Prozeßmodellierung mit ereignisgesteuerten Prozeßketten (EPKs) und Petri-Netzen

Peter Langner[1] Christoph Schneider[2] Joachim Wehler[3]

Versandanschrift: Joachim Wehler, Döllingerstraße 35, D-80639 München, Fax: 089-1708145

Stichworte: Geschäftsprozeßmodellierung, Business Reengineering, Prozeßanalyse, Ereignisgesteuerte Prozeßkette, Petri-Netz, Verifikation, Simulation.

Zusammenfassung: Nach einer Definition der Syntax für EPKs werden Regeln angegeben, um EPKs in Petri-Netze zu übersetzen. Die resultierenden Booleschen Netze sind eine Erweiterung von Stellen/Transitions-Netzen um logische Verknüpfungen und bilden eine einfache Klasse gefärbter Petri-Netze. Das korrekte Verhalten dieser Netze läßt sich bereits durch eine algorithmische Reduktion ihrer Struktur verifizieren, eine Analyse des Fallgraphen ist nicht nötig. Eine Reihe von Modellierungsfehlern bei EPKs aus der Praxis belegt die Notwendigkeit einer solchen Netzanalyse. EPKs mit korrektem Verhalten lassen sich zu semantisch gleichwertigen Stellen/Transitions-Netzen vereinfachen. Damit bietet die Petri-Netz Theorie in ihrer klassischen Form ein tragfähiges Fundament zur Verifikation, Animation und Simulation von EPKs aus dem Bereich der Wirtschaftsinformatik.

Modeling processes using event-driven process chains (EPCs) and Petri nets

Keywords: Process modeling, business reengineering, process analysis, event-driven process chain, Petri net, verification, simulation.

Abstract: After defining the syntax of EPCs we propose a set of rules to translate EPCs into Petri nets. The resulting Boolean nets generalize place/transition nets und constitute a simple class of coloured Petri nets. Well behavedness of these nets can be proven by reducing the net structure, there is no need to consider case graphs. A sample of incorrect EPCs from different projects substantiates the need for such type of net analysis. Well behaved EPCs can be reduced to place/transition nets respecting the original semantic. Hence Petri net theory provides a well founded base to verify, to animate and to simulate EPCs used in the domain of Management Information Systems.

[1] INNOBIS Unternehmensberatung und Software GmbH, Willhoop 7, D-22453 Hamburg, email: P.Langner@innobis.de

[2] Christoph Schneider Datenerfassung, Paul Robeson Straße 40, D-10439 Berlin, email: 100 422.133@compuserve.com

[3] Softlab GmbH, Zamdorfer Straße 120, D-81677 München, email: WEJ@softlab.de

1 Prozeßmodellierung in der Wirtschaftsinformatik

Prozeßmodellierung ist der seit langem fehlende Bestandteil des Software Engineering. Die Betrachtung übergreifender Geschäftsprozesse erfordert in Ergänzung zur Datenmodellierung neue Methoden, um auch den dynamischen Aspekt eines Unternehmens zu modellieren. Die Geschäftsprozeßmodellierung hat für ein Unternehmen die gleiche Bedeutung wie die Datenmodellierung - aber auch die gleiche Komplexität.

1.1 Zielsetzung

Diese Arbeit verfolgt das Ziel, Prozeßmodellierung im Rahmen der Wirtschaftsinformatik auf eine tragfähige mathematische Basis zu stellen. Diese Basis sehen wir in der Theorie der Petri-Netze. Der Vorteil dieser theoretischen Fundierung liegt in der Möglichkeit zur maschinellen Verifikation der Prozeßmodelle, nicht nur hinsichtlich ihrer Syntax, sondern auch hinsichtlich ihres Verhaltens. Wir halten die maschinelle Überprüfbarkeit für eine Grundanforderung an eine ordnungsmäßige Prozeßmodellierung, siehe auch [BRS1995][4].

Für verifizierte Prozeßmodelle ist anschließend eine Simulation sinnvoll - nicht wie bisher zur Validierung des Modells, sondern um jetzt an einem formal korrekten Modell zeitliche Abläufe der Realität durchzuspielen[5]. Eine weitergehende Aufgabe ist dann die computergesteuerte Ausführung der Prozesse, z.B. durch eine Workflow Engine.

In zahlreichen kommerziellen Projekten wird in Deutschland die Methode der ereignisgesteuerten Prozeßketten (EPK) zur Modellierung von Geschäftsprozessen eingesetzt. Im einzelnen werden EPKs zur

- Dokumentation
- Analyse im Rahmen eines Soll-Istvergleichs
- EDV-Unterstützung

von Prozessen herangezogen. Auf Grund ihrer Bedeutung in der Praxis erscheint es unabdingbar, diese Prozeßmodelle maschinell auf Fehler zu überprüfen - ebenso wie es bei Programmen ein Compiler tut.

In dieser Arbeit wird die Syntax von EPKs festgelegt, es wird eine Übersetzung von EPKs in Petri-Netze vorgeschlagen, und es wird gezeigt, wie sich die Semantik der resultierenden Netze maschinell verifizieren läßt. Für diese Prüfung ist es nicht nötig, den Fallgraphen zu konstruieren, statt dessen kann man einen Reduktionsalgorithmus anwenden.

---

[4] Vgl. das Verbundprojekt *Grundsätze ordnungsmäßiger Modellierung (GoM)* unter Leitung des Instituts für Wirtschaftsinformatik an der Westfälischen Wilhelms-Universitat Münster

[5] Gute Simulationsmöglichkeiten bietet z. B. SIMPLE ++, ein Produkt der Firma AESOP GmbH

Die Arbeit richtet sich an WirtschaftsinformatikerInnen und BetriebswirtschaftlerInnen. Für InformatikerInnen, die zusätzlich an den mathematischen Details, an vollständigen Definitionen oder Beweisen interessiert sind, verweisen wir auf unsere Ausarbeitung [LSW1997].

1.2 Das Vorbild Datenmodellierung

Bei einem Vergleich von Daten- und Prozeßmodellierung sind drei Faktoren erkennbar, auf denen der unbestrittene Erfolg der Datenmodellierung in der Wirtschaftsinformatik beruht:

- Mathematische Fundierung: Die Relationenalgebra
- Formale Methode: Das Entity Relationship-Modell
- Geradlinige Implementierung: Relationale Datenbanken unter SQL.

Durch die formale Methode wird eine gemeinsame Sprache festgelegt, in der alle Beteiligten eindeutig über den Gegenstandsbereich sprechen. Hinter dem Entity Relationship-Modell steht bekanntlich die von Codd entwickelte Relationenalgebra [Cod1972], die auf einem einzigen abstrakten Datentyp, der Relation, basiert. Die Sprache SQL implementiert diesen abstrakten Datentyp für relationale Datenbanken.

Datenmodellierung auf der Basis des Entity Relationship-Modells beeindruckt als erfolgreiche Methode des Software Engineering, die von der Wirtschaftsinformatik übernommen und auf praktische Fragen angewendet wurde. Wegbereiter in Deutschland ist Scheer [Sch1994], der auf dieser Basis ein unternehmensweites Referenzdatenmodell entwickelte.

1.3 Prozeßmodellierung

In diesem Abschnitt werden zwei Methoden zur Prozeßmodellierung vorgestellt.

Unter einem ***Prozeß*** verstehen wir ein Netz von Funktionen, das durch eine Menge von Startereignissen angestoßen wird und dazu dient, eine Menge von Zielereignissen zu erreichen. Im folgenden berücksichtigen wir ausschließlich die Ablaufsicht der Prozesse, die Organisationssicht werden wir nicht weiter behandeln. Man kann mit Prozessen drei Grundoperationen ausführen, man kann sie

- sequentialisieren: Zielereignisse des ersten Prozesses sind Startereignisse eines zweiten Prozesses,
- parallelisieren: Mehrere Prozesse laufen exklusiv (xor), optional (or) oder nebenläufig (and) ab,
- hierarchisieren: Eine Funktion des ersten Prozesses wird zu einem eigenen, zweiten Prozeß verfeinert.

Aus formaler Sicht kann man auf die *Funktion* als separates Objekt verzichten und ausschließlich von Ereignissen und Prozessen sprechen. Denn ob eine Aufgabe Funktion oder Prozeß genannt wird, ist allein eine Frage des Abstraktionsgrads des Modells. Das sagt obiges Prinzip der Hierarchisierung.

In der Regel beruhen *formale Methoden* zur Prozeßmodellierung auf gerichteten Graphen: Man zeichnet ein Diagramm mit verschiedenen Knoten und verbindet diese durch gerichtete Kanten. Die

Knoten stellen Ereignisse bzw. Aktivitäten dar, und die gerichteten Kanten modellieren einen Fluß - entweder den Kontrollfluß oder den Datenfluß.

Die von Keller, Nüttgens und Scheer [KNS1991] entwickelte Methode der *ereignisgesteuerten Prozeßketten (EPKs)* wird in einer schnell wachsenden Anzahl von Projekten mit Erfolg zur Prozeßmodellierung eingesetzt. Die EPK-Methode besitzt von Haus aus keine theoretische Begründung, durch Übersetzung in ein Petri Netz läßt sie sich aber fundieren.

Die von Petri 1962 erfundenen *Petri-Netze* [Bau1990] zeichnen sich unter allen Konkurrenten um die Prozeßmodellierung gerade durch ihre *mathematische Fundierung* aus. Damit hat eine Prozeßmodellierung auf der Basis von Petri-Netzen eine vergleichbar präzise Semantik wie die oben angesprochene Datenmodellierung. Man kann ein Petri-Netz sogar als ein ausführbares Programm ansehen und kompilieren. Bei der Kompilation wird das Netz zunächst auf formale, d.h. syntaktische Fehler geprüft. In manchen Fällen, insbesondere bei den im folgenden auftretenden Booleschen Netzen, ist darüber hinaus sogar eine algorithmische Verifikation des Netzverhaltens (Semantik) möglich. Geprüfte Netzmodelle lassen sich *implementieren*: Man kann sie animieren, in Testläufen simulieren und zur Steuerung von Workflows heranziehen. Die entscheidenden Vorzüge der Petri-Netze als Methode der Prozeßmodellierung liegen in der Möglichkeit zur

- Verifikation ihrer Syntax
- Verifikation ihres Verhaltens
- Animation und Simulation.

Mitunter wird gegen die Präzisierung einer EPK der Einwand erhoben, daß gerade die Vagheit der EPK-Methode von Vorteil sei, wenn am Anfang eines Projektes eine bewußte Unschärfe des Modells angestrebt wird. Wir betrachten den in dieser Arbeit zur Diskussion gestellten Ansatz daher als Angebot für Situationen, in denen eine Präzisierung erwünscht ist. Allerdings erscheint es uns auch bei noch nicht vollständig analysierten Prozessen ein gangbarer Weg, wenn man zunächst auf einer höheren Abstraktionsebene, aber dennoch in einer präzisen Sprache modelliert, und das Modell dann im Laufe des Projekts durch Hierarchisierung detailliert.

2 Übersetzungsregeln für EPKs

Die EPK-Methode basiert im wesentlichen auf der Petri-Netz Theorie (...)
und kann als eine Variante des Bedingungs-Ereignisnetzes, welche um
logische Verknüpfungsoperationen erweitert wurde, verstanden werden.
Scheer, Nüttgens, Zimmermann [SNZ1995]

2.1 Syntax einer EPK

Die Übersetzung einer EPK in ein Petri-Netz läßt sich als Algorithmus formulieren und maschinell ausführen, aber man muß zuvor die Syntax der EPK definiert haben. Dies ist in der Literatur bisher nicht geschehen. Wir definieren:

*Eine **ereignisgesteuerte Prozeßkette** (EPK)*[6] *ist ein gerichteter Graph.*

- *Er hat drei Arten von Knoten: Ereignisse, Funktionen, binäre logische Konnektoren (xor, and, or).*
- *Alle Kanten verbinden zwei Knoten von jeweils unterschiedlichem Typ.*
- *Nur die logischen Konnektoren verzweigen, sie verbinden Ereignisse mit Funktionen und vice versa.*
- *Die Eingänge eines logischen Konnektors sind entweder alle vom Typ Ereignis oder alle vom Typ Funktion, ebenso sind seine Ausgänge entweder alle vom Typ Ereignis oder alle vom Typ Funktion.*
- *Am Rand der EPK liegen nur Ereignisse. Es gibt mindestens ein Start- und mindestens ein Zielereignis.*

In der Praxis verwendet man logische Konnektoren mit beliebig vielen Ein- und Ausgängen. Um eine EPK in die obige Syntax zu bringen, muß man diese Konnektoren in eine Folge von binären Konnektoren auflösen, wobei je zwei Konnektoren aus einer solchen Auflösung durch ein Ereignis zu trennen sind. Diese Reduktion auf binäre Konnektoren verändert nichts an der Semantik der EPK, erlaubt aber die Beschränkung auf drei Grundelemente der Logik. Nach erfolgter Übersetzung und Netzanalyse kann diese Zerlegung wieder rückgängig gemacht werden.

In der Literatur [KNS1991] findet man manchmal das Verbot, auf ein Ereignis einen or- bzw. xor-Konnektor folgen zu lassen. Das wird damit begründet, daß Ereignisse als passive Elemente keine Entscheidungen treffen können. Wir halten diese Einschränkung für zu restriktiv, da man an Beispielen aus der Literatur sieht, daß auch diese Fälle praxisrelevant sind, vgl. [Sch1994], p.450.

Die folgende EPK *Beschaffungslogistik* von Abbildung 2-1 stammt aus [Sch1994], p. 420. Wir haben lediglich die Beschriftungen einer objektorientierten Notation angepaßt:

<Klasse>.<Funktion> bzw. <Klasse>.<Ereignis>

Die EPK läßt sich durch kleine Korrekturen auf die oben vorgeschlagene Syntax bringen.

---

[6] Die Parallelisierung von Prozessen und Funktionen trifft besser der Begriff des ereignisgesteuerten Prozeß*netzes* (EPN).

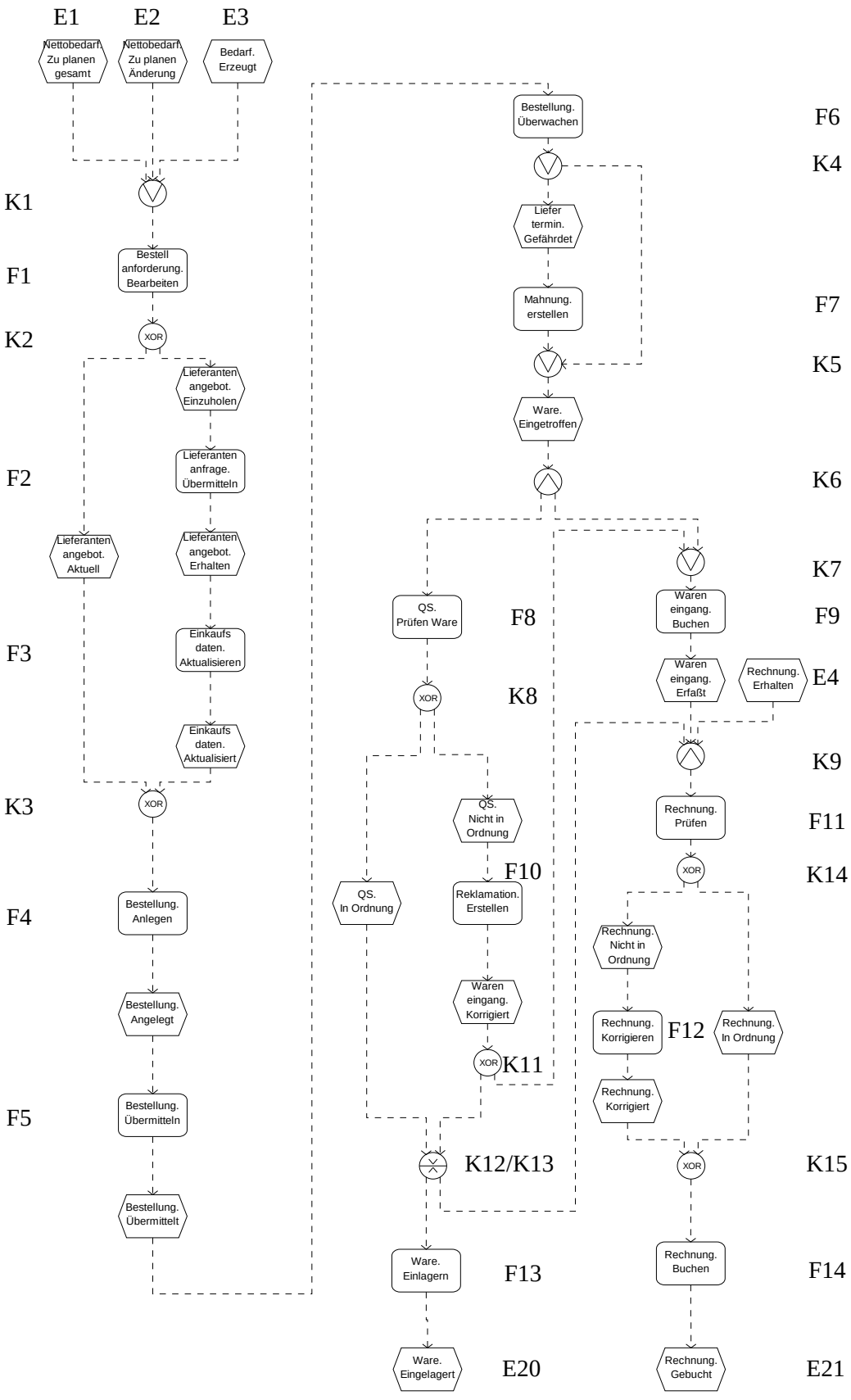
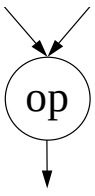
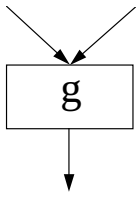


Abbildung 2-1 EPK Beschaffungslogistik

2.2 Übersetzungsregeln

Bevor wir die EPK *Beschaffungslogistik* aus Abbildung 2-1 in ein Petri-Netz nach den Regeln von Abbildung 2-2 übersetzen, korrigieren wir zwei Druckfehler:

- Die beiden or-Konnektoren K4, K5 nach der Funktion *Bestellung.Überwachen* werden durch xor-Konnektoren K40, K50 ersetzt. Denn entweder wird eine Mahnung erstellt oder sie wird nicht erstellt.
- Nach dem Ereignis *Wareneingang.Korrigiert* muß der öffnende xor-Konnektor K11 durch einen and-Konnektor ersetzt werden. Denn mit dem xor-Konnektor würde entweder eine Korrektur gebucht, aber keine Ware eingelagert, oder es würde Ware eingelagert, aber keine Korrektur gebucht.

EPK	Petri-Netz
Logischer Konnektor	**Boolesche Transition**
op ∈ { xor, or, and }	g ∈ { Branch, Branch/Fork, Fork }
	
op ∈ { xor, or, and }	g ∈ { Merge, Merge/Join, Join }

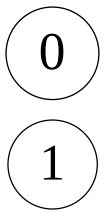
EPK	Petri-Netz
Schleifenangelpunkt	**Stelle**
Ereignis	**Stelle**
	
Funktion	**Transition**
	
nicht vorhanden	**Markierung** 

Abbildung 2-2 Übersetzungsregeln

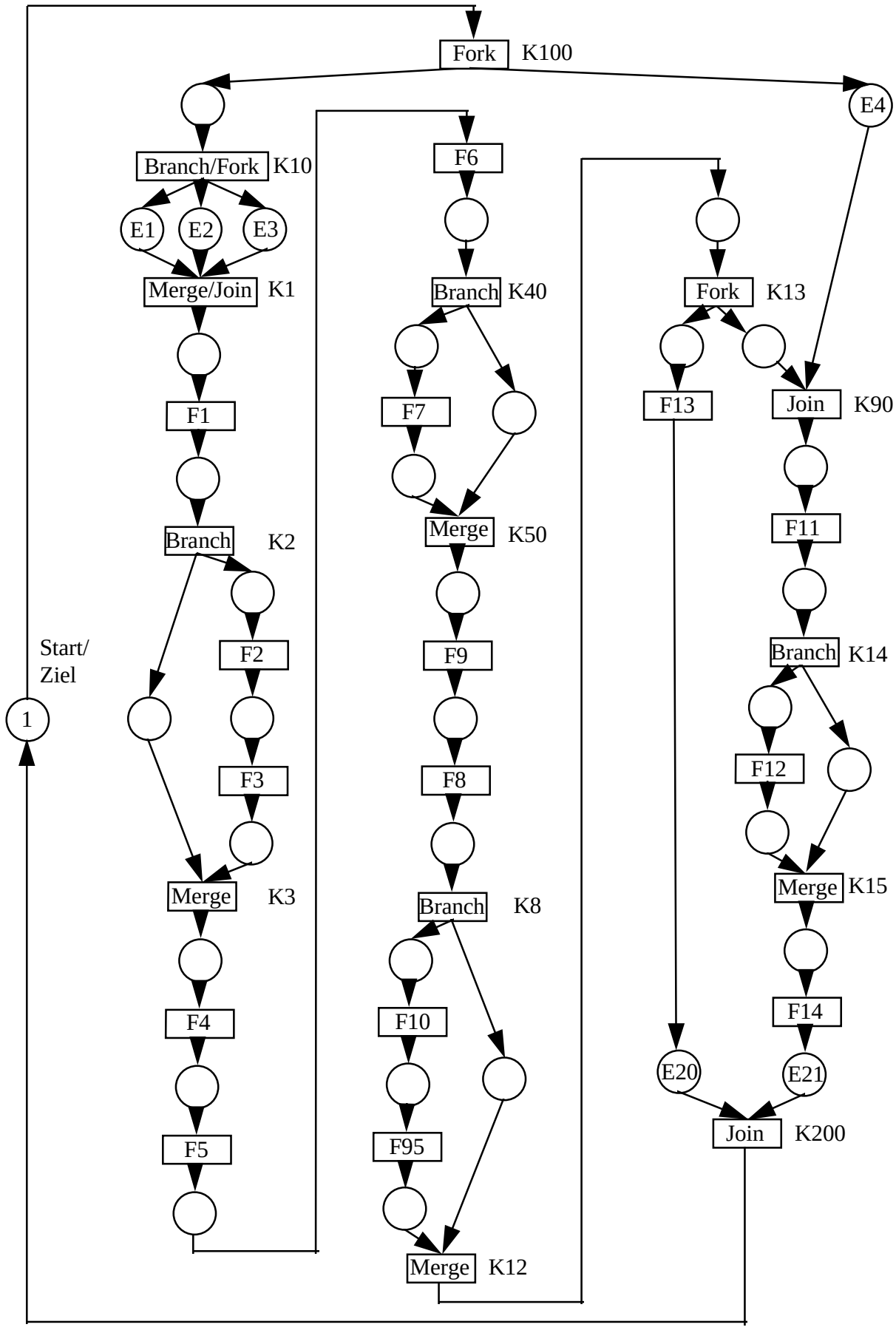


Abbildung 2-3 Boolesches Netz Beschaffungslogistik

Nach der Übersetzung der Netzelemente fügen wir einige zusätzliche Stellen ein, um die Syntax eines Petri-Netzes zu erfüllen. Danach schließen wir das resultierende Netz durch eine weitere ausgezeichnete Stelle *Start/Ziel*, die wir mit den möglichen Startereignissen E1, E2, E3, E4 und den möglichen Zielereignissen E20, E21 verbinden. Die zulässigen Kombinationen dieser Ereignisse werden durch die Logik der eingefügten Konnektoren K100, K10, K200 wiedergegeben. Abbildung 2-3 zeigt das resultierende Petri-Netz, nachdem es allerdings schon einer Reihe von Korrekturen aufgrund der Netzanalyse unterworfen wurde, die wir in Abschnitt 3.4 erläutern werden. Im Zuge dieser Netzanalyse wurde die neue Funktion F95 eingefügt, der Konnektor K9 wurde durch den Konnektor K90 ersetzt, und die Konnektoren K7 und K11 fielen weg.

Obwohl die Autoren [SNZ1995] mit EPKs den Kontrollfluß modellieren wollen, enthält die EPK-Methode kein Netzelement, welches diesen Fluß repräsentieren könnte: In einer EPK gibt es keine Marken. Man wird daher das Petri-Netz, das aus der Übersetzung einer EPK resultiert, so mit Marken versehen, daß markierte Stellen die momentane Aktivität anzeigen. Die Marken bewegen sich dann aufgrund einer Schaltregel durch das Netz und modellieren auf diese Art den Kontrollfluß. So ist in dem Booleschen Netz *Beschaffungslogistik* in Abbildung 2-3 die Stelle *Start/ Ziel* markiert, wenn der Logistikprozeß startbereit ist. Bei diesem Ansatz stellt sich insbesondere die Frage nach der Schaltregel des schließenden or- bzw. xor-Konnektors:

- Schaltet ein schließender binärer or- bzw. xor-Konnektor das erste Mal beim Eintreffen einer Marke auf dem einen seiner Eingänge und ein weiteres Mal beim Eintreffen einer weiteren Marke auf dem anderen Eingang?
- Hängt das Schaltverhalten vom zeitlichen Eintreffen beider Marken ab? Soll der or-Konnektor nur einmal schalten, wenn beide Marken gleichzeitig eintreffen?

Wir fassen EPKs und Petri Netze so auf, daß beide Methoden kausale Abhängigkeiten modellieren: Ereignisse bzw. Stellen bedeuten notwendige Voraussetzungen, wenn sie erfüllt sind, kann die nachfolgende Transition schalten. Aus der kausalen Abhängigkeit ergibt sich a posteriori eine zeitliche Reihenfolge. Aber beide Methoden enthalten keinen globalen Zeitmaßstab und damit auch keinen Begriff der Gleichzeitigkeit. Wir verwenden daher eine Schaltregel, nach der ein schließender binärer Konnektor erst dann schalten kann, wenn an seinen *beiden* Eingängen eine explizite Aktivierungsinformation vorliegt. Als technisches Hilfsmittel führen wir zwei verschiedene Arten von Marken ein:

- Marken mit der Aufschrift „1“ bedeuten eine Aktivierung, Marken mit der Aufschrift „0“ eine explizite Deaktivierung.
- Unmarkierte Stellen bedeuten Bedingungen, über deren Gültigkeit keine Information vorliegt.

Durch die Einführung der 0-Marken können gewisse Netzteile explizit deaktiviert werden. Falls im Modell solche Netzteile durch schließende logische Konnektoren synchronisiert werden, müssen die

0-Marken auch durch die deaktivierten Netzteile weiterfließen. Das bedeutet, daß die Schaltregeln der Transitionen auch den Fall erfassen, in dem alle Eingänge mit einer 0-Marke markiert sind. In diesem Fall sollen beim Schalten der Transition auch alle Ausgänge eine 0-Marke erhalten: Deaktivierte Transitionen reichen die Deaktivierung an alle Folgestellen weiter.

2.3 Boolesche Netze

Wir präzisieren die Klasse der Petri-Netze, die aus der Übersetzung einer EPK resultieren, durch folgende Definition:

*Ein **Boolesches Netz** entsteht aus einem Stellen/Transitions-Netz durch eine Beschriftung in der Sprache der Aussagenlogik. Hierbei erhält*

- *jede Kante eine Boolesche Variable mit den möglichen Werten „0“ oder „1“*
- *und jede Transition einen Booleschen Ausdruck (Guard-Formel) in allen Variablen der angrenzenden Kanten.*

Eine *Markierung* legt auf gewisse Stellen des Booleschen Netzes eine Marke, und zwar entweder eine 0-Marke oder eine 1-Marke. Im Unterschied zu einer Transition in einem Stellen/Transitions-Netz reicht bei einer Booleschen Transition die Markierung ihrer Eingangsstellen jedoch noch nicht zur Aktivierung aus. Im Rahmen der Semantik der Petri-Netze bestimmt zusätzlich die Guard-Formel die **Schaltregel** einer Transition:

- *Ein zulässiger Schaltmodus einer Transition ist eine Wertzuweisung an die Booleschen Variablen aller Ein- und Ausgangskanten der Transition, so daß die Guard-Formel den Booleschen Wert „wahr“ annimmt.*
- *Eine Transition ist in einem zulässigen Schaltmodus aktiviert, wenn jede ihrer Eingangsstellen eine Marke mit dem Booleschen Wert der entsprechenden Eingangskante enthält.*
- *Wenn eine Boolesche Transition in einem zulässigen Schaltmodus aktiviert ist und schaltet, so entfernt sie aus jeder Eingangsstelle die zugehörige Marke und legt auf jede Ausgangsstelle eine dem Schaltmodus entsprechende Marke.*

In diesem Sinne hat die Guard-Formel die Rolle eines „Wächters“. Sie entscheidet über das Schaltverhalten der Booleschen Transition bei einer Belegung ihrer Eingangsstellen. Zum Beispiel besitzt eine xor-Transition mit zwei Eingangsstellen keinen zulässigen Modus, in dem beide Stellen mit einer 1-Marke markiert sind. Statt dessen ist die Transition bei dieser Markierung blockiert. Dieser Deadlock bedeutet, daß beide Marken bis zum Ende des Prozesses an dieser Stelle liegenbleiben und einen Modellierungsfehler anzeigen.

Die Booleschen Transitionen, die aus der Übersetzung einer syntaktisch korrekten EPK resultieren, haben entweder zwei Eingänge und einen Ausgang oder einen Eingang und zwei Ausgänge. Wir

beschriften die beiden gleichgerichteten Kanten mit den Variablen *x* bzw. *y* und die dritte Kante mit der Variablen *z* und geben jeder Booleschen Transition die Guard-Formel

$$x\,or\,y\,or\,z \quad \Rightarrow \quad (x\,op\,y)\ and\ z$$

wobei op ∈ { xor, or, and } den logischen Typ des zugehörigen Konnektors der EPK bezeichnet. Diese Guard-Formel hängt nicht von der Richtung der Kanten ab, so daß jeweils öffnende und schließende Boolesche Transitionen desselben logischen Typs auch dieselbe Guard-Formel haben. Auf Grund ihrer Guard-Formeln können die einzelnen Booleschen Transitionen in folgenden zulässigen Modi schalten:

op = and		
x	y	z
0	0	0
1	1	1

op = xor		
x	y	z
0	0	0
1	0	1
0	1	1

op = or		
x	y	z
0	0	0
1	1	1
1	0	1
0	1	1

Abbildung 2-4 Zulässige Schaltmodi Boolescher Transitionen

Alle Markierungen, die von einer Anfangsmarkierung aus durch sukzessives Schalten eintreten können, heißen *erreichbare* Markierungen. Die Verhaltensanalyse eines Booleschen Netzes prüft nun, ob das Netz unter seinen erreichbaren Markierungen ein ordentliches Verhalten zeigt. Wir definieren:

Ein Boolesches Netz zeigt unter einer gegebenen Anfangsmarkierung ein ***ordentliches Verhalten*** *(well-behaved), wenn es sicher und lebendig ist, d.h. wenn*

- *jede Stelle unter jeder erreichbaren Markierung höchstens eine Marke trägt*
- *und jede Transition von jeder erreichbaren Markierung aus aktiviert werden kann.*

Die *Sicherheit* des Verhaltens ist die notwendige Voraussetzung, um die Stellen des Netzes als Bedingungen interpretieren zu können. Insofern kann ein Boolesches Netz mit ordentlichem Verhalten als „eine Variante des Bedingungs-Ereignisnetzes, welche um logische Verknüpfungsoperationen erweitert wurde, verstanden werden“ [SNZ1995]. Durch die Beschriftung der Transitionen mit Guard-Formeln und die Unterscheidung von zwei verschiedenen Typen von Marken wird die Klasse der Stellen/Transitions-Netze verlassen. Die resultierenden Booleschen Netze stellen einen einfachen Spezialfall von gefärbten Petri-Netzen dar [Jen1992]. Für andere Ansätze zur Übersetzung einer EPK in ein Petri-Netz vgl. [CS1994], [Brö1996], [Rod1997]. Letztgenannte Arbeit enthält auch eine ausführliche Diskussion des in [CS1994] dargestellten Vorgehens.

2.4 Modellierung von Alternativen

Man sollte die logischen Konnektoren einer EPK nicht isoliert für sich betrachten, sondern bei einem öffnenden Konnektor jeweils beide zugehörigen Prozeßalternativen verfolgen und prüfen, ob sie an späterer Stelle wieder synchronisiert werden. Daher nennen wir die verschiedenen Booleschen Transitionen, die aus der Übersetzung einer EPK resultieren,

- ***Fork***- bzw. ***Join-Transition*** (öffnender bzw. schließender and-Konnektor)
- ***Branch***- bzw. ***Merge-Transition*** (xor-Konnektor)
- ***Branch/Fork***- bzw. ***Merge/Join-Transition*** (or-Konnektor).

Unter den logischen Alternativen enthält die Branch/Fork-Alternative die Verzweigungsmöglichkeiten der beiden anderen Alternativen, der Fork- und der Branch-Alternative, gemäß der logischen Formel

$$x \; or \; y \Leftrightarrow \left(x \; xor \; y\right) \; xor \; \left(x \; and \; y \; \right).$$

Die Schwierigkeit der or-Modellierung wird bei der Merge/Join-Transition deutlich. Ihre Guard-Formel muß sowohl die beiden Teilprozesse einer Fork-Alternative als auch die beiden Teilprozesse einer Branch-Alternative synchronisieren. Bei jedem konkreten Prozeßablauf hat man sich nun bei der Branch/Fork-Transition für eine der beiden Verzweigungsarten entschieden - entweder verzweigt der Prozeß im Fork-Modus, oder er verzweigt im Branch-Modus. Daher ist es sinnvoll, für eine Merge/Join-Transition zu fordern:

Jeder Merge/Join-Transition geht eine Branch/Fork-Transition voran, und beide Transitionen schalten im selben Schaltmodus, entweder beide im Branch-Modus oder beide im Fork-Modus.

Diese Forderung läßt sich erfüllen, indem nur solche or-Alternativen zugelassen werden, die als Stellenverfeinerung aus einer *elementaren Branch/Fork-Alternative* hervorgehen. Als Konsequenz lassen sich alle Branch/Fork-Alternativen gemäß obiger Formel in eine Schachtelung von Branch- und Fork-Alternativen auflösen (Branch/Fork-Auflösung), und der or-Konnektor läßt sich als selbständiges Netzelement einer EPK eliminieren, vgl. Abbildung 2-5.

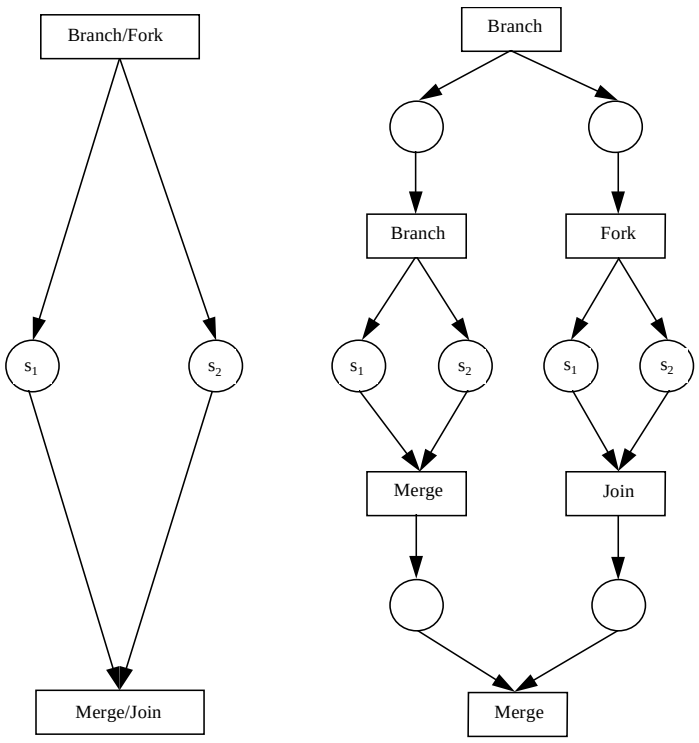


Abbildung 2-5 Elementare Branch/Fork-Alternative und ihre Branch/Fork-Auflösung

2.5 Modellierung von Schleifen

In [KNS1991], [Sch1994] wird der Grundsatz aufgestellt, Verzweigungen nur an den logischen Konnektoren einer EPK zuzulassen. Anfang und Ende einer inneren Schleife wird dabei als schließender bzw. öffnenden xor-Konnektor modelliert. Dadurch treten diese xor-Konnektoren jedoch in zwei ganz verschiedenen Bedeutungen auf mit der Folge, daß ein grundlegender Unterschied der Realität an dieser Stelle der EPK nicht mehr erkennbar ist:

- An einem xor-Konnektor, der eine Branch-Alternative öffnet, verzweigt sich der Kontrollfluß: Der eine Teilprozeß wird aktiviert, der andere Teilprozeß wird explizit deaktiviert. Diese Aktivierungs- bzw. Deaktivierungsinformationen fließen über beide Ausgänge weiter und werden an anderer Stelle synchronisiert.
- An einem xor-Konnektor, der eine Schleife öffnet, fließt der Kontrollfluß entweder in den Schleifenrumpf hinein oder an der Schleife vorbei - ohne sich jedoch zu verzweigen. In jedem Fall wird nur eine der beiden Alternativen aktiviert, die andere bleibt unberücksichtigt. Insbesondere wird die andere Alternative nicht deaktiviert und daher auch nicht an späterer Stelle synchronisiert.

Ein entsprechender Unterschied gilt für schließende Konnektoren.

Wir haben bei unseren Übersetzungsregeln Wert auf die Unterscheidung beider Fälle gelegt und übersetzen daher die Konnektoren einer Branch-Alternative in eine verzweigte *Transition*, die Konnektoren einer Schleifen-Alternative dagegen in eine verzweigte *Stelle*. Im Unterschied zu EPKs respektiert das Petri-Netz die unterschiedlichen Situationen der Realität auch im Modell:

- Im Falle einer Branch-Alternative fließt über beide Ausgänge der verzweigten Transition je eine Marke.
- Im Falle einer Schleifenverzweigung gibt es nur eine einzige Marke, und diese fließt nur über einen der beiden Ausgänge der verzweigten Stelle.

Aus der Übersetzung von EPKs mit inneren Schleifen entstehen daher Boolesche Netze, bei denen sowohl die Transitionen als auch die Stellen verzweigt sein können. Allerdings sind nicht beliebige Verzweigungen zugelassen, sondern verzweigende Stellen dürfen nur als *Angelpunkt* einer Schleife auftreten, wobei aus Gründen der Syntax anschließend zwei zusätzliche Transitionen eingefügt werden. Darüber hinaus sind keine Ein- oder Aussprünge aus dem Schleifenrumpf zugelassen, und mehrfache Schleifen müssen in wohldefinierter Weise geschachtelt sein.

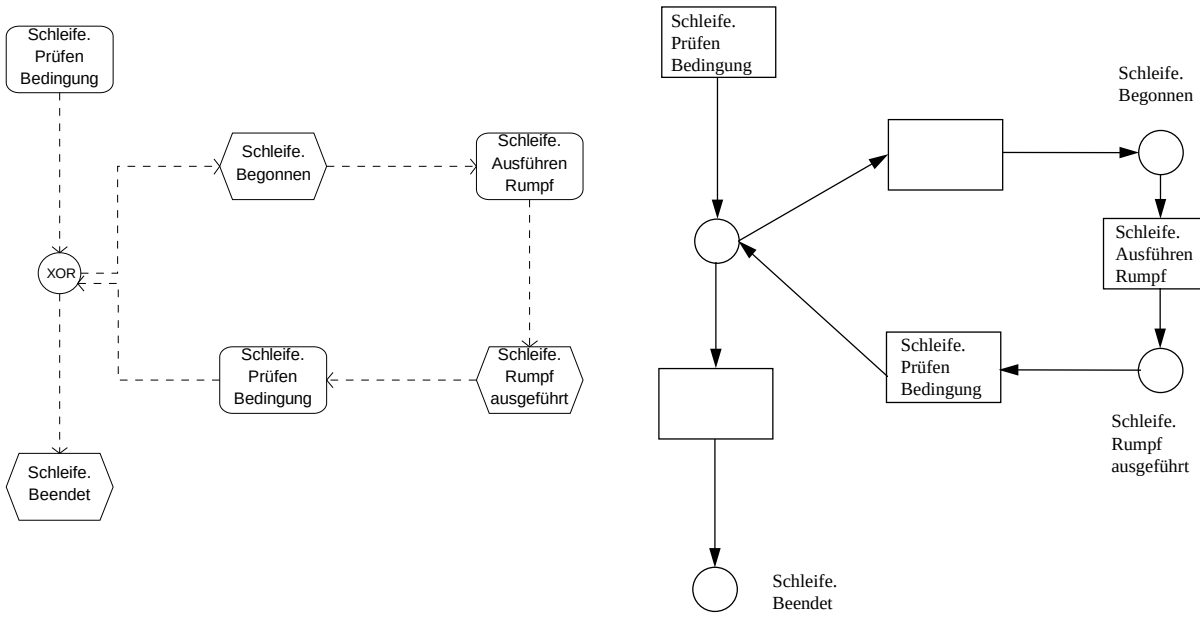


Abbildung 2-6 Schleifenmodellierung mit EPKs und Booleschen Netzen

3 Analyse Boolescher Netzsysteme

An das Verhalten einer EPK stellen wir zwei grundlegende Forderungen:

- Die EPK soll keine Abläufe ermöglichen, bei denen der Kontrollfluß vor einer Funktion „hängenbleibt“.
- Die EPK soll keine toten Netzteile enthalten, sondern jede Transition kann immer wieder in einem zulässigen Schaltmodus aktiviert werden.

Wir haben diese beiden Eigenschaften unter dem Namen *Lebendigkeit* als Anforderung an das resultierende Boolesche Netz in unsere Definition von *ordentlichem* Verhalten aufgenommen. Man kann die zweite Bedingung noch verschärfen zu der Forderung, daß *jeder* zulässige Schaltmodus einer gegebenen Booleschen Transition aktiviert werden kann, der eine 1-Marke enthält. Diese Forderung betrifft die beiden Arten von schließenden Transitionen vom Typ Merge bzw. Merge/Join. Für eine

Merge/Join-Transition z. B. mit Eingangsvariablen *x*, *y* und Ausgangsvariablen *z* wird verlangt, daß alle drei zulässigen Schaltmodi

$$(x,\ y,\ z) \in \{\ (1,1,1)\ ,\ (1,0,1)\ ,\ (0,1,1)\ \}$$

aktiviert werden können.

3.1 Modellierungsfehler

Ein Blick auf die betriebliche Praxis zeigt, daß gegenwärtig in vielen kommerziellen Projekten Fachkonzepte verfaßt werden, bei denen die Spezifikation mit EPKs an erster Stelle steht. Hier stellt sich folgendes Problem: Wie lassen sich die vorliegenden Prozeßmodelle prüfen, wie kann man erkannte Fehler beheben, mit welchen Regeln lassen sich Fehler bereits im Vorfeld ausschließen? Die Brisanz dieses Problems steigt in dem Maße, wie Prozeßmodellierung immer mehr zum Rückgrat der Projekte gemacht wird. Die fehlende Modellverifikation wird sich spätestens beim Übergang von der Fachkonzeptphase zur Realisierung als Mangel herausstellen. Aber selbst eine syntaktisch korrekte EPK modelliert deswegen nicht schon einen sinnvollen Prozeß. Zahlreiche Beispiele aus der Praxis machen vielmehr deutlich, wie leicht die syntaktische Freiheit der EPKs bei der Modellierung mißbraucht werden kann. Dabei entstehen dann EPKs, die keinen realen Prozeß mehr darstellen. Die folgenden Praxisbeispiele Abbildung 3-1 und Abbildung 3-2 zeigen eine Auswahl möglicher Fehlersituationen bei der Modellierung mit EPKs.

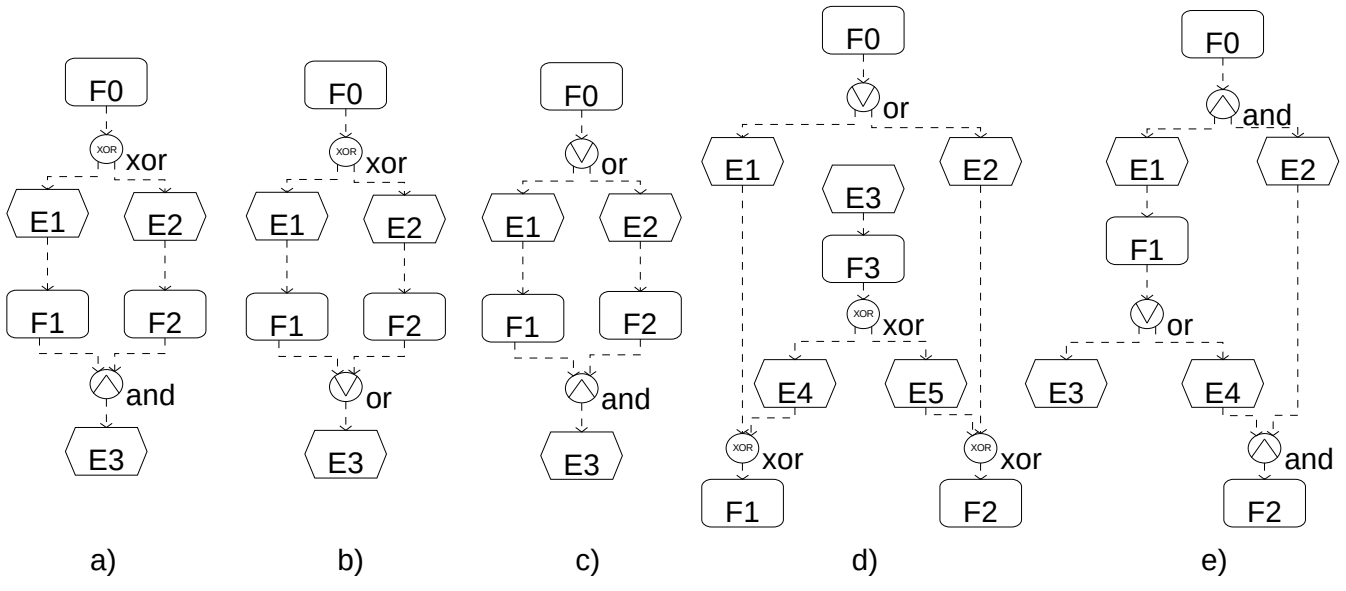
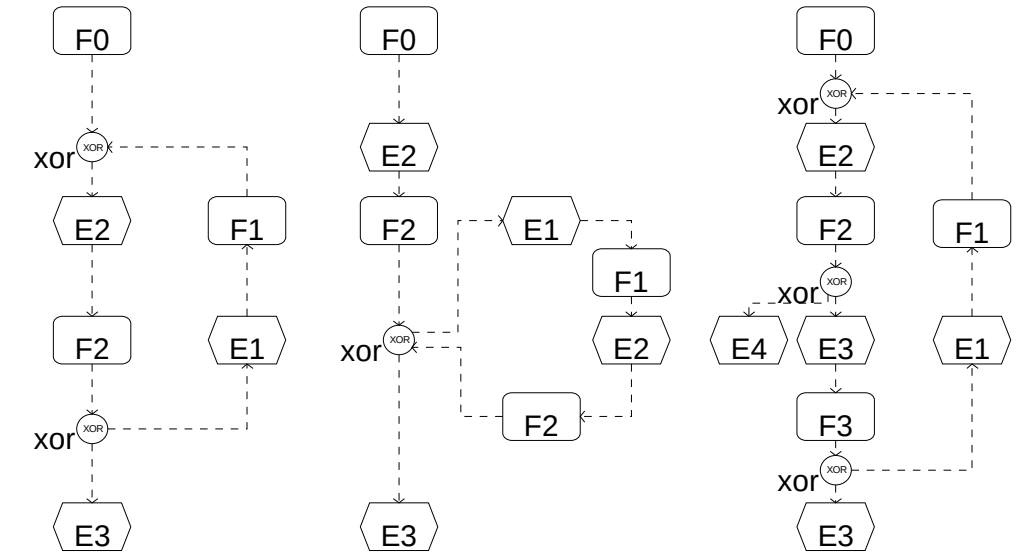


Abbildung 3-1 Modellierungsfehler mit EPKs

Abbildung 3-1, a) Schließen eines öffnenden xor-Konnektors mit einem and-Konnektor: Der schließende and-Konnektor blockiert, da aufgrund des öffnenden xor-Konnektors nicht beide Funktionen F1 und F2 gemeinsam ausgeführt werden können.
Abbildung 3-1, b) Schließen eines öffnenden xor-Konnektors mit einem or-Konnektor: Der schließende or-Konnektor läßt mehr Fälle zu, als der öffnende xor-Konnektor zur Verfügung stellt: Der Fall, daß beide Funktionen F1 und F2 zusammen ausgeführt werden, kann nicht eintreten.
Abbildung 3-1, c) Schließen eines öffnenden or-Konnektors mit einem and-Konnektor: Der schließende and-Konnektor blockiert, falls der öffnende or-Konnektor nur eine der beiden Funktionen F1 oder F2 aktiviert.
Abbildung 3-1, d) Falsche Schachtelung von or- und xor-Konnektoren: Falls die Funktion F0 beide Ereignisse E1 und E2 auslöst und zusätzlich die Funktion F3 ausgeführt wird, blockiert einer der beiden schließenden xor-Konnektoren.
Abbildung 3-1, e) Aussprung aus einer Fork-Alternative mit einem öffnenden or-Konnektor: Falls die Funktion F1 das Ereignis E4 nicht aktiviert, blockiert der schließende and-Konnektor.

Bei der Modellierung von Schleifen muß in Analogie zu Nassi-Shneiderman-Diagrammen zunächst der Schleifenrumpf identifiziert werden, d.h. derjenige Teil, welcher iteriert werden kann. Bei der Schleifenmodellierung sind Aus- und Einsprünge in den Schleifenrumpf an anderer Stelle als nach der Abbruchbedingung überflüssig und können durch eine sorgfältige Modellierung vermieden werden. Als nächstes unterscheidet man, ob die Schleife fuß- oder kopfgesteuert ist: Wenn der Prozeß den Schleifenrumpf mindestens einmal durchläuft und erst danach die Abbruchbedingung getestet wird, so handelt es sich um eine fußgesteuerte Schleife. Wenn die Abbruchbedingung dagegen schon vor dem ersten Betreten des Schleifenrumpfes getestet wird, so spricht man von einer kopfgesteuerten Schleife. Die Modellierung mit EPKs unterstützt kopfgesteuerte Schleifen. Jedoch kann jede fußgesteuerte Schleife in eine kopfgesteuerte Schleife überführt werden, indem man den Schleifenrumpf verdoppelt und vorweg einmal außerhalb der Schleife durchläuft.



f) Modellierungsfehler f) Korrekte Modellierung g) Modellierungsfehler

Abbildung 3-2 Fehler bei der Schleifenmodellierung mit EPKs

Abbildung 3-2, f) Schleife mit Ein- und Aussprung verschieden: Der xor-Konnektor für den Einsprung in die Schleife ist verschieden vom xor-Konnektor des Aussprungs. Der Schleifenrumpf besteht aus der Sequenz der Funktionen F1, F2. Die Schleife ist kopfgesteuert. Vor einem evtl. Schleifendurchlauf soll die Funktion F2 ausgeführt werden. Zur Korrektur dieser Schleife wird die Funktion F2 verdoppelt und zusätzlich vor den Schleifenrumpf gestellt.

Abbildung 3-2, g) Zusätzlicher Aussprung bzw. Einsprung an beliebiger Stelle des Schleifenrumpfes: Der Rumpf der Schleife ist die Sequenz der Funktionen F1, F2, F3. Die Schleife ist kopfgesteuert. Vor einem evtl. Schleifendurchlauf wird die Sequenz der Funktionen F2, F3 ausgeführt. Der Aussprung am xor-Konnektor nach der Funktion F2 geschieht nicht am Ende des Schleifenrumpfes. Überdies fallen Schleifeneinsprung und -aussprung nicht zusammen.

Wir empfehlen, den Unterschied zwischen Branch-Alternativen und Schleifen auch graphisch durch verschiedene Verzweigungskonstrukte deutlich zu machen.

3.2 Statische Analyse

Zur Analyse übersetzen wir eine gegebene EPK in ein Boolesches Netz und spannen dieses Netz in einen „Testtreiber“ ein: Wenn die EPK mehrere Startereignisse besitzt, so muß man unterscheiden, ob der Prozeß bereits startet, sobald eines dieser Ereignisse eintritt, oder ob erst alle Ereignisse oder zumindest eine bestimmte zulässige Kombination von Ereignissen eintreten muß. Dieser Sachverhalt läßt sich explizit machen, indem wir in das Boolesche Netz eine zusätzliche Stelle *Start/Ziel* einführen. An diesen *Basispunkt* des Netzes schließen wir alle zugelassenen Kombinationen von Startereignissen durch

Boolesche Transitionen an. Analog führen wir die möglichen Zielereignissen durch geeignete Boolesche Transitionen an dieselbe Stelle *Start/Ziel* heran. Die Strukturanalyse prüft nun,

- ob jede Transition vom Basispunkt aus auf einem gerichteten Weg erreicht werden kann und ob von jeder Transition ein gerichteter Weg zum Basispunkt weiterführt (Starker Zusammenhang)
- und ob alle gerichteten Kreise, die den Basispunkt nicht enthalten, durch das sukzessive Anheften von Schleifen an jeweils einem Angelpunkt entstanden sind (Schleifenbaum).

Wir nennen diese inneren Schleifen die *Schleifenkomponenten* des Netzes und die ausgezeichnete Schleifenkomponente mit dem Basispunkt die *Basiskomponente*. Alle genannten Struktureigenschaften des Netzes lassen sich leicht mit Standardalgorithmen aus der Graphentheorie testen. Diese Prüfung nennt man die statische Netzanalyse.

3.3 Verhaltensanalyse

Die Verhaltensanalyse prüft eine EPK, welche die statische Netzanalyse bestanden hat, auf ordentliches Verhalten. EPKs, welche diese weitere Prüfung erfolgreich bestanden haben, heißen *wohlgeformt (well-formed)*. Bei allgemeinen Netzen muß diese Verhaltensanalyse in Form einer dynamischen Netzanalyse alle Schaltfolgen überprüfen, die von der Basismarkierung ausgehen. Wesentlich einfacher ist es jedoch im vorliegenden Falle einer EPK. Zum Start des Prozesses aktivieren wir den Basispunkt des Netzes mit einer 1-Marke und prüfen, ob das Netz unter dieser Basismarkierung von ordentlichem Verhalten ist. Dazu genügt es wiederum, daß jede Schleifenkomponente für sich betrachtet ein Boolesches Netz von ordentlichem Verhalten darstellt, wenn man sie an ihrem Angelpunkt abtrennt und die ausgezeichnete Stelle als Basispunkt mit einer 1-Marke markiert.

Man prüft eine gegebene Schleifenkomponente, indem man jeweils zusammengehörige Branch/Fork- und Merge/Join-Transitionen identifiziert und ihre Branch/Fork-Alternative gemäß Abbildung 2-5 in eine Schachtelung von Branch- und Fork-Alternativen auflöst. Bei diesem Schritt werden i.a. Netzteile dupliziert. Die Branch/Fork-Auflösung enthält nun ausschließlich Branch-, Merge-, Fork- bzw. Join-Transitionen. Netze dieser Art wurden in [GT1984] unter dem Namen *Bipolare Synchronisationsschemata* studiert. Genrich und Thiagarajan gaben einen Reduktionsalgorithmus an, mit dem sie ein solches Netz auf ordentliches Verhalten prüfen können, *ohne* den Fallgraphen zu betrachten: Statt der üblichen kombinatorischen *Explosion* des Fallgraphen wird eine *Reduktion* des Netzes auf ein triviales Netz durchgeführt. Durch diesen Algorithmus ist die dynamische Analyse einer EPK ebenfalls auf eine statische Netzanalyse zurückgeführt.

Geprüfte Boolesche Netze von ordentlichem Verhalten lassen sich in Stellen/Transitions-Netze mit gleicher Semantik übersetzen. Man ersetzt dazu jede Branch- und jede Merge-Transition durch ein Teilnetz, vgl. Abbildung 3-3, und läßt bei allen anderen Netzelementen die Beschriftung von Kanten und

Transitionen weg. Außerdem entfernt man alle 0-Marken. Das resultierende Netz ist ein Stellen/Transitions-Netz aus einer gut untersuchten Klasse von Petri-Netzen:

- Wohlgeformte EPKs sind sichere und lebendige Free-Choice Netze.

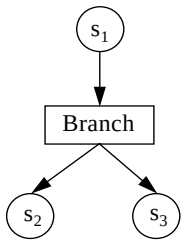
3.4 Netzanalyse der EPK Beschaffungslogistik

Wir führen die vollständige Netzanalyse am Beispiel der EPK *Beschaffungslogistik* von Abbildung 2-1 vor. Das aus der Übersetzung resultierende Netz enthält keine inneren Schleifen. Es ist stark-zusammenhängend und besitzt daher die bei der statischen Netzanalyse geforderten Eigenschaften.

Die nachfolgende Verhaltensanalyse des Booleschen Netzes erkennt zwei Merge/Join-Transitionen K12 und K7, die zu keiner Branch/Fork-Alternative gehören. Wir fassen Transition K12 als Gegenstück zur Branch-Transition K8 auf und ändern K12 daher ab in eine Merge-Transition. Die Merge/Join-Transition K7 soll eine Korrektur der Wareneingangsdaten im Falle negativ verlaufener Qualitätsprüfung modellieren. Es sind verschiedene Alternativen denkbar, diesen Fall in der Realität abzuwickeln. Wir entscheiden uns dafür, den Wareneingang zunächst unabhängig vom Ausgang der Qualitätsprüfung zu buchen. Sollte die nachfolgende Qualitätsprüfung einen Mangel an der Ware feststellen, so wird eine Korrekturbuchung durchgeführt. Die vorliegende EPK modelliert diesen Fall nicht, denn Transition K7 wartet auf jeden Fall das Ergebnis der Qualitätsprüfung ab. Wir führen daher eine weitere Funktion *Wareneingang.Buchen Korrektur* ein, und diese Funktion F95 ersetzt zugleich die Konnektoren K11 und K7. Als Konsequenz werden außerdem die beiden Join-Transitionen, die aus der Zerlegung des mehrfachen Konnektors K9 entstanden sind, durch eine einzige Join-Transition K90 ersetzt.

Das Netz enthält zwei geschachtelte Branch/Fork-Alternativen, die aus der Zerlegung der beiden mehrfachen Konnektoren K10 und K1 entstehen. Wenn wir zunächst für die äußere und danach für die innere Alternative die jeweilige Branch/Fork-Auflösung gemäß Abbildung 2-5 bilden, enthält das Boolesche Netz anschließend keine Branch/Fork- oder Merge/Join-Transition mehr.

Der anschließende Reduktionsalgorithmus von Genrich und Thiagarajan läßt sich veranschaulichen, wenn man sich überlegt, daß für die Wohlgeformtheit des Netzes allein die Booleschen Transitionen K100, K13, K90 und K200 maßgeblich sind. Die dazwischen liegenden Netzteile lassen sich jeweils auf eine Stelle reduzieren. Man erhält nach diesem Zwischenschritt das linke Netz von Abbildung 3-4, dessen Wohlgeformtheit man leicht überprüft. Der Algorithmus reduziert dieses Netz weiter bis zum rechten Netz in Abbildung 3-4 und erkennt dieses Netz als wohlgeformt. Damit ist die Verhaltensanalyse beendet: Das Boolesche Netz *Beschaffungslogistik* von Abbildung 2-3 ist sicher und lebendig.



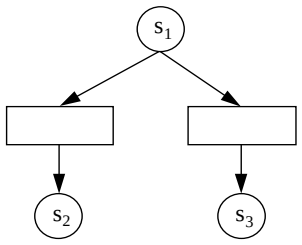


Abbildung 3-3 Substitution einer Branch-Transition

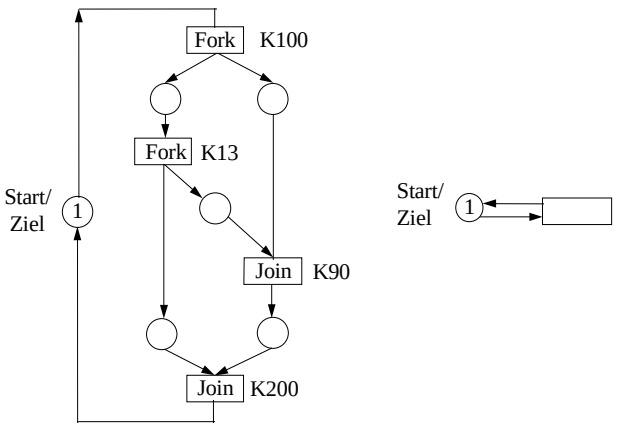


Abbildung 3-4 Zwischenschritt und Ergebnis der Reduktion nach Genrich/Thiagarajan

4 Literatur

[Bau1990] *Baumgarten, Bernd*: Petri-Netze. Grundlage und Anwendungen. BI-Wissenschaftsverlag, 1990

[BRS1995] *Becker, Jörg; Rosemann, Michael; Schütte, Reinhard*: Grundsätze ordnungsmäßiger Modellierung. Wirtschaftsinformatik, 37 (1995) 5, p. 435-445

[Brö1996] *Bröker, Axel*: Transformation ereignisgesteuerter Prozeßketten in Prädikat-Transitions-Netze sowie Vergleich der Modellierungstools ARIS-Toolset und INCOME. Diplomarbeit FH Pforzheim, Hochschule für Gestaltung, Technik und Wirtschaft. Pforzheim 1996

[Cod1972] *Codd, Edgar F.*: Relational completeness of Data Base Sublanguages. In *Courant Computer Science Symposia Series, Vol. 6, Data Base Systems*, Englewood Cliffs, N.J., Prentice-Hall 1972

[CS1994] *Chen, R.; Scheer, August-Wilhelm*: Modellierung von Prozeßketten mittels Petri-Netz Theorie. Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 107, Saarbrücken 1994

[GT1984] *Genrich, Hartmann; Thiagarajan, Pazhamaneri*: A Theory of Bipolar Synchronization Schemes. Theoretical Computer Science 30 (1984), p. 241-318

[Jen1992] *Jensen, Kurt*: Coloured Petri Nets. Basis Concepts, Analysis Methods and Practical Use. Springer 1992

[KNS1991] *Keller, Gerhard; Nüttgens, Markus; Scheer, August-Wilhelm*: Semantische Prozeßmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK)“. Veröffentlichungen des Instituts für Wirtschaftsinformatik, Heft 89, Saarbrücken 1991

[LSW1997] *Langner, Peter; Schneider, Christoph; Wehler, Joachim*: Ereignisgesteuerte Prozeßketten und Petri-Netze. Universität Hamburg, Fachbereich Informatik, Bericht Nr. 196, FBI-HH-B-196/97, 1997

[Rod1997] *Rodenhagen, Jörg*: Darstellung ereignisgesteuerter Prozeßketten (EPK) mit Hilfe von Petrinetzen. Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Hamburg 1997

[Sch1994] *Scheer, August-Wilhelm*: Wirtschaftsinformatik. Springer, Berlin u.a., 5. Auflage 1994

[SNZ1995] *Scheer, August-Wilhelm; Nüttgens, Markus; Zimmermann, Volker*: Rahmenkonzept für ein integriertes Geschäftsprozeßmanagement. Wirtschaftsinformatik 37 (1995), p. 426-434