Einfache EPK-Semantik durch praxistaugliche Stilregeln

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

Abstract: Bekannte Ansätze zur Semantikdefinition von EPKs beschreiten zwei grundsätzlich unterschiedliche Wege, um die bestehenden Probleme (insbesondere mit nichtlokalen Konnektoren) zu lösen: Entweder die Klasse der „gültigen“ EPKs wird stark eingeschränkt oder ein komplexer Algorithmus berechnet die Semantik (oder stellt fest, dass für die vorliegende EPK keine vernünfige Semantik existiert).

Wir versuchen, einen Mittelweg zu finden und fragen, ob es wenige einfache Stilregeln für EPKs gibt, die eine einfache und eindeutige Semantikdefinition ermöglichen. Die Stilregeln sollen nicht unnötig streng sein. d.h. sie sollen möglichst viele in der Praxis vorhandenen Modelle abdecken.

Wir schlagen einige einfache Stilregeln vor, die die o.g. Anforderungen erfüllen. An 285 EPK-Modellen, die wir aus verschiedensten Quellen gewonnen haben, wurde geprüft, ob das Modell die Regeln einhält. Diese Untersuchung zeigte, dass in den Fällen, in denen die Stilregeln verletzt waren, tatsächlich auch das Modell einer Verbesserung bedurfte.

1 Einführung

Ereignisgesteuerte Prozessketten (EPKs) wurden ursprünglich als eine nicht vollständig formalisierte Notation entwickelt, um Geschäftsprozesse zu modellieren. Dies reicht zwar aus, um Prozesse zu dokumentieren und über die Modelle zu diskutieren, für eine automatische Verarbeitung von EPK-Modellen (etwa in Werkzeugen zur Simulation oder Verifikation) muss jedoch eine formale Semantik der Modelle definiert werden.

Hierfür gibt es verschiedene Ansätze, die wir in Abschnitt 2 besprechen. Dort stellen wir auch die Probleme dar, die es bei der Definition einer EPK-Semantik gibt und geben an, wie diese von den verschiedenen Autoren gelöst wurden.

Im Abschnitt 3 werden die bekannten Ansätze bewertet. Insbesondere stellen wir die Frage, wie gut die verschiedenen Ansätze mit den in der Praxis vorhandenen (mitunter wenig strukturierten) Modellen arbeiten können. In Abschnitt 4 schlagen wir Stilregeln für „gut modellierte EPKs“ vor. EPKs, die diesen Regeln folgen, lassen sich in Petri-Netze übersetzen, was kurz in Abschnitt 6 besprochen wird. In Abschnitt 5 zeigen wir häufige Fehler,

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

die zur Verletzung der Stilregeln führen und deren (automatisch durchführbare) Korrektur. Im Abschnitt 7 werten wir 285 aus den verschiedensten Quellen gesammelte EPKs aus und überprüfen an diesen die Praxistauglichkeit unserer vorgeschlagenen Stilregeln.

2 Semantikdefinitionen für EPKs und auftretende Probleme

EPKs wurden ursprünglich eingeführt und verwendet, ohne ihre Semantik formal zu definieren. Demgegenüber bieten Petri-Netze, deren Semantik klar definiert ist, ebenso die Möglichkeit, Abläufe mit Parallelität und Entscheidungsknoten zu beschreiben. Die große Zahl bekannter Forschungsergebnisse aus dem Bereich der Petri-Netze sowie die gute Werkzeug-Unterstützung etwa zur Simulation von Petri-Netzen legen es nahe, zur Definition der Semantik einer EPK diese in ein Petri-Netz zu übersetzen.

Solche Übersetzungen wurden von verschiedenen Autoren vorgeschlagen, darunter van der Aalst[Aal99], Chen/Scheer[CS94], Langner, Schneider und Wehler[LSW97a, LSW97b] und Dehnert/van der Aalst[DA04]. Bei diesen Übersetzungen müssen jeweils Einschränkungen gemacht werden, die wir in den folgenden Unterabschnitten besprechen werden.

Einen anderen Weg beschreiten die Autoren Kindler und Cuntz[Kin04, Cun04, CK04]. Sie beschreiben die möglichen Abläufe eines EPK-Modells mit Hilfe eines Transitionssystems und geben ein Verfahren an, mit dem dessen Transitionsrelation bestimmt werden kann. Dieses Verfahren ist im dem Programm EPCtools implementiert, das zudem einige algorithmische Tricks benutzt, um die Berechnung auch für große EPKs effektiv ausführen zu können.

Der Grund dafür, dass über Jahre hinweg immer wieder neue Vorschläge zur Definition einer Semantik gemacht wurden, liegt im Wesentlichen in den Problemen begründet, die in den beiden folgenden Unterabschnitten besprochen werden: der Nichtlokalität von Verknüpfungskonnektoren und der fehlenden Unterstützung mehrfacher Instanziierung.

2.1 Probleme durch nichtlokale Konnektoren

Abbildung 1 zeigt ein typisches OR-Konstrukt. Es könnte einer EPK, die die Auswahl von Bewerbern für eine Stelle eines wissenschaftlichen Mitarbeiters beschreibt, entnommen sein. Nach dem OR-Split werden eine, zwei oder alle der dargestellten Aktivitäten parallel ausgeführt, und der Ablauf wird nach dem OR-Join erst fortgesetzt, wenn alle begonnenen Aktivitäten beendet sind. Bei einer Übersetzung der EPK in ein Petri-Netz ergibt sich nun die Frage, wie der OR-Join-Konnektor in ein Petri-Netz-Konstrukt zu übersetzen ist. Das Problem hierbei ist, dass am OR-Join nicht bekannt ist, wie viele parallele Abläufe am OR-Split gestartet wurden, d.h. auf wie viele vom OR-Split gesendete Tokens gewartet werden soll. Diese Frage lässt verschiedene Antwortmöglichkeiten zu, die unter anderem in [Aal99], [Rit00] und [WEAtH05] diskutiert werden. Meist wird die Frage so beantwortet, dass der OR-Join erst dann Kontrollfluss-Tokens („Prozessordner“) weiterreicht, wenn an einem ankommenden Zweig ein Kontrollfluss-Token anliegt und für alle anderen

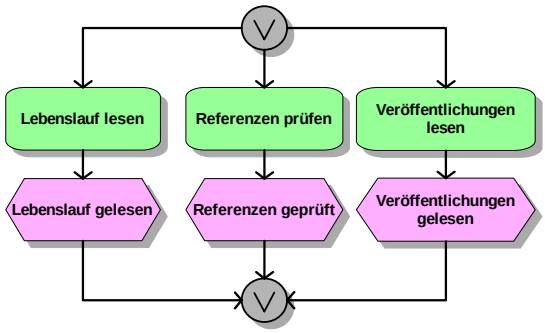

Abbildung 1: OR-Konstrukt

ankommenden Zweige bestimmt werden kann, ob noch weitere Tokens zu erwarten sind. Diese Entscheidung erfordert aber in der Regel nicht nur die Kenntnis der lokalen Situation direkt vor dem OR-Join, sondern eine Analyse des gesamten EPK-Modells. Daher wird der OR-Join-Konnektor als *nichtlokaler Konnektor* bezeichnet.

Eine ähnliche Situation finden wir bei XOR-Join-Konnektoren, deren Zweck darin besteht, alternative Kontrollflüsse zusammenzuführen. Liegt ein Token an genau einem der Eingänge eines XOR-Joins an, wird es an den Ausgang des XOR-Joins weitergeleitet. Wie die Semantik eines XOR-Joins ist, wenn an mehr als einem Eingang Kontrollfluss-Tokens eintreffen, wird unterschiedlich beantwortet. Während bei [Aal99] und [DA04] in diesem Falle die ausgehende Kontrollflusskante mehrfach durchlaufen wird, gehen andere Ansätze [ADK02, Kin04, Cun04] davon aus, dass der Ablauf blockiert wird. Folgt man dieser Interpretation, erhält auch der XOR-Join eine nichtlokale Semantik, da vor einem Weitergeben von Tokens stets zu prüfen ist, an welchen Eingängen noch Tokens ankommen könnten.

[ADK02] liefert das zentrale Ergebnis der Betrachtung nichtlokaler Konnektoren: Es ist unmöglich, eine befriedigende Semantik für EPKs anzugeben, die mit der informellen Semantik (mit nichtlokalen Konnektoren) übereinstimmt. Dies wird an einem Gegenbeispiel, bei dem zwei XOR-Joins jeweils darauf warten, dass der andere zuerst schaltet, gezeigt.

Für die Definition einer Semantik nichtlokaler Konnektoren wurden verschiedene Ansätze vorgeschlagen.

[Aal99] betrachtet nur solche EPKs, die keine OR-Joins enthalten und geht außerdem davon aus, dass XOR-Joins mehrfach schalten, wenn an mehreren eingehenden Kontrollflusskanten Kontrollfluss-Tokens ankommen. Dadurch gibt es keine nichtlokalen Konnektoren mehr, und das EPK-Modell lässt sich leicht in ein Petri-Netz übersetzen.

[LSW97a] und [CS94] stellen zusätzliche Forderungen zur Wohlgeformtheit von EPKs, insbesondere zur sauberen Verschachtelung von Prozessblöcken. Für Modelle, die diese erfüllen, ist eine Übersetzung von EPKs in (markierte) Petri-Netze möglich.

Einen grundsätzlich anderen Weg beschreiten [Kin04] und [Cun04]. Hier untersucht ein relativ komplexer Algorithmus alle möglichen Abläufe einer EPK, um deren Semantik zu

berechnen. Ist diese Berechnung erfolgreich, so stimmt sie mit der intuitiven Vorstellung einer EPK-Semantik mit nichtlokalen Konnektoren überein. Es ist aber durchaus auch möglich, dass der Algorithmus das Resultat „keine Semantikdefinition möglich“ liefert (was nach [ADK02] auch immer so sein muss).

[WEAtH05] behandelt das Problem nichtlokaler OR-Konnektoren im Kontext der Sprache YAWL[AH02], die Ergebnisse lassen sich auch auf EPKs übertragen. Hier werden Reset-Netze, eine spezielle Variante von Petri-Netzen, als formale Grundlage benutzt. Eine Entscheidung, ob noch weitere Tokens eintreffen können, wird durch Rückwärtssuche im Zustandsraum getroffen.

Abweichend von diesen Ansätzen, betrachtet [DA04] OR- und XOR-Joins als Elemente mit lokaler Semantik. Obwohl dies - wie auch einer der Autoren von [DA04] an anderer Stelle[Aal99] schrieb - nicht den üblichen Vorstellungen einer EPK-Semantik entspricht, wurde diese Semantik erfolgreich angewendet, indem während der Analyse der EPK zusätzliche Informationen vom Modellierer erfragt wurden[vDAV05].

2.2 Probleme durch die Blockierung des Tokenflusses

Die üblichen EPK-Semantiken erlauben nicht die mehrfache Instanziierung von Ereignissen oder Funktionen.[1] Wird wie in [Rum99] der Zustand einer EPK durch die Zahl der Tokens („Prozessmappen“) auf den einzelnen Modellelementen (Ereignissen, Funktionen, Prozessegweisern und Konnektoren) definiert, kann es dennoch vorkommen, dass z.B. ein Kontrollfluss-Token eine Funktion erreicht, die bereits aktiv ist. Wie sich das Modell in diesem Falle verhalten soll, ist unklar.

Analoge Überlegungen gelten, wenn wir, wie in [Kin04] vorgeschlagen, einen Zustand einer EPK durch die Zahlen der Tokens auf den *Kontrollflusskanten* definieren. In der Regel wird hier davon ausgegangen, dass eine Kontrollflusskante, die schon durch ein Token belegt ist, „blockiert“, d.h. keine weitere Tokens annehmen kann.

Ausführlich wird diese Frage in [Cun04] analysiert, wo auch eine alternative Semantikdefinition (die eine Belegung einer Kontrollflusskante mit mehreren Tokens erlaubt) besprochen wird.

3 Diskussion der vorhandenen Lösungsansätze

Die bekannten Lösungsvorschläge für die genannten Probleme lassen sich grob in zwei Klassen einteilen.

In die eine Klasse fallen die Ansätze, die die „gültigen“ EPKs durch zusätzliche Regeln für Wohlgeformtheit beschränken. So betrachtet [Aal99] nur EPKs ohne OR-Join. [LSW97a] und [CS94] verlangen strukturierte EPK-Modelle (zu jedem Split-Konnektor gehört ein

[1] [MNN05] schlägt zwar vor, EPKs um zusätzliche Notationselemente für mehrfache Instanziierung zu erweitern, definiert jedoch dazu keine formale Semantik.

Join-Konnektor gleichen Typs, die Split/Join-Konstrukte sind sauber verschachtelt, außerdem fordert [LSW97a] zusätzliche Eigenschaften von per XOR-Konnektoren modellierten Iterationen).

Bei der Bewertung dieser Ansätze teilen wir die Einschätzung aus [vDAV05]: Sie sind von einem theoretischem Standpunkt aus interessant und wichtig, aus der Sicht eines Praktikers aber oft weniger nützlich. Die Klasse der betrachteten EPKs wird durch zusätzliche Wohlgeformtheits-Regeln eingeschränkt, die vorrangig mit dem Ziel formuliert wurden, eine elegante Übersetzung wohlgeformter EPKs in Petri-Netze (oder andere Formalismen) zu erreichen. Viele in der Praxis modellierte EPKs sind nach diesen restriktiven Regeln nicht wohlgeformt und können demnach nicht übersetzt werden, selbst wenn Praktiker diese EPKs problemlos verstehen und einsetzen können.

Die Arbeiten [Kin04] und [WEAtH05][2] fallen in die andere Klasse. In beiden Fällen wird ein komplexer Algorithmus bemüht, um für *jede* denkbare EPK eine Semantik zu bestimmen (oder - wenn dies nicht möglich ist - festzustellen, dass es keine sinnvolle Semantik gibt). Selbstverständlich ist das die bestmögliche Lösung aus theoretischer Sicht. Wir meinen allerdings, dass man für praktische Belange auf die Verwendung solch komplexer Algorithmen verzichten kann, da es eigentlich gar nicht wünschenswert ist, für jede (auch noch so wirr modellierte) EPK eine Semantik zu bestimmen. Wenn (wie in [Cun04] erwähnt) ein Computerprogramm 20 Minuten benötigt, um eine Entscheidung über die Bedeutung einer EPK zu treffen, ist es sehr unwahrscheinlich, dass dieses Modell einem der Hauptanliegen der Geschäftsprozessmodellierung - der Möglichkeit, über ein gezeichnetes Modell zu diskutieren - gerecht werden kann. Daher sollte man ein solches Modell in jedem Falle durch ein einfacher modelliertes ersetzen.

Um die Nachteile beider Klassen zu vermeiden, haben wir uns die Frage gestellt, ob es möglich sei, wenige „Stilregeln für gut modellierte EPKs“ zu finden, die folgende Bedingungen erfüllen:

1. EPKs, die in der Praxis vorkommen und über deren Semantik sich Praktiker einig sind, sollten nicht durch zu strenge Einschränkungen ausgeschlossen werden.
2. Es soll möglich sein, für EPKs, die den Stilregeln entsprechen, eine Semantik (etwa als Übersetzung in Petri-Netze) anzugeben.
3. Die Stilregeln sollen Modellierern, die mit EPKs vertraut sind, nicht „unnatürlich“ vorkommen. Statt dessen sollen sie möglichst ohnehin schon (unbewusst) angewendet werden.
4. Die Einhaltung der Regeln soll leicht und automatisiert überprüft werden können.

Modelle, die nicht den Stilregeln entsprechen, nennen wir *unzulässig*. Solche Modelle sollen unserer Auffassung nach nicht benutzt werden. Insbesondere werden wir auch auf jeglichen Versuch verzichten, die Frage nach der Semantik für diese Modelle zu beantworten. Dass wir mit diesem Ansatz trotzdem nahezu keine Probleme bei in der Praxis anzutreffenden EPKs haben, zeigt unsere Untersuchung in Abschnitt 7.

[2] [WEAtH05] behandelt keine EPKs, sondern OR-Joins in der Sprache YAWL, die Ergebnisse lassen sich aber auf EPKs übertragen.

Im nächsten Abschnitt beschreiben wir einen Vorschlag für solche Stilregeln.

4 Stilregeln

4.1 Blockierung des Tokenflusses

EPKs sehen „von Haus aus“ keine mehrfache Instanziierung von Funktionen und Ereignissen vor. Eine mögliche Definition eines Zustands der EPK sagt aus, dass der Zustand durch die Zahlen der Tokens auf den Kontrollflusskanten bestimmt ist, ohne diese Zahl zu beschränken[Cun04]. Ist es dann in einem Modell möglich, dass ein Token eine Kontrollflusskante erreicht, die bereits durch ein Token belegt ist, lässt sich daraus (wenn beide Tokens synchron „weiterwandern“) eine Situation ableiten, in der nachfolgende Funktionen oder Ereignisse von beiden Tokens erreicht werden können. Dies würde einer mehrfachen Instanziierung entsprechen. Es ist also die Schlussfolgerung naheliegend, dass dieses Modell fehlerhaft ist.

Als erste Stilregel fordern wir demnach, dass eine „gut modellierte“ EPK garantiert, dass nie mehrere Tokens zugleich eine Kontrollflusskante erreichen. Andere EPKs betrachten wir als unzulässig. Die genannte Stilregel kann leicht mit einem Petri-Netz-Analysetool überprüft werden, wenn die EPK in ein Petri-Netz übersetzt wurde.

4.2 XOR-Joins

Verschiedene wissenschaftliche Veröffentlichungen zur EPK-Syntax wie [NR02] und [Kin04] gehen von einer nichtlokalen Semantik von XOR-Joins aus: Ein XOR-Join reicht Tokens genau dann weiter, wenn an genau einer eingehenden Kontrollfluss-Kante ein Token anliegt und vor Durchlaufen des XOR-Joins *an keiner weiteren eingehenden Kante weitere Tokens eintreffen können*. Treffen also mehrere Tokens am XOR-Join ein, blockiert dieser.

Unserer Erfahrung nach entspricht dieses Blockieren jedoch nicht den Vorstellungen, die Modellierer aus der Praxis vom Verhalten des XOR-Joins haben. Wir wählen daher die auch in [Aal99] und [DA04] vorgeschlagene *lokale* Semantik für XOR-Joins: Sobald ein Token den XOR-Join erreicht, wird dieses weitergeleitet. Das spiegelt den „Normalfall“ wider, den ein Modellierer bei der Benutzung eines XOR-Joins im Sinn hat: dass er sich darauf verlassen kann, dass ohnehin nur an einer eingehenden Kontrollflusskante des XOR-Joins ein Token eintrifft.

Ist nun das Modell so gestaltet, dass (mit dieser *lokalen* Semantik für XOR-Joins) an mehreren eingehenden Kontrollflusskanten Tokens an einem XOR-Join ankommen können, gelangen diese (da der XOR-Join die Tokens ja sofort weiterleiten darf) beide an die vom XOR-Join abgehende Kontrollflusskante. Gerade das führt aber nach Abschnitt 4.1 dazu, dass wir das Modell als unzulässig betrachten.

Modelle mit XOR-Joins mit unklarer (nichtlokaler) Semantik wie der in [ADK02] vorgestellte „Teufelskreis“ (vicious circle) werden durch auf diese Weise als unzulässige EPKs eingestuft, da (wenn XOR-Joins immer Tokens weiterleiten dürfen) mehrere Eingänge eines anderen XOR-Joins belegt werden können.

Bei der Untersuchung vorhandener EPK-Modelle (siehe Abschnitt 7) fanden wir *kein* Modell, dass inhaltlich korrekt war und nach dem beschriebenen Ansatz unnötigerweise als unzulässig eingestuft würde. Keines der von uns untersuchten EPK-Modelle macht bewusst von der Deutung Gebrauch, dass der Kontrollfluss an XOR-Joins, die von mehr als einem Token erreicht werden, blockiert wird. (Dies wäre auch ohnehin eine schlechte Modellierung.) Mehr noch: Wenn immer im Modell mehr als ein Token am XOR-Join ankommen konnte (wiederum unter Annahme einer *lokalen* Semantik für alle anderen XOR-Joins), zeigte eine genauere Analyse, dass das Modell tatsächlich fehlerhaft war. Diese beiden Tatsachen belegen unserer Ansicht nach die Berechtigung, eine lokale Semantik für XOR-Joins zu betrachten und wie beschrieben gewisse Modelle als unzulässig einzustufen.

4.3 OR-Join-Konnektoren

OR-Joins sind das EPK-Notationselement, über dessen semantische Bedeutung am meisten diskutiert wurde[Aal99, LSW97a, Rit00, WEAtH05].

Die informelle Semantik für OR-Joins, die eindeutig einem OR-Split zugeordnet sind, lautet: „Warte, bis alle vom OR-Split losgeschickten Tokens eingetroffen sind und leite das Token dann an die abgehende Kontrollflusskante weiter“. Es ist dann u.a. ein Weg zu finden, wie der OR-Join erfährt, auf welche Tokens er warten muss. [LSW97a] beschreibt hierfür eine Lösung mit Hilfe boolescher Netze (also 0/1-markierter Petri-Netze). Der OR-Split sendet mit 1 markierte Tokens auf den Kontrollflusskanten, auf denen Kontrollfluss stattfinden soll und mit 0 markierte Tokens auf den anderen „nicht aktivierten“ Kanten. Der OR-Join wartet nun, bis an allen eingehenden Kontrollflusskanten ein Token eintrifft. Ist mindestens eines davon ein 1-Token , dann wird der Kontrollfluss an die vom OR-Join abgehende Kante weitergeleitet.

Unglücklicherweise ist für diese elegante Lösung ein hoher Preis zu bezahlen: [LSW97a] betrachtet nur EPKs, die sehr strengen Wohlgeformtheits-Kriterien entsprechen. Ein erheblicher Teil der in der Praxis vorkommenden EPKs mit OR-Joins entspricht diesen Kriterien nicht. Somit ist [LSW97a] mit seinen strengen Kriterien ein gangbarer Weg, wenn man Regeln für Modellierer in einem neuen Projekt festschreiben kann. Liegen aber (wie oft anzutreffen) schon EPKs vor, erfüllen diese meist nicht die Kriterien und können folglich nicht analysiert werden.

Ausgehend von den von uns untersuchten EPKs haben wir uns nun die Frage gestellt, ob auch weniger strenge (und damit mehr praxisnahe) Stilregeln für EPKs ausreichen, um den Ansatz von [LSW97a], boolesche Netze für die Modellierung von OR-Konstrukten zu verwenden, zu ermöglichen.

Tatsächlich war es möglich, solche Stilregeln aufzustellen. Hierzu definieren wir zunächst,

was wir unter einem *wohlstrukturierten Konstrukt* verstehen wollen. (Wir abstrahieren hier von Funktionen und Ereignissen in der EPK, da die kritischen Elemente allein die Konnektoren sind. Funktionen und Ereignisse sind gemäß den in [NR02] gestellten Forderungen einzusetzen.)

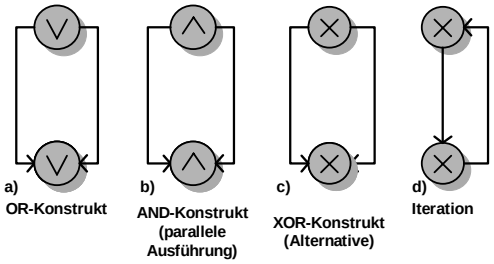

Abbildung 2: Workflow-Konstrukte

Definition 1 *(wohlstrukturierte Konstrukte):*

1. *Die in Abb. 2 gezeigten Workflow-Konstrukte sind wohlstrukturiert. In den Abb. 2 a)-c) sind auch mehr als zwei Kontrollflusskanten zwischen Split- und Join-Konnektor zulässig.*
2. *Wird in eine Kante eines wohlstrukturierten Konstruktes ein weiteres wohlstrukturiertes Konstrukt eingesetzt (siehe Abb. 3), ist das erhaltene Konstrukt wiederum wohlstrukturiert.*
3. *Wird ein „Aussprung“ (siehe Abb. 4, der XOR-Konnektor kann auch durch OR oder AND ersetzt werden) in eine Kante eines wohlstrukturierten Konstruktes eingesetzt, ist das erhaltene Konstrukt wiederum wohlstrukturiert.*
4. *Wird eine Kante eines wohlstrukturierten Konstrukts unterbrochen und durch ein Endereignis beendet (siehe Abb. 5), ist das erhaltene Konstrukt wiederum wohlstrukturiert, wenn der erhaltene Graph noch zusammenhängend ist.*

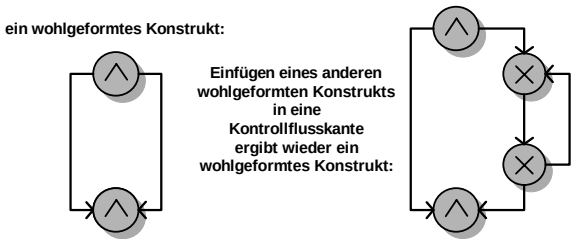

Abbildung 3: Definition, Regel 2

Die ersten beiden Regeln gewährleisten wie auch bei [LSW97a] gefordert, dass zu jedem Join-Konnektor ein zugehöriger Split-Konnektor gleichen Typs gehört. Die Regeln 3 und

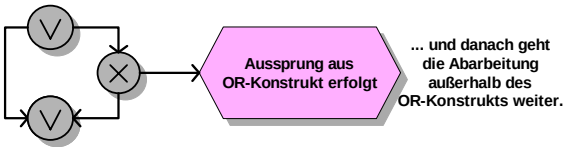

Abbildung 4: Definition, Regel 3

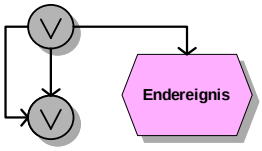

Abbildung 5: Definition, Regel 4

4 tragen der in der Praxis verbreiteten Gewohnheit Rechnung, den Kontrollfluss durch „Aussprünge“ aus einem Split/Join-Konstrukt zu unterbrechen oder durch ein Endereignis ganz abzubrechen.[3]

Wir betrachten nun nur solche EPKs als *zulässig*, für die alle OR-Joins ein wohlstrukturiertes Konstrukt entsprechend Def. 1 beenden. Insbesondere muss dieses Konstrukt dann mit einem OR-Split beginnen, d.h. jedem OR-Join muss ein OR-Split zugeordnet sein. Die Klasse der lt. unserer Definition zulässigen EPKs ist größer als die Klasse der in [LSW97a] betrachteten EPKs, da zum einen weniger strenge Anforderungen an XOR-Konnektoren in Iterationen gestellt werden, zum anderen die Regeln 3 und 4 in Def. 1 hinzukommen. Im Abschnitt 7 werden wir sehen, dass unsere Definition für zulässige EPKs nahezu keine der von uns gesammelten 285 in der Praxis vorkommenden EPKs unnötigerweise als „unzulässig“ ausschließt.

Im Gegensatz zu den in 4.1 und 4.2 genannten Stilregeln, zu deren Test eine Verhaltensanalyse des Modells nötig ist, lassen sich die Stilregeln für OR-Konstrukte mittels statischer Analyse der EPK nachweisen. Dies kann ähnlich zu dem aus [LSW97a] bekannten Vorgehen geschehen, wobei jedoch „Aussprünge“ aus Split/Join-Konstrukten und an beliebigen Stellen erlaubte Endereignisse zusätzliche Überlegungen erfordern.

5 Häufige Fehler und deren Korrektur

[LSW97a] nennt zahlreiche Modellierungsfehler, die teilweise mittels statischer Analyse einer EPK gefunden werden können. Eine solche Analyse müssen wir nach unserem Ansatz ohnehin durchführen, um die Gültigkeit der in Def. 1 aufgeführten Regeln zu prüfen. Bei dieser Gelegenheit können einige typische Modellierungsfehler gleich erkannt und verbessert werden.

Wir haben bei den von uns gesammelten EPKs typische Fehler an OR-Joins analysiert. Oft wurden OR-Joins genutzt, wenn ein XOR- oder AND-Join angebracht gewesen wäre (siehe Abb. 6a)-c), wo wieder Funktionen und Ereignisse weggelassen sind.) Ebenso wurde oft die optionale Ausführung fehlerhaft modelliert (siehe Abb. 6d)). Natürlich ändert sich bei allen in Abb. 6 gezeigten Änderungen nicht die Semantik der EPK, vom theoretischen Standpunkt aus sind die Korrekturen sogar unnötig. Wir bezeichnen diese Modelle trotz-

[3] Man kann natürlich einwenden, dass entsprechend der genannten Gewohnheiten modellierte EPKs schlecht modelliert sind, da ein Endereignis erreicht werden kann, wenn noch Synchronisationen ausstehen. Die Zielrichtung dieser Veröffentlichung ist aber eine andere: Wir nehmen zur Kenntnis, welche Modelle in der Praxis vorhanden sind und versuchen, diese so gut wie möglich zu analysieren.

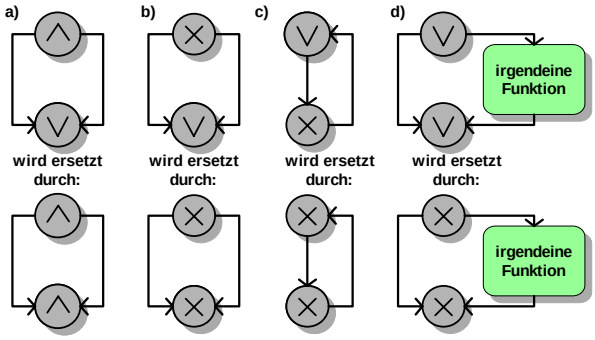

Abbildung 6: Korrektur typischer Fehler

dem bewusst als fehlerhaft, da die geänderten Modelle den zu modellierenden Sachverhalt offensichtlich besser wiedergeben. Da ein Hauptzweck von EPKs oft ist, über das Modell zu kommunizieren, sollten die „schlechten“ Konstrukte durch die besseren Varianten ersetzt werden, um die Gefahr von Missverständnissen zu verringern.

6 Übersetzung in Petri-Netze

Um nun EPKs, die unseren Stilregeln entsprechen und somit gültig sind, in Petri-Netze zu übersetzen, können wir die in [LSW97a] angegebene Übersetzung übernehmen. Zusätzlich müssen Iterationen (Abb. 2d)) entsprechend der Semantik der XOR-Konnektoren behandelt werden, und es muss dafür gesorgt werden, dass „Aussprünge“, die ein Split-Join-Konstrukt vorzeitig beenden (vgl. Abb. 4) 0-Tokens an den nachgeordneten OR-Join senden (und dieser die 0-Tokens ggf. an weitere nachgeordnete OR-Joins weiterreicht). Beide Erweiterungen des Übersetzungsalgorithmus aus [LSW97a] sind recht einfach, denn wir wissen nach der statischen Analyse zu der in Def. 1 gegebenen Stilregel, ob ein XOR-Split die Situation aus Abb. 2c) oder d) oder einen „Aussprung“ wie in Abb. 4 darstellt.

Tab. 1 fasst die wesentliche Ergänzung zu der in [LSW97a] angegebenen Übersetzung zusammen. Sie stellt für alle möglichen Split-Konnektoren dar, wie mit 0 und 1 markierte Tokens an die ausgehenden Kontrollflusskanten (symbolisiert durch die Variablen x und y) weitergeleitet werden, wenn auf der eingehenden Kontrollflusskante (symbolisiert durch die Variable z) ein mit 0 oder 1 markiertes Token eintrifft. Die Angabe „0“ oder „1“ in der entsprechenden Spalte besagt, dass ein entsprechend markiertes Token gesendet wird; ein Strich zeigt an, dass kein Token gesendet wird. Im Fall „xor-Split in Iteration“ ist die Variable x der Kontrollflusskante zugeordnet, die die Schleife beendet, y der anderen. In den Fällen „. . . -Aussprung“ ist x der Kontrollflusskante zugeordnet, die im wohlstrukturierten Konstrukt liegt, und y der Kontrollflusskante , die „herausspringt“. Alle anderen Bezeichnungsweisen und das generelle Vorgehen entsprechen dem in [LSW97a] angegebenen.

	and-Split		xor-Split		or-Split		xor-Split in Iteration		xor-Aussprung		or-Aussprung		and-Aussprung	
Abb.	2a)		2b)		2c)		2d)		4		analog 4		analog 4	
z	x	y	x	y	x	y	x	y	x	y	x	y	x	y
0	0	0	0	0	0	0	0	-	0	-	0	-	0	-
1	1	1	1 0	0 1	1 0 1	0 1 1	- 1	1 -	0 1	1 -	0 1 1	1 - 1	1	1

Tabelle 1: Schaltungen an Split-Verknüpfern

7 Untersuchung von EPK-Modellen

Die Zielstellung dieser Arbeit lag darin, Stilregeln für EPKs zu finden, die einerseits eine korrekte Übersetzung der EPKs in Petri-Netze erlauben, andererseits jedoch die Klasse der gültigen EPKs nicht so stark einschränken, dass der Ansatz für viele EPKs aus der Praxis unbrauchbar ist.

Um dies zu überprüfen, haben wir EPKs aus allen uns zugänglichen Quellen zusammengesucht, insgesamt 285 Stück. Die Quellen hierfür waren 23 Diplomarbeiten, 2 Seminararbeiten, 4 Promotionen, 5 Bücher (eines davon mit zahlreichen Beispielen des SAP-Referenzmodells), 30 wissenschaftliche Veröffentlichungen, Kursfolien einer Lehrveranstaltung zum Thema „EPKs“ sowie eines unserer eigenen Projekte, bei dem Abläufe bei einem Energieversorger modelliert wurden. Die komplette Quellenliste ist auf [Lau05] nachzulesen. EPKs mit kleinen syntaktischen Fehlern haben wir toleriert; 9 Modelle, die in unseren Quellen als „EPK“ bezeichnet wurden, hatten jedoch solch gravierende syntaktische Fehler (etwa Aktivitäten, von denen mehrere Kontrollflusskanten ausgehen), dass wir sie beim besten Willen nicht mehr als EPK ansehen konnten und sie daher für die weitere Analyse ignorieren mussten.

Für die verbleibenden 276 EPK-Modelle fanden wir folgendes heraus:

- *Keines* der Modelle benutzt bewusst die Interpretation, dass zwei an einem XOR-Join zugleich ankommende Tokens den Ablauf blockieren. Wann immer in einem Modell mehr als ein Token zugleich am XOR-Join ankommen konnte, war das Modell *klar erkennbar fehlerhaft*. Damit ist die in Abschnitt 4.2 vorgeschlagene lokale Semantik von XOR-Joins bei gleichzeitigem Verbot von EPKs, bei denen mehrere Tokens zugleich einen XOR-Join erreichen, sinnvoll.
- 190 EPKs verwendeten keine OR-Joins.
- Die verbleibenden 86 EPKs enthielten insgesamt 151 OR-Joins. 94 davon entsprachen den Stilregeln aus Abschnitt 4.3.

Das eigentlich interessante Resultat ergab sich nun bei der („in Handarbeit“ durchgeführten) Untersuchung der 57 OR-Joins, die nicht den Stilregeln aus Abschnitt 4.3 entsprachen.

Auf 45 davon traf einer der in Abb. 6 gezeigten Fälle zu, d. h. sie sollten besser durch einen anderen Join-Konnektor ersetzt werden. Wie schon erwähnt, kann diese Ersetzung automatisch erfolgen.

Für 10 weitere EPKs, deren OR-Joins nicht den Stilregeln entsprachen, ergab eine genauere Betrachtung, dass diese offensichtlich fehlerhaft waren.[4]

Wir fanden nur zwei EPKs, die nicht den Stilregeln entsprechen und trotzdem als korrekt modelliert angesehen werden könnten. Beide benutzten das in Abb. 7 gezeigte Konstrukt. Da in diesem Konstrukt ein Token am AND-Join „steckenbleibt“, wenn nur eine der Ausnahmen eintritt (was übrigens dann auch verbietet, dieses Konstrukt in einer Schleife mehrfach durchlaufen zu lassen), sollte eine solche Modellierung vermieden werden, so dass eine Einstufung des Modells als „unzulässig“ durchaus vertretbar ist.

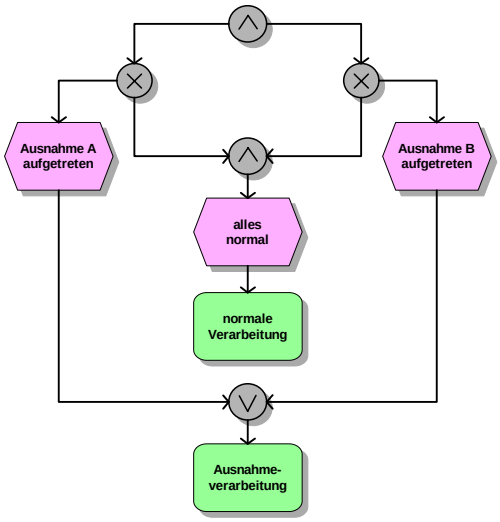

Abbildung 7: EPK, die unsere Stilregeln verletzt

8 Zusammenfassung

Die im letzten Kapitel genannten Zahlen belegen, dass unsere Stilregeln von nahezu allen in der Praxis anzutreffenden Modellen befolgt werden. Dies geschieht meist schon intuitiv, man kann die Regeln dem Modellierer aber auch mit wenigen (bewusst informell

[4] „Fehlerhaft“ heißt an dieser Stelle „inhaltlich falsch“, was nur durch Betrachtung der modellierten Geschäftsprozesse herauszufinden ist. Ein Beispiel aus einem unserer eigenen Modelle war die Modellierung eines Falles, in dem ein Stromzähler zugleich defekt und in Ordnung war. Ein anderes Beispiel eines fehlerhaften Modells war das bereits von verschiedenen Autoren untersuchte Modell „Beschaffungslogistik“[Rit00]

formulierten) Sätzen vermitteln:

1. Beachte (bei Modellen mit Zyklen), dass keine Funktion mehrfach zugleich aktiviert werden kann.
2. Beachte, dass ein XOR-Join immer nur von genau einer Seite erreicht wird.
3. Denke bei der Verwendung von OR-Konstrukten immer zunächst darüber nach, ob du nicht eigentlich „AND“ oder „XOR“ meinst. Wenn du wirklich OR-Konstrukte verwenden musst, beachte, dass innerhalb dieser Konstrukte zu jedem Split- ein Join-Konnektor gleichen Typs gehört und dass man nicht von außerhalb des OR-Split/OR-Join-Konstrukts in dieses hineinspringen kann.

Mehr noch, die Ergebnisse zeigen, dass Modelle, die den Stilregeln *nicht* entsprechen, auch tatsächlich *ausnahmslos* fehlerhaft[5] waren. Das nächste interessante Resultat war, dass sich die Mehrzahl dieser Fehler automatisch mit den in der statischen Analyse gewonnenen Erkenntnissen korrigieren lässt.

Die Abdeckung von EPKs aus der Praxis ist mit unseren Stilregeln deutlich besser als bei den in [LSW97a] genannten Regeln, jedoch kann die in [LSW97a] vorgeschlagene Übersetzung von EPKs in boolesche Netze auch für die Klasse der nach unserer Definition gültigen EPKS erfolgen.

Damit erhalten wir das positive Resultat, dass die meisten in der Praxis vorhandenen EPKs auf relativ einfache (zumindest im Vergleich zu den Algorithmen aus [CK04] oder [WEAtH05]) Weise in eine Sprache übersetzt werden können, die zur Weiterverarbeitung in Simulations- oder Model-Checking-Tools dienen kann.

Literatur

[Aal99] Wil M.P. van der Aalst. Formalization and verification of event-driven process chains. *Information & Software Technology*, 41(10):639–650, 1999.

[ADK02] Wil M.P. van der Aalst, Jörg Desel und Ekkart Kindler. On the semantics of EPCs: A vicious circle. In *EPK 2004, Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten*, Seiten 71–79, 2002.

[AH02] Wil M.P. van der Aalst und A. Hofstede. YAWL: Yet Another Workflow Language. Bericht FIT-TR-2002-06, Queensland University of Technology, Brisbane, 2002.

[CK04] Nicolas Cuntz und Ekkart Kindler. On the semantics of EPCs: Efficient calculation and simulation. In *EPK 2004: Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, Proceedings*, Seiten 7–26, 2004.

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

[5] Hier wieder „fehlerhaft“ im weiteren Sinne: Gemeint sind inhaltlich falsche Modelle wie auch solche, für die eine klar bessere Darstellung mit gleicher Semantik existiert.

[Cun04] Nicolas Cuntz. Über die effiziente Simulation von Ereignisgesteuerten Prozessketten. Diplomarbeit, Universität Paderborn, 2004.

[DA04] Juliane Dehnert und Wil M.P. van der Aalst. Bridging The Gap Between Business Models And Workflow Specifications. *Int. J. Cooperative Inf. Syst.*, 13(3):289–332, 2004.

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

[Lau05] Ralf Laue. ebus.informatik.uni-leipzig.de/∼laue, 2005.

[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.

[MNN05] Jan Mendling, Gustaf Neumann und Markus Nüttgens. Towards Workflow Pattern Support of Event-Driven Process Chains (EPC). In *Second GI-Workshop XML4BPM XML for Business Process Management*, 2005.

[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.

[Rit00] Peter Rittgen. Quo vadis EPK in ARIS? *Wirtschaftsinformatik*, 42(1):27–35, 2000.

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

[vDAV05] Boudewijn F. van Dongen, Wil M.P. van der Aalst und H. M. W. Verbeek. Verification of EPCs: Using Reduction Rules and Petri Nets. In *CAiSE*, Seiten 372–386, 2005.

[WEAtH05] Moe Thandar Wynn, David Edmond, Wil M.P. van der Aalst und Arthur H. M. ter Hofstede. Achieving a General, Formal and Decidable Approach to the OR-Join in Workflow Using Reset Nets. In *ICATPN*, Seiten 423–443, 2005.