb: --> `<throw faultName="..." faultVariable="..."/>`

<!-- Den Prozess beenden: --> `<exit>`

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

<!-- Nichts tun (syntaktischer Zucker): --> `<empty>`

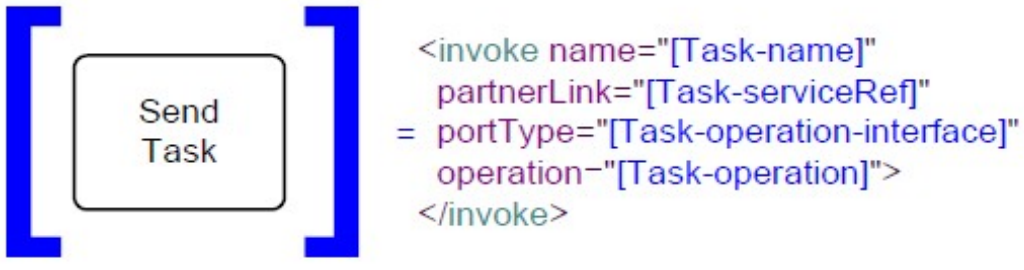
<!-- Sequenzielles Ausführen von Aktivitäten: --> `<sequence>`

<!-- Paralleles Ausführen von Aktivitäten: --> `<flow>`

<!-- Iterieren der Ausführung von Aktivitäten bis Bedingung nicht erfüllt ist: --> `<while>`

<!-- Mehrere Event-Aktivitäten (z.B. Annehmen von Nachrichten, Zeit-Event) angesetzt für parallele Ausführung; erste eintretende wird ausgewählt und passender Code ausgeführt: --> `<pick>`

<!-- Definiert einen Kontrollzusammenhang zwischen einer Startaktivität und einem Ziel: --> `<link ...>`



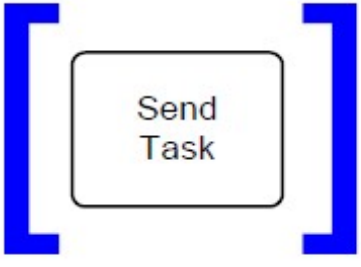
Transformation: Aktivitäten

=

```
<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>
```

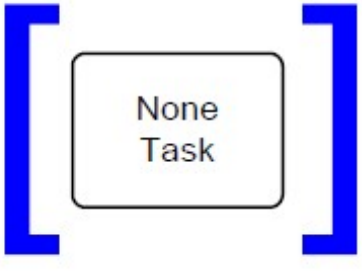
=

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



=

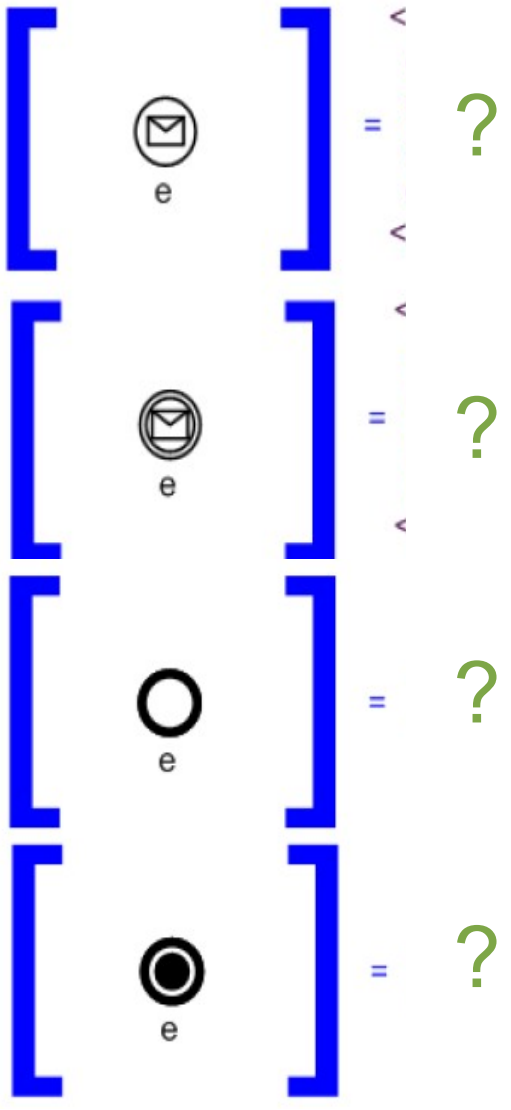
```
<invoke name="[Task-name]"
  partnerLink="[Task-serviceRef]"
  portType="[Task-operation-interface]"
  operation="[Task-operation]">
</invoke>
```



=

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

Transformation: Ereignisse



<!-- Der Prozess aktiviert eine Operation beim Partner: -->
`<invoke partnerLink="..." portType="..." operation="..." inputVariable="..." outputVariable="..."/>`

<!-- Der Prozess erhält einen Aufruf des Partners: -->
`<receive partnerLink="..." portType="..." operation="..." variable="..."[createInstance="..."]/>`

<!-- Der Process sendet eine Antwortnachricht in einem Partneraufruf: --> `<reply partnerLink="..." portType="..." operation="..." variable="..."/>`

<!-- Datenbelegung zwischen Variablen: --> `<assign> <copy> <from variable="..."/> <to variable="..."/> </copy>+ </assign>`

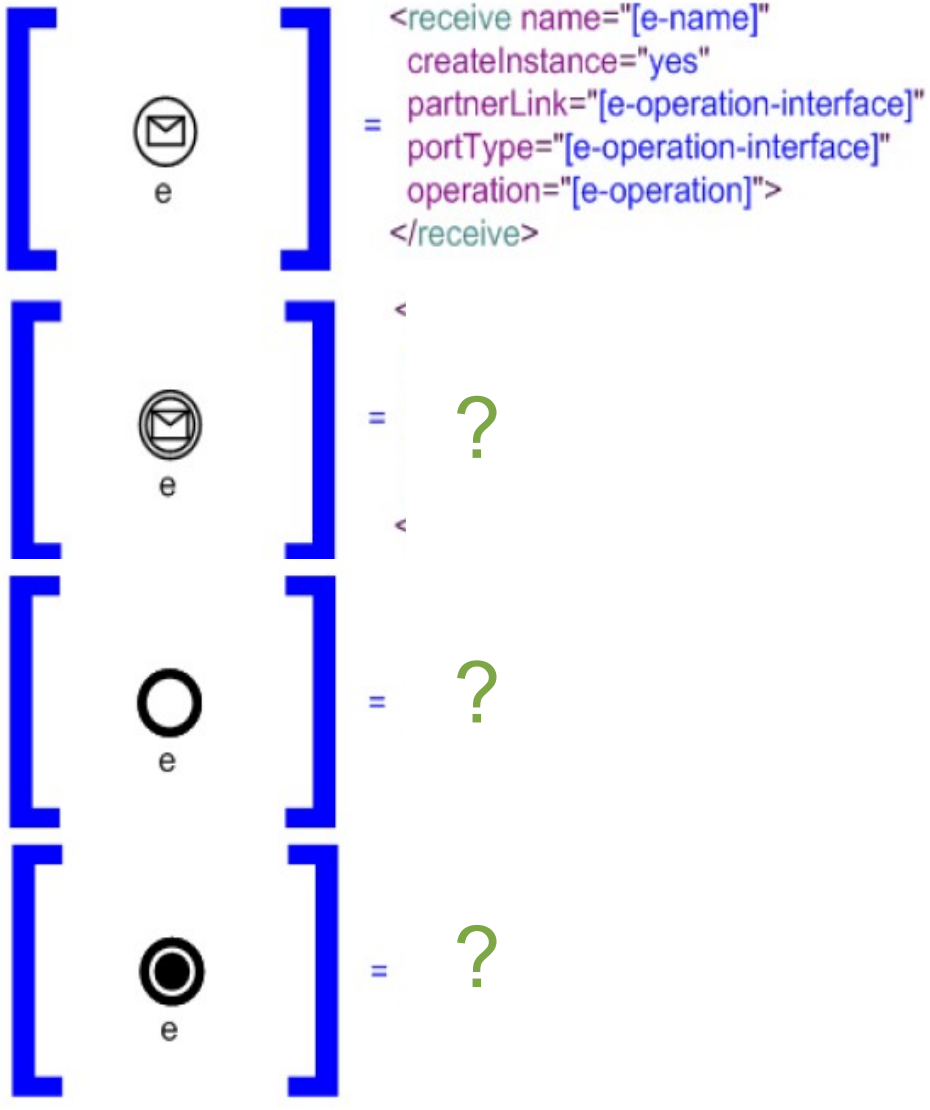
<!-- Der Prozess entdeckt einen Ausführungsfehler und wechselt in den Fehlerausführungsbetrieb: --> `<throw faultName="..." faultVariable="..."/>`

<!-- Den Prozess beenden: --> `<exit>`

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

<!-- Nichts tun (syntaktischer Zucker): --> `<empty>`

Transformation: Ereignisse



<!-- Der Prozess aktiviert eine Operation beim Partner: -->
`<invoke partnerLink="..." portType="..." operation="..." inputVariable="..." outputVariable="..."/>`

<!-- Der Prozess erhält einen Aufruf des Partners: -->
`<receive partnerLink="..." portType="..." operation="..." variable="..."[createInstance="..."]/>`

<!-- Der Process sendet eine Antwortnachricht in einem Partneraufruf: --> `<reply partnerLink="..." portType="..." operation="..." variable="..."/>`

<!-- Datenbelegung zwischen Variablen: --> `<assign> <copy> <from variable="..."/> <to variable="..."/> </copy>+ </assign>`

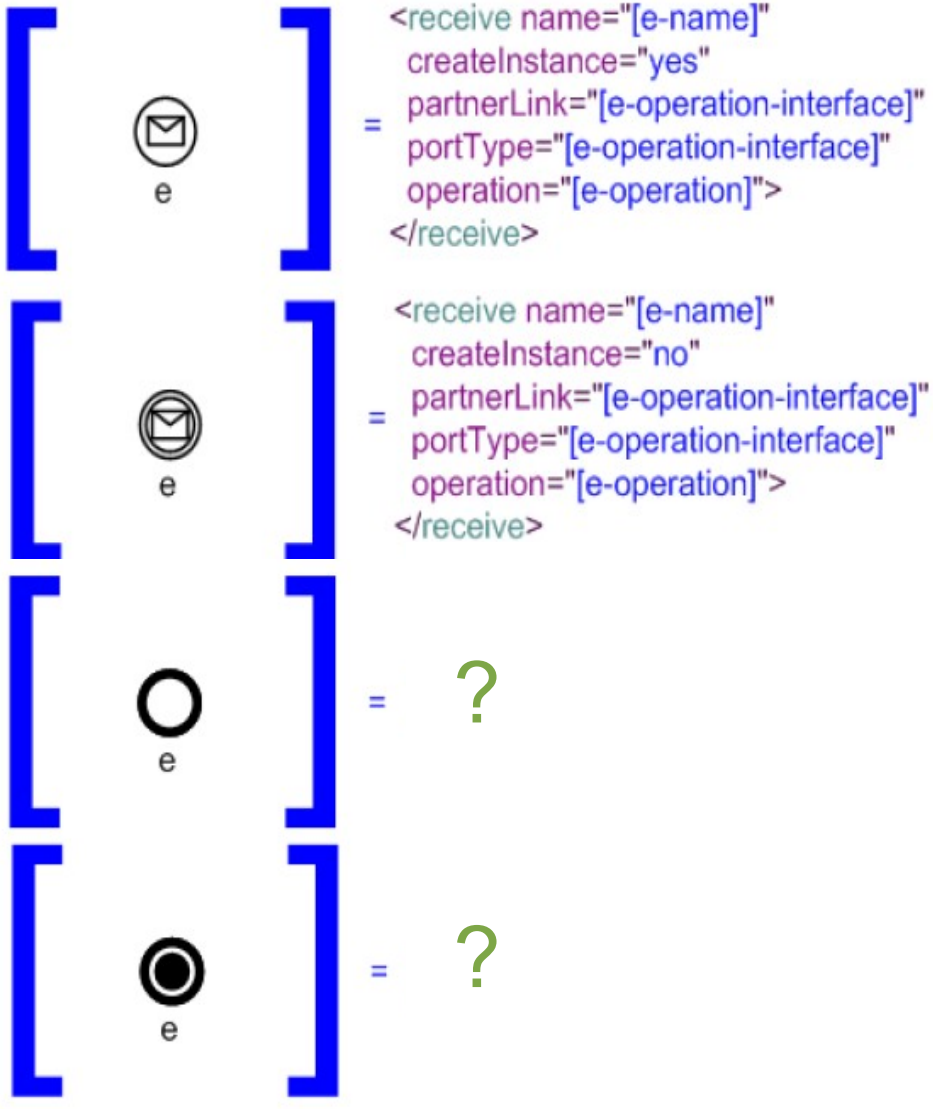
<!-- Der Prozess entdeckt einen Ausführungsfehler und wechselt in den Fehlerausführungsbetrieb: --> `<throw faultName="..." faultVariable="..."/>`

<!-- Den Prozess beenden: --> `<exit>`

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

<!-- Nichts tun (syntaktischer Zucker): --> `<empty>`

Transformation: Ereignisse



<!-- Der Prozess aktiviert eine Operation beim Partner: --> `<invoke partnerLink="..." portType="..." operation="..." inputVariable="..." outputVariable="..."/>`

<!-- Der Prozess erhält einen Aufruf des Partners: --> `<receive partnerLink="..." portType="..." operation="..." variable="..."[createInstance="..."]/>`

<!-- Der Process sendet eine Antwortnachricht in einem Partneraufruf: --> `<reply partnerLink="..." portType="..." operation="..." variable="..."/>`

<!-- Datenbelegung zwischen Variablen: --> `<assign> <copy> <from variable="..."/> <to variable="..."/> </copy>+ </assign>`

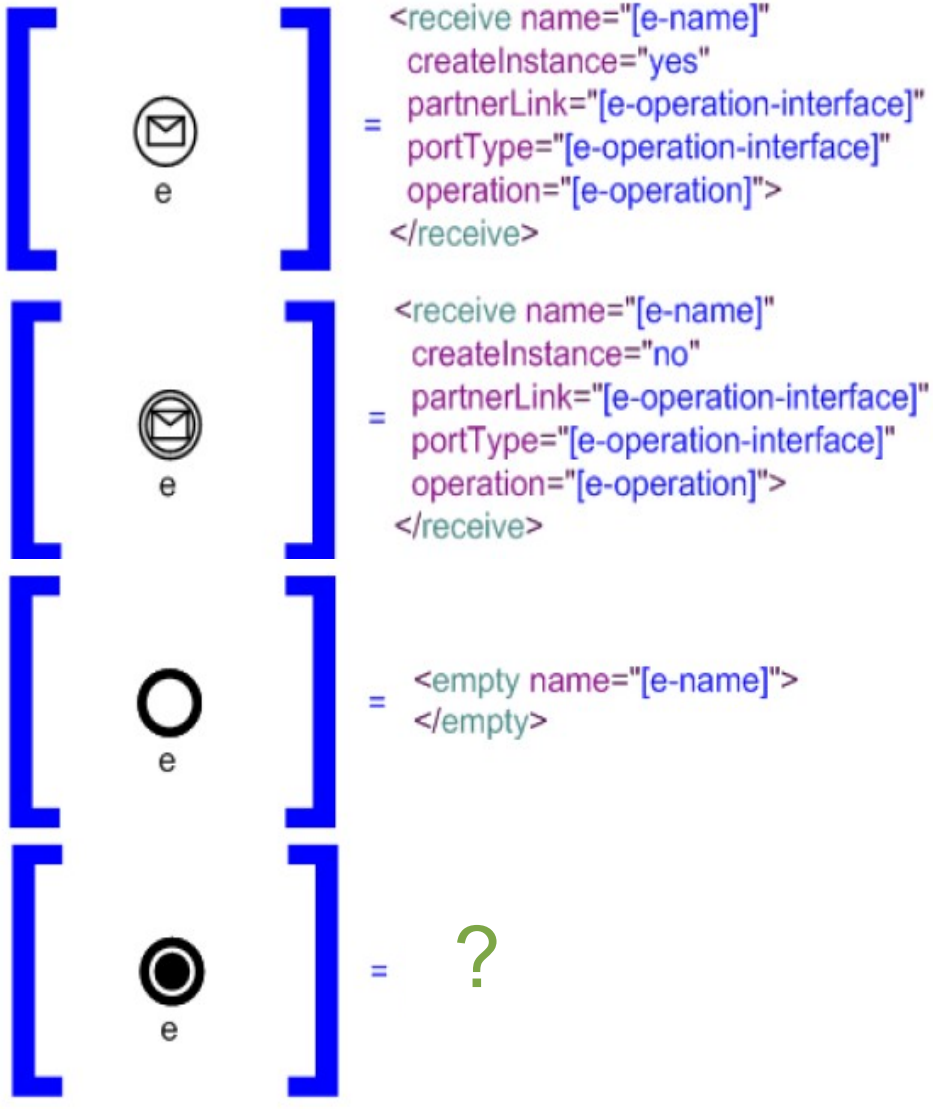
<!-- Der Prozess entdeckt einen Ausführungsfehler und wechselt in den Fehlerausführungsbetrieb: --> `<throw faultName="..." faultVariable="..."/>`

<!-- Den Prozess beenden: --> `<exit>`

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

<!-- Nichts tun (syntaktischer Zucker): --> `<empty>`

Transformation: Ereignisse



<!-- Der Prozess aktiviert eine Operation beim Partner: --> `<invoke partnerLink="..." portType="..." operation="..." inputVariable="..." outputVariable="..."/>`

<!-- Der Prozess erhält einen Aufruf des Partners: --> `<receive partnerLink="..." portType="..." operation="..." variable="..."[createInstance="..."]/>`

<!-- Der Process sendet eine Antwortnachricht in einem Partneraufruf: --> `<reply partnerLink="..." portType="..." operation="..." variable="..."/>`

<!-- Datenbelegung zwischen Variablen: --> `<assign> <copy> <from variable="..."/> <to variable="..."/> </copy>+ </assign>`

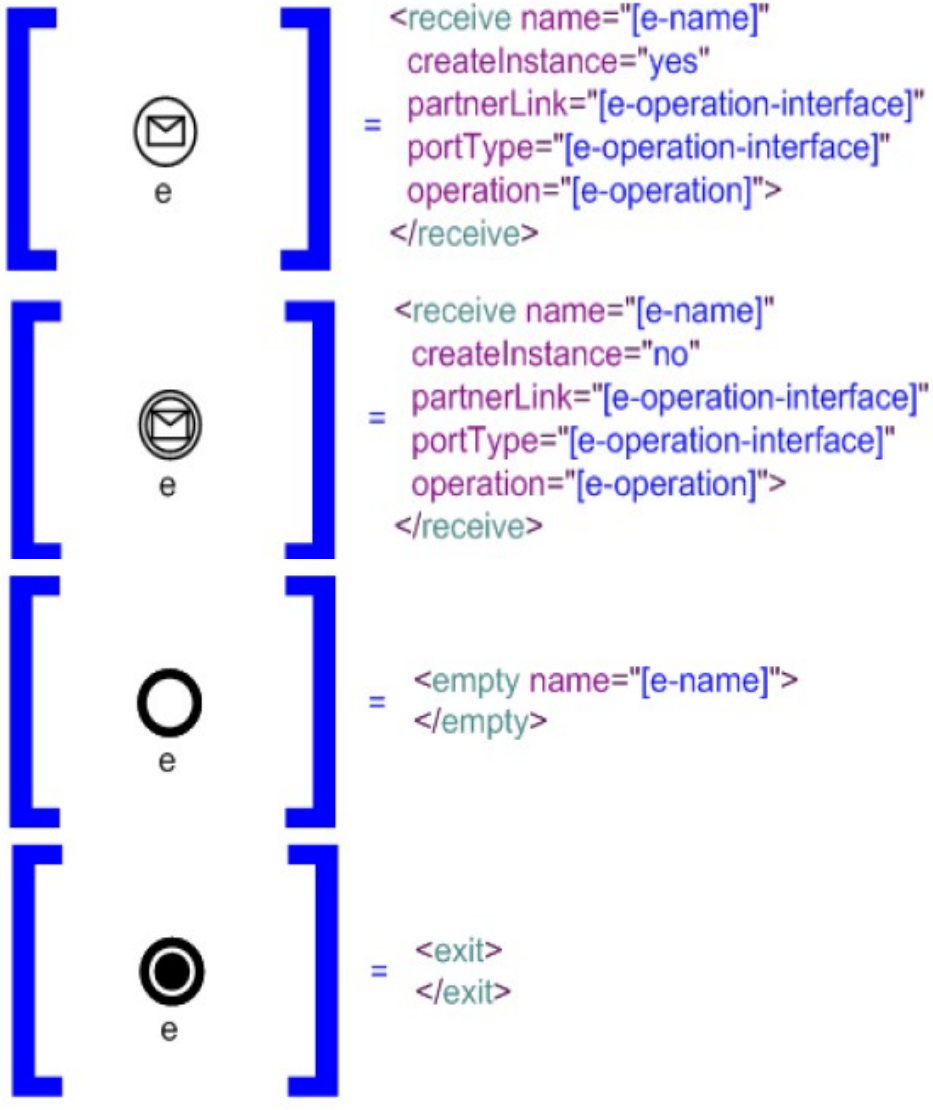
<!-- Der Prozess entdeckt einen Ausführungsfehler und wechselt in den Fehlerausführungsbetrieb: --> `<throw faultName="..." faultVariable="..."/>`

<!-- Den Prozess beenden: --> `<exit>`

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

<!-- Nichts tun (syntaktischer Zucker): --> `<empty>`

Transformation: Ereignisse



<!-- Der Prozess aktiviert eine Operation beim Partner: --> `<invoke partnerLink="..." portType="..." operation="..." inputVariable="..." outputVariable="..."/>`

<!-- Der Prozess erhält einen Aufruf des Partners: --> `<receive partnerLink="..." portType="..." operation="..." variable="..."[createInstance="..."]/>`

<!-- Der Process sendet eine Antwortnachricht in einem Partneraufruf: --> `<reply partnerLink="..." portType="..." operation="..." variable="..."/>`

<!-- Datenbelegung zwischen Variablen: --> `<assign> <copy> <from variable="..."/> <to variable="..."/> </copy>+ </assign>`

<!-- Der Prozess entdeckt einen Ausführungsfehler und wechselt in den Fehlerausführungsbetrieb: --> `<throw faultName="..." faultVariable="..."/>`

<!-- Den Prozess beenden: --> `<exit>`

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

<!-- Nichts tun (syntaktischer Zucker): --> `<empty>`

Transformation: Ereignisse

<!-- Der Prozess aktiviert eine Operation beim Partner: -->
```
<invoke partnerLink="..." portType="..."
operation="..." inputVariable="..."
outputVariable="..."/>
```

<!-- Der Prozess erhält einen Aufruf des Partners: -->
```
<receive partnerLink="..."
portType="..." operation="..."
variable="..."[createInstance="..."]/>
```

<!-- Der Process sendet eine Antwortnachricht in einem Partneraufruf: --> `<reply partnerLink="..." portType="..." operation="..." variable="..."/>`

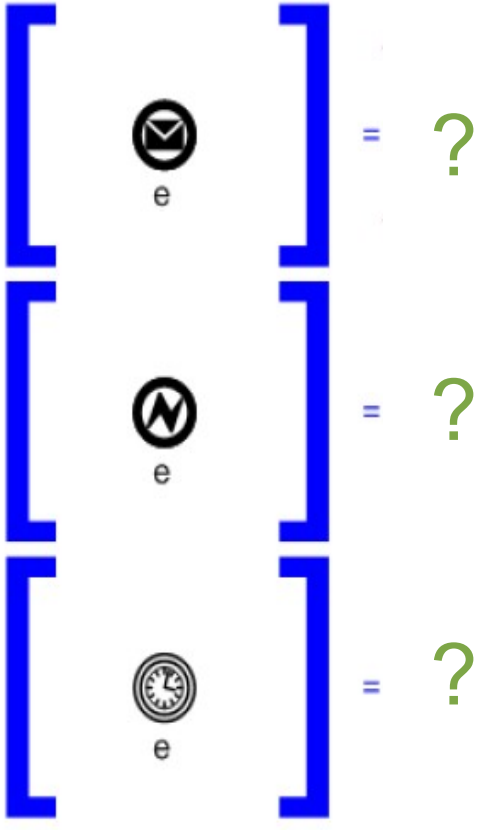
<!-- Datenbelegung zwischen Variablen: --> `<assign> <copy> <from variable="..."/> <to variable="..."/> </copy>+ </assign>`

<!-- Der Prozess entdeckt einen Ausführungsfehler und wechselt in den Fehlerausführungsbetrieb: --> `<throw faultName="..." faultVariable="..."/>`

<!-- Den Prozess beenden: --> `<exit>`

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

<!-- Nichts tun (syntaktischer Zucker): --> `<empty>`



Transformation: Ereignisse

<!-- Der Prozess aktiviert eine Operation beim Partner: -->
```
<invoke partnerLink="..." portType="..."
operation="..." inputVariable="..."
outputVariable="..."/>
```

<!-- Der Prozess erhält einen Aufruf des Partners: -->
```
<receive partnerLink="..."
portType="..." operation="..."
variable="..."[createInstance="..."]/>
```

<!-- Der Process sendet eine Antwortnachricht in einem Partneraufruf: --> `<reply partnerLink="..." portType="..." operation="..." variable="..."/>`

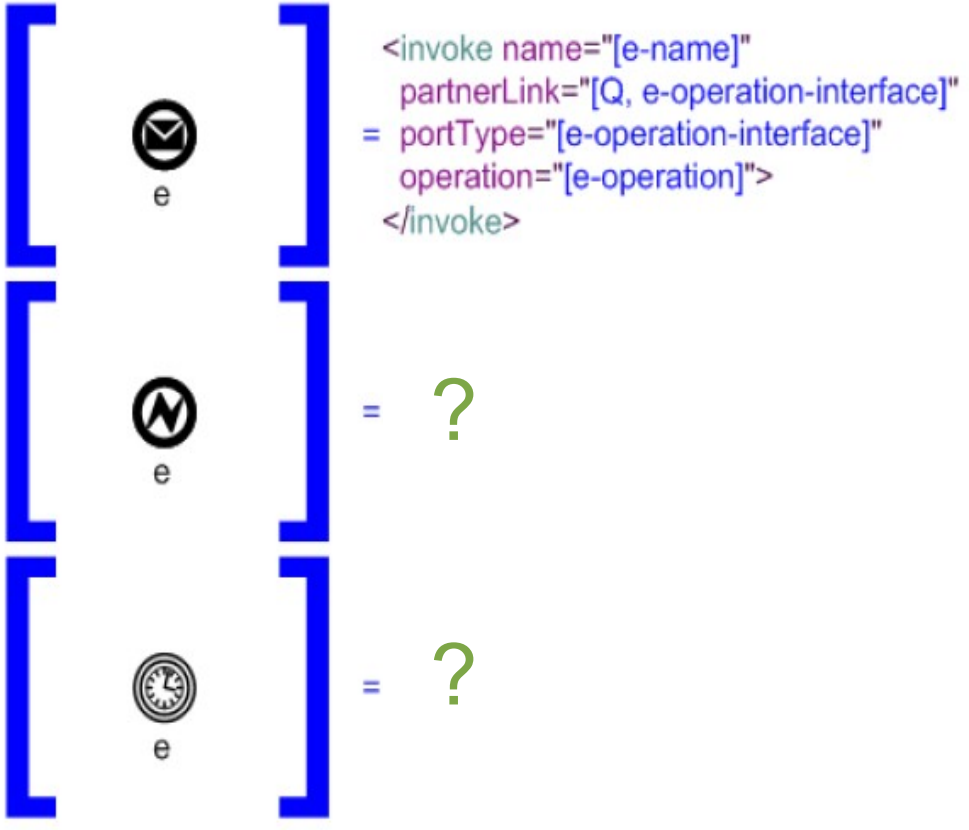
<!-- Datenbelegung zwischen Variablen: --> `<assign> <copy> <from variable="..."/> <to variable="..."/> </copy>+ </assign>`

<!-- Der Prozess entdeckt einen Ausführungsfehler und wechselt in den Fehlerausführungsbetrieb: --> `<throw faultName="..." faultVariable="..."/>`

<!-- Den Prozess beenden: --> `<exit>`

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

<!-- Nichts tun (syntaktischer Zucker): --> `<empty>`



Transformation: Ereignisse

<!-- Der Prozess aktiviert eine Operation beim Partner: -->
```
<invoke partnerLink="..." portType="..."
operation="..." inputVariable="..."
outputVariable="..."/>
```

<!-- Der Prozess erhält einen Aufruf des Partners: -->
```
<receive partnerLink="..."
portType="..." operation="..."
variable="..."[createInstance="..."]/>
```

<!-- Der Process sendet eine Antwortnachricht in einem Partneraufruf: --> `<reply partnerLink="..." portType="..." operation="..." variable="..."/>`

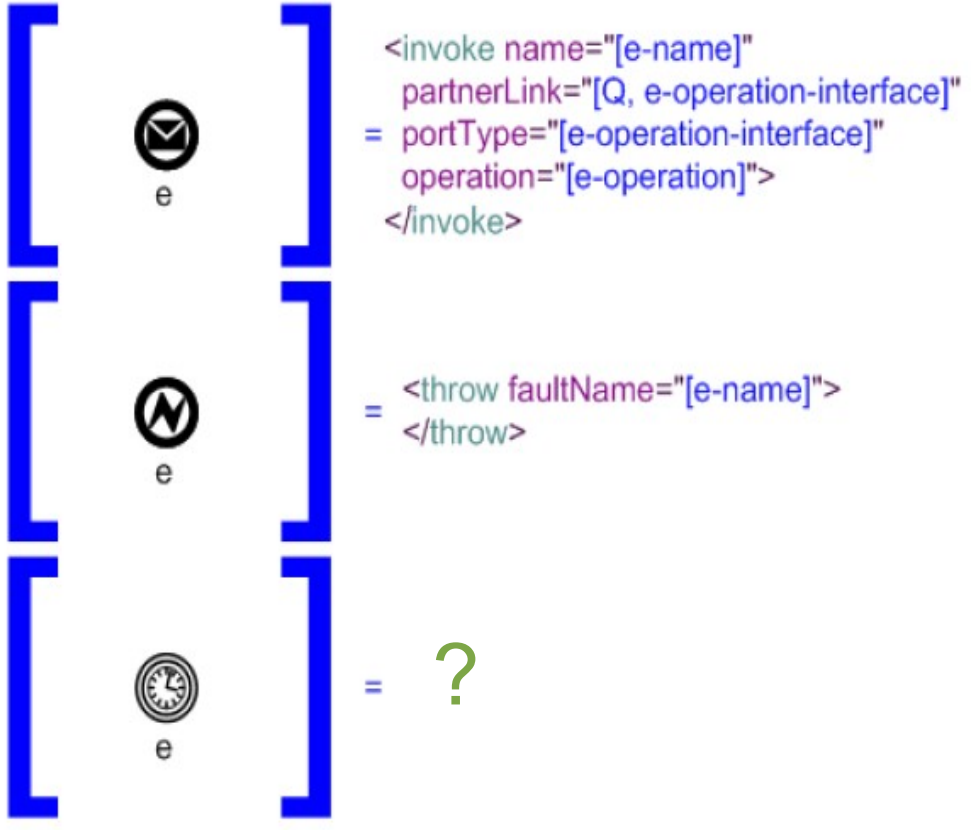
<!-- Datenbelegung zwischen Variablen: --> `<assign> <copy> <from variable="..."/> <to variable="..."/> </copy>+ </assign>`

<!-- Der Prozess entdeckt einen Ausführungsfehler und wechselt in den Fehlerausführungsbetrieb: --> `<throw faultName="..." faultVariable="..."/>`

<!-- Den Prozess beenden: --> `<exit>`

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

<!-- Nichts tun (syntaktischer Zucker): --> `<empty>`



Transformation: Ereignisse

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

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

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

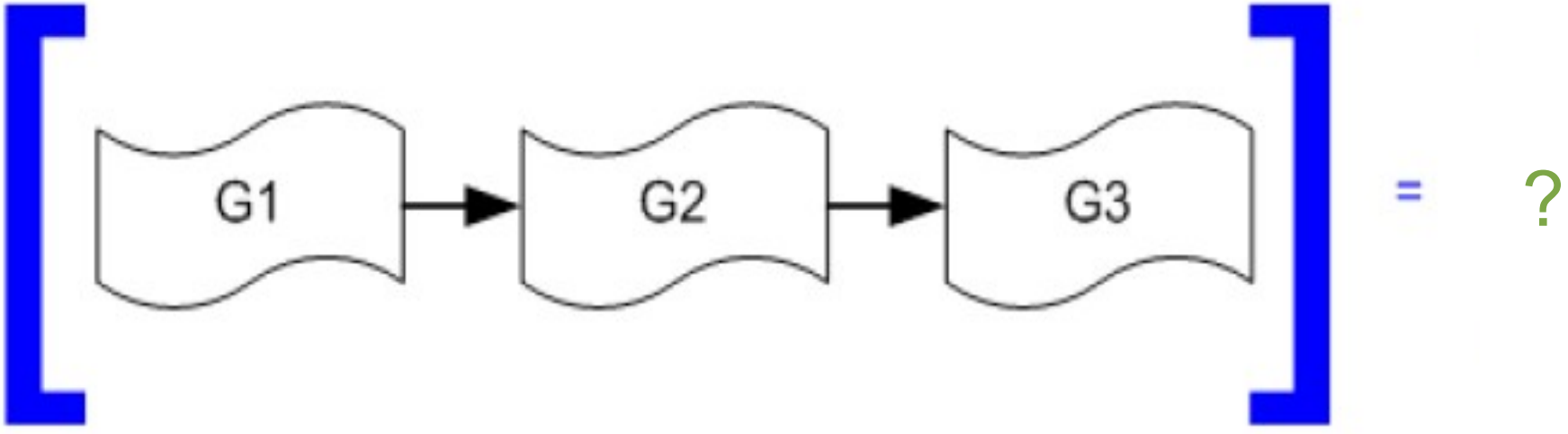
[e] =
```
<exit>
</exit>
```

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

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

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

Transformation: Sequenzen



<!-- Der Prozess entdeckt einen Ausführungsfehler und wechselt in den Fehlerausführungsbetrieb: -->
`<throw faultName="..." faultVariable="..."/>`

<!-- Den Prozess beenden: --> `<exit>`

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

<!-- Nichts tun (syntaktischer Zucker): --> `<empty>`

<!-- Sequenzielles Ausführen von Aktivitäten: --> `<sequence>`

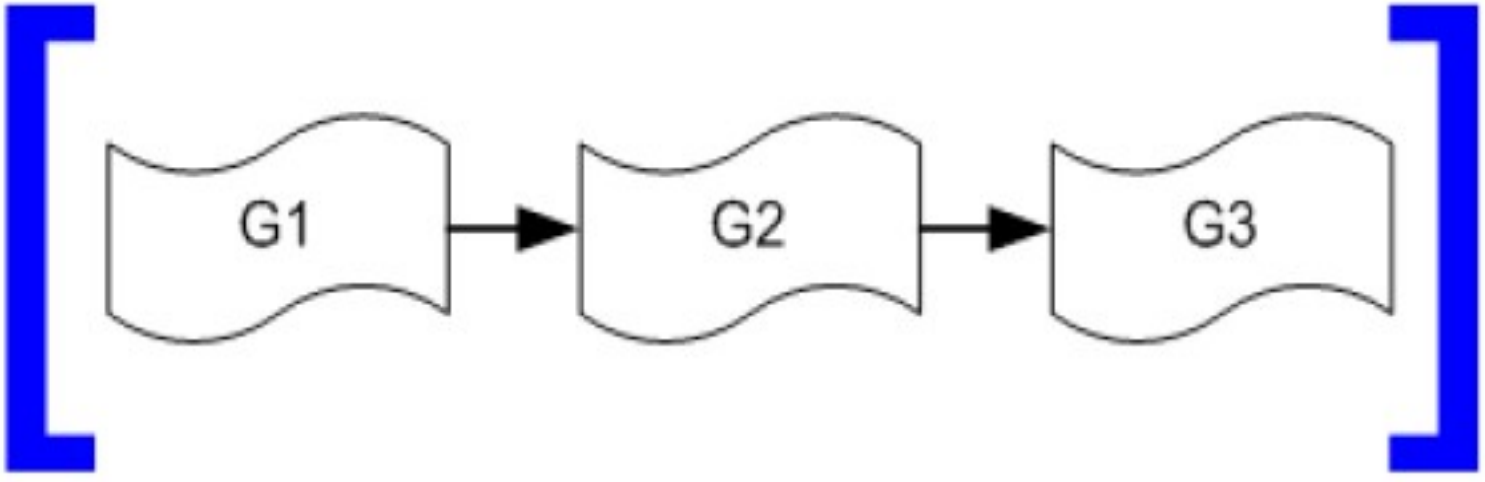
<!-- Paralleles Ausführen von Aktivitäten: --> `<flow>`

<!-- Iterieren der Ausführung von Aktivitäten bis Bedingung nicht erfüllt ist: --> `<while>`

<!-- Mehrere Event-Aktivitäten (z.B. Annehmen von Nachrichten, Zeit-Event) angesetzt für parallele Ausführung; erste eintretende wird ausgewählt und passender Code ausgeführt: --> `<pick>`

<!-- Definiert einen Kontrollzusammenhang zwischen einer Startaktivität und einem Ziel: --> `<link ...>`

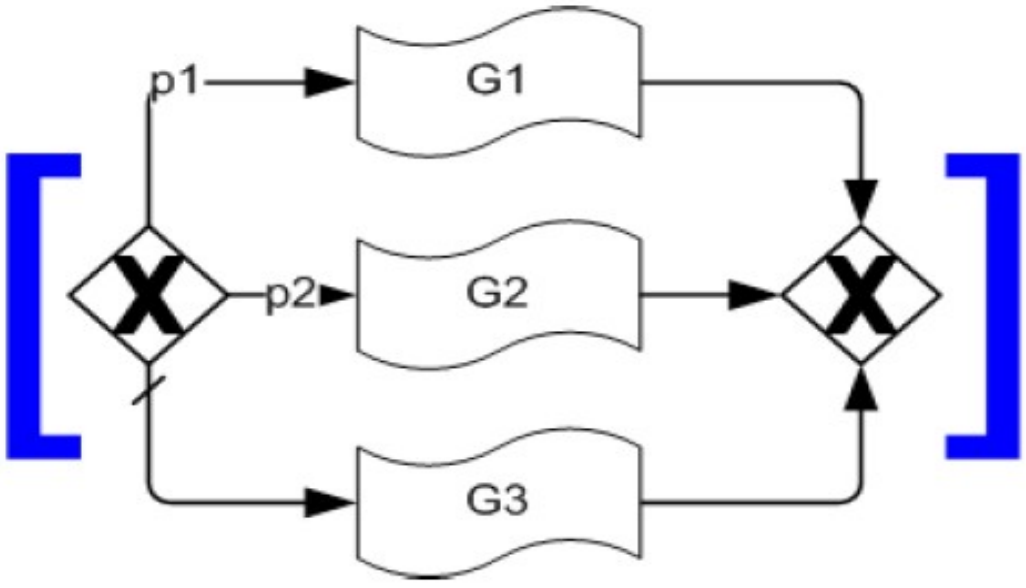
Transformation: Sequenzen



=

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

Transformation: If-Then-Else



=

?

<!-- Der Prozess entdeckt einen Ausführungsfehler und wechselt in den Fehlerausführungsbetrieb: --> `<throw faultName="..." faultVariable="..."/>`

<!-- Den Prozess beenden: --> `<exit>`

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

<!-- Nichts tun (syntaktischer Zucker): --> `<empty>`

<!-- Sequenzielles Ausführen von Aktivitäten: --> `<sequence>`

<!-- Paralleles Ausführen von Aktivitäten: --> `<flow>`

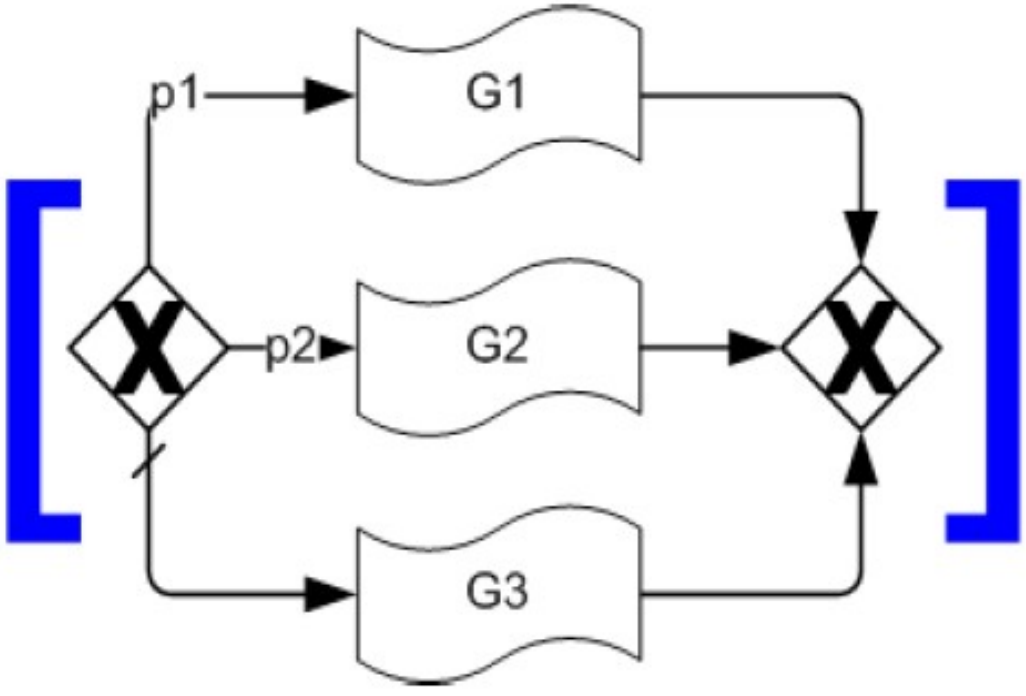
<!-- Iterieren der Ausführung von Aktivitäten bis Bedingung nicht erfüllt ist: --> `<while>`

<!-- Mehrere Event-Aktivitäten (z.B. Annehmen von Nachrichten, Zeit-Event) angesetzt für parallele Ausführung; erste eintretende wird ausgewählt und passender Code ausgeführt: --> `<pick>`

<!-- Definiert einen Kontrollzusammenhang zwischen einer Startaktivität und einem Ziel: --> `<link ...>`

<!-- Bedingte Verzweigung: --> `<if name="..."> <condition> ... </condition> ... <elseif> <condition> … </condition> ... </elseif> <else> ... </else> </if>`

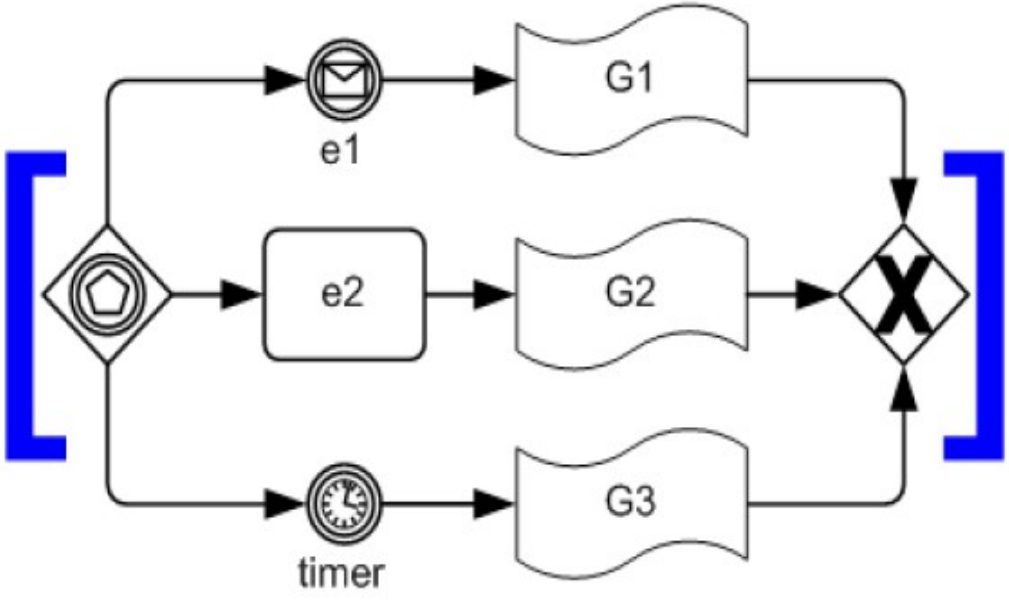
Transformation: If-Then-Else



=

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

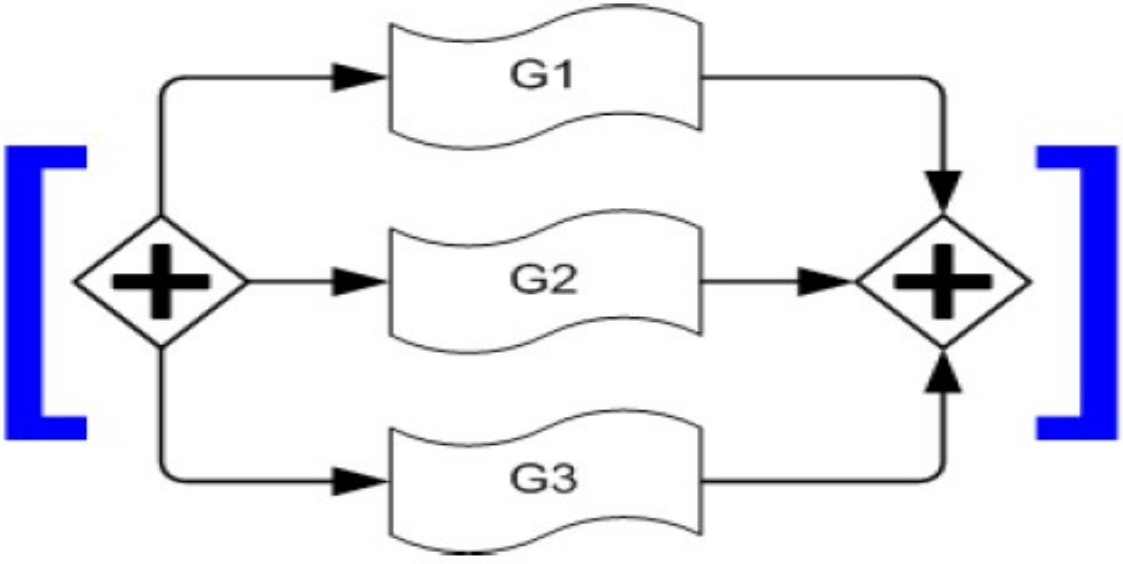
Transformation: Auswahl



=

```
<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: Parallele Ausführung



= ?

<!-- Der Prozess entdeckt einen Ausführungsfehler und wechselt in den Fehlerausführungsbetrieb: --> `<throw faultName="..." faultVariable="..."/>`

<!-- Den Prozess beenden: --> `<exit>`

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

<!-- Nichts tun (syntaktischer Zucker): --> `<empty>`

<!-- Sequenzielles Ausführen von Aktivitäten: --> `<sequence>`

<!-- Paralleles Ausführen von Aktivitäten: --> `<flow>`

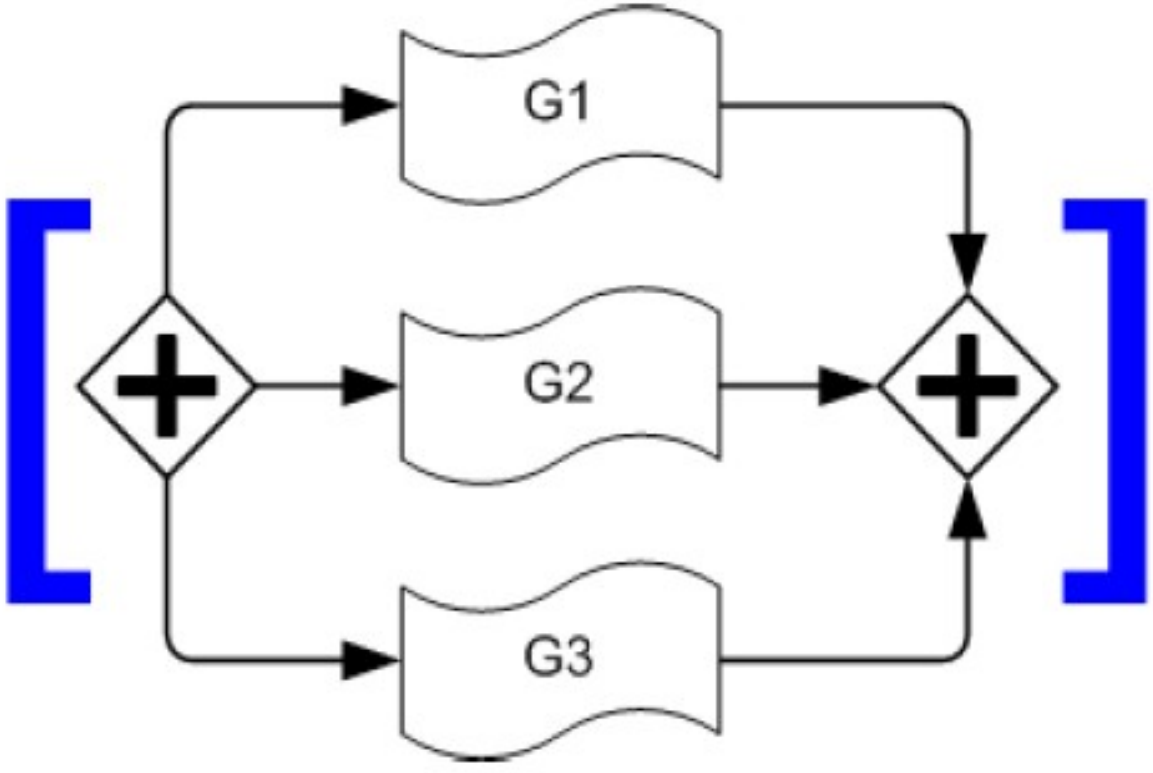
<!-- Iterieren der Ausführung von Aktivitäten bis Bedingung nicht erfüllt ist: --> `<while>`

<!-- Mehrere Event-Aktivitäten (z.B. Annehmen von Nachrichten, Zeit-Event) angesetzt für parallele Ausführung; erste eintretende wird ausgewählt und passender Code ausgeführt: --> `<pick>`

<!-- Definiert einen Kontrollzusammenhang zwischen einer Startaktivität und einem Ziel: --> `<link ...>`

<!-- Bedingte Verzweigung: --> `<if name="..."> <condition> ... </condition> ... <elseif> <condition> … </condition> ... </elseif> <else> ... </else> </if>`

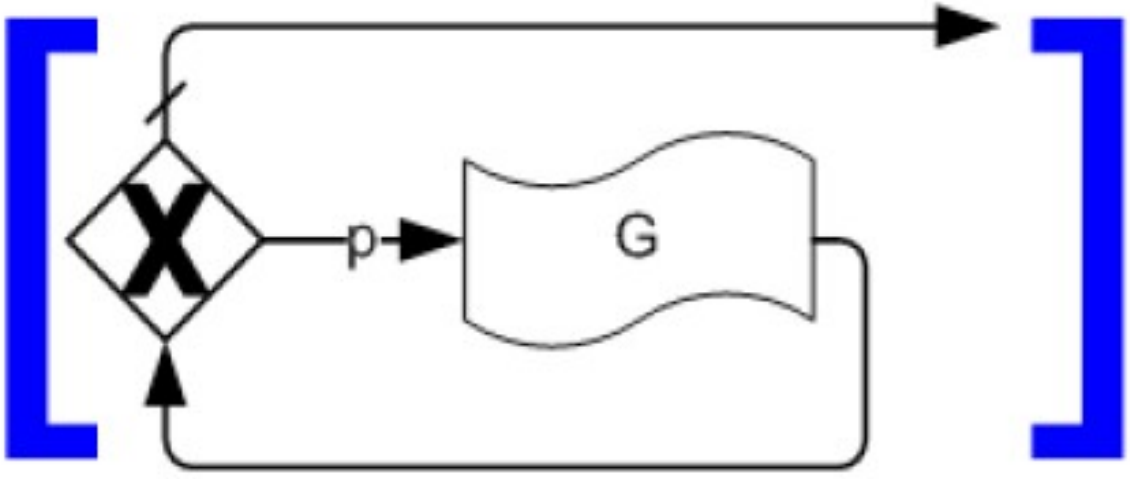
Transformation: Parallele Ausführung



=

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

Transformation: While-Schleifen



= ?

<!-- Der Prozess entdeckt einen Ausführungsfehler und wechselt in den Fehlerausführungsbetrieb: --> `<throw faultName="..." faultVariable="..."/>`

<!-- Den Prozess beenden: --> `<exit>`

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

<!-- Nichts tun (syntaktischer Zucker): --> `<empty>`

<!-- Sequenzielles Ausführen von Aktivitäten: --> `<sequence>`

<!-- Paralleles Ausführen von Aktivitäten: --> `<flow>`

<!-- Iterieren der Ausführung von Aktivitäten solange Bedingung erfüllt ist: --> `<while><condition>...</condition>...</while>`

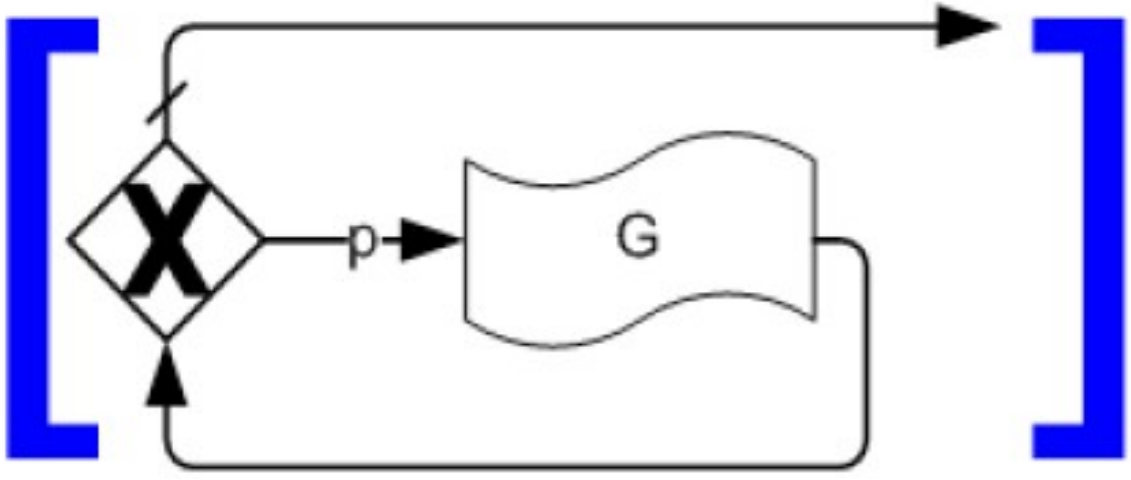
<!-- Iterieren der Ausführung von Aktivitäten bis Bedingung erfüllt ist: --> `<repeatUntil><condition>...</condition>...</repeatUntil>`

<!-- Mehrere Event-Aktivitäten (z.B. Annehmen von Nachrichten, Zeit-Event) angesetzt für parallele Ausführung; erste eintretende wird ausgewählt und passender Code ausgeführt: --> `<pick>`

<!-- Definiert einen Kontrollzusammenhang zwischen einer Startaktivität und einem Ziel: --> `<link ...>`

<!-- Bedingte Verzweigung: --> `<if name="..."> <condition> ... </condition> ... <elseif> <condition> … </condition> ... </elseif> <else> ... </else> </if>`

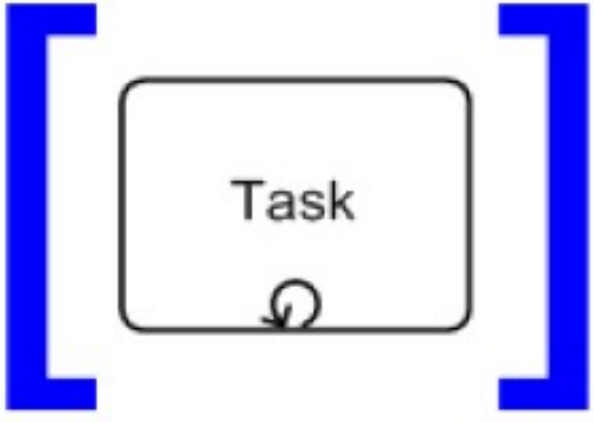
Transformation: While-Schleifen



=

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

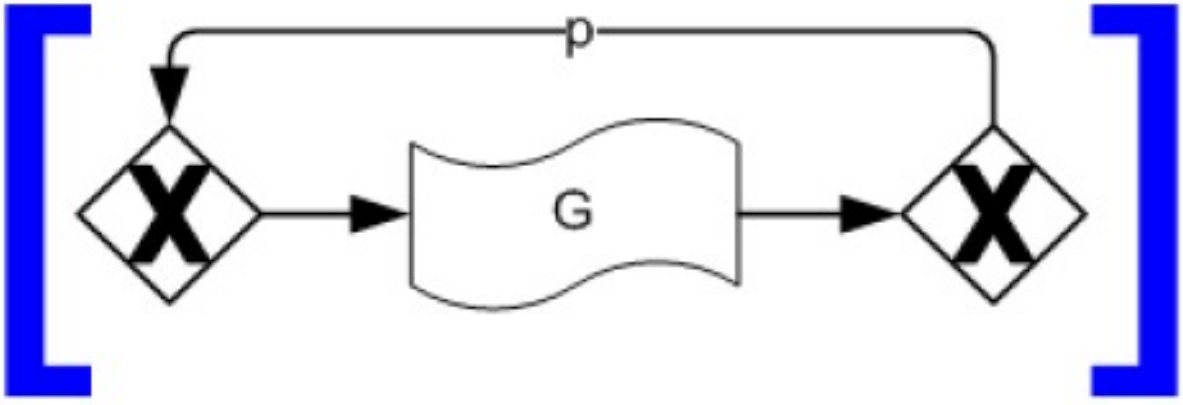
Analog:



=

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

Transformation: Until-Schleifen



= ?

<!-- Der Prozess entdeckt einen Ausführungsfehler und wechselt in den Fehlerausführungsbetrieb: --> `<throw faultName="..." faultVariable="..."/>`

<!-- Den Prozess beenden: --> `<exit>`

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

<!-- Nichts tun (syntaktischer Zucker): --> `<empty>`

<!-- Sequenzielles Ausführen von Aktivitäten: --> `<sequence>`

<!-- Paralleles Ausführen von Aktivitäten: --> `<flow>`

<!-- Iterieren der Ausführung von Aktivitäten solange Bedingung erfüllt ist: --> `<while><condition>...</condition>...</while>`

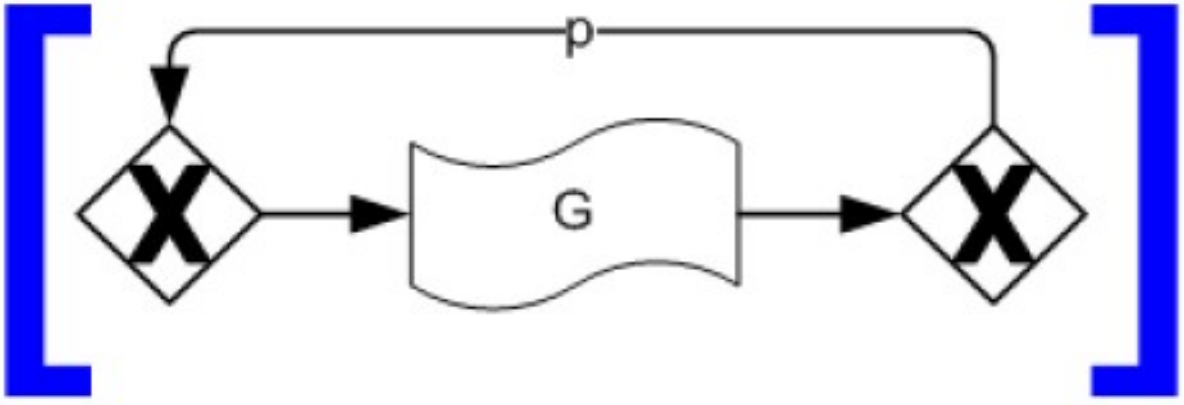
<!-- Iterieren der Ausführung von Aktivitäten bis Bedingung erfüllt ist: --> `<repeatUntil><condition>...</condition>...</repeatUntil>`

<!-- Mehrere Event-Aktivitäten (z.B. Annehmen von Nachrichten, Zeit-Event) angesetzt für parallele Ausführung; erste eintretende wird ausgewählt und passender Code ausgeführt: --> `<pick>`

<!-- Definiert einen Kontrollzusammenhang zwischen einer Startaktivität und einem Ziel: --> `<link ...>`

<!-- Bedingte Verzweigung: --> `<if name="..."> <condition> ... </condition> ... <elseif> <condition> … </condition> ... </elseif> <else> ... </else> </if>`

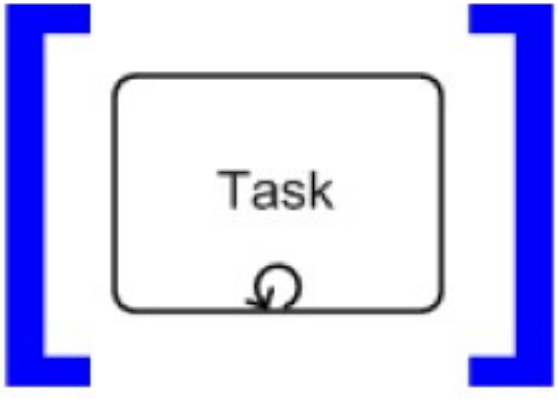
Transformation: Until-Schleifen



=

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

Analog:



=

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

Zusammenfassung: 2.6 Workflow-Automatisierung

In diesem Abschnitt haben wir folgendes behandelt:

- Native Metamodelle
- Modell-Transformationen
- Kurze Einführung in die BPEL-Notation
- Transformation von BPMN nach BPEL, mittels derer die BPMN 2.0-Notation eine Teilnotation enthält, die isomorph zu BPEL ist.

Zusammenfassung und Ausblick: Teil 2. Geschäftsprozesse

- Anwendungsbeispiel Finanz- und Versicherungsdomäne
- Geschäfts-Prozesse
  - Grundlagen Geschäfts-Prozesse
  - Einführung in die BPMN
  - Elektronische Prozessketten
  - Grundlagen der GP-Modellierung: Petri-Netze
  - Workflow-Management-Systeme
  - Workflow-Automatisierung
- Qualitätsmanagement
- Testen
- Sicherheit
- Sicheres Software Design