Forderungen an hierarchische EPK-Schemata

Volker Gruhn, Ralf Laue
{gruhn,laue}@ebus.informatik.uni-leipzig.de
Lehrstuhl für Angewandte Telematik und E-Business*
Universität Leipzig, Fakultät für Informatik

Abstract: Ereignisgesteuerte Prozessketten (EPK) sind eine Notation zur Modellierung von Geschäftsprozessen. Komplexe Geschäftsprozesse werden üblicherweise nicht als eine einzige EPK modelliert, sondern durch mehrere EPK-Schemata, die durch Prozesswegweiser und hierarchisierte Funktionen miteinander verbunden sind. In diesem Beitrag wird untersucht, welche Forderungen an solche EPK-Schemata zu stellen sind, um eine eindeutige Interpretation zu gewährleisten.

1 Einführung

Ereignisgesteuerte Prozessketten (EPK) sind eine Notation zur Modellierung von Geschäftsprozessen. Da sie im SAP-Referenzmodell angewendet wurden und zentraler Bestandteil der ARIS-Plattform sind, fanden sie insbesondere im deutschsprachigen Raum eine weite Verbreitung.

Ursprünglich wurden EPKs eingeführt, ohne deren Syntax und Semantik formell zu definieren[KNS92]. Zahlreiche Autoren schlugen später formale Definitionen zur Syntax und Semantik vor, um diese Lücke zu schließen. Eine (bei weitem nicht vollständige) Auswahl von einschlägigen Arbeiten ist [CS94, LSW97b, van99, NR02, Kin04, Men07].

Alle Arbeiten zur Definition einer Syntax und Semantik richten ihr Augenmerk allerdings kaum auf die Konstrukte, die dazu dienen, mehrere EPKs miteinander zu verbinden: Prozesswegweiser und hierarchisierte Funktionen. Diese Konstrukte sind in der Praxis sinnvoll, um zum einen eine bessere Übersichtlichkeit in der Modellierung zu erhalten, zum anderen Modellteile mehrfach wiederverwenden zu können.

[NR02] und [Rum99] formulieren grundlegende Anforderungen an Prozesswegweiser und hierarchisierte Funktionen, die den Ausgangspunkt für unsere weitergehenden Untersuchungen bilden.

In allen uns bekannten Veröffentlichungen wird davon ausgegangen, dass bei Einhaltung der durch [NR02] und [Rum99] aufgestellten Forderungen durch hierarchisierte Funktionen und Prozesswegweiser miteinander verbundene EPKs problemlos zu einer einzigen

*Der Lehrstuhl für Angewandte Telematik und E-Business ist ein Stiftungslehrstuhl der Deutschen Telekom AG

EPK zusammengefasst („flachgeklopft“) werden können. Wir zeigen in diesem Beitrag, dass dies keineswegs in jedem Falle ohne Schwierigkeiten möglich ist. Wir zeigen weiterhin einige Ansätze, um die Probleme zu lösen. Hauptanliegen dieses Beitrags soll es jedoch sein, anhand von Beispielen zu verdeutlichen, welche Schwierigkeiten die Aufgabe des „Flachklopfens“ mehrerer EPK-Schemata zu einem einzelnen flachen EPK-Schema aufwirft.

2 Grundlegende Definitionen

Für die weiteren Ausführungen setzen wir das Konzept der EPK als bekannt voraus. Bei der Formalisierung der EPK-Syntax und -Semantik orientieren wir uns an den in [NR02] und [Rum99] eingeführten Bezeichnungen. Wir vereinfachen diese jedoch auf diejenigen Elemente, die für unsere weiteren Betrachtungen relevant sind.

Definition 1 *(flaches EPK-Schema):*
Ein flaches EPK-Schema A ist ein Tupel $A = (E, F, P, V, J)$. Dabei ist

- *E eine nichtleere Menge von Ereignissen*
- *F eine nichtleere Menge von Funktionen*
- *P eine Menge von Prozesswegweisern*
- *V eine Menge von Verknüpfern (Konnektoren)*
- *$K = E \cup F \cup P \cup V$ die Menge aller Notationselemente*
- *$J \subset K \times K$ eine Menge von Kontrollflusskanten.*

E, F, P und V sind paarweise disjunkt.

Wir wollen im Weiteren ein flaches EPK-Schema kurz als EPK bezeichnen. An die Elemente eines flachen EPK-Schemas werden zusätzliche Anforderungen (etwa zur Multiplizität) gestellt, die in [NR02] und [Rum99] zu finden sind.

Zusätzlich werden die folgenden Bezeichnungen eingeführt:

- Für zwei Notationselemente $j, k \in K$ schreiben wir $j \rightarrow k$ genau dann, wenn $(j, k) \in J$, wenn es also eine Kontrollflusskante von j nach k gibt.
- $\rightarrow^*$ bezeichnet die transitive Hülle der durch $\rightarrow$ beschriebenen Relation.
- Die Menge der **direkten Vorgänger** eines Notationselements $k \in K$ ist $\bullet k = \{x \in K : (x, k) \in J\}$.
- Die Menge der **direkten Nachfolger** eines Notationselements $k \in K$ ist $k\bullet = \{x \in K : (k, x) \in J\}$.

- Für Notationselemente $j \in F \cup P$ definieren wir die Menge $VE(j)$ der **vorangehenden Ereignisse** als die Menge aller $e \in E$, für die $e \rightarrow j$ gilt oder es eine Folge von Verknüpfern $v_1, \ldots, v_n \in V$ gibt so dass gilt: $e \rightarrow v_1 \rightarrow \cdots \rightarrow v_n \rightarrow j$. Analog definieren wir die Menge $NE(j)$ der **nachfolgenden Ereignisse** als die Menge aller $e \in E$, für die $j \rightarrow e$ gilt oder es eine Folge von Verknüpfern $v_1, \ldots, v_n \in V$ gibt so dass gilt: $j \rightarrow v_1 \rightarrow \cdots \rightarrow v_n \rightarrow e$.
- Die Menge end(A) aller Endereignisse des flachen EPK-Schemas $A = (E, F, P, V, J)$ ist die Menge aller $e \in E$, für die kein $e' \in E$ mit $e \rightarrow^* e'$ existiert.
- Die Menge start(A) aller Startereignisse des flachen EPK-Schemas $A = (E, F, P, V, J)$ ist die Menge aller $e \in E$, für die kein $e' \in E$ mit $e' \rightarrow^* e$ existiert.[1]
- Ein $p \in P$ heißt **Startprozesswegweiser**, wenn $\bullet p = \emptyset$ und **Endprozesswegweiser**, wenn $p\bullet = \emptyset$.
- Ein Konnektor x heißt Split, wenn $\bullet x$ genau ein und $x\bullet$ mindestens zwei Elemente hat und Join, wenn $x\bullet$ genau ein und $\bullet x$ mindestens zwei Elemente hat.

3 Prozesswegweiser und hierarchisierte Funktionen

Mehrere flache EPK-Schemata können durch Prozesswegweiser und hierarchisierte Funktionen miteinander verbunden werden. Informell können wir die Bedeutung dieser Notationselemente wie folgt beschreiben:

Prozesswegweiser: Ein Prozesswegweiser leitet den Kontrollfluss von einer EPK zu einer anderen weiter. Es wird also „von einer EPK zu einer anderen gesprungen“, was mit einer GOTO-Anweisung in einer imperativen Programmiersprache vergleichbar ist.

Hierarchisierte Funktion: Trifft der Kontrollfluss in einer EPK auf eine hierarchisierte Funktion, so referenziert diese eine andere EPK. Der Kontrollfluss verzweigt zu der referenzierten EPK und kehrt zur „rufenden“ EPK zurück, wenn der Ablauf in der referenzierten EPK beendet ist. Dies ist vergleichbar mit einem Unterprogrammaufruf in einer imperativen Programmiersprache.

Wir führen wir die Menge der hierarchisierten Funktionen einer EPK in der folgenden Definition ein:

Definition 2 *(Hierarchisierte Funktion): Für eine EPK $A = (E, F, P, V, J)$ bildet die Menge $\widehat{F}$ der hierarchisierten Funktionen eine Untermenge der Funktionsmenge F.*

Formal werden Prozesswegweiser und hierarchisierte Funktionen in [NR02] und [Rum99] mit Hilfe einer Hierarchierelation beschrieben. Um die Schreibweise ein wenig zu vereinfachen, verwenden wir (ähnlich wie auch in [Men07] eingeführt) statt dessen den Begriff

[1] Die in manchen Quellen anzutreffende Definition von Start- und Endereignissen als Menge aller Ereignisse mit $e\bullet = \emptyset$ bzw. $\bullet e = \emptyset$ kann nicht verwendet werden, wenn Prozesswegweiser zugelassen sind.

der **Verweisfunktion**, die jedem Prozesswegweiser und jeder hierarchisierten Funktion das flache EPK-Schema zuweist, auf das sie verweisen.

Definition 3 *(Verweisfunktion, referenziertes EPK-Schema, EPK-Schemamenge, Verfeinerung):*
Sei $S = \{(E_1, F_1, P_1, V_1, J_1), \ldots, (E_n, F_n, P_n, V_n, J_n)\}$ *eine Menge flacher EPK-Schemata.*

Die Verweisfunktion h ist eine Funktion, die allen Prozesswegweisern, die in einem der EPK-Schemata aus S *vorkommen, sowie allen hierarchisierten Funktionen, die in einem der EPK-Schemata aus* E *vorkommen, ein EPK-Schema aus* S *zuweist. Dieses zugewiesene EPK-Schema nennen wir im Folgenden auch „referenziertes Schema"; für ein* $f \in \widehat{F}_i$ *heißt die EPK h(f) Verfeinerung von f.*

Somit ist die Abbildung h also erklärt als

$h : \bigcup_{i=1}^{n} \widehat{F}_i \cup P_i \longrightarrow S$

S selbst bezeichnen wir dann als EPK-Schemamenge.

Zur Bedeutung der Verweisfunktion ist zu beachten: Bei hierarchisierten Funktionen $f \in \widehat{F}$ ist h(f) die Verfeinerung von f. Für einen Endprozesswegweiser p_e ist $h(p_e)$ die EPK, zu der im Kontrollfluss „gesprungen" wird. Für einen Startprozesswegweiser p_s jedoch ist $h(p_s)$ die **„rufende"** EPK.[2] Wollen wir nur die „rufenden" Verweise betrachten, müssen wir die Startprozesswegweiser aus dem Definitionsbereich herausnehmen. Dies führt zu:

Definition 4 *(Aufruffunktion): Für ein EPK-Schema* $A_i \in S$ *bezeichne* P_i^{end} *die Menge der Endprozesswegweiser in* A_i*. Die Aufruffunktion h' ist die auf die Menge der Endprozesswegweiser und hierarchisierten Funktionen eingeschränkte Verweisfunktion h.*
h' ist also eine Abbildung $h' : \bigcup_{i=1}^{n} \widehat{F}_i \cup P_i^{end} \longrightarrow S$

4 Einfache Syntaxanforderungen an EPK-Schemamengen

[NR02] formuliert nun einige Anforderungen an EPK-Schemamengen wie folgt:

$E = \{A_1, \ldots, A_n\}$ sei eine EPK-Schemamenge mit der Verweisfunktion h. Dann muss gelten:

F 1 Die Menge der vorangehenden Ereignisse einer hierarchisierten Funktion entspricht der Menge der Startereignisse des referenzierten EPK-Schemas. Es gilt also:
$\forall f \in \widehat{F} : VE(f) = start(h(f))$

F 2 Die Menge der nachfolgenden Ereignisse einer hierarchisierten Funktion entspricht der Menge der Endereignisse des referenzierten EPK-Schemas. Es gilt also:

[2] In [NR02] und [Rum99] wird dieser Unterschied nicht beachtet, was dort zu Inkorrektheiten in den Definitionen führt.

$\forall f \in \widehat{F} : NE(f) = end(h(f))$

F 3 Die Menge der vorangehenden Ereignisse eines Endprozesswegweisers ist eine Teilmenge der Startereignisse des referenzierten EPK-Schemas. Es gilt also:

$\forall p \in P : p\bullet = \emptyset \Rightarrow VE(p) \subseteq start(h(p))$

F 4 Die Menge der nachfolgenden Ereignisse eines Startprozesswegweisers ist eine Teilmenge der Endereignisse des referenzierten Schemas. Es gilt also:

$\forall p \in P : \bullet p = \emptyset \Rightarrow NE(p) \subseteq end(h(p))$

F 5 Verbot der Rekursion: Ein EPK-Schema $A \in S$ ist nicht über mehrfache Anwendung der Aufruffunktion h' mit sich selbst verbunden. Es gibt also keine Folge $A_0, \ldots A_n \in S$, so dass für jedes $i = 0, \ldots, n-1$ ein Element $x_i \in \widehat{F}_i \cup P_i^{end}$ existiert, so dass $h'(x_i) = A_{i+1}$ und $A_0 = A_n = A$ gilt.

Man beachte bei den Punkten 3 und 4, dass die vorangehenden Ereignisse eines Endprozesswegweisers in der „aufrufenden“, die nachfolgenden Ereignisse eines Startprozesswegweisers in der „gerufenen“ EPK liegen.

In den folgenden Abschnitten werden wir Unzulänglichkeiten dieser Definitionen benennen und Verbesserungen vorschlagen.

5 Verbesserte Forderungen zur Aufrufhierarchie

Das Verbot der Rekursion in F5 soll vermeiden, dass sich eine EPK unendlich oft selbst aufrufen kann. Allerdings erscheint uns das Verbot eines solchen Selbstaufrufs im Falle von Prozesswegweisern als zu strikt. Gegen Modelle wie in Abb. 1 (zu finden z.B. im Referenzmodell „Ergebnisplanung“ in [Kel99]) ist wenig einzuwenden. Der Verweis eines Endprozesswegweisers auf einen Startprozesswegweiser der selben EPK dient hier nur der Übersichtlichkeit und ersetzt lediglich einen Kontrollflusspfeil. In Abb. 1 ist dies sicher wenig sinnvoll, bei größeren Modellen kann es aber die Übersichtlichkeit erhöhen. Tatsächlich rekursive Aufrufe finden beim Verfolgen von Prozesswegweisern nicht statt.

Zu fordern ist lediglich, dass es in den Zyklen von EPKs, die sich per Prozesswegweiser einander aufrufen, in mindestens einer EPK Start- bzw. Endereignisse gibt, damit die Bearbeitung auch tatsächlich begonnen bzw. beendet werden kann.

Kritisch dagegen ist eine „Nacheinanderschaltung“ von Aufrufen hierarchisierter Funktionen und dem Verfolgen von Prozesswegweisern zu sehen. Abb. 2 zeigt einen Ausschnitt aus einer EPK, die eine hierarchisierte Funktion aufruft. Will man mit dieser EPK arbeiten, muss man sich darauf verlassen können, dass nach Eintreten des Ereignisses „Vorbedingung“ die hierarchisierte Funktion H ausgeführt wird und anschließend immer das Ereignis „Nachbedingung“ eintritt. Eine genaue Betrachtung der durch H referenzierten EPK h(H), also ein Ändern der Abstraktionsebene, darf für das Verständnis der EPK nicht erforderlich sein.

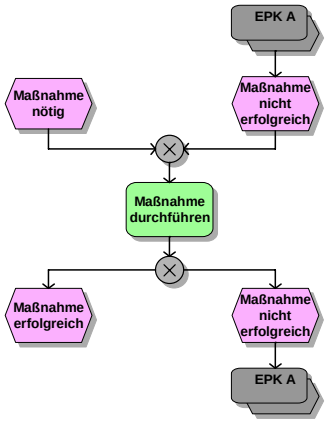

Abbildung 1: Rekursiver Prozesswegweiser

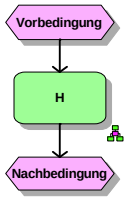

Abbildung 2: Aufruf einer hierarchisierten Funktion

Enthielte aber nun die referenzierte EPK h(H) ausgehende Prozesswegweiser, die auf eine andere EPK als h(H) selbst verweisen, könnte dies dazu führen, dass der Kontrollfluss derart verzweigt, dass das Ereignis „Nachbedingung“ möglicherweise nie eintritt und die Kontrolle nicht an die übergeordneten EPK zurückgegeben wird. Besonders schlimm ist dabei, dass man einen solchen möglichen Ablauf nicht durch Betrachten der übergeordneten EPK erkennen kann. Ähnliche bei Betrachtung der übergeordneten EPK unerwartete Abläufe sind denkbar, wenn die von einer hierarchisierten Funktion referenzierte EPK eingehende Prozesswegweiser enthält. Um solchen Problemen zu begegnen, kann man die Forderung aufstellen, dass eine EPK, die Verfeinerung einer hierarchisierten Funktion ist, keine heraus- oder hereinführenden Prozesswegweiser haben darf. [3] Uns ist bewusst, dass diese Forderung eine recht starke Einschränkung ist, die durchaus der heute teilweise üblichen Praxis widerspricht.

Zusammenfassend ergeben sich die folgenden Forderungen, die Punkt 5 aus Abschnitt 4 ersetzen:

[3] Darüberhinaus sollte eine EPK, die Verfeinerung einer hierarchisierten Funktion ist, sogar stark sound[van99] sein, um Effekte ähnlich der beschriebenen zu vermeiden. Dies ist im Übrigen ein gutes Argument gegen schwächere Soundnesskriterien[DR01]: EPKs, die nicht stark sound sind, eignen sich nicht als Verfeinerung einer hierarchisierten Funktion. Diese Betrachtungen sind aber schon semantischer Natur und führen daher über das Anliegen dieses Beitrags hinaus.

F 5’ Verbot der Rekursion über hierarchisierte Funktionen: Ein EPK-Schema $A \in S$ ist nicht über mehrfache Anwendung der Aufruffunktion h’ auf hierarchisierte Funktionen mit sich selbst verbunden. Es gibt also keine Folge $A_0, \ldots A_n \in S$, so dass für jedes $i = 0, \ldots, n-1$ eine Funktion $x_i \in \widehat{F_i}$ existiert, für die $h'(x_i) = A_{i+1}$ und weiterhin $A_0 = A_n = A$ gilt.

F 6 Gibt es eine Folge von EPK-Schemata $A_0, \ldots A_n \in S$ mit $A_0 = A_n$, so dass für jedes $i = 0, \ldots, n-1$ ein Prozesswegweiser p_i in A_i enthalten ist, für den $h'(p_i) = A_{i+1}$, so muss mindestens ein EPK-Schema der genannten Folge ein Startereignis s mit $\bullet s = \emptyset$ und mindestens ein EPK-Schema der genannten Folge ein Endereignis e mit $e\bullet = \emptyset$ enthalten.

F 7 Verbot von ein- und abgehenden Prozesswegweisern in EPK-Schemata, die eine hierarchisierte Funktion verfeinern: Ist eine EPK A Verfeinerung einer hierarchisierten Funktion, so darf A keine Prozesswegweiser enthalten, die auf eine andere EPK als A selbst referenzieren.

6 Korrespondenz zwischen End- und Startprozesswegweisern

In den in Abschnitt 4 genannten Forderungen fehlt bisher noch eine naheliegende Bedingung, nämlich dass es zu jedem rufenden Prozesswegweiser einen gerufenen gibt und umgekehrt.

Man könnte daher folgende Forderung formulieren:

Seien A_1 und A_2 EPK-Schemata, die zu einer Schemamenge gehören. Gibt es in A_1 einen Endprozesswegweiser e mit $h(e) = A_2$, so muss es in A_2 einen Startprozesswegweiser s mit $h(s) = A_1$ geben. Existiert umgekehrt in A_2 ein Startprozesswegweiser s mit $h(s) = A_1$, so muss es in A_1 einen Endprozesswegweiser e mit $h(e) = A_2$ geben.

Aber auch mit dieser Forderung ist noch nicht garantiert, dass Prozesswegweiser aus einem rufendem und einem gerufenem EPK-Schema zusammenpassen. Die drei in Abb. 3 gezeigten Modelle erfüllen sämtliche bisher genannten Forderungen, jedoch stimmen die Ereignisse an den aus- und abgehenden Prozesswegweisern nicht überein, was ein Zusammenfügen der drei Modelle zu einem Gesamtmodell unmöglich macht.

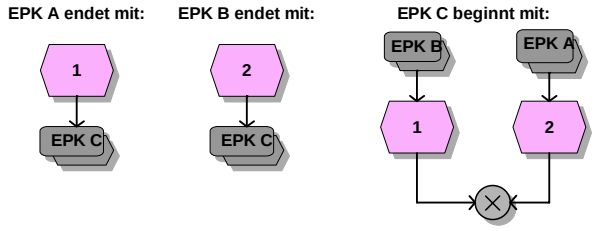

Abbildung 3: Ereignisse an rufender und gerufener EPK stimmen nicht überein

Bei dem Versuch, allgemeine Regeln aufzustellen, wann Ereignisse an Start- und Endprozesswegweisern „zusammenpassen“, trifft man auf die Schwierigkeit, dass sich in der Praxis im Einklang mit den bisher betrachteten syntaktischen Forderungen verschiedene Varianten der Benutzung von Prozesswegweisern durchgesetzt haben.
Abb. 4 zeigt zwei Möglichkeiten, von einer EPK per Prozesswegweiser auf eine andere zu verweisen. Auf den ersten Blick mag man vermuten, dass beide Varianten gleichwertig sind. Dies ist jedoch nicht der Fall. Der Unterschied besteht darin, dass der AND-Kontrollblock im Fall b) bereits abgeschlossen ist. Wie ist das Modell zu interpretieren, wenn die gerufene EPK die in Abb. 5 gezeigte Form hat? Im Falle a) lassen sich rufende und gerufene EPK leicht zu einem Modell zusammenfügen: Sie werden an den doppelt vorkommenden Ereignissen A und B verbunden. Zwar kann man dem erhaltenen Modell wegen des Aussprungs aus dem XOR-Kontrollblock schlechten Modellierungsstil vorwerfen, es ist jedoch durchaus akzeptabel.

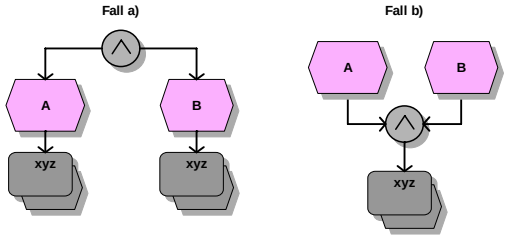

Abbildung 4: Verschiedene Arten der Benutzung von Prozesswegweisern

Anders ist die Situation im Falle b). In der aufrufenden EPK wird davon ausgegangen, dass beide Ereignisse A und B schon eingetreten sind, ehe zur aufgerufenen EPK gesprungen wird. Die gerufene EPK modelliert jedoch einen Ablauf, bei dem die Funktion F durchaus eintreten darf, wenn zwar B, aber noch nicht A eingetreten ist. Folglich können die beiden Modelle nicht sinnvoll zusammengefügt werden.

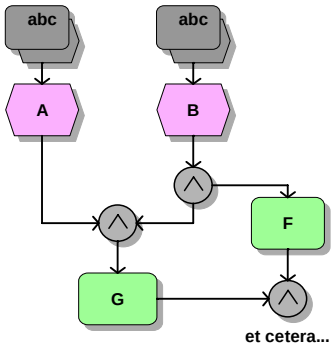

Abbildung 5: Gerufene EPK

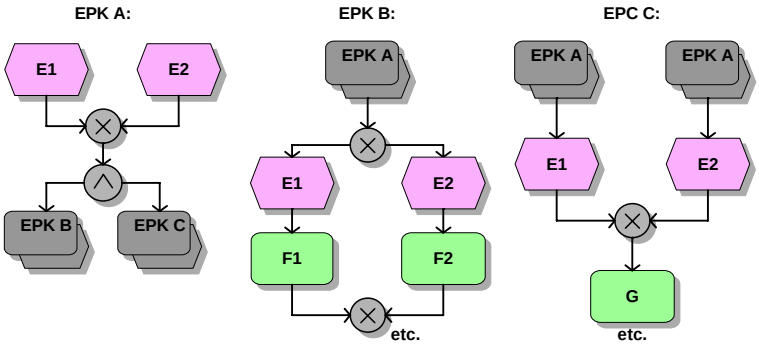

Abbildung 6: Problematische Konstruktionen, kein „Flachklopfen“ möglich

Einen ähnlichen Fall zeigt Abb. 6. Einzeln betrachtet, kann man sowohl EPK A mit EPK B als auch EPK A mit EPK C verbinden. Probleme bereitet aber das Zusammenfügen aller drei Modelle, da die Frage „Darf F1 gleichzeitig mit G ausgeführt werden?“ nicht befriedigend beantwortet werden kann: Die Kontrolllogik von EPK B verlangt, dass F1 und F2 *vor* dem XOR-Join in die EPK A eingefügt werden. F1 könnte demnach nicht gleichzeitig mit G ausgeführt werden. Das widerspricht aber der Kontrollflusslogik in EPK A, wonach F1 und F2 offenbar erst *nach* dem AND-Split einzufügen sind und somit F1 und G durchaus gleichzeitig ausgeführt werden können. Ein sinnvolles Zusammenfügen der drei EPKs ist unmöglich.

Abb. 7 zeigt schließlich den Aufruf einer hierarchisierte Funktion. Das Modell der Verfeinerung besagt, dass die Ausführung der Funktion „Planung“ bewirkt, dass die Ereignisse 1 und 2 eintreten. Das aufrufende Modell jedoch besagt etwas anderes, nämlich dass die Ereignisse 1 und 2 erst eintreten dürfen, wenn auch die Funktion „Bestellung“ ausgeführt wurde. Wegen dieses offensichtlichen Widerspruchs ist ein Zusammenfassen der beiden Modelle nicht möglich.

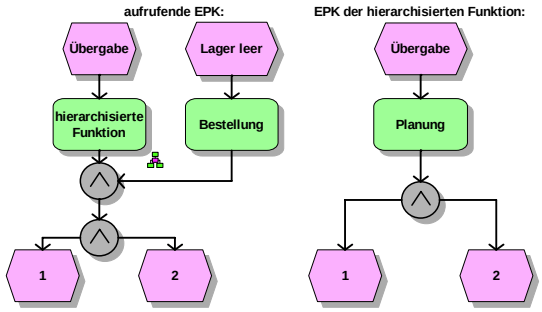

Abbildung 7: Problematische Konstruktionen, kein „Flachklopfen“ möglich

Zur Vermeidung der gezeigten Problemen führen wir im kommenden Abschnitt Beschränkungen für die Verwendung von Prozesswegweisern und hierarchisierten Funktionen ein und zeigen, wie unter diesen Restriktionen ein Flachklopfen erfolgen kann.

7 Restriktionen der Schnittstellen an Prozesswegweisern und hierarchisierten Funktionen

Wir definieren zunächst zwei Typen von Prozesswegweisern wie folgt:

Definition 5 *Ein Startprozesswegweiser s, für den $s\bullet$ ein Ereignis ist sowie ein Endprozesswegweiser e, für den $\bullet e$ ein Ereignis ist, heißt Prozesswegweiser vom Typ 1. Ein Startprozesswegweiser s, für den $s\bullet$ ein Konnektor ist sowie ein Endprozesswegweiser e, für den $\bullet e$ ein Konnektor ist, heißt Prozesswegweiser vom Typ 2.*

In Abb. 4 zeigt Fall a) zwei Prozesswegweiser vom Typ 1, während Fall b) einen Prozesswegweiser vom Typ 2 zeigt.

Durch zusätzliche Forderungen wollen wir nun sicherstellen, dass End- und Anfangsprozesswegweiser zusammenpassen (vgl. Abb. 8).

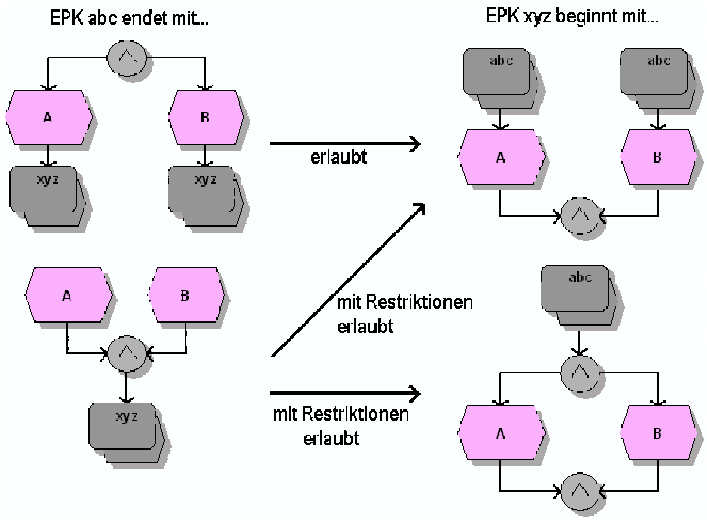

Abbildung 8: Restriktionen bei Aufrufen zwischen Prozesswegweisern

7.1 Endprozesswegweiser vom Typ 1

Prozesswegweiser vom Typ 1 stellen den einfachereren Fall dar. Jedem Endprozesswegweiser geht direkt ein Ereignis voran und jedem Startprozesswegweiser folgt direkt ein Ereignis. Um sicherzustellen, dass zwei EPKs zusammenpassen (vgl. Abb. 3 als Beispiel, wo das nicht der Fall ist), fordern wir:

F 8 Gibt es in einer EPK A_1 einen Endprozesswegweiser p_1 vom Typ 1 mit $h(p_1) = A_2$, so muss in der EPK A_2 ein Startprozesswegweiser p_2 vom Typ 1 mit $h(p_2) = A_1$ existieren, so dass $\bullet p_1 = p_2 \bullet$.

F 9 Gibt es in einer EPK A_1 einen Startprozesswegweiser p_1 vom Typ 1 mit $h(p_1) = A_2$, so muss in der EPK A_2 ein Endprozesswegweiser p_2 vom Typ 1 mit $h(p_2) = A_1$ existieren, so dass $p_1 \bullet = \bullet p_2$.

Das Zusammenfassen („Flachklopfen“) der beiden EPKs ist in diesem Fall problemlos: Die Prozesswegweiser werden entfernt und die EPKs an den in beiden EPKs vorkommenden Ereignissen zusammengefügt.

7.2 Endprozesswegweiser vom Typ 2

Während bei Prozesswegweisern vom Typ 1 rufende und gerufene EPK nur gemeinsame Ereignisse haben, besitzen sie bei Prozesswegweisern vom Typ 2 gemeinsame Konstrukte aus Ereignissen und Konnektoren. Man kann das so interpretieren, dass in der gerufenen EPK auf den Prozesswegweiser zunächst ein Block aus Ereignissen und Konnektoren folgt, der die in der rufenden EPK genannten Vorbedingungen für das Verfolgen des Prozesswegweisers wiederholt. Wie in Abb. 5 zu sehen ist, kann es Probleme geben, wenn in diesem Block bereits etwas von der eigentlichen Ausführungslogik der gerufenen EPK enthalten ist.

Um solche Fälle auszuschließen, fordern wir, dass in der gerufenen EPK dem Startprozesswegweiser ein Teilgraph folgt, dessen einzige Aufgabe es ist, die „Übergabelogik“ zu wiederholen, die durch die dem Prozesswegweiser der rufenden EPK vorausgehenden Ereignisse und Konnektoren vorgegeben ist.

Was wir hierbei unter „Übergabelogik“ verstehen, soll zunächst anhand von Abb. 9 erläutert werden. Hier gelangt der Prozesswegweiser genau dann zur Ausführung, wenn Ereignis 3 sowie entweder Ereignis 1 oder Ereignis 2 eintritt. Die logische Formel (1 xor 2) $\wedge$ 3, die diesen Sachverhalt beschreibt, nennen wir Übergabelogik von P_1.

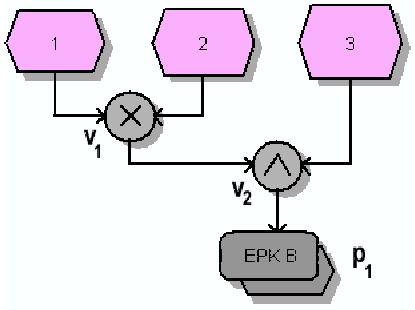

Abbildung 9: Übergabelogik (1 xor 2) $\wedge$ 3

Formal bestimmen wir die Formel, die die Übergabelogik an einem Endprozesswegweiser bestimmt, mit folgendem Algorithmus:

Zur Initialisierung enthalte die Zeichenkette Z den betroffenen Endprozesswegweiser, die Mengen S und J seinen leer. Solange dann die Zeichenkette Z noch EPK-Elemente enthält, die keine Ereignisse sind, führe folgenden Algorithmus aus:

1. Bestimme $\bullet x$ für alle in Z enthaltenen EPK-Elemente x, die keine Ereignisse sind.
2. Ist $\bullet x$ ein Split, so ersetze in Z x durch $\bullet x$ und füge $\bullet x$ der Menge S hinzu.
3. Ist $\bullet x$ ein Join vom Typ $\oplus$ (d.h. $\oplus \in \{xor, \vee, \wedge\}$) so ersetze in Z x durch $(x_1 \oplus \cdots \oplus x_n)$, wobei $\bullet x = \{x_1, \ldots, x_n\}$ die Menge der Vorgänger des Joins ist, und füge $\bullet x$ zur Menge J hinzu.

Ist der Algorithmus abgeschlossen, enthält Z als Zeichenkette eine logische Formel, die wir die Übergabelogik des Endprozesswegweisers nennen.

Außerdem werten wir noch die Mengen S und J aus. Gibt es ein $(s, j) \in S \times J$ mit $s \rightarrow^* j$, wird Z als unbestimmt betrachtet. Bei Endprozesswegweisern mit unbestimmter Übergabelogik ist zu vermuten, dass sie ein falsch modelliertes oder zumindest zweifelhaftes Konstrukt abschließen, so dass wir solche Endprozesswegweiser ausschließen wollen. Abb. 10 zeigt ein Beispiel für ein solches Konstrukt. Auch wenn das gezeigte Modellfragment theoretisch völig korrekt ist, spricht einiges dafür, solche Konstrukte zu verbieten. Weniger restriktive Einschränkungen zu finden, um manche „gutartigen“ Modelle mit unbestimmter Übergabelogik doch nicht auszuschließen, wäre eine Aufgabe für weiterführende Untersuchungen.

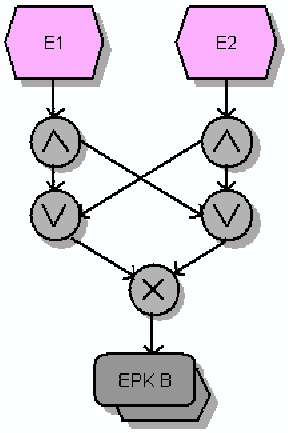

Abbildung 10: Endprozesswegweiser mit unbestimmter Übergabelogik

Für die in Abb. 9 gezeigte EPK sieht die Abarbeitung des Algorithmus wie folgt aus: Zunächst initialisieren wir Z durch $Z = P_1$, wobei P_1 den Prozesswegweiser bezeichnet. Im nächsten Schritt wird $\bullet P_1 = V_2$ bestimmt. Da V_2 ein Join vom Typ $\wedge$ ist, dessen

Vorgänger der Konnektor V_1 und das Ereignis 3 sind, setzen wir im nächsten Schritt $Z = (V_1 \wedge 3)$. Da V_1 ein Join vom Typ xor ist und dessen Vorgänger die Ereignisse 1 und 2 sind, lautet das Ergebnis des nächsten Ausführungsschrittes $Z = ((1 \text{ xor } 2) \wedge 3)$. Da alle in dieser Zeichenkette enthaltenen EPK-Elemente Ereignisse sind, kommt der Algorithmus zum Ende und liefert mit Z die Übergabelogik des Prozesswegweisers P_1.

Sei nun allgemein p ein Endprozesswegweiser und Z(p) seine Übergabelogik. Dann ist zu fordern, dass sich die durch Z(p) beschriebene Übergabelogik am Anfang der referenzierten EPK h(p) wiederfindet. Dazu fordern wir, dass h(p) eine Kante (x,y) enthält, die den Übergabeteil (der die Übergabelogik beschreibt) vom eigentlichen Ausführungsteil (der die Ablauflogik beschreibt) trennt. Für eine solche Kante gilt, dass der die EPK beschreibende Graph nach Entfernen der Kante nicht mehr zusammenhängend ist. (In der Graphentheorie wird hierfür die Bezeichnung cut-vertex verwendet.) Somit ist x das letzte Element des Übergabeteils und y das erste Element des Ausführungsteils.

Abb. 11 zeigt die obigen Bezeichnungen an einem Beispiel.

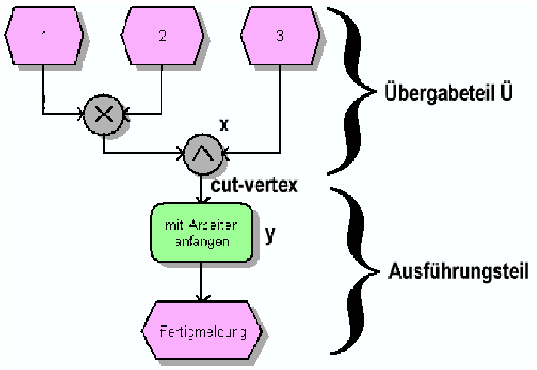

Abbildung 11: Cut Vertex trennt Übergabeteil von Ausführungsteil (Die Startprozesswegweiser wurden in dieser Abbildung weggelassen.)

Die Übergabelogik des Übergabeteils (Symbol: Z(h(p))) definieren wir wieder mit Hilfe des oben angeführten Algorithmus. Für die Anfangsinitialisierung wählen wir jetzt allerdings y, also das der cut-vertex folgende Element.

Um eine Korrespondenz zwischen ausgehendem Prozesswegweiser und Übergabeteil der referenzierten EPK zu gewährleisten, fordern wir dann, dass die logischen Formeln Z(p) und Z(h(p)) logisch identisch sind. Damit ist gewährleistet, dass die durch Ereignisse und Konnektoren beschriebenen Vorbedingungen für die Ausführung eines Prozesswegweisers mit den für die Abarbeitung der referenzierten EPK nötigen Bedingungen übereinstimmen. Es ist zu beachten, dass wir nicht die Gleichheit der Zeichenketten Z(p) und Z(h(P)) fordern. Es reicht die logische Identität. Für den in Abb. 9 beschriebenen Fall mit $Z(P_1) = (1 \text{ xor } 2) \wedge 3)$ wäre also z.B. durchaus ein Übergabeteil der referenzierten EPK möglich, der die Übergabelogik $Z(h(P_1)) = (1 \wedge 3) \text{ xor } (2 \wedge 3)$ besitzt.

Wir formulieren nun die folgende Forderung:

F 10 Hat eine EPK e_1 einen Endprozesswegweiser p_1 vom Typ 2, und sei $Z(p_1)$ die Übergabelogik von p_1. Dann muss die referenzierte EPK $e_2 = h(p_1)$ eine cut-vertex (x,y) mit der folgenden Eigenschaft besitzen:

Bezeichnen wir den Teilgraph von e_2, der x enthält, als Übergabeteil Ü, dann gilt:

Ü ist zyklenfrei und enthält keine Funktionen. Für die Übergabelogiken gilt, dass $Z(p_1)$ und Z(y) logisch identisch sind. Ferner trifft einer der beiden folgenden Fälle zu:

(a) Jedes Startereignis s in Ü ist hat als Vorgänger einen Startprozesswegweiser $\bullet s = p_i$ mit $h(p_i) = e_1$. Darüberhinaus enthält Ü keine Splits.

(b) In Ü gibt es genau einen Startprozesswegweiser vom Typ 1. Dieser referenziert auf e_1. Ferner ist Ü ein sauber geschachtelter Kontrollblock.[4]

Für die in Abb. 9 gezeigte EPK sind in Abb. 12 Beispiele für referenzierte EPKs zu sehen, die den angeführten Regeln entsprechen. Das „Flachklopfen“ besteht in beiden Fällen darin, in der gerufenen EPK den Übergabeteil Ü zu entfernen und das der cut-vertex folgende Element an Stelle des rufenden Prozesswegweisers zu setzen.

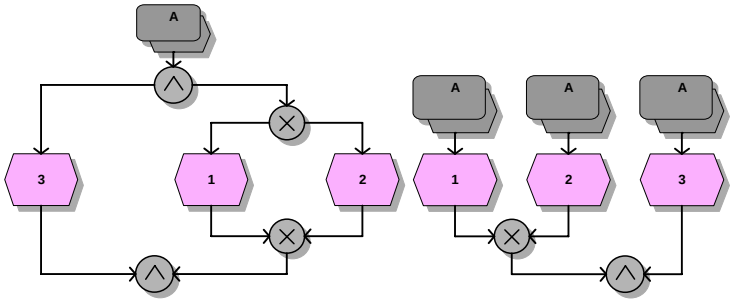

Abbildung 12: Passende EPK zu Abb. 9, links: Startprozesswegweiser haben Typ 1, rechts: Startprozesswegweiser haben Typ 2

7.3 Zusammenfassung der Prozesswegweiser-Forderungen

In den beiden vorangehenden Unterkapiteln haben wir für Endprozesswegweiser vom Typ 1 und vom Typ 2 Forderungen aufgestellt, denen die referenzierte EPK genügen soll. Wir haben ferner beschrieben, wie die EPKs an den Prozesswegweisern zusammengefügt werden können, wenn unsere Forderungen erfüllt sind.

Der Vollständigkeit halber erwähnen wir noch, dass natürlich auch zu fordern ist, dass es zu jeder EPK mit Startprozesswegweiser auch eine zugehörige rufende EPK gibt:

[4] „Sauber geschachtelt“ heißt hier: Splits und Joins treten paarweise auf und sind vom gleichen Typ, eine formale Definition findet sich z.B. in [LSW97a].

F 11 Enthält eine EPK e_1 einen Anfangsprozesswegweiser p mit $h(p) = e_2$, so muss e_2 einen Endprozesswegweiser q mit $h(q) = e_1$ enthalten.

Die von uns beschriebenen Restriktionen an die Verwendung von Prozesswegweisern stellen sicher, dass die zu verbindenden EPKs zu einer EPK zusammengefügt werden können. Unserer Erfahrung nach entsprechen die meisten in der Praxis anzutreffenden EPKs mit Prozesswegweisern den genannten Restriktionen. Es gibt jedoch durchaus Fälle, wo sinnvoll modellierte EPKs mit Prozesswegweisern unsere Restriktionen nicht erfüllen, z.B. sind Konstruktionen wie in Abb. 13 durch die von uns beschriebenen Fälle noch nicht abgedeckt. Es gibt also noch zahlreiche Möglichkeiten zur Erweiterung der in diesem Beitrag beschriebenen Überlegungen. Auf Grund der Komplexität selbst zunächst einfach scheinender Fälle haben wir zunächst darauf verzichtet, alle theoretisch nur denkbaren Konstruktionen zu erfassen.

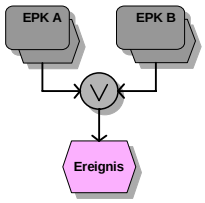

Abbildung 13: Noch nicht erfasster Fall

7.4 Hierarchisierte Funktionen

Nachdem wir im vorangegangenen Unterkapitel das Zusammenfügen von Prozesswegweisern untersucht haben, können wir nun auch das Flachklopfen hierarchisierter Funktionen mit Hilfe der dort beschriebenen Methode erklären.

Wie in Abb. 14 gezeigt, überführen wir den Aufruf einer hierarchisierten Funktion in einen Aufruf via Prozesswegweiser. Dazu setzen wir in der rufenden EPK einen Endprozesswegweiser an Stelle der hierarchisierten Funktion. In der referenzierten EPK setzen wir vor jedes Startereignis einen Startprozesswegweiser, der auf die rufende EPK referenziert.

Die Prüfung der Korrespondenz zwischen einer EPK mit hierarchisierter Funktion und der durch sie referenzierten Verfeinerung erfolgt dann wie zuvor für Prozesswegweiser beschrieben.

Analog lässt sich die Prüfung der Korrespondenz beim „Rücksprung“ von der Verfeinerung an die rufende EPK durchführen.

Die an Prozesswegweiser gestellten Anforderungen garantieren, dass für eine hierarchisierte Funktion f zwischen den Ereignissen in VE(f) und f keine Splits sowie zwischen f und den Ereignissen in NE(f) keine Joins vorkommen, was Situationen wie in Abb. 7 gezeigt vermeidet.

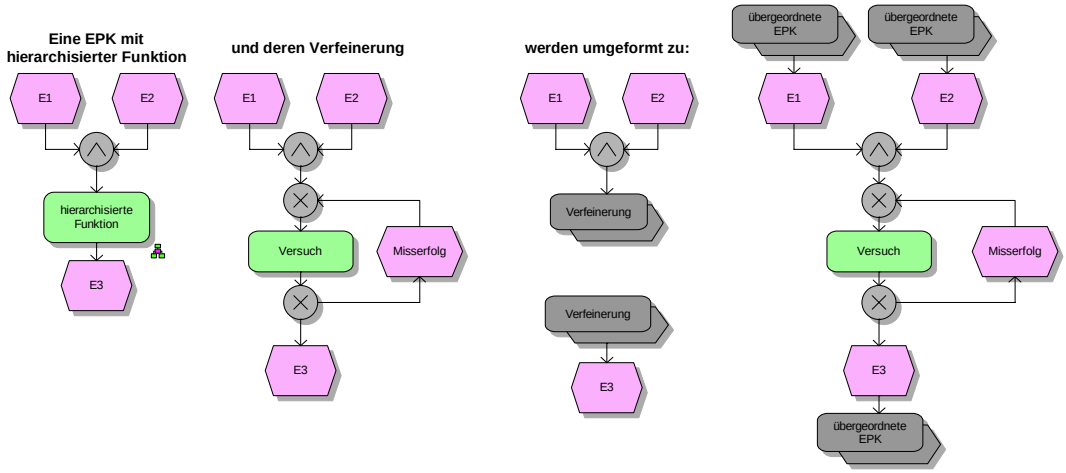

Abbildung 14: Ersetzten des Aufrufs einer Verfeinerung durch Prozesswegweiser

8 Weiterführende Fragestellungen

Wir haben bisher schon an einigen Beispielen gezeigt, dass es beim „Flachklopfen“ hierarchisierter EPKs verschiedene unschöne Situationen geben kann, die in der Literatur bisher nicht benannt wurden. Da das Thema komplexer ist, als man auf den ersten Blick vermutet, halten wir es für durchaus wahrscheinlich, dass noch weitere Schwierigkeiten gefunden werden.

Wie soll beispielsweise eine Situation behandelt werden, in der mehrere hierarchisierte Funktionen einer EPK auf die selbe Verfeinerung verweisen? Während es im linken Modell in Abb. 15 keine Probleme geben dürfte, resultiert das rechte Modell darin, dass die Verfeinerung mehrfach parallel auszuführen wäre, was in der Praxis sicher zu Ressourcenkonflikten führen würde. Ähnliche Fragestellungen sind für Prozesswegweiser denkbar. Hier wäre zunächst die Frage zu beantworten, welche Semantik ein mehrfaches Vorkommen einer Funktion oder eines Ereignisses in einer EPK haben soll. Obwohl solche Modelle in der Praxis nicht selten sind und z.B. auch durch das Austauschformat EPML[MN04] ausdrücklich unterstützt werden, ist diese Frage bisher nicht formal beantwortet.

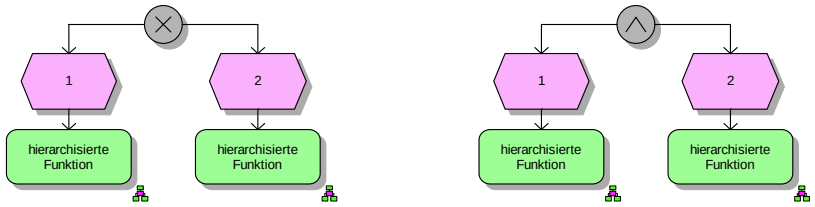

Abbildung 15: Mehrfaches Vorkommen einer Verfeinerung

Eine weitere wünschenswerte Forderung könnte es sein, dass man für jede EPK erkennen sollte, welche Startbelegungen gültig sind. Dies wäre z.B. für das in Abb. 16 gezeigte Modell nicht der Fall, da man hier erst an der Übergabelogik der referenzierten EPK 2 erkennen kann, ob die Ereignisse A und B gleichzeitig auftreten können bzw. müssen.

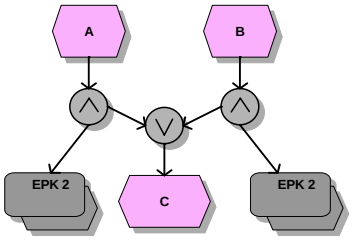

Abbildung 16: Welche Startbelegungen sind hier erlaubt?

9 Zusammenfassung

In unserem Beitrag haben wir syntaktische Anforderungen an EPKs mit hierarchisierten Funktionen und Prozesswegweisern präzisiert. Wir haben gezeigt, dass das „Flachklopfen“ mehrerer EPKs einer EPK-Schemamenge, das in der Literatur bisher immer als problemlos dargestellt wurde, zu verschiedenen unschönen Schwierigkeiten führen kann. Durch die Einführung von Restriktionen für hierarchisierte Funktionen und Prozesswegweiser beschrieben wir eine Klasse von EPKs, für die ein Flachklopfen automatisiert möglich ist. Allerdings stellen wir nicht die Behauptung auf, mit unseren Restriktionen den denkbar allgemeinsten Fall zu beschreiben, in dem ein Zusammenfassen von Modellen einer EPK-Schemamenge zu einem flachen EPK-Schema möglich ist. Die in diesem Beitrag beschriebenen Forderungen sind durchaus subjektiver Natur und sollen vor allem den Ausgangspunkt für weitere wissenschaftliche Diskussionen liefern.

Literatur

[CS94] R. Chen und A.W. Scheer. Modellierung von Prozessketten mittels Petri-Netz-Theorie. *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, (107), 1994.

[DR01] Juliane Dehnert und Peter Rittgen. Relaxed Soundness of Business Processes. In *CAiSE '01: Proceedings of the 13th International Conference on Advanced Information Systems Engineering*, Seiten 157–170, London, UK, 2001. Springer-Verlag.

[Kel99] Gerhard Keller. *SAP R/3 prozessorientiert anwenden*. Addison-Wesley, München, 1999.

[Kin04] Ekkart Kindler. On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. In *Business Process Management*, Seiten 82–97, 2004.

[KNS92] G. Keller, M. Nüttgens und A.W. Scheer. Semantische Prozessmodellierung auf der Grundlage Ereignisgesteuerter Prozessketten (EPK). *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, (89), 1992.

[LSW97a] P. Langner, C. Schneider und J. Wehler. Ereignisgesteuerte Prozessketten und Petri-Netze. *Berichte des Fachbereichs Informatik der Universität Hamburg*, (106), 1997.

[LSW97b] P. Langner, C. Schneider und J. Wehler. Prozessmodellierung mit ereignisgesteuerten Prozessketten (EPKs) und Petri-Netzen. *Wirtschaftsinformatik*, 39(5):479–489, 1997.

[Men07] Jan Mendling. *Detection and Prediction of Errors in EPC Business Process Models*. Dissertation, Vienna University of Economics and Business Administration, 2007.

[MN04] J. Mendling und M. Nüttgens. Exchanging EPC Business Process Models with EPML. In M. Nüttgens und J. Mendling, Hrsg., *XML4BPM 2004, Proceedings of the 1st GI Workshop XML4BPM – XML Interchange Formats for Business Process Management at 7th GI Conference Modellierung 2004, Marburg Germany, March 2004*, Seiten 61–80, March 2004.

[NR02] Markus Nüttgens und Frank J. Rump. Syntax und Semantik Ereignisgesteuerter Prozessketten (EPK). In *Promise 2002 - Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen*, Seiten 64–77, 2002.

[Rum99] Frank J. Rump. *Geschäftsprozeßmanagement auf der Basis ereignisgesteuerter Prozeßketten*. B. G. Teubner Verlag Stuttgart Leipzig, 1999.

[van99] Wil M.P. van der Aalst. Formalization and verification of event-driven process chains. *Information & Software Technology*, 41(10):639–650, 1999.