Übersetzung von graf- u. blockorientierten Prozessmodellierungssprachen

Seminararbeit

Zur Absolvierung der Lehrveranstaltung

Seminar für Diplomanden

im Rahmen des Studiums

Software Engineering / Internet Computing

erstellt von

Thomas Tschach
Matrikelnummer 0525381

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuerin: Mag. Dr. Katharina Kaiser

Wien, 10.07.2012

Kurzfassung

Es gibt eine Vielzahl an Prozessbeschreibungssprachen, die zur Beschreibung von Abläufen in unterschiedlichen Domänen entwickelt wurden. Sie unterscheiden sich meist durch domänenspezifische Konstrukte, können den Kontrollfluss jedoch nur mit dem graf- oder blockorientiert Kontrollflussparadigma darstellen. Wird ein Prozess, der in der Prozessbeschreibungssprache eines Paradigmas dargestellt wird, in eine Sprache des anderen Paradigmas umgewandelt, so sind diese Transformationen nicht immer trivial umzusetzen.
Probleme entstehen dadurch, dass graf- und blockorientierte Modelle unterschiedliche Ausdruckskraft haben, Kontrollflüsse zu modellieren und dadurch möglicherweise Kontrollflussmuster unterstützen, die im Zielmodell nicht abgebildet werden können.
In dieser Arbeit werden Transformationsstrategien für den Kontrollfluss zwischen graf- und blockorientierten Modellen erläutert, die versuchen, die o.a. Einschränkungen zu umgehen. Die Strategien beschreiben Transformationen zwischen einer BPEL-Abstraktion, als Vertreter des blockorientierten Paradigmas, und einer Abstraktion verschiedener graforientierter Workflow-Modelle. Der Grundgedanke der unterschiedlichen Strategien kann in weiterer Folge für Transformationen zwischen anderen Prozessmodellen übernommen und angepasst werden.

Inhaltsverzeichnis

Kurzfassung 2

1. Einführung 4

2. Transformationen von Workflow-Modellen 5

   2.1. Definition graforientierter Beschreibungssprachen 6

   2.2. Definition blockorientierter Beschreibungssprachen 8

   2.3. Strukturierte Prozessgrafen und andere Eigenschaften 10

3. Herausforderungen bei der Transformation 11

4. Transformationsstrategien von graf- zu blockorientierten Sprachen 13

   4.1. Element-Preservation Strategie 14

   4.2. Element-Minimization Strategie 17

   4.3. Structure-Identification Strategie 18

   4.4. Structure-Maximization Strategie 20

   4.5. Event-Condition-Action-Rules Strategie 20

   4.6. Zusammenfassung der Strategien 22

5. Transformationsstrategien von block- zu graforientierten Sprachen 24

   5.1. Flattening Strategie 24

   5.2. Hierarchy-Preservation Strategie 25

   5.3. Hierarchy-Maximization Strategie 26

   5.4. Zusammenfassung der Strategien 26

6. Weitere Transformationsansätze 27

   6.1. Model Transformation by-Example (MTBE) 28

   6.2. Musterbasierte Modelltransformation 30

7. Konklusion und weitere Studien 31

Referenzen 32

1. Einführung

Prozessbeschreibungssprachen haben sich sowohl in geschäftlichen wie auch technischen Bereichen etabliert, um z.B. geschäftliche, technische Prozesse oder Verhalten von Softwarekomponenten zu beschreiben. Auch in vielen anderen Bereichen finden Workflow-Modelle ihren Einsatz. So vielfältig wie die Anwendungsgebiete sind, so vielfältig sind auch die vorhandenen Sprachen.

Da Modelle Abstrahierungen der realen Welt sind, unterscheiden sich die Sprachen nach den Möglichkeiten, Entitäten der realen Welt, deren Eigenschaften und Verhalten zu modellieren. Aber auch die Fähigkeit, verschiedenste Abfolgen von Aktivitäten auszudrücken (Kontrollfluss) und wie diese repräsentiert werden, sind Unterscheidungsmerkmale [1].
Es gibt zwei Paradigmen zur Darstellung des Kontrollflusses [2]:

(1) Graforientierte Modelle benutzen gerichtete Kanten, um die Abfolge von Aktivitäten und deren Abhängigkeiten zu definieren (Beispiel siehe Abb. 1). Vertreter dieser Sprachen sind z.B. Business Process Model Notation (BPMN) [3], UML Activity Diagrams [4], Petri Nets [5], Event-Driven Process Chains (EPC) [6], YAWL [7] und andere.

(2) Blockorientierte Modelle stellen strukturierte Konstrukte zu Verfügung, um die Abfolge von Aktivitäten darzustellen. Verschachtelt man diese, werden komplexe Prozesse geschaffen (Beispiel siehe Abb. 3). Vertreter dieses Paradigmas sind die Business Process Execution Language for Web Services (BPEL4WS oder BPEL) [8], XLANG oder auch Asbru [9].

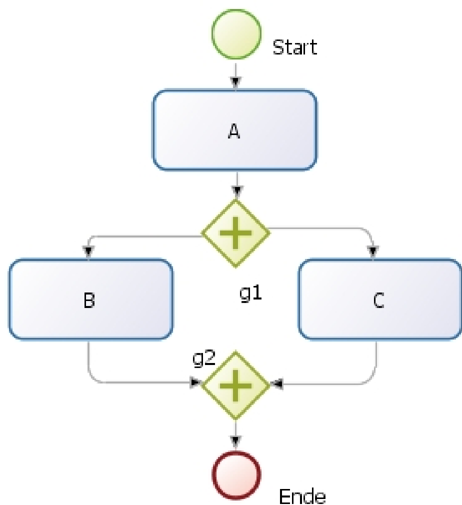

Abb. 1: Beispielprozess in graforientierter BPMN Notation (grafische Darstellung)

```
<?xml version="1.0" encoding="UTF-8"?>
definitions>
  <process id="Beispielprozess" name="Beispielprozess">
    <startEvent id="Start" name="Start"/>
    <endEvent id="Ende" name="Ende"/>

    <parallelGateway id="g1" name="g1"/>
    <parallelGateway id="g2" name="g2"/>

    <task id="A" name="A"/>
    <task id="C" name="C"/>
    <task id="B" name="B"/>

    <sequenceFlow sourceRef="Start" targetRef="A" ... />
    <sequenceFlow sourceRef="A" targetRef="g1" ... />
    <sequenceFlow sourceRef="g1" targetRef="C" ... />
    <sequenceFlow sourceRef="C" targetRef="g2" ... />
    <sequenceFlow sourceRef="g1" targetRef="B" ... />
    <sequenceFlow sourceRef="B" targetRef="g2" ... />
    <sequenceFlow sourceRef="g2" targetRef="Ende" ... />
  </process>
</definitions>
```

Abb. 2: Beispielprozess aus Abb. 1 in BPMN Modell-Code

Abb. 1 zeigt die grafische Darstellung eines Prozesses in BPMN Notation, bei dem zuerst Aktivität A ausgeführt wird, und danach die Aktivitäten B und C gleichzeitig gestartet und parallel ausgeführt werden. Abb. 2 zeigt den dazugehörigen BPMN Modell-Code im XML-Format. Die Knoten werden durch die startEvent-, endEvent-, task- und parallelGateway-Elemente repräsentiert. Die Kanten werden durch das sequenceFlow-Element dargestellt, die die Knoten per sourceRef- und targetRef-Attribut miteinander verbinden.

```
<process name="Beispielprozess">...
  <sequence>
    <invoke operation="A".../>
    <flow>
       <invoke operation="B".../>
       <invoke operation="C".../>
     </flow>
  </sequence>
</process>
```

Abb. 3: Beispielprozess aus Abb. 1 als blockorientierter BPEL-Code

Im blockorientierten BPEL-Code (Abb. 3) wird dies durch die Verschachtelung des flow-Elements im sequence-Element ausgedrückt.

Modelltransformationen wandeln einen Workflow bzw. eine Modellinstanz in einen Workflow eines anderen Metamodells um. Modelltransformationen werden eingesetzt, um z.B. Workflows an das Eingangsformat eines anderen Modellierungstools anzupassen oder um das Modell in einen ausführbaren Prozess zu übersetzen [2].

Graforientierte Modelle sind ausdruckstärker, da sie durch die Verkettung von Knoten mit Hilfe gerichteter Kanten Abfolgen von Aktivitäten definieren können. So können z.B. Schleifen durch zyklische Kantenfolgen konstruiert werden, die mehrere Ein- und Ausstiegspunkte haben. Dies lässt sich mit strukturierten Konstrukten, wie sie blockorientierte Sprachen verwenden, nicht darstellen. Da strukturierte Abläufe besser verständlich und weniger Fehleranfällig als unstrukturierte sind, sollten sie jedoch möglichst vermieden werden [10].

Probleme treten bei Modelltransformationen dann auf, wenn im Quellmodell Kontrollflüsse konstruiert werden können, die im Zielmodell nicht dargestellt werden können. Dies führt entweder zu einer Einschränkung des Eingangsmodells oder zu einem fehlerhaften bzw. nicht äquivalenten Übersetzungsresultat.
In dieser Arbeit werden nun einige Themen besprochen, um korrekte Modelltransformation zwischen graf- und blockorientierten Modellen durchzuführen.
Zuerst werden grundlegende Informationen zu Prozessmodellen vermittelt (Abschnitt 2). Dazu gehört auch ein formales Modell um graf- bzw. blockorientiert Workflow-Modelle zu beschreiben.
In Abschnitt 2.3 werden wichtige Eigenschaften von Workflow-Modellen definiert, um Einschränkungen des Eingangsmodells besser zu verdeutlichen.
Anschließend werden typische Diskrepanzen zwischen graf- und blockorientierten Modellen beschrieben, und strukturierte Komponenten definiert (Abschnitt 3).
Das Kernstück der Arbeit sind die Erläuterungen der Strategien von graf-zu-blockorientierten Modellen (Abschnitt 4) und block-zu-graforientierten Modellen (Abschnitt 5).

2. Transformationen von Workflow-Modellen

Verschiedenste Sprachen wurden entwickelt, um Business Prozess Modelle (BPM) bzw. Workflow-Modelle (WfM) zu beschreiben. Die Zielsetzungen dieser Sprachen unterscheiden sich und sind vielfältig. Sie werden u.a. zur Dokumentation, zur formalen Analyse von Geschäftsprozessen oder zur Komposition von Services herangezogen [1].

Workflows sind Instanzen eines WfM und beschreiben, im Wesentlichen, Aktivitäten und die Abfolge wie sie ausgeführt werden, um einen Geschäftsprozess darzustellen. Bei der Definition der Abfolge

der auszuführenden Aktivitäten spricht man auch vom Kontrollfluss (control flow) des Prozesses. Dient der Workflow nur zur Dokumentation, spricht man von einem abstrakten Prozess (abstract process). Ist der Workflow auch ausführbar, spricht man von einem ausführbaren Prozess (executable process) [11].

In den letzten Jahrzehnten wurde versucht, die Vielfalt an BPM Sprachen in einem Standard zu bündeln. Diese Bemühungen scheiterten jedoch hauptsächlich an den unterschiedlichen, domänenspezifischen Konstrukten der verschiedenen Sprachen und an den ungleichen Paradigmen, wie der Kontrollfluss dargestellt wird [2]. Wie schon in Abschnitt 1 angeführt, sind dies das graf- und das blockorientiert Paradigma (genaueres dazu folgt in Abschnitt 2.1 und 2.2).

Modelle sind abstrahierte, vereinfachte Darstellungen des zu beschreibenden Objekts, im Fall von WfM eben Workflows. In der Softwareentwicklung waren Modelle meist Entwürfe für das zu implementierende System und dienten als Vorlage für den Programmierer, der die entsprechende Implementierung realisierte.
Model-Driven-Development (MDD) geht einen Schritt weiter und generiert auf Basis des Modells ausführbaren Code. Manchmal sind davor noch Modelltransformationen notwendig, um die Code-Generierung zu optimieren. Dabei wird das Quellmodell nicht in ausführbaren Code, sondern in ein anderes, das Zielmodell, transformiert.
Modelltransformationen werden auch verwendet, um Modelle an das Eingangsformat eines anderen Tools oder eines Standards anzupassen, da der Entwicklungsprozess möglicherweise von mehreren Werkzeugen unterstützt wird, und ein Modell unter ihnen ausgetauscht werden soll. Des Weiteren können Modeltransformationen eingesetzt werden, wenn das Quellmodell analysiert werden soll und es dafür keine entsprechenden Formalismen besitzt, das Zielmodell jedoch schon [2].
Wird ein WfM nur unzureichend durch vorhandene Werkzeuge unterstützt, kann eine Modelltransformation eine Lösung darstellen. Das gewünschte Modell wird in einem anderen, etablierten Modell mit guter Modellierungsinfrastruktur modelliert, und anschließend in das eigentlich gewünschte Workflow-Modell umgewandelt.

Abgesehen von den Gründen, warum ein Modell in ein anderes umgewandelt wird, gibt es generelle Schlüsselanforderungen an diese Transformationen, die wie folgt lauten [12]:

- Vollständigkeit (completness): Jede mögliche Instanz des Quellmodells soll in eine entsprechende Instanz des Zielmodells umgewandelt werden können.
- Automatisierung (automation): Die Instanz des Zielmodells soll automatisch generiert werden und manuelle Transformationsschritte vermieden werden.
- Lesbarkeit (readability): Die generierte Zielmodellinstanz soll gut lesbar und auch ohne weitere Analyseschritte für menschliche Betrachter verständlich sein.

Diese Anforderungen zu erfüllen ist nicht trivial und auch nicht immer vollständig erreichbar. Vor allem wenn das Quellmodell weniger restriktiv als das Zielmodell ist und Konstrukte möglich sind, die im Zielmodell nicht unterstützt werden, leidet die Vollständigkeit der Umwandlung.
In den nächsten Abschnitten wollen wir uns mit allgemeinen Definitionen und Eigenschaften von graf- und blockorientierten Beschreibungssprachen beschäftigen und einige Transformationsstrategien aufzeigen. Zuerst werden jedoch die Grundlagen von BPEL vermittelt.

2.1. Definition graforientierter Beschreibungssprachen

Graforientierte Beschreibungssprachen für Workflow-Modelle beschreiben eine Instanz durch einen Prozessgrafen (process graph, PG), der aus Knoten und Kanten besteht (siehe Abb. 4). Knoten können untrennbare Aktionen, Konnektoren, Start- und Endzustände sein. Diese Knoten werden mit Kanten

verbunden und definieren so die zeitlichen und logischen Abhängigkeiten zwischen den Knoten [2]. Kanten können auch eine Bedingung (guard) haben, die boolsche Ausdrücke sind. Ist die Kantenbedingung erfüllt, dann wird der Zielknoten der Kante aktiviert.
Konnektoren verzweigen bzw. vereinigen den Kontrollfluss und können anhand ihres Verhaltens in AND-, OR- oder XOR-Splits bzw. -Joins unterteilt werden.

Abb. 4 zeigt einen Beispielprozess anhand eines graforientierten Prozessmodells.
Er enthält Knoten und Kanten. Knoten können Start- (Stern im Kreis), Endzustände (Kreis im Kreis), Aktivitäten (A, B,...) oder Konnektoren (Rautensymbole) sein. Kanten verbinden diese Knoten und definieren so die Abfolge der Aktivitäten.

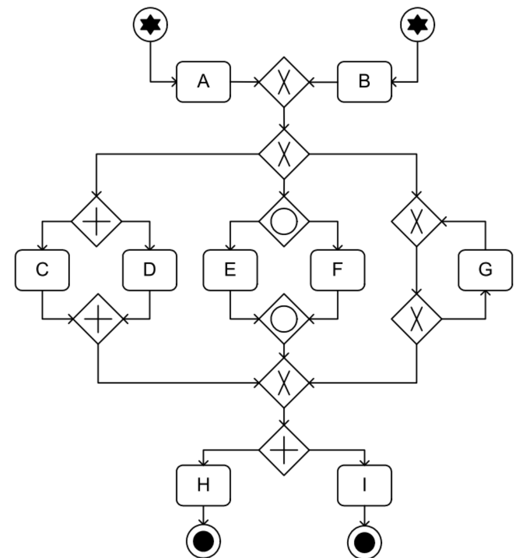

Abb. 4: graforientierte Darstellung eines Beispielprozesses [2]

In [2] wird eine Prozessbeschreibungssprache, namens Process Graph, definiert, die die Kernelemente von YAWL und EPC abstrahiert.
Sie kann auch auf andere graforientierte Modelle angewandt werden und müsste nur um spezifische Elemente erweitert werden. Die Process Graph Definition enthält genau jene Elemente, um den Kontrollfluss mit Hilfe von Kanten und Knoten darzustellen. Die meisten graforientierten Modelle enthalten diese Elemente als Kern ihrer Spezifikation, aber bauen diese auch mit zusätzlichen Elementen aus, die die Ausdruckkraft erhöhen.
Die folgende formale Spezifikation von Process Graph (Definition 1) kann erweitert werden, um den Anforderungen anderer graforientierter Modelle gerecht zu werden. Für BPMN [3] z.B., müsste die u.a. Definition um das Konzept der Events erweitert werden. Im Weiteren dient sie uns als Basis für die Beschreibung einiger Transformationsstrategien.

Die nachfolgende Definition 1 stellt die formale Definition der Sprache Process Graph dar. Wir verwenden diese Spezifikation zur Darstellung eines Prozessgrafen, um dessen Bestandteile zu erläutern und die Transformationsstrategien in Abschnitt 2.5 verständlicher zu machen.

Definition 1: Ein Prozessgraf(PG) ist ein Tupel $PG = \{S, E, F, C, l, A, g\}$, das aus vier disjunkten Mengen S, E, F, C, einer Funktion $l: C \rightarrow \{AND, OR, XOR\}$, einer binären Relation $A \subseteq (S \cup F \cup C) \times (E \cup F \cup C)$ und einer Funktion $g: A \rightarrow expr$ besteht [2]. Weiters muss gelten:

- S ist die Menge an Startzuständen. $|S| \geq 1$und $\forall s \in S: |succ^1(s)| = 1 \wedge |pred^2(s)| = 0$
- E ist die Menge der Endzustände. $|E| \geq 1$ und $\forall e \in E: |succ(e)| = 0 \wedge |pred(e)| = 1$
- F ist die Menge an Aktivitäten. $\forall f \in F: |succ(f)| = 1 \wedge |pred(f)| = 1$
- C ist die Menge an Konnektoren. $\forall c \in C: |succ(c)| = 1 \wedge |pred(c)| > 1 \vee |succ(c)| > 1 \wedge |pred(c)| = 1$
- Funktion l spezifiziert den Typ eines Konnektors $c \in C$ als AND, OR oder XOR [2].$l: C \rightarrow \{AND, OR, XOR\}$

[1] succ(n) gibt die Menge der Nachfolgeknoten eines Knoten n an: $succ(n) = \{x \in N | (n, x) \in A\}$
[2] pred(n) gibt die Menge der Vorgängerknoten eines Knoten n an: $pred(n) = \{x \in N | (x, n) \in A\}$

- A definiert die Menge der gerichteten Kanten die die Knoten im Graf verbinden. Es gibt keine Kanten mit identen Start- und Zielknoten (reflexiv) und keine paarweise identen Kanten (mehrfach Kanten). $A \subseteq (S \cup F \cup C) \times (E \cup F \cup C)$

- Die Funktion g gibt für eine Kante $a \in A$ die Kantenbedingung, also den boolschen Ausdruck $expr$, zurück. Gehen die Kanten von einem Konnektor des Types XOR-Split aus, dann müssen alle Kantenbedingungen so gestaltet sein, dass exakt eine Kantenbedingung erfüllt ist (mutually exclusiv). Kantenbedingungen können nur auf Kanten definiert werden, die von einem XOR- oder OR-Split ausgehen($(c, n) \in A | \, l(c) \neq AND \, \wedge n \in E \cup F \cup C$). Die Bedingungen aller anderen Kanten, also Kanten von AND-Splits und Sequenzen, sind immer erfüllt.

Mit der anschließenden Bestimmung der transitiven Hülle, kann die Erreichbarkeit zweier Knoten innerhalb des PG dargestellt werden.

Definition 2: Transitive Hülle (transitiv closure) A* eines Prozessgrafen $PG = \{S, E, F, C, l, A, g\}$ ist die transitive Hülle über die Kantenmenge A [2]. D.h. ist $(n1, n2) \in A^*$, dann gibt es einen Pfad von n1 nach n2 über eine Folge von Kanten aus A.

2.2. Definition blockorientierter Beschreibungssprachen

Blockorientierte Beschreibungssprachen definieren den Kontrollfluss über strukturierte Aktivitäten. Durch deren Verschachtelung können komplexe Abläufe erstellt werden (siehe Abb. 5).
Es gibt Element für sequentiellen (sequence), alternativen (switch), parallelen Kontrollfluss (flow), so wie Schleifen (while) und alternative Startbedingungen (pick). Über Links können Aktivitäten verbunden werden, und komplexere Synchronisationen erzielt werden.

Abb. 5 zeigt den Prozess aus Abb. 4 in der schematischen Darstellung eines blockorientierten Prozessmodells. Strukturierte Aktivitäten werden durch Rechtecke (Sequence, Pick, ...), Basisaktivitäten durch abgerundete Rechtecke (A, B, ...), Links durch Kanten dargestellt. Durch Verschachtelung wird der gewünschte Kontrollfluss definiert.

BPEL ist ein Beispiel für ein blockorientiertes Workflow-Modell und eignet sich gut zur Veranschaulichung von Transformationsstrategien, da man in der Literatur viele verschiedene Transformationsprozesse findet, die Übersetzungen von oder zu BPEL implementieren [2]. Es können verschiedene Übersetzungsansätze mit BPEL umgesetzt werden, da es kein rein blockorientiertes Prozessmodell darstellt, sondern auch graforientierte Elemente (Links) aufweist. Außerdem besitzt BPEL interessante Konstrukte, wie Event-Handler, mit denen ein Kontrollfluss simuliert werden kann.
Aus diesem Grund eignet sich BPEL gut, um mit ihm Transformationsstrategien von und zu blockorientierten Prozessmodellen zu veranschaulichen.

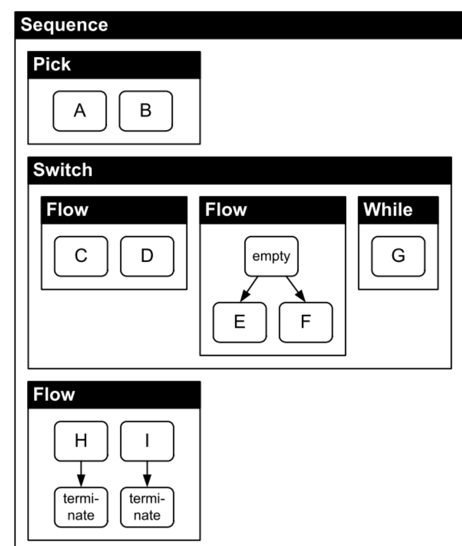

Abb. 5: schematisch blockorientierte Darstellung eines Beispielprozesses [2]

In [2] werden die Kernkonzepte von BPEL abstrahiert, und daraus das Workflow-Modell namens BPEL Control Flow (BCF) definiert. BCF kann jedoch auf jede andere blockorientierte Prozessbeschreibung umgelegt werden.
Da BCF eine vereinfachte Darstellung von BPEL ist, in der die für die Repräsentation des Kontrollflusses relevanten Elemente definiert werden, und die nachfolgenden Strategien auf BPEL

Mechanismen zurückgreifen, vereinfacht ein grundlegendes Wissen über BPEL das Verständnis für die nachfolgenden Abschnitte.

Business Process Executuion Language (BPEL) Grundlagen

BPEL ist eine weitverbreitete Sprache im Bereich der Web Service Orchestration. Dabei werden Geschäftsprozesse auf Basis bestehender Web Services (WS) erstellt und selbst als Web Service verfügbar gemacht [1]. BPEL wird in einem XML-Format dargestellt und ist weitgehend blockorientiert und ausführbar.

BPEL kombiniert Aktivitäten zu einem Workflow. Man kann zwei Arten von Aktivitäten unterscheiden, nämlich Basisaktivitäten (basic activities) und strukturierte Aktivitäten (structured activities) [12].
Basisaktivitäten führen einfache Aktionen aus, wie Aufruf eines Web Services (invoke), warten auf einen Aufruf (receive) und eine Nachricht als Rückgabewert auf einen Aufruf an den Aufrufer zurückschicken (reply) [1] [12]. Weiters gibt es Basisaktivitäten exit, wait und empty, mit denen der gesamte Prozess beendet, eine spezifizierte Zeitspanne gewartet oder nichts gemacht wird [1] [13].

Über strukturierte Aktivitäten wird in BPEL der Kontrollfluss des Prozesses definiert, durch deren Verschachtelung komplexe Abfolgen erzeugt werden können. Zu ihnen zählen sequence, um sequentielle Abläufe, switch, um alternative Abläufe, und flow, um parallele Abläufe zu beschreiben. Mit while werden Schleifen ausgedrückt, und mit pick werden alternative Startbedingungen für eine Aktivität definiert. Scope gruppiert Aktivitäten und begrenzt den Gültigkeitsbereich von event- und exception-handler [13].

Event-Handler können mit einem Scope assoziiert werden und stellen Event-Aktions-Regeln (event-action rule) dar. D.h. wird ein Prozess gerade ausgeführt und befindet sich innerhalb eines Scopes mit dem ein Event-Handler assoziiert ist und es tritt ein passender Event auf, dann wird die entsprechende Aktivität des Handlers ausgeführt [12].
BPEL kann den Kontrollfluss nicht nur blockorientiert festlegen, sondern hat auch graforientierte Elemente, die Links. Innerhalb eines flow-Elements können Aktivitäten mit Links verbunden werden. Mit Hilfe von transition-conditions und join-conditions kann der Kontrollfluss festgelegt werden [11].

Business Prozesse werden häufig mit Hilfe von BPEL und Web Services realisiert. BPEL stellt zwar einen de-facto Standard im Hinblick auf die Realisierung da, eignet sich jedoch nicht zum Modellieren auf technisch unabhängigem Abstraktionsniveau und wird deshalb nur selten von Analysten eingesetzt. BPMN oder UML Activity Diagrams eignen sich besser dafür. Auf Grund der zunehmenden Relevanz und um das Business-IT-Gap zu schließen, ist die Transformation von diversen WfM zu BPEL häufig Thema wissenschaftlicher Arbeiten.

Nach dieser Einführung in BPEL folgt nun die formale Definition von BCF. Definition 3 gibt die Bestandteile eines Prozesses in BCF an und definiert somit ein repräsentatives, formales Modell einer blockorientierten Beschreibungssprache, das die Beschreibung der Transformationsstrategien in Abschnitt 4 und 5 erleichtert.

Definition 3: Ein BPEL Control Flow (BCF) ist ein Tupel $BCF = \{Seq, Flow, Switch, While, Pick, Scope, Basic, Empty, Terminate, Link, de, jc, tc\}$. Die Mengen $Seq, Flow, Switch, While, Pick, Scope, Basic, Empty, Terminate, Link$ sind dabei paarweise disjunkt. $Str = Seq \cup Flow \cup Switch \cup While \cup Pick, \cup Scope$ vereinigt die strukturierten Aktivitäten, während die Basisaktivitäten unter $Bas = Basic \cup Empty \cup Terminate$ zusammengefasst werden. Die Vereinigung der strukturierten und der Basisaktivitäten ergeben alle Aktivitäten des BCF $Act = Str \cup Bas$. Wobei folgende Definitionen gelten:

- Seq, Flow, Switch, While, Pick, Scope, Empty, Terminate, Link ($Link \subseteq Act \times Act$) enthalten jeweils die dazugehörigen BPEL Elemente.
- Basic enthält alle BPEL Basisaktivitäten, ausgenommen Empty und Terminate.
- de: ist eine Dekompositionsrelation und bildet eine strukturierte Aktivität auf die Menge der darin geschachtelten Aktivitäten ab. $de: Str \rightarrow \mathbb{P}(Act)\backslash\emptyset$
- jc: stellt die Aktivierungsbedingung(join condition) der Aktivitäten dar. $jc: Act \rightarrow expr$
- tc: hier wird die Übergangsbedingung(transition condition) der Links definiert.
  $tc: Link \rightarrow expr$

Definition 4 Aktivierungsbedingung: Aktivierungsbedingungen von BPEL-Aktivitäten sind zusammengesetzte Ausdrücke und stellen eine konjunktive, disjunktive oder antivalente(xor) Verknüpfung der Übergangsbedingung der in die Aktivität eingehenden Links dar. Mit AND, OR oder XOR wird, in verkürzter Form, stellvertretend von folgenden Ausdrücken gesprochen:

$AND: jc(x) = tc(y_1, x) \wedge tc(y_2, x) \wedge \ldots \wedge tc(y_n, x)$,
$OR: jc(x) = tc(y_1, x) \vee tc(y_2, x) \vee \ldots \vee tc(y_n, x)$,
$XOR: jc(x) = tc(y_1, x) \oplus tc(y_2, x) \oplus \ldots \oplus tc(y_n, x)$,
wobei succ(x)={y, y, ... y} gilt.

Definition 5 Teilbaumfragment: Die Verschachtelung strukturierter Aktivitäten stellt ein baumartiges Konstrukt dar, auf der eine Teilbaumrelation definiert werden kann. Wenn $Strukt \subseteq Act \times Act$ ist, dann ist $(a1, a2) \in Strukt$ genau dann, wenn $a2 \in de(a1)$ gilt und $Strukt^*$ ist die transitive Hülle von $Strukt$ [2].

Nach der formalen Beschreibung der beiden Modellansätze werden verschiedene strukturelle Eigenschaften besprochen, die die Beschreibung der nachfolgenden Strategien erleichtert.

2.3. Strukturierte Prozessgrafen und andere Eigenschaften

Prozessgraf-, wie auch BCF-Modelle, werden als Eingaben für den Transformprozess betrachtet. Einige Strategien erfordern restriktivere Einschränkungen des Eingangsmodells, damit die Transformation fehlerfrei durchgeführt werden kann. Die im Zuge der hier vorgestellten Transformationsstrategien relevanten Modelleigenschaften werden in diesem Abschnitt erläutert.

Definition 6 zyklische Prozessgrafen: Ein Prozessgraf kann zyklische Kantenfolgen enthalten. D.h. wenn es eine oder mehrere Folgen gerichteter Kanten gibt, die von einem Knoten n zu n zurückführt, dann ist der PG zyklisch [2]. Formal gilt $\exists n \in F \cup C$: (n, n) ∈ A*. Andernfalls ist der Prozessgraf kreisfrei.

Definition 7 strukturierte BPEL Control Flows (BCFs): Ein BPEL Control Flow ist strukturiert, wenn es kein Link-Element innerhalb des beschriebenen Prozesses gibt [2]. Also, $Link = \emptyset$. Sonst gilt der BCF als unstrukturiert.

Definition 8 strukturierte Prozessgrafen (PG): Ein Prozessgraf ist strukturiert, wenn er mit Hilfe der nachfolgenden Reduktionsregeln zu einer einzigen Aktivität vereinfacht werden kann. Die folgenden Reduktionsregeln beschreiben eine Komponente des PG, und wie sie durch eine Funktion ersetzt werden [2]:

- Sequenz Reduktion: Aktivitäten $f_1, f_2, \ldots f_n \in F$, die sequenziell miteinander verbunden sind $(f_1, f_2), (f_2, f_3), \ldots, (f_{n-1}, f_n) \in A$, werden durch ein Funktion f_C ersetzt. $F \coloneqq (F \cup \{f_C\}) \backslash \{f_1, f_2, \ldots f_n\}$, $A \coloneqq (A \cup \{(x, f_C) | x \in pred(f_1)\} \cup \{(f_C, x) | x \in succ(f_n)\} \backslash$
  $(f_1, f_2), (f_2, f_3), \ldots, (f_{n-1}, f_n)$

- Konnektorpaar Reduktion: Ein Konnektorpaar setzt sich aus dem ersten Konnektor $c_1 \in C$, der den Kontrollfluss verzweigt $|pred(c_1)| = 1$, und dem zweiten Konnektor $c_2 \in C$, der den Kontrollfluss wieder vereinigt $|succ(c_2)| = 1$, zusammen. Die Menge der direkten Nachfolgeknoten des ersten Konnektors und die Menge der direkten Vorgängerknoten des zweiten Konnektors müssen ident sein $succ(c_1) = pred(c_2) \in F$, und beide Konnektoren müssen denselben Typ(AND, OR, XOR) haben $l(c_1) = l(c_2)$. Ist dies erfüllt, kann das Konnektorpaar und die darin eingeschlossenen Aktivitäten durch die Aktivität fc ersetzt werden. $F := (F \cup \{f_C\}) \backslash succ(c_1)$, $A := (A \cup \{(x, f_C) | x \in pred(c_1)\} \cup \{(f_C, x) | x \in succ(c_2)\}) \backslash (\{(x, c_1) | x \in pred(c_1)\} \cup \{(f_C, c_2) | x \in succ(c_2)\})$, $C := C \backslash \{c_1, c_2\}$

- Schleifen Reduktion: ein Schleife beginnt mit einem XOR-Konnektor $c_1 \in C$ (Einstiegspunkt in die Schleife), der den Kontrollfluss vereinigt $|succ(c_1)| = 1$, der über den vorwärtsgerichteten Zweig mit einem XOR-Konnektor$c_2 \in C$ verbunden ist. c_2 (Ausstiegspunkt der Schleife), der den Kontrollfluss splittet $|pred(c_1)| = 1$, ist über den rückführenden Zweig mit dem Einstiegspunkt verbunden. Je nachdem ob im vorwärtsgerichteten, im rückführenden, in beiden oder in keinem Zweig eine Aktivität vorhanden ist, ergeben sich vier Schleifenvarianten. Unabhängig von der Schleifenvariante kann der Ein- u. Ausstiegspunkt und die eingeschlossene(n) Aktivität(en) $succ(c1) \cap pred(c2), succ(c2) \cap pred(c1) \subseteq F$ durch eine Aktivität f_c ersetzt werden. $F := (F \cup \{f_C\}) \backslash ((succ(c1) \cap pred(c2)) \cup (succ(c2) \cap pred(c1)))$, $A := (A \cup \{(x, f_C) | x \in pred(c_1) \setminus succ(c_2)\} \cup \{(f_C, x) | x \in succ(c_2) \backslash pred(c1)\}) \setminus (\{(x, c_1) | x \in pred(c_1)\} \cup \{(c_1, x) | x \in succ(c_1)\} \cup \{(x, c_2) | x \in pred(c_2)\} \cup \{(c_2, x) | x \in succ(c_2)\})$, $C := C \backslash \{c_1, c_2\}$

- Startblock Reduktion: Ein Startblock besteht aus XOR-Join Konnektor $c \in C$, dessen Vorgängerknoten der Menge der Startzustände $S = pred(c)$ entsprechen. Ein solcher Block kann durch eine Aktivität fc ersetzt werden. $S := \emptyset$, $F := F \cup \{f_C\}$, $A := (A \cup \{(f_C, x) | x \in succ(c)\}) \backslash (\{(x, c) | x \in pred(c)\} \cup \{(c, x) | x \in succ(c)\})$, C:=C\{c}

- Endblock Reduktion: Ein Endblock besteht aus XOR-Split Konnektor $c \in C$, dessen Nachfolgeknoten der Menge der Endzustände $E = succ(c)$ entsprechen. Ein solcher Block kann durch eine Aktivität fc ersetzt werden. $E := \emptyset$, $F := F \cup \{f_C\}$, $A := (A \cup \{(x, f_c) | x \in pred(c)\}) \backslash (\{(x, c) | x \in pred(c)\} \cup \{(c, x) | x \in succ(c)\})$, C:=C\{c}

3. Herausforderungen bei der Transformation

Transformationen können problemlos durchgeführt werden, wenn alle Entitäten des Quellmodells in Konstrukte des Zielmodells umgewandelt werden können. Ist dies nicht der Fall, treten Diskrepanzen auf. Werden nun Workflows eines weniger restriktiven Metamodells in einen Workflow eines restriktiveren Metamodells übergeführt, kann es zu Informationsverlusten kommen. Abhängig vom zu übersetzenden Prozess, ist der resultierende Prozess möglicherweise nicht mehr semantisch äquivalent.

Grundsätzlich können drei Klassen von Diskrepanzen unterschieden werden, die bei der Transformation zwischen unterschiedlichen BPM-Sprachen auftreten können [14]:

(1) Diskrepanzen domänenspezifische Entitäten auszudrücken: Modelle sind meist vereinfachte Darstellungen der realen Welt, und besitzen unterschiedliche Möglichkeiten, Konstrukte der realen Welt und ihre Eigenschaften auszudrücken. Kommen Modelle aus unterschiedlichen Anwendungsbereichen und haben unterschiedliche Ausdruckskraft, geht bei der Transformation des einen Modells in das andere die Semantik verloren. Dies ist jedoch zu vermeiden bzw. zu minimieren.

(2) Diskrepanzen Kontrollflussmuster auszudrücken: Kontrollflussmuster beschreiben immer wiederkehrende Ablaufsituationen in Workflows. Nicht unterstützte Muster im Zielmodell können nicht transformiert werden und verhindern eine vollständige Modelltransformation. In [15] wurden 20 Kontrollflussmuster definiert, die als Vergleichssystem herangezogen werden können, um nicht unterstützte Muster zu identifizieren.
(3) Diskrepanzen in der Repräsentation des Kontrollflusses: Hier entsteht der Unterschied der Modelle dadurch, wie der Kontrollfluss ausgedrückt wird - also entweder graf- oder blockorientiert.

In dieser Arbeit wird angenommen, dass Diskrepanzen, domänenspezifische Entitäten auszudrücken, nicht auftreten. Betrachtet man graforientierte Prozessbeschreibungssprachen fällt auf, dass die Ausdruckkraft durch die Darstellung des Kontrollflusses durch Kanten und Knoten stärker ist. Dadurch sind Transformationen von block- zu graforientierten Modellen meist unproblematischer als in die entgegengesetzte Richtung. Dies manifestiert sich in den beiden folgenden Beobachtungen und variiert aber abhängig vom betrachteten WfM:

In graforientierten WfM können

- Schleifen mit mehreren Ein- und/oder Ausstiegspunkten konstruiert werden.
- Blöcke konstruiert werden, deren Split- und Join-Konnektor nicht vom gleichen Typ ist. Somit sind auch Kombinationen wie z.B. AND-Split/XOR-Join möglich.
-

Schleifen

Schleifen werden in blockorientierten WfM über strukturierte Aktivitäten dargestellt, die nur einen Ein- (g1) und einen Ausstiegspunkt (g2) definieren(Abb. 6). In BPEL z.B. wird dies über das While-Element ausgedrückt oder in Asbru über einen zyklischen Plan [16] [17].

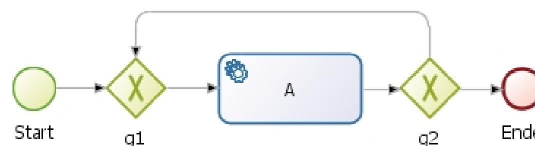

Abb. 6: strukturierte Schleife mit einem Ein- (g1) u. Ausstiegspunkt (g2)

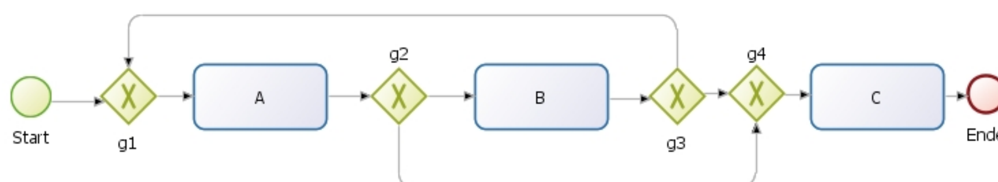

Abb. 7: unstrukturierte Schleife m. zwei Ausstiegspunkten (g2, g3) u. einem Einstiegspunkt (g1)

Abb. 7 zeigt eine Schleife mit einem Einstiegspunkt g1 und zwei Ausstiegspunkten g2 und g3. Dies kann weder mit einem While-Element in BPEL, noch mit einem zyklischen Plan in Asbru dargestellt werden.

Blöcke

Blöcke, wie z.B. And-Blöcke (Abb. 9) oder XOR-Blöcke, werden in blockorientierten WfM über strukturierte Aktivitäten repräsentiert. Das Verhalten am Synchronisationspunkt (Join-Konnektor) ist meist durch das jeweilige Konstrukt vorgegeben und lässt sich nicht beeinflussen. Blöcke mit unterschiedlichen Konnektortypen (Abb. 8) lassen sich deshalb, wie z.B. in BPEL, meist nicht umsetzen. In BPEL wird dafür das Flow- (And-Blöcke) bzw. Switch-Element (XOR-Blöcke) verwendet

[16]. Asbru hat dafür den parallelen Plan bzw. den If-Then-Else Plan [17]. strukturierter paralleler AND-Block. unstrukturierter Block mit And-Verzweigung (g1) u. XOR-Vereinigung (g2)

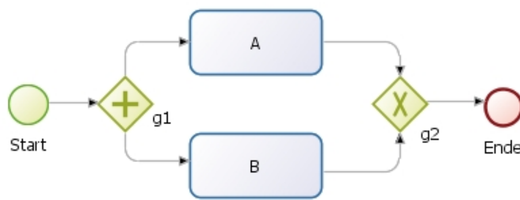

Abb. 8: unstrukturierter Block mit And-Verzweigung (g1) u. XOR-Vereinigung (g2)

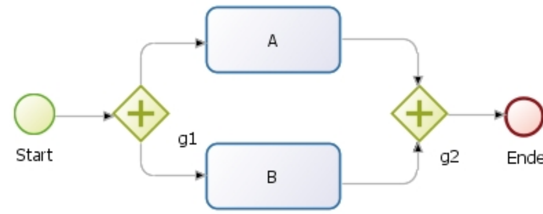

Abb. 9: strukturierter paralleler AND-Block

Wenn man von strukturierten Komponenten spricht, dann sind dies Teile des Prozesses im Quellmodell, die auf Konstrukte im Zielmodell abgebildet werden können. Ob eine Komponente strukturiert ist oder nicht, ist deshalb auch vom Quell- und Zielmodell abhängig und deshalb keine absolute Eigenschaft.

4. Transformationsstrategien von graf- zu blockorientierten Sprachen

Nachdem die Definitionen sowohl für die graf- als auch die blockorientierten Workflow-Modelle und deren strukturellen Eigenschaften vorhanden sind, können nun in diesem Abschnitt die Transformationsstrategien besprochen werden. Zuerst werden jene Strategien erläutert die einen graforientierten Workflow in einen blockorientierten umwandeln. Jene Umwandlungsstrategien, die blockorientierte Workflows in graforientierte transformieren, werden im nächsten Abschnitt 5 behandelt.
Um ein besseres Verständnis für die in folgenden Unterabschnitten angeführten Transformationen zu bekommen, werden noch weitere Konzepte erklärt.

Definition 9: Wird eine Funktion $f: A \rightarrow B$ auf eine Menge von Elementen angewandt, dann wird die Funktion auf alle Elemente einzeln angewandt und die daraus abgebildeten Elemente zu einer Ergebnismenge zusammengefasst. $f(X) = \cup_{x \in X} f(x), X \subseteq A$

Definition 10 Attributierter Prozessgraf: Ein attributierter Prozessgraf (APG) ist ein Tupel $APG = \{S, E, F, C, l, A, B\}$, wobei S, E, F, C, l wie in Definition 1 definiert sind [2].

- A ist die Menge an Kanten, zwischen den Knoten des APGs. $A = (S \cup F \cup C \cup B) \times (E \cup F \cup C \cup B)$
- B ist die Menge aller PG-Knoten, die ein Attribut mit der BCF-Übersetzung enthalten. Somit ist ein Element in B ein bereits übersetzter Knoten des Prozessgrafs.

Definition 11 APG Knoten Mapping: M stellt eine Mapping-Funktion dar, die einen Knoten des APGs in BCF-Aktivitäten übersetzt. $M: S \cup E \cup F \cup C \cup B \rightarrow Basic \cup Empty \cup Terminate \cup B$

$$M(x) = \begin{cases} Empty(x), if\ x \in C; \\ Basic(x), if\ x \in F \cup S \\ Terminate(x), if\ x \in E; \\ x, if x \in B \end{cases}$$

Die grundsätzliche Idee ist Startzustände und Aktivitäten auf Basic-Elemente, Konnektoren auf Empty-Elemente und Endzustände auf Terminate-Elemente abzubilden [2].

Die in den folgenden Unterabschnitten vorgestellten Übersetzungsstrategien nützen für die Modelltransformation unterschiedliche Mechanismen des Zielmodells. Als Zielmodell dient hier BPEL, da es einerseits repräsentativ für das blockorientierte Paradigma ist, andererseits sind

Transformationen von verschiedenen graforientierten Modellen zu BPEL ein wissenschaftlich ausführlich erörtertes Thema (z.B. [1], [12], [13], [16]).

In dieser Arbeit werden 5 Strategien diskutiert, manche davon kombinieren andere Strategien zu einer eigenständigen Strategie und manche bauen aufeinander auf und verfeinern den ursprünglichen Ansatz.
Folgende Ansätze werden erörtert.

(1) Element-Preservation: Hier werden Aktivitäten entsprechend einer Mapping-Funktion (Definition 11) in Basic-Aktivitäten umgewandelt. Der Kontrollfluss wird dann mit Hilfe von BPEL-Links definiert.
(2) Element-Minimization: verfeinert das Ausgangsmodell der Element-Preservation Strategie, indem die Empty-Elemente eliminiert werden, die die Konnektoren darstellen.
(3) Structure-Identifikation: Der zu übersetzende Eingangsgraf wird in Komponenten geteilt, die in strukturiert BPEL-Aktivitäten übersetzt werden können. Dies eignet sich jedoch nur für einen strukturierten Prozessgrafen, da unstrukturierte Komponenten nicht übersetzt werden können.
(4) Structure-Maximization: Strukturierte Komponenten des Eingangsgrafs werden mit der Structure-Identifikation Strategie transformiert, während unstrukturierte Komponenten mit Hilfe der Element-Preservation bzw. -Minimization Strategie übersetzt werden.
(5) Event-Condition-Action-Rules: Der Eingangsgraf wird mit der Structure-Identifikation Strategie so weit wie möglich transformiert. Aktivitäten innerhalb unstrukturierter Komponenten werden in BPEL-Event-Handler umgewandelt, die den entsprechenden Event der nachfolgenden Aktivität (Event-Handlers) aufrufen.

Die folgenden Unterabschnitte enthalten die detaillierten Beschreibungen der oben überblicksmäßig angeführten Strategien.

4.1. Element-Preservation Strategie

BPEL ist keine rein blockorientierte Beschreibungssprache, sondern besitzt auch grafbasierte Konzepte, nämlich Links. Mit Links können logische Abhängigkeiten modelliert werden, vergleichbar mit Kanten in Prozessgrafen. BPEL-Links können nur innerhalb eines BPEL Flow-Elements eingesetzt werden, und die damit definierten Kantenfolgen müssen kreisfrei sein. Eine weitere Einschränkungen ist, dass Links nie aus einer bzw. in eine Schleife führen dürfen [2].

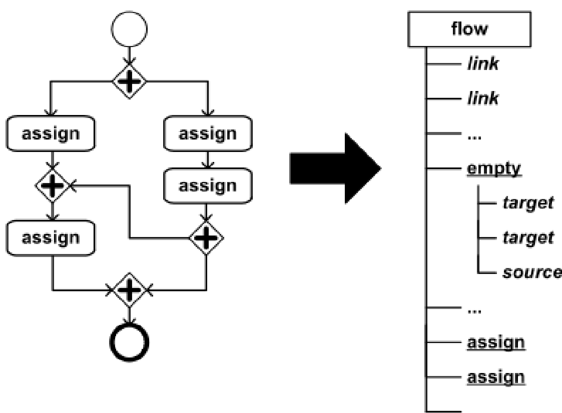

Abb. 10: Element-Preservation Strategie [2]

procedure: Element-Preservation(PG)

1: $Empty \leftarrow M(C)$
2: $Basic \leftarrow M(F \cup S)$
3: $Terminate \leftarrow M(E)$
4: $Flow \leftarrow flow$
5: $de(flow) \leftarrow Empty \cup Basic \cup Terminate$
6: $Link \leftarrow \emptyset$
7: **for all** $(x, y) \in A$ **do**
8: $\quad Link \leftarrow Link \cup (M(x), M(y))$
9: **end for**
10: $jc(x) = \begin{cases} AND, |\bullet M^{-1}(x)| > 1 \land l(M^{-1}(x)) = and; \\ XOR, |\bullet M^{-1}(x)| > 1 \land l(M^{-1}(x)) = xor; \\ OR, otherwise. \end{cases}$
11: $tc(x, y) = guard(M^{-1}(x), M^{-1}(y))$
12: **return**(BCF)

Algorithmus 1: Element-Preservation Pseudo-Code [2]

Der generelle Gedanke dieser Strategie ist, dass jeder Knoten des PG, entsprechend der Mapping-Funktion (Definition 11), in eine BPEL-Basisaktivität übersetzt wird, und die Kanten durch entsprechende Links dargestellt werden (siehe Abb. 10) [2].

Der entstandene BPEL-Prozess besitzt genau ein Flow-Element. In diesem Flow-Element sind die Links und die übersetzten Aktivitäten des PG enthalten. In den Aktivitäten sind source- bzw. target-Elemente definiert, die die Quell- bzw. Zielaktivität des Links darstellen.
Über alle target-Elemente einer Aktivität kann eine Aktivierungsbedingung angegeben werden. Und für jeden Link kann, über das source-Element, eine Übergangsbestimmung spezifiziert werden.

Algorithmus 1 zeigt, wie ein BCF-Prozess erstellt wird, der ein Flow-Element (Zeile 4) enthält, in dem alle übersetzten Elemente (Zeile 1-3), also Funktionen, Start-, Endzustände und Konnektoren, verschachtelt sind (Zeile 5).
Anschließend wird jede Kante in einen BCF-Link transformiert (Zeile 7-9).
Aktivitäten erhalten ihre Aktivierungsbedingung entsprechend ihrer Knoten im Prozessgraf[3] (Zeile 10), AND für AND-Konnektoren XOR für XOR-Konnektoren und OR für die restlichen Knoten.
Die Übergangsbestimmungen der Links werden von den korrespondierenden Kantenbedingen abgeleitet (Zeile 11).

Diese Strategie kann nur auf kreisfreie PGs angewendet werden, die keine zyklischen Pfade spezifizieren. Der Vorteil liegt im geringen Implementierungsaufwand, und dass für jeden Knoten im PG eine Aktivität im BCF gebildet wird. Da auch ein Empty-Element für jeden Konnektor im PG gebildet wird, enthält der BCF mehr Aktivitäten als eigentlich nötig wären, was ein klarer Nachteil dieser Strategie ist [2].

Beispiel 1:

Folgendes Beispiel soll die Element-Preservation Strategie veranschaulichen und ist an das Beispiel aus [1] angelehnt.

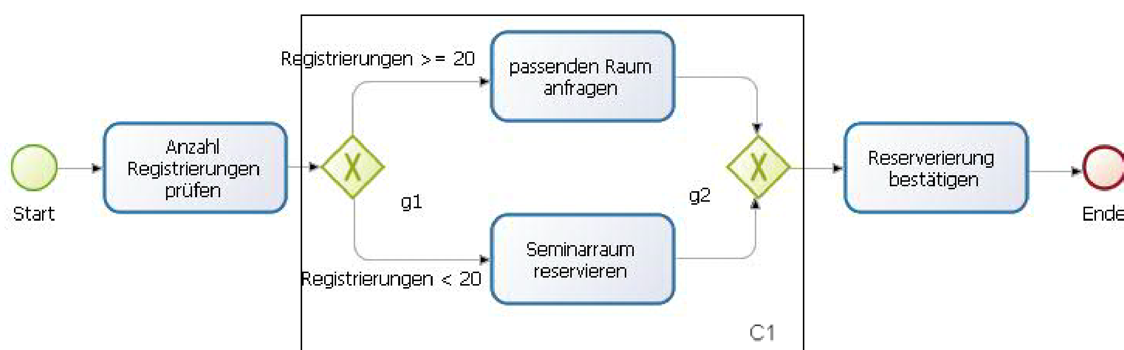

Abb. 11: Raumreservierungsprozess [1]

Abb. 11 zeigt einen Raumreservierungsprozess. Zu Beginn wird überprüft, wie viele Teilnehmer sich für den Kurs registriert haben. Abhängig von der Anzahl wird der institutseigene Seminarraum oder ein externer Vortragssaal für den Kurs verwendet. Ist die Anzahl kleiner als 20 Teilnehmer, wird der Seminarraum reserviert, sonst ein passender Hörsaal. Anschließend wird die Reservierung bestätigt.

Der folgende Codeabschnitt zeigt den in Abb. 11 gezeigten Prozess in schematischen BPEL. Man sieht wie mit source- und target-Elementen und den dazugehörigen join- und transitionConditions der Kontrollfluss nachgebildet wird.

[3] In den Algorithmen wird für die Vorgängermenge als $\bullet x$ und für Nachfolgermenge als $x\bullet$ dargestellt.

```
<process name="Raumreservierung">...
	<flow>
		<links>
			<link name="Anzahl2Split"/>
			<link name="Split2Seminar"/>
			<link name="Split2Raumanfrage"/>
			<link name="Seminar2join"/>
			<link name="Raumanfrage2join"/>
			<link name="join2Bestätigung"/></links>

		<invoke operation="Registrierungen_prüfen"….>
		<source linkName=" Anzahl2Split"/></invoke>

		<empty>
			<target linkName="Anzahl2Split"/>
			<source linkName="Split2Seminar">
				<transitionCondition>Teilnehmer < 20
				</transitionCondition></source>
			<source linkName="Split2Raumanfrage">
				<transitionCondition>Teilnehmer >= 20
				</transitionCondition></source>
		</empty>

		<invoke operation="Seminarraum_reservieren"….>
			<target linkName="Split2Seminar"/>
			<source linkName="Seminar2join"/></invoke>

		<invoke operation="Raum_anfragen"….>
			<target linkName="Split2Raumanfrage"/>
			<source linkName="Raumanfrage2join"/></invoke>

		<empty>
			<targets>
				<joinCondition>$Seminar2join xor $Raumanfrage2join
				</joinCondition>
				<target linkName="Seminar2join"/>
				<target linkName="Raumanfrage2join"/>
			</targets>
			<source linkName="join2Bestätigung"/></empty>

		<invoke operation="Reservierung_bestätigen"….>
			<target linkName="join2Bestätigung"/></invoke>

	</flow></process>
```

4.2.Element-Minimization Strategie

Diese Strategie setzt bei dem Nachteil an, dass in der Element-Preservation Strategie Konnektoren explizit in Empty-Elemente übersetzt werden und vereinfacht das Ergebnis daraus.
D.h. die Empty-Elemente des in der Element-Preservation Strategie entstandenen BCF werden eliminiert, indem die Links um die Empty-Elemente herumgeleitet werden und die Aktivierungsbedingung der nachfolgenden Aktivitäten angepasst werden (siehe Abb. 12). Danach werden die Empty-Elemente und die darin hinein- und hinausführenden Links entfernt [2].

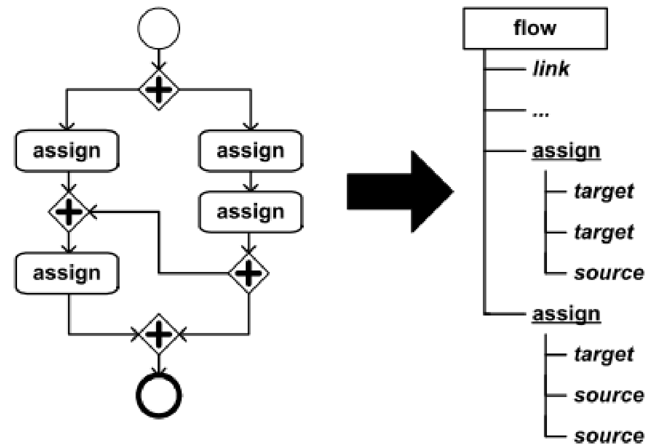

Abb. 12: Illustration d. Element-Minimization Strategie [2]

In Algorithmus 2 wird zuerst der PG mit der Element-Preservation Strategie übersetzt (Zeile 1). Im Anschluss wird über alle Empty-Elemente $x \in X$ iteriert, die keinen anderen Konnektor als direkten Vorgängerknoten im PG haben (Zeile 2).
Nun wird ein Link von jeder eingangsverlinkten Aktivität von x zu jeder ausgangsverlinkten Aktivität gebildet (Zeile3).
Die angepasste Aktivierungsbedingung $jc'(y)$ der ausgangsverlinkten Aktivität $y \in Y$ wird in der Form angepasst, dass die ursprüngliche Aktivierungsbedingung von y, $jc(y)$, mit der Aktivierungsbedingung des Empty-Elements x, $jc(x)$, konjunktiv verknüpft wird (Zeile 5). $jc'(y) = jc(y) \wedge jc(x)$. Im Anschluss werden alle Empty-Elemente x und die mit ihnen verbundenen Links entfernt (Zeile 7-8) [2]. $M^{-1}(Empty) \cap C = \emptyset$, $Links = (Links \cup \{(y1, y2) | y1 \in pred(x) \wedge y2 \in succ(x)\}) \setminus (\{(x, y) | y \in succ(x)\} \cup \{(y, x) | y \in pred(x)\})$

Der Vorteil der Element-Minimization Strategie ist, dass der generierte BCF nur noch Elemente enthält, die zum Ausdrücken des BPEL-Prozesses notwendig sind.
Der Nachteil liegt darin, dass die korrespondierenden Knoten in PG und BCF nicht mehr so intuitiv erkennbar sind, wie z.B. in der Element-Preservation Strategie, da nicht mehr alle ursprünglichen Knoten vorhanden sind [2].

procedure: Element-Minimization(PG)
1: $BCF \leftarrow$ **Element-Preservation**(PG)
2: **while** $\exists x \in Empty : M^{-1}(\bullet x) \cap C = \emptyset$ **do**
3: $\quad Link \leftarrow Link \cup \{(y_1, y_2) \mid y_1 \in \bullet x \wedge y_2 \in x\bullet\}$
4: $\quad$ **for all** $y \in x\bullet$ **do**
5: $\quad\quad jc \leftarrow \left(jc'(y') = \begin{cases} jc(y'), & y' \neq y; \\ jc(y') \wedge jc(x), & otherwise. \end{cases} \right)$
6: $\quad$ **end for**
7: $\quad Link \leftarrow Link \setminus (\{(x, y) \mid y \in x\bullet\} \cup \{(x, y) \mid y \in \bullet x\})$
8: $\quad Empty \leftarrow Empty \setminus \{x\}$
9: **end while**
10: **return**(BCF)

Algorithmus 2: Element-Minimization Pseudo-Code [2]

Beispiel 2:

Diese Strategie wird ebenfalls anhand des Raumreservierungsprozesses (Abb. 11) erläutert und vereinfacht den entstandenen BPEL-Prozess aus Beispiel 1. Dabei werden die Empty-Elemente eliminiert und die join- und transitionConditions in die Vorgänger- bzw. Nachfolgeaktivitäten entsprechend angepasst.

```
<process name="Raumreservierung">...
	<flow>
		<links> <link name="Anzahl2Seminar"/>
			<link name="Anzahl2Raumanfrage"/>
			<link name="Seminar2Bestätigung"/>
			<link name="Raumanfrage2Bestätigung"/></links>

		<invoke operation="Registrierungen_prüfen"....>
			<source linkName="Anzahl2Seminar">
				<transitionCondition>
					Teilnehmer <20
				</transitionCondition> </source>
			<source linkName="Anzahl2Raumanfrage">
				<transitionCondition>
					Teilnehmer>=20 </transitionCondition></source></invoke>

		<invoke operation="Seminarraum_reservieren"....>
			<target linkName="Anzahl2Seminar"/>
			<source linkName="Seminar2Bestätigung"/></invoke>

		<invoke operation="Raum_anfragen"....>
			<target linkName="Anzahl2Raumanfrage"/>
			<source linkName="Raumanfrage2Bestätigung"/></invoke>

		<invoke operation="Reservierung_bestätigen"....>
			<targets>
				<joinCondition>$Seminar2Bestätigung xor
					$Raumanfrage2Bestätigung
				</joinCondition>
				<target linkName="Seminar2Bestätigung"/>
				<target linkName="Raumanfrage2Bestätigung"/>
			</targets></invoke>
	</flow></process>
```

4.3. Structure-Identification Strategie

Bei der Structure-Identification Strategie werden strukturierte Teile des Prozessgrafen identifiziert, um sie durch die Reduktionsregeln (Definition 8) zu vereinfachen. Nur werden die strukturierten Komponenten nicht durch eine Funktion, sondern durch einen Knoten eines attributierten Prozessgrafen ersetzt, dessen Attribut die BCF-Übersetzung der Komponente enthält [2]. Handelt es sich um einen strukturierten PG, kann mit dieser Strategie eine vollständige Reduktion erfolgen. Somit wird der PG in einen APG umgewandelt, dessen Serialisierung der Attribute den BCF-Prozess enthält (siehe Abb. 13). Zusammengefasst bedeutet das, dass der PG ausschließlich mit strukturierten BCF-Aktivitäten übersetzt wird.

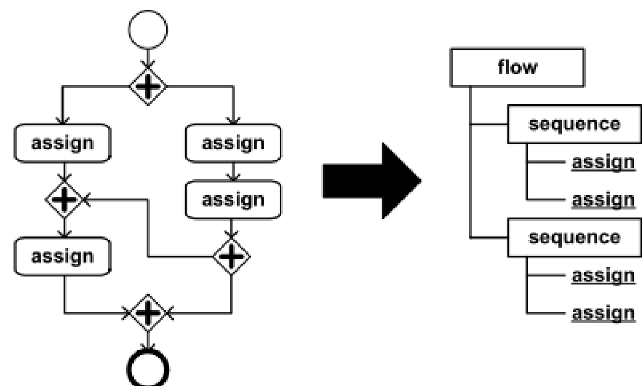

Abb. 13: Illustration d. Structure-Identification Strategie [2]

Daraus kann gleich gefolgert werden, dass diese Strategie nur für strukturierte Prozessgrafen sinnvoll ist.

Definition 12 Übersetzungsregeln: Entsprechend der angewandten Reduktionsregeln werden folgende strukturierte BCF-Aktivitäten zur Übersetzung verwendet:

- Sequenz: Sequenzen eines PG wird mit einem Sequence-Element dargestellt, dessen Subaktivität den Knoten in der Reihenfolge des PG entsprechen.

- AND-Block: Ein AND-Block wird in ein Flow-Element umgewandelt, wobei jeder Zweig einer Subaktivität des Flow-Elements entspricht.

- OR-Block: Dafür wird ein Empty-Element in ein Flow-Element verschachtelt, und jeder Zweig des OR-Blocks wird als Subaktivität des Flow-Elements dargestellt [2]. Zusätzlich werden Links, ausgehend vom Empty-Element zu den Subaktivitäten, erstellt, deren Übergangsbedingungen mit den Kantenbedingungen des OR-Splits übereinstimmen [2].

- XOR-Block: Ein XOR-Block wird in ein Switch-Element umgewandelt, wobei jeder Zweig einer Subaktivität des Switch-Elements entspricht.

- Schleifen: Die Übersetzung wird, entsprechend der Schleifenvariante, mit While-, Sequenc-, oder Empty-Elementen umgesetzt. Genauere Informtionen siehe [2].

- Start-Block: Der Start-Block wird mit einem Pick-Element umgesetzt, das für jeden Zweig ein Empty-Element als Subaktivität besitzt.

- End-Block: Wird entsprechend dem Eingangsmodell mit einem AND-, OR, oder XOR-Block, gefolgt von einem Terminate-Element für jeden Zweig, umgesetzt.

Bei der Übersetzung geht man wie Algorithmus 3 vor:
Zuerst wird ein attributierter Prozessgraf entsprechend dem Eingangsgrafen erstellt $APG = \{S, E, F, C, l, A, \emptyset\}$ (Zeile 1). Danach werden die Reduktionsregeln solange angewandt, bis der Graf auf einen Knoten reduziert ist (Zeile 2).
Dabei wird bei jeder Iteration, nach einer passenden Komponente APG' gesucht (Zeile 3), die durch Anwendung einer Reduktionregel (Definition 8) ersetzt werden kann. APG' wird mit Hilfe der Regeln in Definition 12 übersetzt (Zeile 4), mit ihrer Übersetzung ersetzt und somit der APG reduziert (Zeile 5).

procedure: Structure-Identification(PG)
1: $APG \leftarrow (S, E, F, C, l, A, \emptyset)$
2: **while** $|F \cup C \cup B| > 1$ **do**
3: $APG' \leftarrow match(APG)$ {Using rules in Definition 6}
4: $b \leftarrow translate(APG')$ {Using the described translations above}
5: Reduce APG substituting APG' with b {Using rules in Definition 6}
6: **end while**
7: **return**(BCF)

Algorithmus 3: Structure-Identification Pseude-Code [2]

Der Vorteil liegt darin, dass der entstandene Prozess nur aus strukturierten Aktivitäten besteht und dadurch lesbarer ist. Als Nachteil kann man anführen, dass die Zuordnung zu den ursprünglichen Elementen des PG nicht mehr so offensichtlich erkennbar ist [2].

Beispiel 3
Es wird der Prozess aus Abb. 11 zur Veranschaulichung der Strategie herangezogen. Da es sich um einen strukturierten Prozess handelt, kann er mit dieser Strategie auch vollständig übersetzt werden.

In Schritt 1 wird der XOR-Block, Komponente C1, mit der entsprechenden Übersetzungsregel transformiert und danach reduziert. Hier der dazugehörige schematische BPEL-Code:

```
<switch name="C1">
        <case conditon="registrierung >= 20">
                <invoke operation="Raum_anfragen".../></case>
        <case conditon="registrierung < 20">
                <invoke operation="Seminarraum_reservieren".../></case>
</switch>
```

Im Schritt 2 wird der aus Schritt 1 reduzierte Prozess übersetzt und transformiert. Die einzig anwendbare Regel ist die für die Sequenz. Hier der dazugehörige schematische BPEL-Code:

```
<sequence>
        <invoke operation="Registrierungen_prüfen".../>
        <switch name="C1">....</switch>
        <invoke operation="Reservierung_bestätigen".../>
</sequence>
```

Da der reduzierte Prozess jetzt nur mehr aus einer Restaktivität besteht, ist der Transformationsprozess beendet.

4.4. Structure-Maximization Strategie

Die Idee dieser Übersetzung ist es, die Reduktionsregeln aus Structure-Identification Strategie so oft wie möglich anzuwenden. Erst wenn dies nicht mehr möglich ist, da unstrukturierte Teilgrafen nicht reduziert werden können, werden diese mit der Element-Preservation bzw. Element-Minimization Strategie weiter übersetzt (siehe Abb. 14) [2].

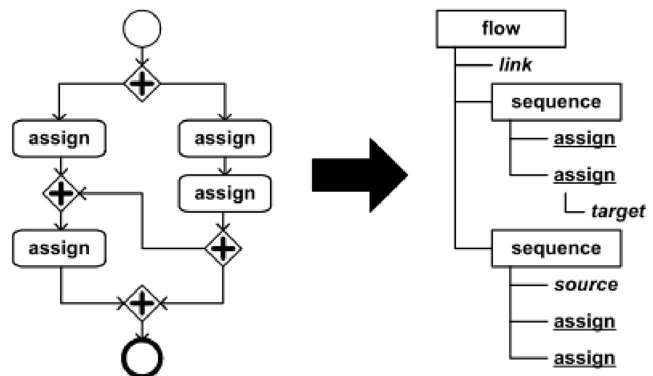

Abb. 14: Illustration d. Structure-Maximization Strategie [2]

Der Vorteil dieser Strategie ist, dass sie auch auf unstrukturierte Prozessgrafen, abgesehen von unstrukturierten Schleifen, angewandt werden kann. Da diese Strategie eigentlich zwei o.a. Strategien zur Umsetzung verwendet, ist nachteilig der erhöhte Implementierungsaufwand anzuführen [2].

4.5. Event-Condition-Action-Rules Strategie

Hier ist die Idee ähnlich wie bei der vorangegangenen Strategie. D.h. es wird versucht, möglichst viele Teile des Eingangsgrafen mit Hilfe des Structure-Indentification Verfahrens zu übersetzen. Die unstrukturierten Teile des PG, die nicht mit Definition 8 reduziert werden können,

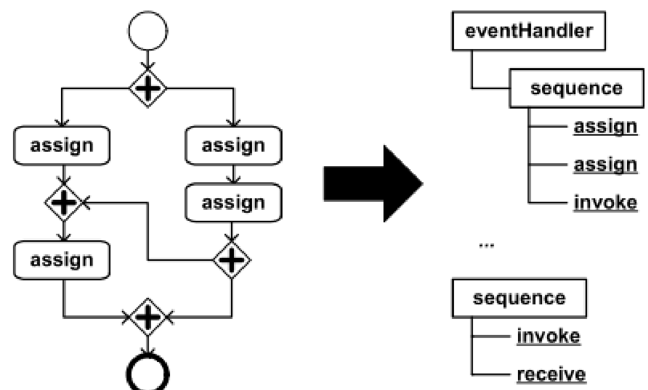

Abb. 15: Illustration d. Event-Condition-Action-Rules Strategie [2]

werden jedoch mit Event-Condition-Action (ECA) Regeln im BCF-Prozess abgebildet (siehe Abb. 15). In BPEL können ECA-Regeln mit Event-Handler implementiert werden [2].
Ein Event-Handler ist ein Konstrukt, das einen Event-Typ und eine Startbedingung mit einer Aktivität assoziiert. Tritt ein Event des assoziierten Typs auf und die Startbedingung des Event-Handlers ist erfüllt, wird die spezifizierte Aktivität ausgeführt. Durch Aufrufen des entsprechenden Event-Typs innerhalb des Prozesses können Event-Handler gestartet werden, und somit können auch unstrukturierte Teilgrafen realisiert werden [2].

Formal muss die Definition des BCF um folgende Elemente erweitert werden [2]:

- $Handler$ stellt die Menge der Event-Handler im Prozess dar und $handler(prc, Act)$ assoziiert die Menge der Startbedingungen mit einer Aktivität.
- $invoke(s)$ ist jene Aktivität, die einen Event-Typ s aufruft und so einen Event-Handler startet. $invoke(s) \in Invoke$
- $receive(s)$ ist jene Aktivität, die auf den Aufruf eines Event-Typs wartet und, bei dessen Empfang, die nachfolgenden Aktivitäten ausführt.

Zuerst wird der Prozessgraf mittels Structure-Identification so weit wie möglich reduziert. Danach wird jene Komponente, die beim Starten des Prozess als erstes ausgeführt wird herangezogen und folgende Sequenz gebildet (siehe Algorithmus 4 Zeile 1-5):

$$receive(startProcess);$$
$$M(getFirstComponent(PG));$$
$$invoke(firstCompleted)$$

Danach wird für jede weitere Funktion(Aktivität) $f \in F$ ein Event-Handler erzeugt (Zeile 7-9).

$$handler(getPreconditionSet(f), M(f); invoke(< i^4 > Completed\ )$$

Nachdem die erste Basisaktivität beendet wurde, ruft die erste Sequenz jene Event-Handler auf, die nach der ersten Aktivität ausgeführt werden und diese die darauf folgenden.

Der Vorteil dieser Strategie ist, dass es hier keine Einschränkungen gibt, und jeder beliebige PG transformiert werden kann.
Ein Nachteil ist hier, dass der erzeugte BPEL-Prozess komplizierter aufgebaut und schwerer zu lesen ist [2].

procedure: Event-Condition-Action-Rules(PG)
1: $Receive \leftarrow receive(processStarted)$
2: $Basic \leftarrow M(\mathbf{getFirstFunction}(PG))$
3: $Invoke \leftarrow invoke(firstCompleted)$
4: $Sequence \leftarrow sequence$
5: $de(sequence) \leftarrow Receive \cup Basic \cup Invoke$
6: $Handler \leftarrow \emptyset$
7: **for all** $f \in F \setminus \{\mathbf{getFirstFunction}(PG)\}$ **do**
8: $\quad prc \leftarrow \mathbf{getPreconditionSet}(f)$
9: $\quad Handler \leftarrow Handler \cup handler(prc, M(f))$
10: **end for**

Algorithmus 4: ECA-Rule Pseudo-Code [2]

Beispiel 4:
Es wird wieder unser laufendes Beispiel aus Abb. 11 übersetzt.
Als ersten Schritt werden die PreconditionSets aller Aktivitäten gebildet:

Aktivität	Precondition Set
Anzahl Registrierungen prüfen (a_1)	$startProcess$
Passenden Raum anfragen (a_2)	$Completion(a_1) \wedge Registrierungen \geq 20$
Seminarraum reservieren (a_3)	$Completion(a_1) \wedge Registrierungen < 20$
Reservierung bestätigen (a_4)	$Completion(a_2), \quad Completion(a_3)$

[4] i steht hier für die Nummer der jeweilige Komponente

Im zweiten Schritt können aus den Preconditon Sets Event-Condition-Action(ECA) Regeln abgeleitet werden.Für unser Beispiel ergeben sich folgende ECA-Regeln (Notation: Event[Condition]{Action}):

startProzess[]{do „Anzahl Registrierungen prüfen“; invoke Completion(a_1)}
Completion(a_1)[Regist.>= 20]{do „passenden Raum anfragen“; invoke Completion(a_2)}
Completion(a_1)[Regist.<20]{do „Seminarraum reservieren“; invoke Completion(a_3)}
Completion(a_2)[]{do „Reservierung bestätigen“; invoke Completion(a_4)}
Completion(a_3)[]{do „Reservierung bestätigen“; invoke Completion(a_4)}

Im dritten Schritt werden die ECA-Regeln in BPEL-EventHandler übersetzt. Für unser Beispiel wäre der folgende BPEL-Prozess eine mögliche Übersetzung:

```
<process name="Raumreservierung">...
    <eventHandlers>
        <onEvent Completion(a1)>
            <switch name="g1">
                <case conditon="registrierung >= 20">
                    <invoke switch_a1_c1)/></case>
                <case conditon="registrierung < 20">
                    <invoke switch_a1_c2)/></case>
            </switch></onEvent>
        <onEvent switch_a1_c1>
            <sequence>
                <invoke operation="Raum_anfragen".../>
                <invoke Completion(a2)/>
            </sequence></onEvent>
        <onEvent switch_a1_c2>
            <sequence>
                <invoke operation="Seminarraum_reservieren".../>
                <invoke Completion(a3)/>
            </sequence></onEvent>
        <onEvent Completion(a2)>
            <sequence>
                <invoke Completion(a3).../>
            </sequence></onEvent>
        <onEvent Completion(a3)>
            <sequence>
                <invoke operation="Reservierung_bestätigen".../>
            </sequence></onEvent>
    </eventHandlers>

    <sequence>
        <invoke operation="Registrierungen_prüfen".../>
        <invoke Completion(a1)/></sequence>

</process>
```

4.6.Zusammenfassung der Strategien

Die im Anschluss angeführte Tabelle 1 fasst die Vor- und Nachteile der Transformationsstrategien von graf- zu blockorientierten Prozessmodellen zusammen.

Die Element-Preservation Strategie kann nur auf kreisfreie Eingangsmodelle angewandt werden, und der Ergebnisprozess enthält mehr Elemente als zur Definition des Kontrollflusses benötigt würden. Die Zuordnung der Elemente zum Ursprungselement im Eingangsmodell ist jedoch intuitiv. Sie kann eingesetzt werden, wenn die Lesbarkeit des Ergebnisprozess wichtig ist.

Bei der Element-Minimization Strategie werden weniger Elemente im Ergebnisprozess erzeugt. Dies erhöht die Performance des BPEL-Prozesses, verringert jedoch die Lesbarkeit.

Die Structure-Indentification Variante erzeugt einen strukturierten und daher gut lesbaren Prozess. Sie kann jedoch nur auf strukturierte Eingangsmodelle angewandt werden, und die Zuordnung der Elemente zu den Ursprungselementen im Eingangsprozess ist nicht mehr intuitiv.

Die Structure-Maximization Strategie kann, ausgenommen unstrukturierte Schleifen, beliebige Eingangsgrafen transformieren, jedoch auf Kosten eines erhöhten Aufwands durch Implementieren zweier Strategien.

Die universellste Strategie ist die Event-Condition-Action-Rules. Sie kann auf beliebige Eingangsgrafen angewendet werden, der transformierte Prozess ist jedoch nur schwer lesbar.

Strategie	Vorteile	Nachteile
Element-Preservation	Ursprungselemente gut zuordenbar	Nur kreisfreie Eingangsgrafen übersetzbar; beinhaltet mehr Elemente als nötig
Element-Minimization	Überflüssige Elemente minimiert	Ursprungselemente schlechter zuordenbar
Structure-Identification	Gut lesbar, da nur strukturierte Elemente genützt werden	Nur strukturierte Eingangs-grafen übersetzbar ; Ursprungs-elemente schwer zuordenbar
Structure-Maximization	Eingangsgraf uneingeschränkt übersetzbar (ausgenommen unstrukturierte Schleifen)	Zwei Strategien müssen implementiert werden
Event-Condition-Action-Rules	Eingangsgraf uneingeschränkt übersetzbar	Ergebnisprozess schwer lesbar

Tabelle 1: Zusammenfassung der Vor- u. Nachteile der graf-zu-blockorientierten Transformationsstrategien

5. Transformationsstrategien von block- zu graforientierten Sprachen

Nachdem Transformationsstrategien im vorangegangenen Abschnitt besprochen wurden, die graforientierte in blockorientierte Workflows umwandeln, wird in diesem Abschnitt die entgegengesetzte Richtung besprochen. Somit dient hier ein blockorientierter Workflow als Eingangsmodell für den Übersetzungsprozess, der als Resultat einen graforientierten Workflow zurückgibt.
Bevor die Strategien besprochen werden, hier noch eine für das Verständnis notwendige Definition:

Definition 13 Mapping-Funktion M: M ist eine Funktion, die einer BPEL Basisaktivität $b \in Basic$ eine Funktion $f \in F$ zuordnet [2]. Dadurch wird die Basisaktivität b aus dem BCF (Eingangsmodell) in die Funktion f eines PG (Ausgangsmodell) übersetzt.

In den folgenden Abschnitten werden drei Transformationsstrategien vorgestellt:

(1) Flattening: Hier wird der BCF-Workflow in einen einzigen flachen Prozess, ohne Subprozesse, übersetzt.
(2) Hierachy-Preservation: Bei dieser Variante wird jede strukturierte Aktivität in einem Subprozess dargestellt, um so die hierarchische Struktur eines blockorientierten Prozesses beizubehalten. Dies funktioniert jedoch nur bei strukturierten BCF-Prozessen (siehe Definiton 7)
(3) Hierachy-Maximization: Dies ist eine Kombination aus den beiden o.a. Strategien. Es werden die strukturierten Teile des TBCF-Workflow mit Hierarchy-Preservation transformiert. Erst wenn dies nicht mehr möglich ist, wird mit der Flattening Strategie weiter übersetzt.

Die folgenden Unterabschnitte beschreiben detailliert die gerade überblicksmäßig angeführten Übersetzungsstrategien.

5.1. Flattening Strategie

Die grundsätzliche Idee dieser Strategie ist es, jede strukturierte BCF Aktivität in einen passenden Teilgrafen abzubilden. Der gesamte BCF-Workflow wird dadurch in einen einzigen Prozessgrafen (siehe Abb. 16) ohne Hierarchien, also flach, übersetzt [2].

Der Vorteil dieser Strategie liegt darin, dass all BCF-Workflows übersetzt werden und es keine Einschränkungen bzgl. des Eingangsmodells gibt. Als Nachteil kann gesehen werden, dass die durch das blockorientierte Paradigma vorgegebene Struktur im übersetzten PG verloren geht [2].

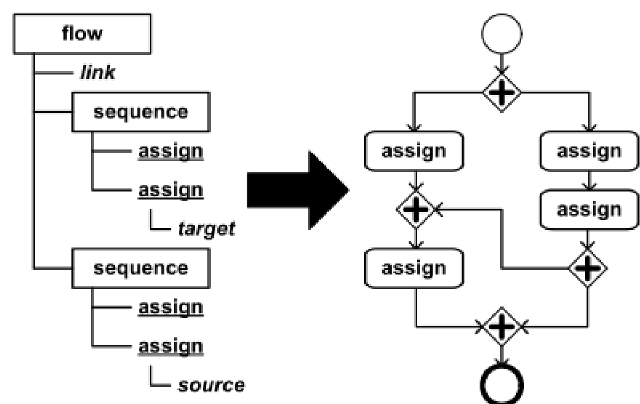

Abb. 16: Illustration d. Flattening Strategie [2]

procedure: Flattening(BCF)
1: $Struct \leftarrow Seq \cup Flow \cup Switch \cup While \cup Pick \cup Scope$
2: $S \leftarrow \{s\}$; $E \leftarrow \{e\}$; $F \leftarrow \emptyset$; $C \leftarrow \emptyset$; $A \leftarrow \emptyset$
3: $root \leftarrow a$, where $a \in Struct \wedge \nexists s \in Struct : de(s) = a$
4: **BCFtransform**$(root, s, e, PG)$
5: **for all** $(l_1, l_2) \in Link$ **do**
6: $\quad A \leftarrow A \cup \{(c_1, c_2)\}$
7: $\quad guard(c_1, c_2) = tc(l_1, l_2)$
8: **end for**
9: **return** PG

Algorithmus 5: Flattening Pseudo Code [2]

Algorithmus 5 transformiert einen BCF-Workflow und gibt einen flachen Prozessgraf zurück. Dabei wird ein leerer Prozessgraf initialisiert (Zeile 2), mit einem Startknoten s und einem Endknoten e, und die Wurzel des BCF ermittelt (Zeile 3). Danach wird die Methode BCFtransform (Algorithmus 6) aufgerufen, und die vorhin genannten Elemente als Parameter übergeben (Zeile 4). Diese

traversiert von oben nach unten durch die Hierarchie des BCF-Workflows, um den PG rekursiv zu vervollständigen. Anschließend werden Links in Kanten abgebildet, und dem resultierten Prozessgraf, mit den dazugehörigen Kantenbedingungen, hinzugefügt (Zeile 5-8) [2].

```
procedure: BCFtransform(activity, pred, succ, PG)
 1: if ∃(l_1, activity) ∈ Links then
 2:    C ← C ∪ {c_1}; l(c_1) = jc(activity)
 3:    A ← A ∪ {(pred, c_1)}; pred ← c_1
 4: end if
 5: if ∃(activity, l_2) ∈ Links then
 6:    C ← C ∪ {c_2}, l(c_2) = OR
 7:    A ← A ∪ {(c_2, succ)}; succ ← c_2
 8: end if
 9: if activity ∈ Seq then
10:    PG ← transSeq(activity, pred, succ, PG)
11: else if activity ∈ Flow then
12:    PG ← transBlock(activity, pred, succ, AND, PG)
13: else if activity ∈ Switch then
14:    PG ← transBlock(activity, pred, succ, XOR, PG)
15: else if activity ∈ While then
16:    PG ← transWhile(activity, pred, succ, PG)
17: else if activity ∈ Pick then
18:    PG ← transPick(activity, pred, succ, PG)
19: else if activity ∈ Scope then
20:    PG ← BCFtransform(de(activity), pred, succ)
21: else if activity ∈ Basic then
22:    F ← F ∪ {M(activity)}
23:    A ← A ∪ {(pred, activity), (activity, succ)}
24: else if activity ∈ Empty then
25:    A ← A ∪ {(pred, succ)}
26: else if activity ∈ Terminate then
27:    E ← E ∪ {e}
28:    A ← A ∪ {(pred, e)}
29: end if
30: return PG
```

Algorithmus 6: BCFtransform Pseudo Code [2]

BCFtransform hat eine BCF Aktivität, deren Vorgänger- und Nachfolgeraktivität und den, bisher nur teilweise übersetzten, Ergebnisgrafen als Parameter. Für die BCF-Wurzel erfolgt der Aufruf mit s, e und dem in Algorithmus 1, Zeile 2 initialisierten Prozessgrafen.
Algorithmus 2 prüft zu Beginn, ob die gegenwärtige Aktivität Ursprung oder Ziel eines BCF-Links ist und fügt, gegebenenfalls, einen entsprechenden Konnektor vor oder nach der Aktivität ein (Zeile 1-8).

Basisaktivitäten werden mit der Mapping-Funktion M übersetzt (Zeile 21-23), Empty Aktivitäten werden durch Kanten, zwischen Vorgänger und Nachfolgerknoten, dargestellt (Zeile 24-25) und Terminate-Aktivitäten werden zu zusätzlichen Endknoten (Zeile 26-28). Für Scop-Elemente wird BCFtransform mit der darin verschachtelten Aktivität aufgerufen (Zeile 20).

Die restlichen strukturierten BCF-Aktivitäten werden in eigenen Prozeduren (transSeq, transBlock, etc.) übersetzt [2]:

- transSeq verbindet alle Subaktivitäten mit Kanten in der entsprechenden Reihenfolge und ruft für jede Subaktivität BCFtransform rekursiv auf.
- transBlock umschließt die Subaktivitäten mit einem Split-Join-Konnektorpaar, deren Konnektor-Typ vom vierten Eingangsparameter abhängt.
- transWhile erstellt eine Schleife mit einem XOR-Join, XOR-Split Konnektor.
- transPick generiert für jede Subaktivität einen Start-Event[5] und ersetzt damit den ursprünglichen Start-Event des PG.

5.2. Hierarchy-Preservation Strategie

Viele graforientierte Workflow-Modelle haben die Möglichkeit ihren Prozess in Subprozesse zu teilen und in einem Prozess auf einen Subprozess zu verweisen. BPMN kann Subprozesse als Aktivität darstellen, die speziell als Teilprozess gekennzeichnet ist [3]. EPC hat dafür hierarchische Funktionen und in YAWL kann der Workflow in Subprozesse geteilt werden [2]. Um dies formal darzustellen, wird das Konzept des Prozessgraf Schemas definiert.

[5] Unter der Voraussetzung, das Pick-Elemente nur zum Modellieren alternativer Start-Events genutzt werden.

Definition 14 Prozessgraf Schema (PGS): ist ein Tupel PGS={PG, s}, wobei PG die Menge an Prozessgrafen darstellt, und s die Subprozessrelation darstellt [2]. $s: F \rightarrow \{\emptyset, pg\}\ mit\ pg \in PG$
s verweist von einer Funktion f auf einen Subprozess pg oder auf die leere Menge, wenn für f kein Subprozess definiert wurde [2].

Der Gedanke dieser Strategie besteht darin, alle strukturierten Aktivitäten des BCF Eingangsmodells in Subprozesse des Prozessgrafen zu transformieren (siehe Abb. 17). Die Struktur des BCF-Modells bleibt dadurch erhalten. Die Strategie ist jedoch dadurch limitiert, dass Links über die Grenzen einer strukturierten Aktivität hinaus nicht dargestellt werden können, der BCF also strukturiert sein muss [2].
Für die Transformation können die Algorithmen 1 und 2 verwendet werden, die Transformationsprozeduren für die strukturierten Aktivitäten müssten jedoch angepasst werden, da hier jeweils ein eigener Subprozess erstellt werden muss.

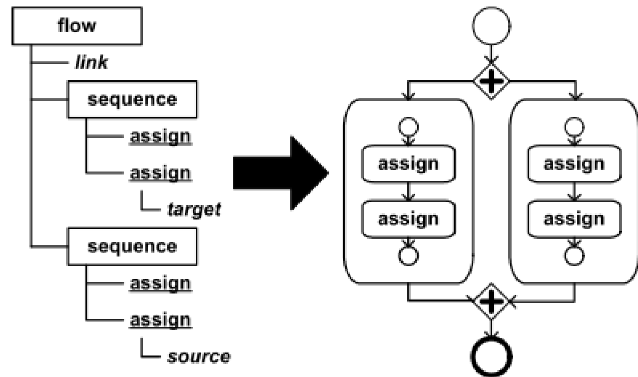

Abb. 17: Illustration d. Hierarchy-Preservation Strategie [2]

Der Vorteil dieser Strategie ist, dass die Struktur der strukturierten Aktivitäten erhalten bleibt. Dem gegenüber steht der Nachteil, dass durch die Hierarchie der Subprozesse navigiert werden muss, um den gesamten Prozess zu erkennen [2].

5.3. Hierarchy-Maximization Strategie

Die Hierarchy-Maximization Strategie versucht die Einschränkung auf strukturierte BCF-Workflows zu umgehen, indem versucht wird, so viel blockorientierte Struktur wie möglich zu erhalten. D.h. dort wo keine BCF-Links über strukturierte Aktivitäten hinausgehen, werden diese Subprozesse transformiert. Wo dies nicht möglich ist, werden flache Prozessgrafen, wie in der Flattening Strategie, konstruiert [2].

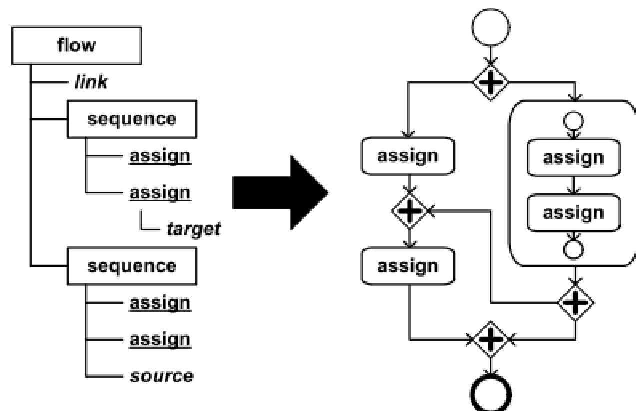

Abb. 18: Illustration d. Hierarchy-Maximization Strategie [2]

Der Vorteil dieser Strategie ist, dass möglichst viel Struktur erhalten bleibt und es doch keine Einschränkung auf das Eingangsmodell gibt. Es müssen jedoch beide Strategien implementiert werden [2].

5.4. Zusammenfassung der Strategien

Tabelle 2 fasst die Vor- und Nachteile der Transformationsvarianten von block- zu graforientierten Prozessmodellen zusammen.

Die Flattening Strategie kann beliebige BCF-Prozesse übersetzen, dabei geht jedoch die hierarchische Struktur von blockorientierten Prozessen verloren. Sie kann eingesetzt werden, um z.B. mit Geschäftsanalysten zu kommunizieren, da diese meist an graforientierte Modelle gewöhnt sind.

Die Hierarchy-Preservation Strategie kann nur auf strukturiert BCF-Modelle angewandt werden. Es bleibt jedoch die hierarchische Struktur erhalten. Sie kann angewandt werden, wenn graforientierte Repräsentationen von BPEL-Prozessen, z.B. nach Änderungen, wieder nach BPEL zurücktransformiert werden.
Die Hierarchy-Maximization Strategie vereinigt die Vorteile der Flattening und Hierarchy-Preservation Strategie, ohne deren Nachteile zu besitzen. Sie kann eingesetzt werden, wenn eine grafische Repräsentation eines BCF-Prozesses erforderlich ist.

Strategie	Vorteile	Nachteile
Flattening	BCF-Prozesse uneingeschränkt übersetzbar	verliert hierarchische Struktur
Hierarchy-Preservation	Hierachische Struktur bleibt erhalten	nur Strukturierte BCF-Prozesse übersetzbar; für Gesamtübersicht Navigation durch Subprozesse erforderlich
Hierarchy-Maximization	BCF-Prozesse uneingeschränkt übersetzbar; Hierachische Struktur bleibt erhalten	Zwei Strategien müssen implementiert werden

Tabelle 2: Zusammenfassung der Vor- u. Nachteile der block-zu-graforientierten Transfromationsstrategien

6. Weitere Transformationsansätze

In diesem Abschnitt soll ein Überblick über verschiedene Ansätze gegeben werden, Modelltransformationen durchzuführen.
Geschäftsmodelle sind inzwischen integraler Bestandteil des Arbeitsalltags, und Unternehmen besitzen oft eine Vielzahl von modellierten Prozessen. Diese Modelle werden mit Geschäftspartnern ausgetauscht oder müssen durch Firmenfusionen in bestehende Systeme integriert werden. Durch diese und ähnliche Szenarien steigt der Bedarf an, Modelle zu transformieren [18].

Modelltransformation ist eigentlich ein Sammelbegriff, unter dem man ganz allgemein Transformationen von Modell-zu-Modell, Modell-zu-Text oder Text-zu-Modell verstehen kann. In dieser Arbeit ist immer die Transformation von Modell-zu-Modell gemeint, wenn von Modelltransformation gesprochen wird. Wird ein Modell in ein Modell derselben Abstraktionsstufe übersetzt, spricht man von einer horizontalen Transformation. Hat das Zielmodell eine andere Abstraktionsstufe, spricht man von einer vertikalen Transformation [19].

Bei Modell-zu-Modell Transformationen wird das Quellmodell der Transformation, welches konform zum Quellmetamodell modelliert wurde, durch Ausführen der Transformationsdefinition durch den Transformations-Engine in das Zielmodell der Transformation umgewandelt, welches konform zum Zielmetamodell ist (Abb. 19) [20].

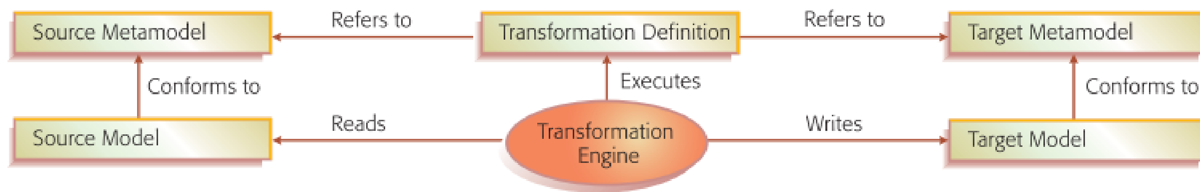

Abb. 19: Illustration d. Zusammenhänge zwischen Quell-/Ziel(meta)modell und d. Transformationsdefinition [21]

Es gibt verschiedene Ansätze diese Transformationsdefinition zu spezifizieren [21]:

- Direct Manipulation Approach: Hier wird das Quellmodell in eine interne Repräsentation umgewandelt, die über ein API manipuliert werden kann. Die Modelltransformation wird meist mit Hilfe einer imperativen Programmiersprache implementiert, die das API nützt.
- Structure-driven Approach: Dieser Ansatz charakterisiert sich dadurch, dass die Transformation zwei Phasen aufweist. Die erste Phase baut die Struktur des Zielmodells auf, während in der zweiten Phase die Attribute und Verbindungsreferenzen erstellt werden.
- Operational Approach: Er ähnelt dem Direct Manipulation Approach, allerdings wird dieser Ansatz um Formalismen erweitert, um auch komplexere Programme zu spezifizieren. Dadurch wird keine zusätzliche allgemeine Programmiersprache mehr benötigt. Dies könnte z.B. eine Abfragesprache mit imperativen Elementen, wie OCL [22], sein, die ein eigenständiges Programmiersystem bildet.
- Template-based Approach: Hier werden Modell Templates verwendet, die Teilmodelle mit eingebettetem Meta-Code darstellen. Der Meta-Code wird bei der Transformation ausgewertet und die Ergebnisse davon stattdessen eingefügt. Dadurch können die dynamischen Anteile mit einem statischen Teilmodell kombiniert werden.
- Graph-transformation-based Approach: Der Ansatz beruht auf Graftransformationen und kann auf typisierte, attributierte, beschrifftete Grafen angewandt werden. Eine Transformationsregel besteht aus einer linken (LHS) und einer rechten Seite (RHS) mit jeweils einem Grafmuster. Die LHS beschreibt das Muster, das im Graf vorgefunden werden muss, um die Transformationsregel zu aktivieren. Die RHS beschreibt das Muster, durch das die LHS ersetzt wird, um den gewünschten Zielgrafen zu erhalten.
- Hybride Approach: Bei diesem Ansatz werden beliebige oben genannte Ansätze miteinander kombiniert, um die Transformationsdefinition zu spezifizieren.

Diese Aufzählung soll nur einen Überblick über mögliche Wege geben, wie Transformationsregeln definiert werden können. Für weitere Details sei auf [21] verwiesen.
Alle hier angeführten Ansätze haben jedoch gemeinsam, dass sie das Quell- und Zielmetamodell nützen, um die Transformationsdefinition zu spezifizieren. Dadurch ist Wissen über den Aufbau der Metamodelle eine Voraussetzung für den Transformationsdesigner.

Der im nächsten Abschnitt gezeigte Ansatz unterscheidet sich grundsätzlich von den bisher gezeigten und arbeitet mit einer anderen, benutzerfreundlichen Art der Transformationsspezifikation, mit der auch Personen arbeiten können, die mit den Metamodellen nicht vertraut sind.

6.1. Model Transformation by-Example (MTBE)

Modelle werden in unterschiedlicher Art dargestellt. Der Designer/Modellierer, der die Modelle meist mit einem grafischen Modellierungswerkzeug erstellt, arbeitet mit der grafischen Notation (auch konkrete Syntax (concrete syntax, CS) genannt, Abb. 20 oben) des Modells, in der jeder Entität des Metamodells eine grafisches Symbol zugeordnet ist. In einem UML Klassendiagramm z.B. werden Klassen als Rechtecke und Relationen zu anderen Klassen als Kanten zwischen den Klassensymbolen dargestellt [23].

Intern wird das Modell jedoch entsprechend dem Metamodell in abstrakter Syntax (abstract syntax, AS, Abb. 20 unten) abgebildet. Die oben in Abschnitt 6 angeführten Spezifizierungsmethoden arbeiten auf Ebene der abstrakten Syntax bzw. des Metamodells, indem Transformationsregeln unter Verwendung der Metamodellelemente definiert werden.

Dadurch ergeben sich zwei problematische Aspekte [23]:

(1) Die Art wie dem Benützer das Modell repräsentiert wird und wie er es sieht, unterscheidet sich von der elektronischen Repräsentationsform (Metamodell).
(2) Die Intention des Metamodell liegt darin, eine Modellierungssprache zu definieren, die effizient implementiert werden kann. Daher werden nicht alle Konzepte der Modellierungssprache explizit im Metamodell ausgedrückt, sondern verbergen sich in Attributen oder Relationen und müssen später rekonstruiert werden. Man spricht hier auch von versteckten Konzepten (concept hiding).

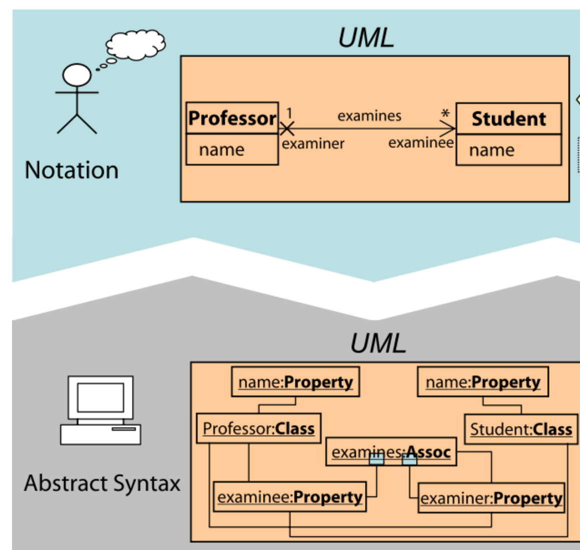

Abb. 20: UML Klassendiagramm oben in konkreter, unten in abstrakter Syntax [23]

Beide Aspekte erschweren die Spezifikation von Modelltransformationsregeln.

Die Idee von Model Transformation by-Exampel (MTBE) ist nun, ein Mapping auf der Ebene der konkreten Syntax zwischen zwei Modellen zu definieren und daraus Transformationsregeln auf Metamodellebene zu generieren.

Unter der Voraussetzung, dass Relationen für jede Modellierungssprache vorhanden sind, die Elemente der abstrakten in Elemente der konkreten Syntax abbilden, kann MTBE mit Hilfe der folgenden drei Schritte definiert werden [23]:

(1) Als erstes muss der Benutzer ein oder mehrere Quell- und Zielmodell(e) modellieren die dasselbe Problem beschreiben. Die definierten Modelle sollten dabei möglichst alle Konzepte der Modellierungssprache abdecken.
(2) Danach müssen die korrespondierenden Elemente in den Modellen verbunden und so ein Mapping zwischen den Modellelementen des Quell- und Zielmodells definiert werden. Dafür stehen nicht nur 1:1, sondern auch komplexere Mapping-Operatoren zu Verfügung [19].
(3) Im letzten Schritt wird das Mapping durch einen Modelltransformationsgenerator (Model Transformation Generator, MTGen) analysiert und so eine Transformationsdefinition

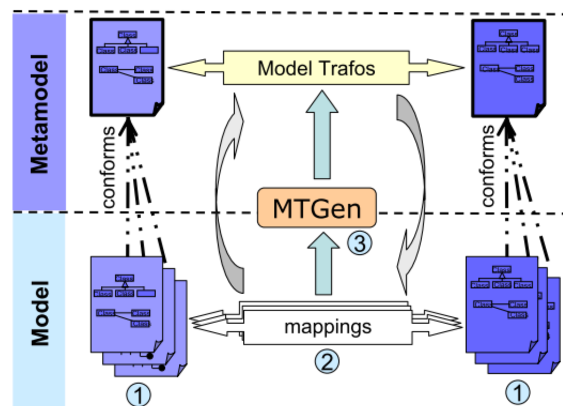

Abb. 21: MTBE Spezifizierungsschritte im Kontext der Architekturschichte [20]

generiert, die auf bewährten metamodellbasierten Transformationstechnologien basiert. Durch das Mapping etwaige ergebende Zweideutigkeiten, müssen durch den Transformationsdesigner aufgelöst werden.

Abb. 21 zeigt die einzelnen Spezifikationsschritte im Kontext der Modellierungsarchitekturschichten. Die für die Spezifikation der von MTBE benötigten Modelle und das benutzerdefinierten Mapping zwischen den Modellelementen befinden sich auch Modellebene (M1). Die von MTGen generierten Transformationsregeln (Model Trafos) befinden sich auch Metamodellebene (M2).

Da bei diesem Ansatz Transformationen auf der Modellebene (M1) definiert werden, muss man nicht mit im Metamodell versteckten Konzepten umgehen. Außerdem wird mit der graphischen Notation des Modells gearbeitet, mit der der Benützer bereits vertraut ist. Dadurch soll es einfach möglich sein Transformationsregeln zu erstellen, auch ohne mit dem Metamodell der Modellierungssprache vertraut zu sein [23].

Die bisher angeführten Transformationen beruhen darauf, das Elemente im Quellmodell in ein oder in mehrere Elemente des Zielmodells umgesetzt werden, und auch die Transformationsregeln werden entsprechend spezifiziert. Für Anwendungsfälle wo eine 1:1 bzw. 1:n Transformation nur schwer anwendbar ist, stellt der im nächsten Abschnitt vorgestellte Ansatz eine Alternative dar.

6.2. Musterbasierte Modelltransformation

Aktuelle Technologien für Modelltransformationen, wie QVT [24] oder ATL [25], beruhen auf der Transformation von Metamodellelementen. Dadurch sind Transformationen schwierig zu definieren, die nicht darauf basieren, ein Element im Quellmodell in ein oder mehrere Element(e) im Zielmodell abzubilden.
Diese komplexen Transformationen benötigen oft Informationen, die erst durch komplexe Abfragen zur Verfügung stehen, die jedoch fehleranfällig, schwer zu warten und kaum wiederverwendbar sind [18].

Abb. 22 zeigt einen Prozess in ADONIS [26] Notation. Die Besonderheit an ADONIS ist, dass hier kein explizites Element vorhanden ist, um die Vereinigung einer alternativen (XOR) Verzweigung auszudrücken. Somit vereinigt das Parallel Join Element (in Abb. 22 rot eingekreist) nicht nur die parallele, sondern auch implizit die alternative Verzweigung.
Wird dieser Beispielprozess in ein Modell transformiert, welches ein explizites Element dafür vorsieht, muss erst durch eine komplexe Abfrage ermittelt werden, ob im Pfad der Vorgängerelemente eine alternative Verzweigung vorhanden ist. Da diese Abfragen fehleranfällig und kaum wiederzuverwenden sind, ist dieser Lösungsansatz unbefriedigend.

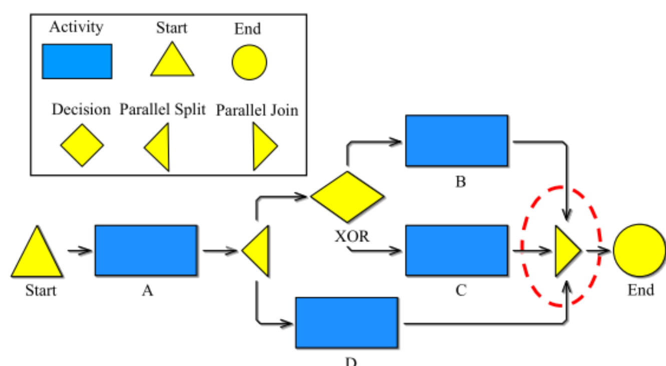

Abb. 22: Prozess in ADONIS Notation; Parallel Join Element (roter Kreis) schließt parallele u. alternative Verzweigung [18]

In [18] wird eine musterbasierte Herangehensweise erläutert, die Übersetzungsmuster (transformation pattern) für den Übersetzungsprozess nutzt. D.h. der zu übersetzenden Prozess wird in einer ersten Phase analysiert und durch eine Abfolge bzw. Verschachtelung von Übersetzungsmustern beschrieben. Diese Abfolge von Übersetzungsmustern wird danach in einer

zweiten Phase in einen Zielprozess umgesetzt. Dabei werden die Transformationsmuster mit Elementen des Zielmodells synthetisiert und bauen so Muster für Muster den Zielprozess auf.

Dadurch, dass der zu transformierende Prozess durch Muster auf einem höheren Abstraktionslevel beschrieben wird, soll die Definition der Transformation erleichtert und die Wiederverwendung verbessert werden.

Für unseren Beispielprozess aus Abb. 22 bedeutet das, dass er ein Sequential-Path Pattern aus A und einem Parallel Pattern darstellt. Dieses Parallel Pattern aus einem Alternative Pattern und D und das Alternative Pattern aus B und C besteht.
Diese musterbasierte Zwischendarstellung des Quellprozesses kann im Zielmodell synthetisiert werden und baut so den Zielprozess auf. Dies funktioniert natürlich nur, wenn das Zielmodell diese Muster auch unterstützt [18].
Dies ist ein vielversprechender Transformationsansatz und eignet sich auch für die Transformation von graf- zu blockorientierten Prozessmodellen.

7. Konklusion und weitere Studien

In dieser Arbeit wurden formale Modelle zur Beschreibung von graf- und blockorientierter Sprachen vorgestellt. Diese dienen als Grundlage, um Einschränkungen des Eingangsmodells und den Transformationsprozess besser zu beschreiben.
Weiters wurden wichtige Eigenschaften von Prozessgrafen und BCF erörtert, die ein Modell erfüllen muss, um von bestimmten Strategien transformiert werden zu können. Es werden auch typische Diskrepanzen zwischen graf- und blockorientierten Prozessmodellen aufgezeigt und mit Beispielen auf die Problematik hingewiesen.
Im Anschluss daran wurden die verschiedenen Transformationsstrategien vorgestellt. Sie zielen darauf ab, Modelle möglichst vollständig und automatisiert zu transformieren und dabei möglichst lesbare Modelle zu erhalten. Dies ist gerade bei Übersetzungen von graf- auf blockorientierte Prozessmodelle nicht immer trivial. Die hier beschriebenen Strategien nützen verschiedene Features von BPEL aus, um typische Diskrepanzen zwischen den beiden Repräsentationsparadigmen zu umgehen.
In dieser Arbeit wird BPEL stellvertretend für viele weitere blockorientierte Workflow-Modelle herangezogen. Weitere Forschung wird dahingehend abzielen, ob diese Strategien auch auf Asbru anwendbar sind, und welche sich dafür am besten eignen.
Asbru ist eine Beschreibungssprache für medizinische Richtlinien, die dafür entwickelt wurde Diagnose- und Behandlungspläne zu beschreiben. Auf Grund fehlender grafischer Modellierungswerkzeuge und um eine implementierungsunabhängige Darstellung zu gewährleisten, soll versucht werden, eine BPMN-zu-Asbru Transformation für den Kontrollfluss zu implementieren. Deshalb wird untersucht werden, ob und welche der o.a. Übersetzungsstrategien darauf anwendbar sind.

Referenzen

[1] Z. Jörg und M. Jan, „Epc-Based Modelling Of Bpel Processes".In Proceedings of MITIP 2005, Italy.

[2] J. Mendling, K. B. Lassen und U. Zdun, „On the transformation of control flow between block-oriented and graph-oriented process modelling languages," International journal of business process integration and management, Bd. 3, Nr. 2, pp. 96-108, Oct. 2008.

[3] I. (. Object Management Group, „Business Process Model and Notation (BPMN)," 2011. [Online]. Available: http://www.omg.org/spec/BPMN/2.0/PDF/. [Zugriff am 11 4 2012].

[4] OMG, „OMG Unified Modeling Language Superstructure, Version 2.4..1," 2011. [Online]. Available: http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/. [Zugriff am 15 6 2012].

[5] W. M. Aalst, „Verification of Workflow Nets," ICATPN '97 Proceedings of the 18th International Conference on Application and Theory of Petri Nets, pp. 407-426, 1997.

[6] G. Keller, M. Nüttgens und A. W. Scheer, „Semantische Prozessmodellierung auf der Grundlage„Ereignisgesteuerter Prozeßketten (EPK)"," Inst. fuer Wirtschaftsinformatik, Saarbruecken, Germany, Nr. 89, 1992.

[7] W. M. P. van der Aalst und A. H. M. ter Hofstede, „YAWL: yet another workflow language," Information Systems, Bd. 30, Nr. 4, pp. 245-275, 2005.

[8] OASIS, „Web Services Business Process Execution Language Version 2.0," 2007. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf. [Zugriff am 15 6 2012].

[9] A. Seyfang, R. Kosara und S. Miksch, „Asbru's Reference Manual, Asbru Version 7.3," [Online]. Available: http://www.asgaard.tuwien.ac.at/asbru_7_3/asbru_7.3_reference.pdf. [Zugriff am 15 6 2012].

[10] J. Mendling, H. Reijers und A. H. Wil, „Seven Process Modeling Guidelines (7PMG).," Information and Software Technology, Nr. 52, pp. 127-136, 2010.

[11] Oliver Kopp, Daniel Martin, Daniel Wutke, Frank Leymann und Ulrich Frank, „The Difference Between Graph-Based and Block-Structured Business Process Modelling Languages," Enterprise Modelling and Information Systems, Bd. 4, Nr. 1, p. 3-13, 2009.

[12] C. Ouyandg, W. M. Aalst, M. Dumas und A. H. Hofstede, „Translating BPMN to BPEL," 2006-01.

[13] C. Ouyang, M. Dumas, A. H. M. ter Hofstede und W. M. P. van der Aalst, „From BPMN Process Models to BPEL Web Services," International Conference on Web Services ICWS '06, pp. 285-292, 2006.

[14] Jan C. Recker und Jan Mendling, „On the Translation between BPMN and BPEL: Conceptual

Mismatch between Process Modeling Languages," Namur University Press, 2006, p. 521-532.

[15] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski und A. P. Barros, „Workflow Patterns," Distributed and Parallel Databases, Bd. 14, Nr. 1, pp. 5-51, 2003.

[16] C. Ouyang, M. Dumas, S. Breutel und A. ter Hofstede, „Translating Standard Process Models to BPEL," Bd. 4001, Springer Berlin / Heidelberg, 2006, pp. 417-432.

[17] N. Mulyar, W. M. Aalst und M. Peleg, „A pattern-based analysis of clinical computer- iterpretable guideline modeling languages".

[18] M. Murzek, G. Kramler und E. Michlmayr, „Structural Patterns for the Transformation of Business Process Models," Models for Enterprise Computing 2006 - International Workshop at EDOC 2006, p. 43-52, 2006.

[19] M. Strommer, M. Murzek und M. Wimmer, „Applying Model Transformation By-Example on Business Process Modeling Languages," LNCS 4802, Springer, 2007, p. 116-125.

[20] M. Strommer, „Model Transformation By-Example," Vienna University of Technology, 01.05.2008.

[21] K. Czarnecki und S. Helsen, „Feature-based survey of model transformation approaches," IBM SYSTEMS JOURNAL, Bd. 45, Nr. 3, pp. 621-645, 2006.

[22] OMG, „Object Constraint Language Version 2.2," 2010. [Online]. Available: http://www.omg.org/spec/OCL/2.2/PDF. [Zugriff am 9 7 2012].

[23] M. Strommer, M. Wimmer, H. Kargl und G. Kramler, „Towards Model Transformation Generation By-Example," 40, IEEE Computer Scociety, 2007, p. 285-286.

[24] OMG, „Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification Version 1.1," 2011. [Online]. Available: http://www.omg.org/spec/QVT/1.1/PDF/. [Zugriff am 9 7 2012].

[25] F. Jouault und I. Kurtev, „Transforming Models with ATL," in s Satellite Events at the MoDELS 2005 Conference, Bd. 3844, Springer Berlin / Heidelberg, 2006, pp. 128-138.

[26] M. Murzek und G. Kramler, „BUSINESS PROCESS MODEL TRANSFORMATION ISSUES The top 7 adversaries encountered at defining model transformations," Proceedings of the ninth international conference on enterprise information systems, p. 144-151, 2007.