8.3 Workflow-Netze

Workflow-Netze oder Workflow-Petrinetze umfassen eine der aktuell wichtigsten Klassen der Petrinetze, insbesondere im Bereich der Wirtschaftsinformatik. Im Folgenden werden die Hintergründe, Definitionen und wichtigsten Konzepte eingeführt.

8.3.1 Historie

Workflow-Systeme steuern und verwalten den Ablauf von Workflow-Prozessen (zu deutsch etwa „Geschäftsprozesse“). In den 60-er Jahren bestanden Informationssysteme aus einer Anzahl von Anwenderprogrammen (engl. application systems, APPL), die direkt auf das Betriebssystem (engl. operating system, OS) aufsetzten (vergl. Abb. 8.12). Für jedes dieser Programme wurde eine eigene Benutzungsschnittstelle und ein eigenes Datenbanksystem entwickelt. In den 70-er Jahren wurde die langfristige Datenhaltung aus den Anwendungsprogrammen (engl. Applications) ausgelagert. Zu diesem Zweck wurden Datenbanksysteme (engl. database management systems, DBMSs) entwickelt. In den 80-er Jahren fand mit den Benutzeroberflächen eine ähnliche Entwicklung statt. Benutzungsschnittstellensysteme (engl. user interface management systems, UIMSs) erlaubten es, die Kommunikation mit dem Benutzer aus dem Anwendungsprogramm auszulagern. Eine ähnliche Entwicklung setzte in den 90-er Jahren mit der Entwicklung von Workflow-Programmen (engl. workflow management systems, WFMSs) ein. Sie erlauben es, die Verwaltung von Geschäftsprozessen aus spezifischen Anwendungsprogrammen herauszuhalten (siehe z.B. [Aal00], [Aal98], [AH02]).

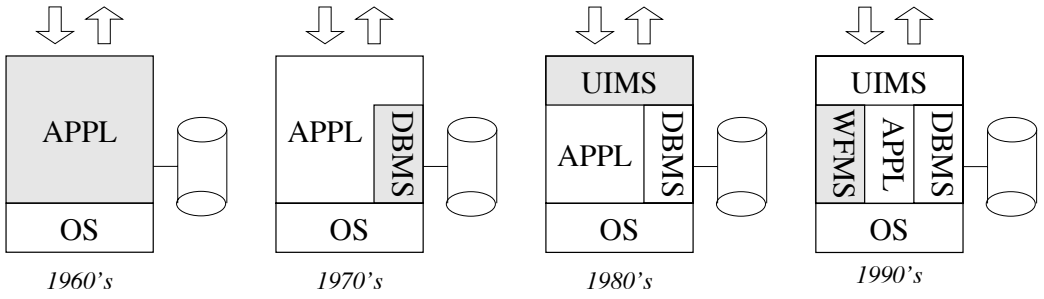

Abbildung 8.12: Workflow-Management-Systeme in historischer Perspektive

Die Aufgabe von Workflow-Systemen (manchmal auch *office logistics* genannt) ist es, sicherzustellen, dass die richtigen Aktionen eines Geschäftsablaufes durch die richtigen Personen zur richtigen Zeit ausgeführt werden. Abstrakter kann man ein Workflow-System als eine Software definieren, die Geschäftsabläufe vollständig modelliert, verwaltet und steuert.

Der Begriff Workflow-System wird häufig nicht so spezifisch gebraucht. Beispielsweise wird so auch Software zur Gruppenkommunikation genannt, die lediglich die Versendung

von Nachrichten und den Austausch von Information unterstützt (wie z.B. Lotus Notes, Microsoft Exchange). Die Abbildung 8.13 setzt allgemeine kooperative Prozesse und Workflow-Systeme in eine Skalierung zwischen informationszentriert und prozesszentriert bzw. weniger und mehr strukturiert.

8.3.2 Modellierung durch Workflow-Netze

Für Workflow-Systeme gibt es hunderte von Modellierungsansätze und Rechnerwerkzeuge, die jedoch häufig keine wohldefinierte und eindeutige Semantik besitzen. Wegen ihrer Orientierung auf Aktionen und deren Koordination bieten sich natürlich besonders Petrinetze an, deren Semantik wohldefiniert und bekannt ist. Ein solcher Ansatz nach [Aal00], [Aal98] und [AH02] wird hier vorgestellt.

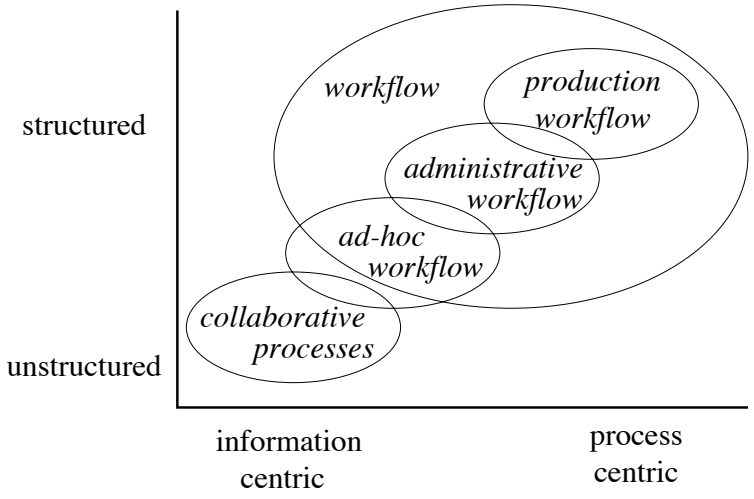

Abbildung 8.13: Workflow Prozesse und kooperative Prozesse im Vergleich

8.3.3 Intuitives Beispiel

Im Folgenden betrachten wir ein Workflowsystem zur Beschwerdebearbeitung (siehe Abbildung. 8.14). Es enthält folgende Aktionen:

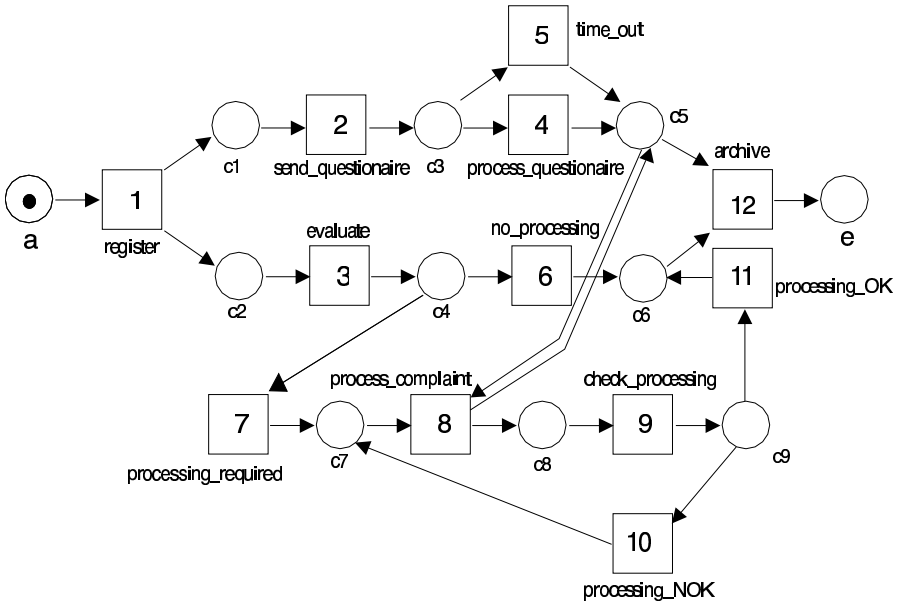

Abbildung 8.14: Ein Petrinetz für einen Vorgang zur Beschwerdebearbeitung

1. Aufnahme einer Beschwerde (register)
2. Fragebogen an Beschwerdeführer (send_questionnaire)
3. Bewertung (evaluate) (nebenläufig zu 2.)
4. Fragebogenauswertung (process_questonaire), falls Rücklauf innerhalb von 2 Wochen, sonst:
5. Nichtberücksichtigung des Fragebogens (time_out).
6. Je nach Ergebnis der Bewertung (3.): Aussetzung der Bearbeitung (no_processing) oder
7. Beginn der eigentlichen Prüfung (processing_required)
8. Bearbeitung der Beschwerde (process_complaint) unter Berücksichtigung des Fragebogens[2]
9. Bewertung der Bearbeitung (check_processing) mit dem Ergebnis
10. erneute Prüfung (processing_nok) oder
11. Abschluss (processing_ok)
12. Ablage (archive)

Auch Plätze haben Bedeutung: z. B. hat c_2 die Bedeutung: „Bewertung kann beginnen“.

[2]Nebenbedingung c_5

8.3.4 Definition

Im Folgenden werden nur einfache Netze betrachtet, d.h. die Kantenbewertung ist 1 auf F, also: $W(x,y) = 1 \iff (x,y) \in F$.
Es ist sinnvoll nur einen Anfangsplatz a und einen Endplatz e vorzusehen, da Anfang und Ende meist hervorgehobene Ereignisse sind. Dann sollten alle Transitionen auf Pfaden zwischen diesen liegen. Wir erhalten so folgende Normalform:

Definition 8.14 *Ein P/T-Netz* $\mathcal{N} = (P, T, F, \mathbf{m}_a)$ *heißt* Workflow-Netz *(WF-Netz), falls*

a) es zwei besondere Plätze $\{a, e\} \subseteq P$ *enthält mit* ${}^\bullet a = \emptyset$ *(„Start, Quelle, Anfangsplatz“) und* $e^\bullet = \emptyset$ *(„Ende, Senke, Endplatz“),*

b) alle Plätze und Transitionen auf Pfaden zwischen a *und* e *liegen und*

c) die Anfangsmarkierung $\mathbf{m}_a$ *durch* $[\mathbf{m}_a(p) :=$ **if** $p = a$ **then** 1 **else** 0 **fi**$]$ *sowie eine* Endmarkierung $\mathbf{m}_e$ *durch:* $[\mathbf{m}_e(p) :=$ **if** $p = e$ **then** 1 **else** 0 **fi**$]$ *festgelegt sind.*

Lemma 8.15 *In jedem Workflow-Netz sind Quelle* a *und Senke* e *eindeutig festgelegt.*

Beweis: Als Übung. □

Das Verhalten der Workflow-Netze ist eindeutig über das Verhalten der P/T-Netze festgelegt. Dies ist für die Fragen nach der jeweiligen Semantik von Bedeutung.

8.3.5 Grundlegende Strukturen in Workflow-Modellen

Der Aufbau von Workflow-Modellen erfolgt meist auf der Grundlage von einfachen und komplexeren Strukturfragmenten. Diese werden im Folgenden *Muster* (engl. „patterns“) genannt.[3]
Einfache Muster sind dabei die folgenden, typische Strukturen sind sequentielle, nebenläufige, wiederholende und alternative Bearbeitung (letztere mit und ohne expliziten Testbedingungen).

- Sequentielle und nebenläufige Bearbeitung (siehe Abb. 8.15)
- Iteration (siehe Abb. 8.16)
- Alternative Bearbeitung mit und ohne expliziten Testbedingungen (siehe Abb. 8.17)

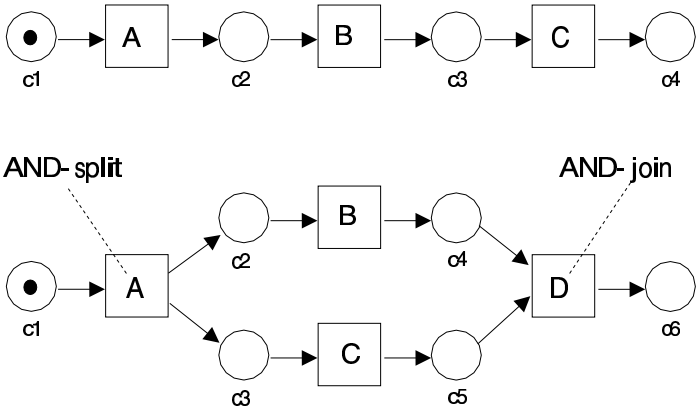

Abbildung 8.15: Sequentielle und nebenläufige Bearbeitung

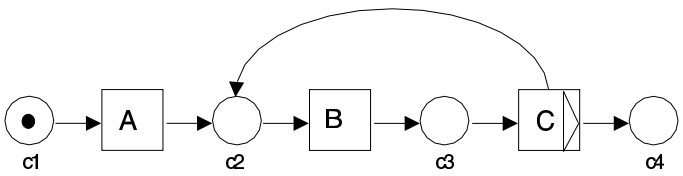

Abbildung 8.16: Iteration

[3]Unter `http://www.workflowpatterns.com/patterns` ist eine Sammlung solcher Strukturmuster für Workflows zu finden.

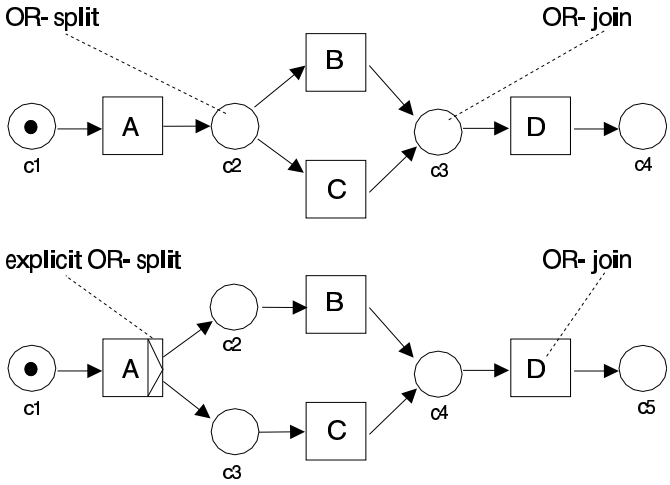

Abbildung 8.17: Alternative Bearbeitung mit und ohne expliziten Testbedingungen

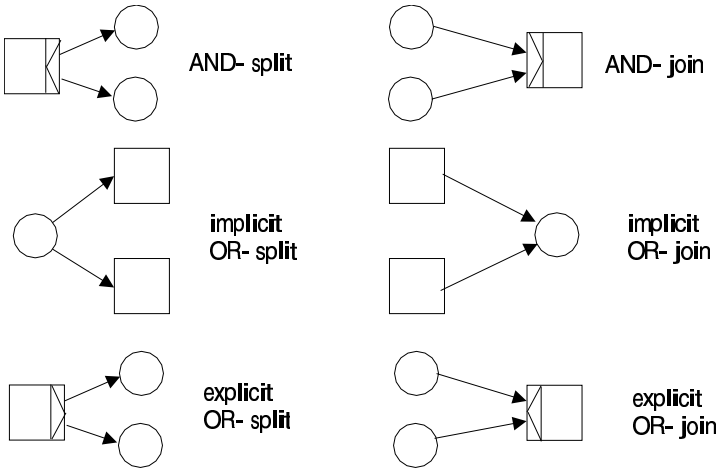

Abbildung 8.18: Graphische Symbole für Verzweigung

Dabei werden zum Teil vereinfachende graphische Symbole benutzt (siehe Abb. 8.18). Hier ist zu beachten, dass diese teilweise fundamental gegen das übliche Schaltverhalten der Transitionen verstoßen. Zur besseren Lesbarkeit in anwendungsnahen Kontexten ist dies aber eine sinnvolle Modellierung. Von entscheidender Bedeutung ist jedoch, dass die Semantik der jeweiligen Symbole den jeweiligen Verwendern / Lesern der Modelle bekannt ist. Die Diskussionen zu diesen Modellierungssprachen oder Modellierungstechniken finden sich in der hier insgesamt angegebenen Literatur.

Ein Workflow kann auch von der Interaktion mit der Umgebung abhängen (restriktives System, Trigger, Ressourcen), wie zum Beispiel bei den Ereignissen: „Bearbeiter ist in Urlaub oder krank“ oder „Antwort auf eine Anfrage trifft nicht ein“.

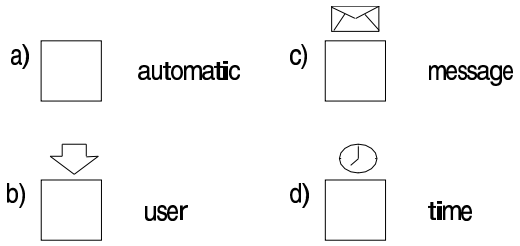

Abbildung 8.19: Trigger eines Workflowsystems

Es gibt folgende Arten von Triggern (Abbildung 8.19):

a) Automatisch (keine externe Eingabe notwendig).
b) Benutzer (user): ein Bearbeiter oder Benutzer oder eine Funktionseinheit nimmt einen Auftrag an und bearbeitet ihn.
c) Nachricht (message): eine Nachricht von außen wird benötigt (Brief, Anruf, e-mail, Fax).
d) Zeit (time): es besteht eine Zeitbedingung für die Bearbeitung.

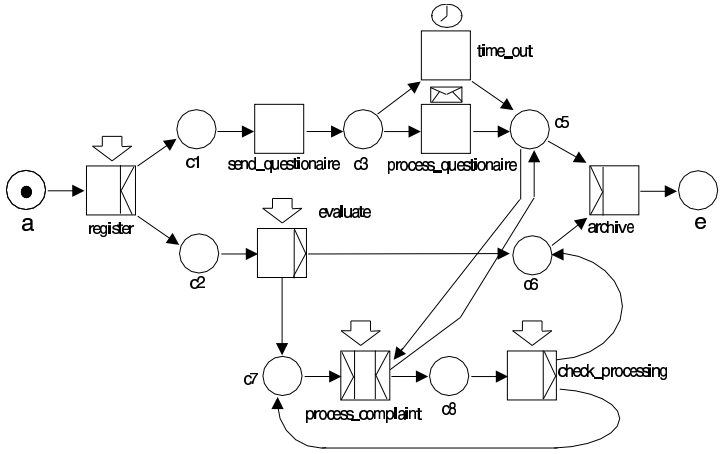

Abbildung 8.20: Workflow aus Abb. 8.14 mit Triggern

Es kann auch Wechselwirkungen zwischen Triggern und Verzweigungen geben (ersetze beispielsweise die implizite Verzweigung in $c3$ der Abbildung 8.20 durch explizite). Ein anderes Problem besteht darin, dass manchmal Zustandsinformation („milestones“) erforderlich ist (Beispiel: $c5$ in Abb. 8.20).

8.3.6 Korrektheit

Workflow-Systeme werden u.U. fehlerhaft entworfen, weil die abzubildenden Prozesse nicht richtig verstanden wurden. Es stellt sich die Frage, ob in diesem Sinne semantisch inkorrekte Entwürfe durch formale Methoden erkannt werden können.
Die Erfahrung aus der Praxis zeigt, dass ...

- Workflow-Prozesse oft nicht richtig verstanden werden (Mehrdeutigkeit, Widersprüche, Verklemmungen),
- allein schon die (versuchsweise) Modellierung durch Petrinetze Mängel aufdeckt und
- bei fertiggestellten Petrinetz-Modellen von Workflow-Systemen Mängel durch strukturelle Untersuchungen aufgedeckt oder durch Werkzeuge (automatisch) gefunden werden.

Beispiel 8.16 Wir betrachten dazu das nicht korrekte Workflow-System von Abbildung 8.21. Es zeigt zum Beispiel die folgenden Fälle von Fehlverhalten:

- Wenn die Transitionen 2 und 5 schalten, dann verbleibt bei Termination (d.h. wenn eine Marke in e ankommt) immer noch eine Marke in c_5. Das deutet darauf hin, dass ein Teilprozess nicht vollendet wurde. (Die Marke kann auch beim nächsten Durchlauf zu Fehlverhalten führen.)
- Wenn die Transitionen 3 und 4 schalten, dann verbleibt bei Termination eine Marke in c_4.
- Wenn die Transitionen 2 und 4 schalten, dann erreichen bei Termination 2 Marken den Endplatz e.

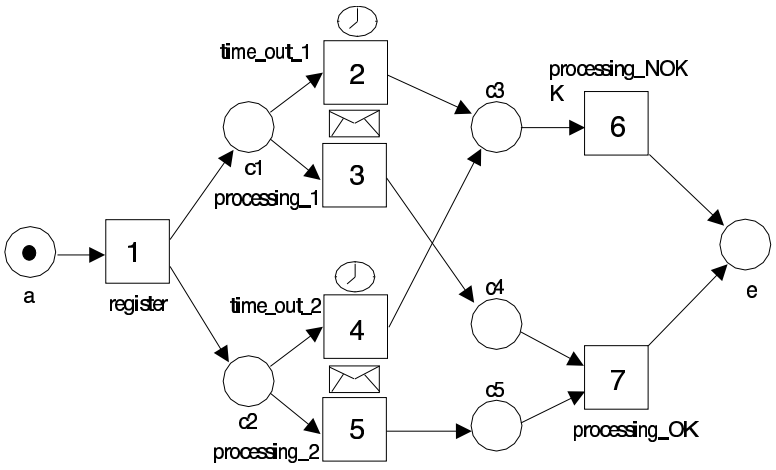

Abbildung 8.21: Problematisches Workflownetz für Beschwerdebearbeitung

Da fehlerhafte Workflow-Systeme oft aus inhaltlichen Missverständnissen entstehen, stellt sich die Frage, ob solche im Grunde semantische Fehler durch formale Kriterien vermieden werden können.

Die Definition 8.17 formalisiert dazu folgende drei Kriterien:

a) Aus jeder erreichbaren Markierung ist eine ordnungsgemäße Termination möglich.
b) Genau eine Marke in dem Endplatz e ist die einzige Möglichkeit zu terminieren.
c) Jede Transition kann in einer möglichen Schaltfolge schalten, denn sonst wäre sie nutzlos.

Definition 8.17 *Ein WF-Netz* $\mathcal{N} = (P, T, F, \mathbf{m}_a)$ *heißt* korrekt, *falls gilt:*

a) $\forall \mathbf{m} \in \mathbf{R}(\mathcal{N})\ \exists w \in T^* : \mathbf{m} \xrightarrow{w} \mathbf{m}_e$ *(Termination möglich)*
b) $\forall \mathbf{m} \in \mathbf{R}(\mathcal{N}) : \mathbf{m}(e) \geq 1 \Rightarrow \mathbf{m} = \mathbf{m}_e$ *(korrekte Termination)*
c) $\forall t \in T\ \exists \mathbf{m} \in \mathbf{R}(\mathcal{N}) : \mathbf{m} \xrightarrow{t}$ *(Nützlichkeit)*

Das problematische WF-Netz aus Abbildung 8.21 erfüllt Eigenschaft c) – aber weder a) noch b).

8.3.7 Abschluss eines WF-Netzes

Wir transformieren ein WF-Netz, indem wir eine neue Transition t^* zwischen e und a hinzufügen (vgl. Abbildung 8.22) und so das Netz „kurzschließen“:

Definition 8.18 *Für ein WF-Netz* $\mathcal{N} = (P, T, F, \mathbf{m}_a)$ *mit Anfangsplatz* a *und Endplatz* e *heißt* $\overline{\mathcal{N}} = (P, T', F', \mathbf{m}_a)$ *der* Abschluss *von* $\mathcal{N}$*, falls gilt:*

a) $T' := T \cup \{t^*\}$ *für eine neue Transition* $t^* \notin T$
b) $F' := F \cup \{(e, t^*), (t^*, a)\}$

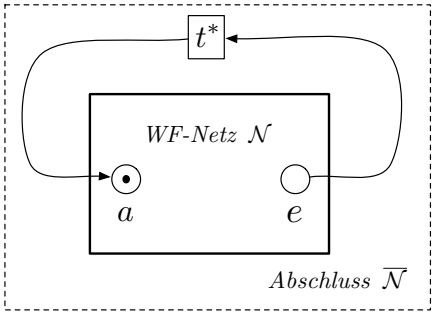

Abbildung 8.22: Transformation eines WF-Netzes: Hinzufügung einer neuen Transition t^*

Aufgabe 8.19 Zeigen Sie, dass für das WF-Netz $\mathcal{N}$ aus Abbildung 8.21 der Abschluss $\overline{\mathcal{N}}$ nicht beschränkt ist (z.B. dadurch dass $\mathbf{R}(\overline{\mathcal{N}})$ nicht endlich ist). Ebenso zeige man, dass $\overline{\mathcal{N}}$ nicht lebendig ist.

Satz 8.20 ([Aal97]) *Ein WF-Netz $\mathcal{N}$ ist genau dann korrekt, wenn sein Abschluss $\overline{\mathcal{N}}$ lebendig und beschränkt ist.*

Beweis: Als Übung. □

Aufgabe 8.21 Analysieren Sie das WF-Netz von Abb. 8.23 mit Hilfe des Verfahrens nach Satz 8.20.

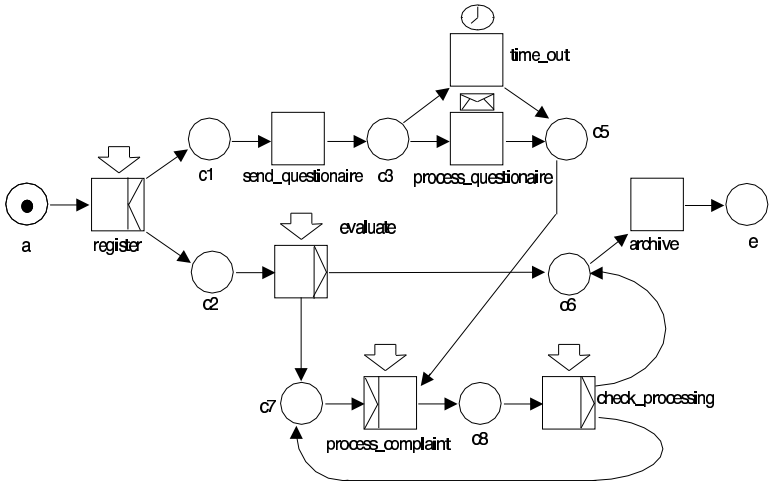

Abbildung 8.23: Ein WF-Netz zur Behandlung von Beschwerden

Die Eigenschaft „korrektes WF-Netz“ ist ein gutes Beispiel dafür, wie eine domänenspezifische Eigenschaft durch eine einfache Transformation auf eine basale Petrinetzeigenschaft zurückgeführt werden kann, die intensiv untersucht wurde und für die Software-Werkzeuge bereitstehen. Es handelt sich um die Eigenschaften „beschränkt“ und „lebendig“, die in der Tabelle 6.1 auf Seiten 105 definiert werden und zu denen im selben Kapitel Entscheidungsalgorithmen entwickelt werden. Beschränktheit bedeutet, dass es eine obere Schranke für die Anzahl der Marken auf den Plätzen gibt. Diese Eigenschaft ist damit äquivalent, dass die Erreichbarkeitsmenge $\mathbf{R}(\mathcal{N})$ endlich ist. Lebendigkeit bedeutet dagegen, dass von jeder erreichbaren Markierung $\mathbf{m} \in \mathbf{R}(\mathcal{N})$ jede Transition über eine geeignete Schaltfolge aktiviert werden kann. Intuitiv bedeutet dies, dass – egal wie sich das System verhält – nie Teile des Systems permanent ausfallen oder terminieren.

Umfangreiche weitere Literatur zu Workflows und angrenzenden Gebieten wie z.B. Process Mining finden sich auf der persönlichen Internetseite von Wil van der Aalst (siehe `http://wwwis.win.tue.nl/~wvdaalst/`).

8.4 Das Bankiersproblem: Sichere Zustände

Die Problematik von Verklemmungen bei der Betriebsmittelvergabe wurde von Dijkstra als Problem des Bankiers dargestellt [Dij68]. Hier wurde auch der Begriff des sicheren Zustands eingeführt.

Ein Bankier besitzt ein Kapital g. Seine n Kunden erhalten wechselnden Kredit. Jeder Kunde muss seinen maximalen Kreditwunsch von vornherein bekanntgeben und wird nur als Kunde akzeptiert, wenn dieser das Kapital nicht übersteigt. Kredite werden nicht entzogen. Dafür muss aber jeder Kunde versprechen, den Maximalkredit auf einmal nach endlicher Zeit zurückzuzahlen. Der Bankier verspricht, jede Bitte um Kredit in endlicher Zeit zu erfüllen. Für den Bankier besteht natürlich das Problem der Verklemmung: es kann sein, dass mehrere Kunden noch nicht ihren Maximalkredit erhalten haben, das Restkapital des Bankiers aber zu klein ist, mindestens einen Kunden total zu befriedigen, um dann nach einer Frist wieder neues Kapital zu haben.

Eine Instanz $\iota = (n,f,\mathrm{g})$ des Bankiersproblems besteht aus einer Zahl $n \in \mathbb{N}$, einem n-Tupel $f = (f_1, \ldots, f_n)$ und einer Zahl $g \in \mathbb{N}$. Ein Zustand einer solchen Instanz ist ein n-tupel $r = (r_1, \ldots, r_n)$ das die Restforderung der Kunden beschreibt. Anfangs gilt $r = f$. Ein Zustand heißt *sicher*, wenn er nicht notwendig zu einer Verklemmung führt.

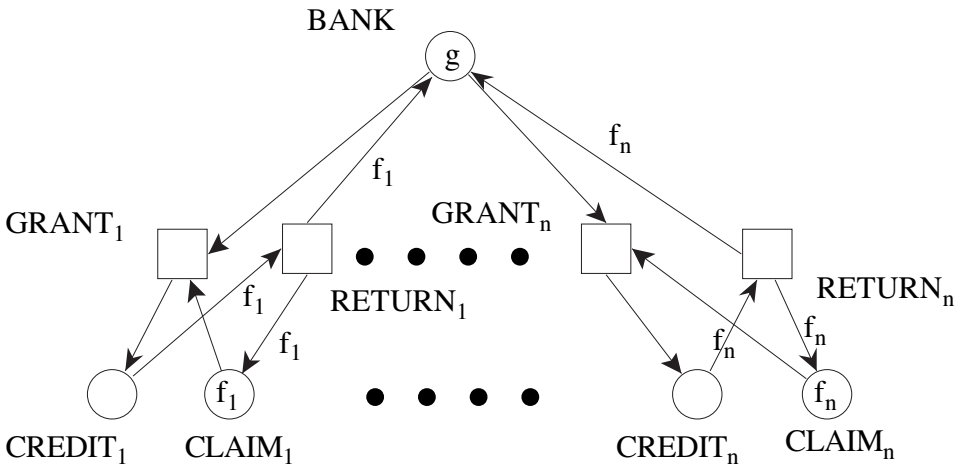

Abbildung 8.24: Das Bankiersproblem mit n Kunden

Das P/T-Netz in Abb. 8.24 modelliert das beschriebene Bankiersproblem. Der Platz *BANK* enthält so viele Marken wie das Kapital des Bankiers in Geldeinheiten umfasst. $CREDIT_i$ und $CLAIM_i$ stehen für Kredit und Restforderung. Durch die Transition $GRANT_i$ erhält der Kunde i so viele Geldeinheiten, wie sie schaltet. $RETURN_i$ transferiert das Geld zurück. Diese Transition kann nur schalten, wenn der Bankier die Maximalforderung f_i des Kunden erfüllt hat. Gleichzeitig wird die Ausgangsforderung wieder hergestellt.

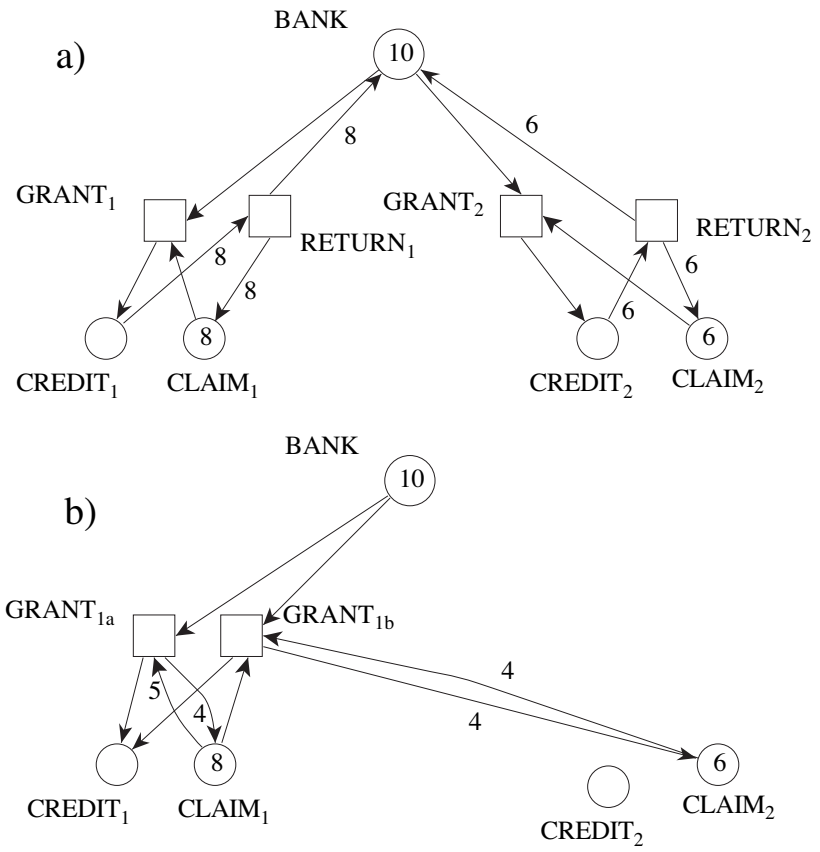

Abbildung 8.25: Eine Bankiersprobleminstanz mit 2 Kunden und Modifikation zur Vermeidung von Verklemmungen

Betrachten wir zwei spezielle Instanzen, nämlich $\iota_1 = (2, (8, 6), 10)$ (Abb. 8.25) und $\iota_2 = (3, (8, 3, 9), 10)$ (Abb. 8.27). An ihnen werden Erreichbarkeitsgraph und P-Invarianten erläutert.

Für die Instanz $\iota_1 = (2, (8, 6), 10)$ in Abb. 8.25a wird jeder Zustand durch eine Markierung dargestellt, d.h. durch einen 5-dimensionalen Vektor. Für alle erreichbaren Markierungen $\mathbf{m}$ gelten die folgenden drei Gleichungen:

- $\mathbf{m}[BANK] + \mathbf{m}[CREDIT_1] + \mathbf{m}[CREDIT_2] = 10$
  (Das Geld ist bei der Bank oder bei den Kunden und zwar genau 10 Einheiten insgesamt.)
- $\mathbf{m}[CLAIM_1] + \mathbf{m}[CREDIT_1] = 8$
  (Kredit und Restforderung des Kunden 1 beträgt zusammen immer genau 8 Einheiten.)
- $\mathbf{m}[CLAIM_2] + \mathbf{m}[CREDIT_2] = 6$
  (Kredit und Restforderung des Kunden 2 beträgt zusammen immer genau 6 Einheiten.)

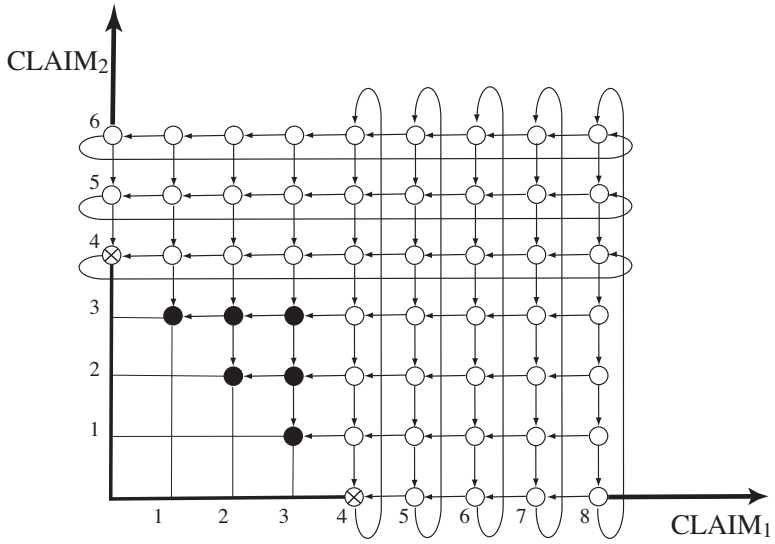

Abbildung 8.26: Erreichbarkeitsgraph des Netzes 8.25

(Sie heißen P-Invarianten-Gleichungen und werden im nächsten Abschnitt 7.5 systematisch behandelt.) Also ist jede erreichbare Markierung schon durch 2 ihrer Komponenten festgelegt, nämlich $(CLAIM_1, CLAIM_2)$. Die anderen 3 Komponenten, $BANK, CREDIT_1$ und $CREDIT_2$ können mit den Gleichungen daraus berechnet werden. Dadurch kann der Erreichbarkeitsgraph in einem ebenen Gitter dargestellt werden (Abb. 8.26).

Die Anfangsmarkierung $\mathbf{m}_0 = (10, 0, 8, 0, 6)$ (wobei die Plätze folgendermaßen geordnet seien: ($BANK$, $CREDIT_1$, $CLAIM_1$, $CREDIT_2$, $CLAIM_2$)) wird auf das Paar $(\mathbf{m}_0(CLAIM_1), \mathbf{m}_0(CLAIM_2)) = (8, 6)$ reduziert. Es entspricht dem Knoten rechts oben im Graphen von Abb. 8.26.

Alle Pfade, die in diesem Knoten anfangen, entsprechen Schaltfolgen. Kanten nach links, rechts, unten und oben entsprechen jeweils dem Schalten der Transition $GRANT_1$, $RETURN_1$,$GRANT_2$ und $RETURN_2$. Werden die Kunden strikt nacheinander bedient, so gibt es keine Probleme. Falls sich jedoch die Ausleihschritte überlappen, kann einer der drei Verklemmungen (1,3),(2,2) und (3,1) kommen.

Dijkstra hat darauf hingewiesen, dass es noch andere kritische Zustände gibt, nämlich diejenigen, die unvermeidlich zu einer Verklemmung führen, wie z.B. (3,3). Er nannte sie *unsicher*. Sie sind als schwarze Knoten dargestellt. In [HV87] und [VJ85] wurde gezeigt, dass *sichere Zustände* (weiße Knoten) durch ihre *minimalen Elemente* darstellbar sind: (0,4) and (4,0), (mit Kreuz).

Wie kann man unsichere Zustände vermeiden? Der Algorithmus von Dijkstra berechnet vor jedem Ausleihschritt, ob von dem dann erreichten Nachfolgezustand der Anfangszustand noch erreichbar ist.

Wie aus Abb. 8.26 ersichtlich kann dies auch dadurch geschehen, dass Transition $GRANT_1$ nur in Markierungen aktivierbar ist, die (in mindestens einer Komponente) größer als $(4, 0)$ oder $(0, 4)$ sind. Dies wird erreicht, wenn man die Transition $GRANT_1$ durch zwei modifizierte Kopien $GRANT_{1a}$ und $GRANT_{1b}$ (Abb. 8.25b)[4] ersetzt. Diese beiden Transitionen haben den gleichen Schalteffekt wie die ursprüngliche, aber einen höheren Aktivierungs-„Schwellwert“. Die entsprechende Konstruktion ist bei $GRANT_2$ anzuwenden. Der allgemeine Algorithmus ist in [VJ85] und teilweise in [JV87] S.216 ff. zu finden.
Die zweite Instanz $\iota_2 = (3, (8, 3, 9), 10)$ ist ein Beispiel aus dem Buch [BH73] über Betriebssysteme. Seine Netzdarstellung befindet sich in Abbildung 8.27. Es enthält sieben

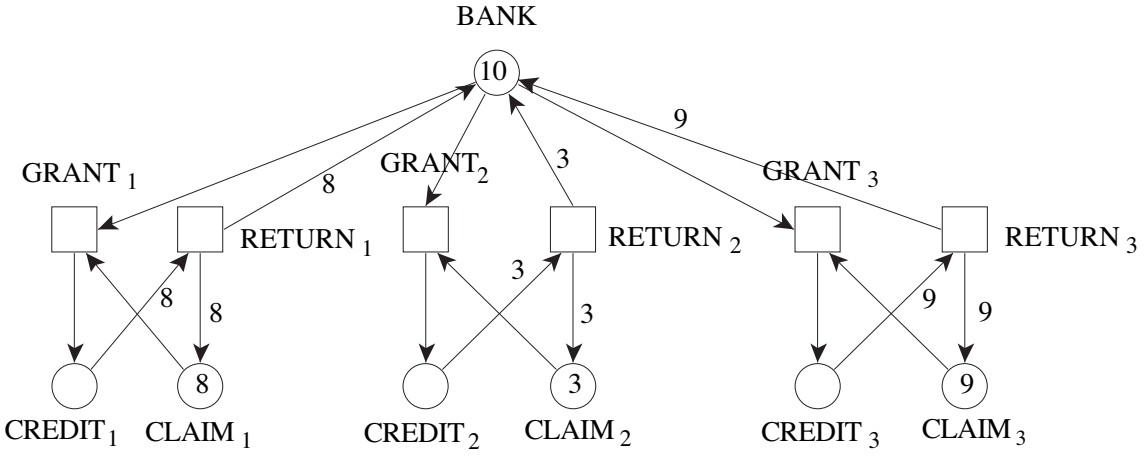

Abbildung 8.27: Eine Instanz des Bankiersproblems mit drei Kunden

Plätze. Wegen der nun vier Gleichungen kann der Erreichbarkeitsgraph in drei Dimensionen dargestellt werden (siehe dazu auch Abb. 8.28). Bezeichnend ist das Anwachsen seiner Größe. Er enthält 195 erreichbare Markierungen. Die Teilmenge von 137 sicheren Markierungen (weiße Knoten) wird von 10 minimalen Elementen (dem **Residuum**: weiße Knoten mit Kreuz)) abgegrenzt. Das Residuum als Charakterisierung der kleinsten Menge der noch gerade sicheren Zustände liefert ein wichtiges Kriterium für die sinnvollerweise erlaubten Handlungen in einem System. Eine allgemeine Methode zu ihrer Berechnung wird in [HV87] gegeben. Verlässt man diesen Bereich, kommt man zwangsweise zu im Allgemeinen unerwünschten (partiellen oder totalen) Verklemmungen: den *unsicheren Markierungen*. Die 58 unsicheren Markierungen sind wieder schwarz dargestellt.
Die zweite und größere Instanz hat aber auch Eigenschaften, die in der ersten nicht zu beobachten waren: sie enthält Markierungen, von denen aus Verklemmungen vermieden werden können, die aber nicht sicher sind, wenn sicher heißt, dass alle Kunden ihre Transaktionen beenden können. Beispielsweise kann von der Markierung $(4, 3, 6)$ ausge-

[4]Diese Abbildung zeigt nur, wie $GRANT_1$ durch zwei Transitionen zu ersetzen ist. Entsprechendes muss auch auf $GRANT_2$ angewandt werden, nicht jedoch auf $RETURN_1$ und $RETURN_2$.

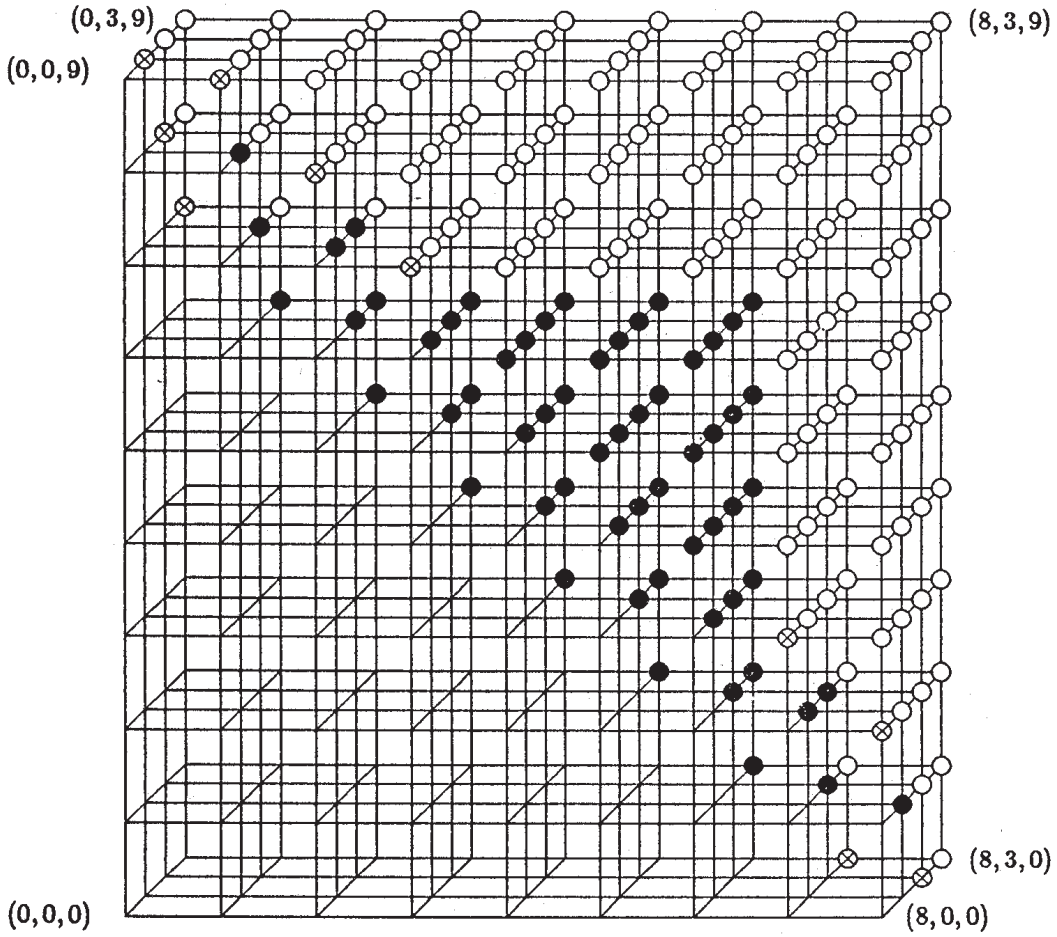

Abbildung 8.28: Erreichbarkeitsgraph des Netzes 8.27

hend, der zweite Kunde beliebig viele Transaktionen ausführen, während die Kunden 1 und 3 nicht einmal einen Ausleihvorgang vollständig zu Ende bringen können. Solche Situationen heißen *partielle Verklemmung*. Ein Netz ohne partielle Verklemmungen heißt *lebendig*. Lebendigkeit wird im Abschnitt 7.1.3 behandelt.

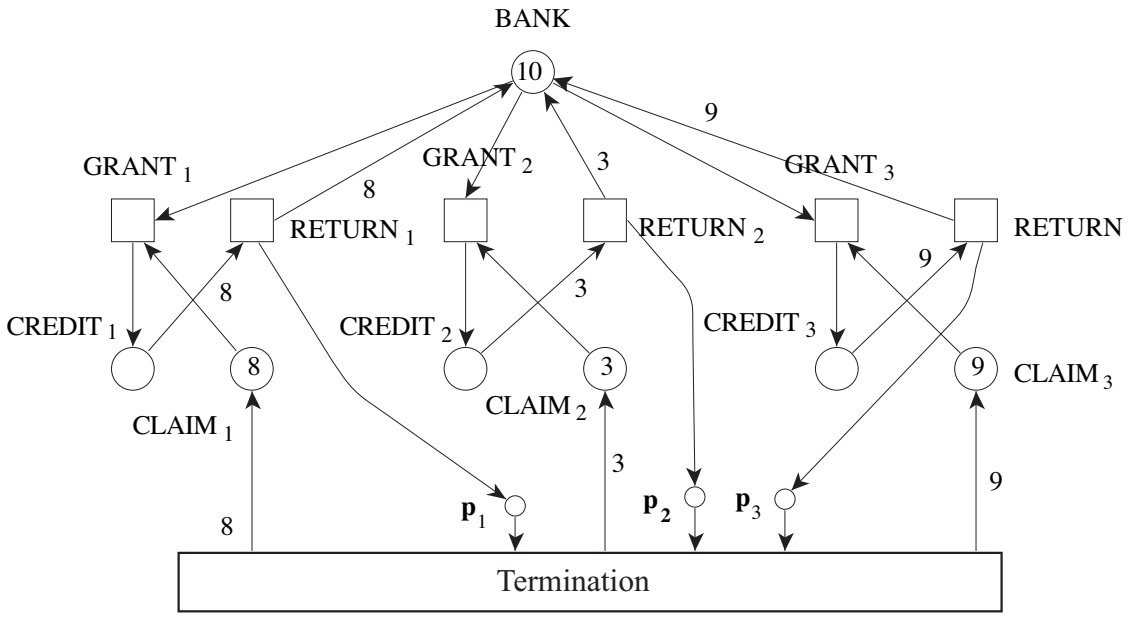

Abbildung 8.29: Bankiersnetz mit Terminationstransition

Um die ursprüngliche Definition von Dijkstra beizubehalten, kann man wie in Abb. 8.29 eine Transition *Termination* einführen, die dafür sorgt, dass alle Kunden einen Ausleihvorgang beendet haben, bevor eine neue Runde started. Die Einschränkungen der möglichen Abläufe ergibt sich aus der expliziten Synchronisation. Um den Begriff der sicheren Markierung sinnvoll auch in dem Netz 8.27 zu benutzen, könnte man folgende Definitionen unter c) benutzen, die eine sinnvolle Übertragungen des Begriffs der Sicherheit auf Netze liefern:

a) Eine Markierung **m** heißt sicher, wenn eine spezifizierte Endmarkierung (hier die Anfangsmarkierung) wieder erreichbar ist.
b) Eine Markierung **m** heißt sicher, wenn eine bestimmte Transition (z.B. *Termination*) zum Schalten gebracht werden kann.
c) Eine Markierung **m** heißt sicher, wenn von ihr aus eine unendliche Schaltfolge möglich ist, die alle Transitionen des Netzes unendlich oft enthält.

Ist in a) die Anfangsmarkierung gemeint, dann wird diese Eigenschaft in der Petrinetzliteratur auch „home property“ genannt. Die Eigenschaft c) hat mit dem „fairen Verhalten“ eines Netzes zu tun. Fairness und Lebendigkeit stehen in komplexen Beziehungen zueinander.

8.5 Das Renew-Werkzeug

Das am Arbeitsbereich TGI in Hamburg entwickelte Werkzeug RENEW [KWD] ermöglicht die graphische Darstellung der hier behandelten Netzmodelle. Unter `http://www.Renew.de` findet sich neben den Installationsdateien und -hinweisen sowie dem vollständigen Quell-Code der Software auch ein Handbuch, das die folgenden Beschreibungen ausführlich ergänzt. Im Folgenden werden anhand einiger Bildschirmdarstellungen zentrale Konzepte und die Nutzung von RENEW erläutert.
Die RENEW-Werkzeugleiste ist in der Abb. 8.30 dargestellt.

Abbildung 8.30: Werkzeugleiste des RENEW-Werkzeugs

RENEW kann auch für die Ausführung der erzeugten Netze benutzt werden. Als Beschriftungssprache wurden Elemente der Programmsprache Java [GJS97] gewählt, die in vielen ihrer Eigenschaften Petrinetze sinnvoll ergänzt. Das Werkzeug ist selbst in Java geschrieben und durch Transitionen können Java-Klassen ausgeführt werden. *Ausführung* wird in der Petrinetzliteratur auch als *Simulation* bezeichnet. Dies bedeutet die Simulation des formalen Modells und nicht eines realen Weltausschnittes. Letzteres gilt natürlich auch, wenn das Petrinetz den realen Weltausschnitt hinreichend genau darstellt. Dazu können auch Zeitschranken für das Schalten benutzt werden. Eine Übersicht zu Petrinetz-Werkzeugen gibt die Internetseite *The Petri Nets World* [Pet]. Über sie sind auch Informationen zu Literatur zu Petrinetzen, Forschungsgruppen, -aktivitäten, -projekte und Anwendungen von Petrinetzen zugänglich.

Die Abbildung 8.31 zeigt ein P/T-Netz in RENEW (nach [Kum01]). Seine drei Marken

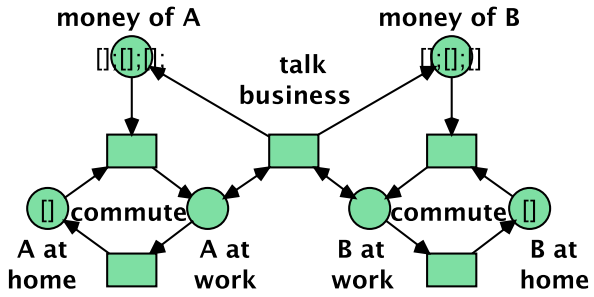

Abbildung 8.31: Das Leben zweier Geschäftsleute

im Platz `money of A` werden als `[];[];[]` dargestellt. Dieses Netz kann folgendermaßen interpretiert werden: zwei Geschäftsleute A und B können jeweils von zu Hause (`at home`) zum Arbeitsplatz (`at work`) wechseln. Die Fahrt kostet Geld. Dieses (und mehr) kann jedoch beim gemeinsamen Handel (`talk business`) wieder verdient werden. Abstrakt gesehen handelt sich hier also um zwei Betriebsmittel verbrauchende und erzeugende Funktionseinheiten.
Durch Faltung erhält man das gefärbte Netz in Abbildung 8.32. Geschäftsleute werden

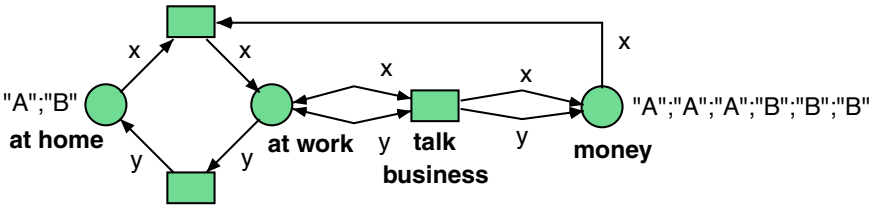

Abbildung 8.32: Faltung von Netz 8.31 zu einem gefärbten Netz

durch *individuelle Marken* `"A"` und `"B"` dargestellt, ebenso wie ihr jeweiliges Guthaben im Platz `money`.

Wird eine Darstellung mit Bezeichner und Wert gewünscht, so kann man wie in Abb. 8.33 vorgehen. Die Kantenbeschriftungen sind hier 2-Tupel (in JAVA-Notation), deren erste

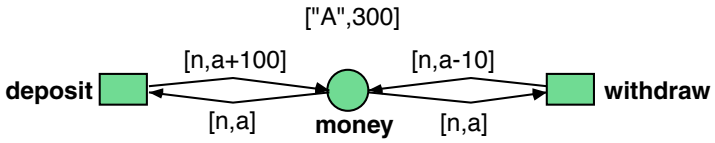

Abbildung 8.33: Ein nicht ganz so einfaches Bankkonto

Komponente den Kontoinhaber und deren zweite Komponente den Wert oder den zu verändernden Wert darstellen.

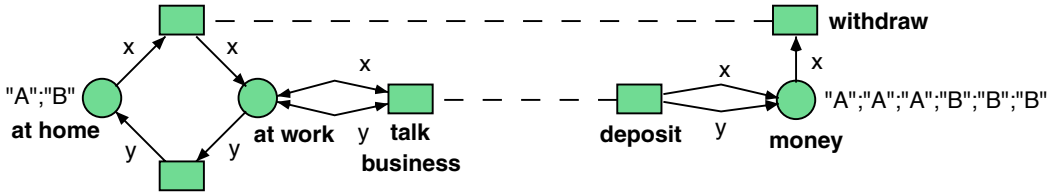

Abbildung 8.34: Personen und Konten werden geteilt

Ein ernst zu nehmender Einwand gegen das gefärbte Netz aus Abb. 8.32 ist, dass die Bewegung von Geld und die Bewegung der Personen zu stark miteinander verwoben sind. Auf den ersten Blick ist es nicht klar, welche Teile des Netzes welchen Aspekt beschreiben.
In Abb. 8.34 bereiten wir eine Teilung des Systems aus Abb. 8.32 vor. Zwei der Transitionen werden doppelt in verschiedenen Teilen des Netzdiagramms aufgezeichnet. Um die Schaltsemantik des Netzes korrekt beizubehalten, müssten die Transitionen wieder verschmolzen werden.

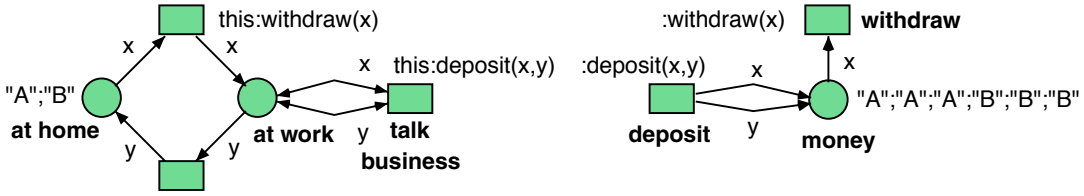

Abbildung 8.35: Textuelle Anschriften kennzeichnen Synchronisation

Wir gehen einen Schritt weiter und deuten die beabsichtigte Bedeutung des Netzes durch eine textuelle Anschrift an. In Abb. 8.35 bedeutet die Anschrift `this:withdraw(x)`, dass in diesem Netz (Englisch *this net*) eine andere Transition vorhanden sein sollte, die die

Auszahlung von Geld vom Konto von `x` übernehmen kann. Wir geben dabei die Variable am Ende der Anschrift an, um klarzumachen, welche Information umhergereicht werden muss.
Solche Anschriften werden als synchrone Kanäle bezeichnet, weil sie die Transitionen zwingen, synchron zu schalten und weil sie den Fluss von Informationen kanalisieren. Die Autoren von [CDH92] haben dieses Konzept für höhere Petrinetze eingeführt. Gegenüber gewöhnlichen Petrinetzen fügen synchrone Kanäle die Fähigkeit hinzu, über gleichzeitige, nicht nur über aufeinanderfolgende oder über nebenläufige Handlungen zu sprechen (Rendez-Vous-Synchronisation).

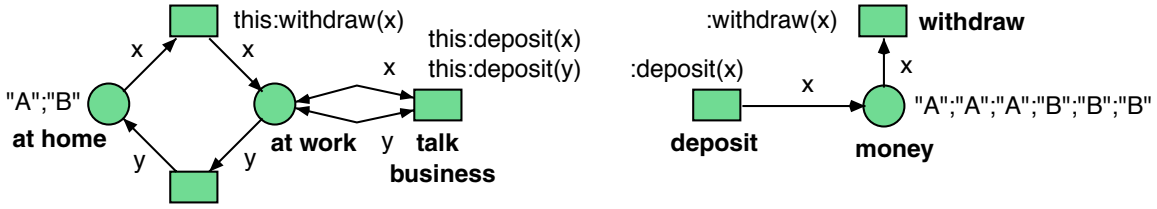

Abbildung 8.36: Wiederverwendung eines Kanals

Weil wir textuelle Anschriften für die Kanäle verwendet haben, können wir mehrere Synchronisationen gleichzeitig anfordern, ohne dass das Diagramm seine Klarheit verliert. In Abb. 8.36 wurde das Netz aus Abb. 8.35 so umstrukturiert, dass von der Transition `talk business` zwei Aufrufe des Kanals `deposit` getätigt werden. Dies veranlasst die Transition `deposit`, zweimal zu schalten, was das gewünschte Verhalten in unserem Beispiel ist.

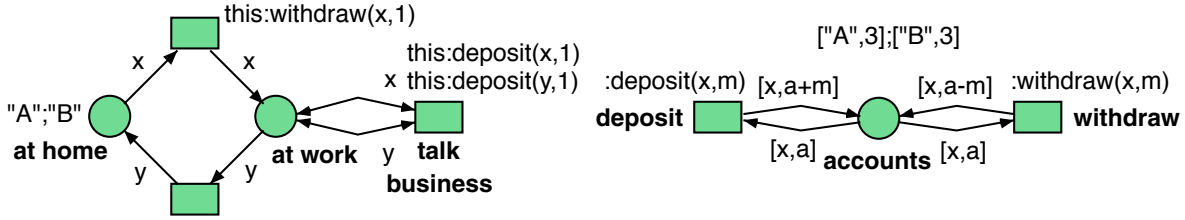

Abbildung 8.37: Buchführung mit Algebra

Jetzt können wir die Lösung aus Abb. 8.36 mit dem gefärbten Netz zur Modellierung eines Bankkontos aus Abb. 8.33 kombinieren und jedes Konto als Wertepaar repräsentieren. Abb. 8.37 zeigt das Resultat. Beachtenswert ist, wie die Anzahl der Geldeinheiten, die ein- oder auszuzahlen ist, als zweiter Parameter über den Kanal übergeben wird.

Üblicherweise werden die beiden Netze in zwei verschieden Fenstern wie in Abbildung 8.38 dargestellt.

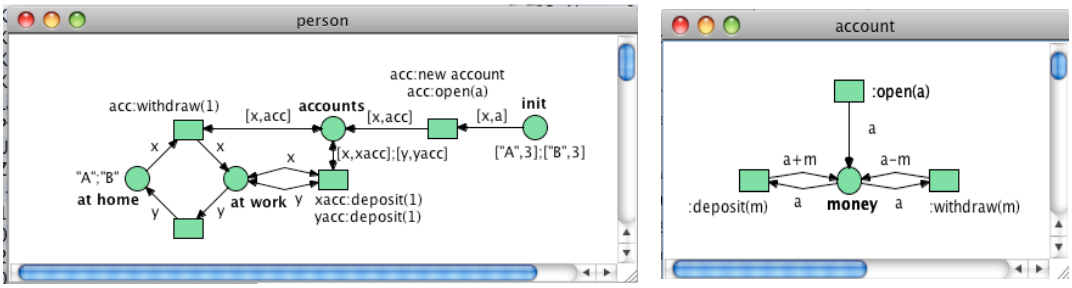

Abbildung 8.38: RENEW: Buchführung mit Algebra in zwei Fenstern

Die Abbildung 8.39 zeigt das Netz `person` nach einem Simulationsschritt.

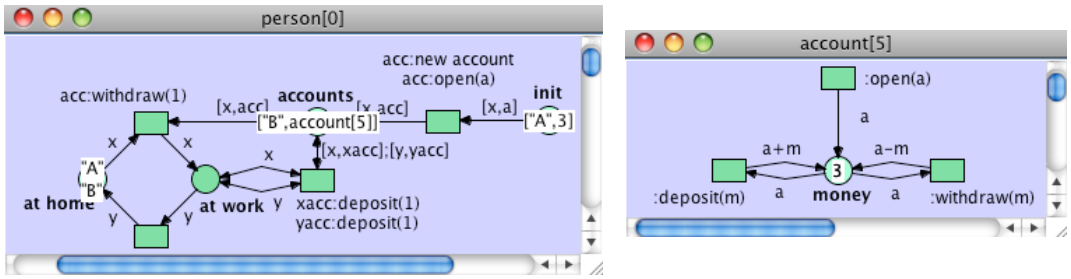

Abbildung 8.39: RENEW (Simulationsmodus): Buchführung mit Algebra in zwei Fenstern

Beim Schalten der Transition rechts oben wurde mit der Bindung $[x = "B", a = 3]$ ein Exemplar `account[5]` des Musters `account` erzeugt und an die Variable `acc` gebunden. Dadurch wird die Bindung zu $[x = "B", a = 3, acc = account[5]]$ erweitert. Im Platz `accounts` verfügt nun der Geschäftsmann `"B"` über eine Referenz auf sein Konto `account[5]` mit Inhalt `3`. Beim nochmaligen Schalten dieser Transition würde entsprechendes für den Geschäftsmann `"A"` gelten, natürlich mit der Referenz auf ein eigenes Exemplar von `account`. Diese Fähigkeit von RENEW Exemplare von Mustern zu bilden und die Referenzen wie Marken zu behandeln ist eine über übliche gefärbte Netze hinausführende Eigenschaft, die aber nicht Gegenstand dieser Vorlesung ist.

8.6 Das Alternierbitprotokoll

Das *Alternierbitprotokoll* steht für die gesamte Klasse der Kommunikationsprotokolle. Diese Klasse ist stark aktionsorientiert. Das entwickelte Netz stellt das Bit und die Daten als Marken eines gefärbten Netzes dar.

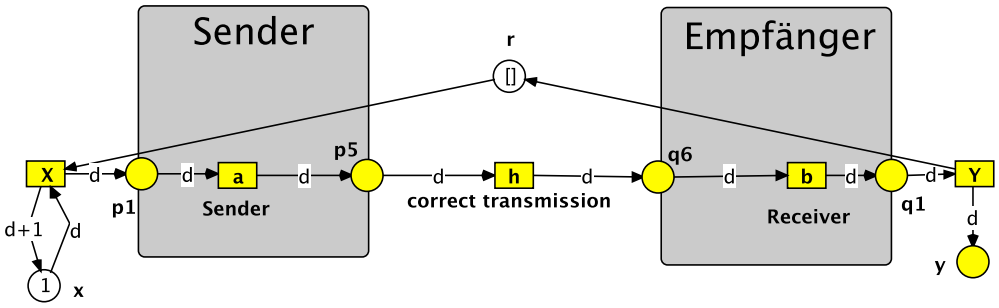

Abbildung 8.40: Spezifikation abp-1 des Alternierbitprotokolls

In diesem Beispiel wird das *Alternierbitprotokoll* als gefärbtes Netz modelliert. Für die Übermittlung einer Nachricht zwischen zwei Arbeitsrechnern (hosts) stehe ein Kanal zur Verfügung, auf dem in beiden Richtungen, aber nur in einer Richtung zur Zeit, eine Nachricht übermittelt werden kann (Halbduplex-Kanal). Bei der Übermittlung kann die Nachricht gestört werden. Durch redundante Kodierung wird aber jeder Fehler erkannt und angezeigt. Das Alternierbitprotokoll soll diese Fehler durch redundante Übermittlung verdecken und so den Benutzern einen fehlerfreien Kanal zur Verfügung stellen. Dazu muss natürlich vorausgesetzt werden, dass der Kanal nicht dauerhaft gestört bleibt, sondern immer wieder einmal ein Datum korrekt übermittelt.

Im folgenden wird das Protokoll schrittweise entwickelt. Zur Einführung in die Systemumgebung ist in Abbildung 8.40 die ungestörte Übermittlung von Daten von einem „Sender“ zu einem „Empfänger“ dargestellt. Die Transition `X` modelliert eine Funktionseinheit (z.B. eine Person, ein Prozessor), die Daten `d` zur Übermittlung an den Sender übergibt. Um dies in Renew ausführbar zu machen, sind dies hier die Werte $1, 2, 3, \ldots$. Das Protokoll soll jedoch für beliebige Daten korrekt arbeiten, d.h. das Protokoll darf nicht auf den Inhalt der Daten zugreifen, um z.B. die Folge als Sequenznummern zu benutzen. Einige oder alle Daten können auch den gleichen Wert haben. Diese Daten werden über die Transittionen `a`, `h` und `b` korrekt übermittelt, damit sie eine Funktionseinheit `Y` (z.B. eine Person, ein Prozessor) empfangen kann. Der mit einer anonymen („schwarzen“) Marke `[]` markierte Platz `r` dient nur dazu, dass die Plätze der mittleren Reihe maximal eine Marke enthalten. Dadurch ist gewährleistet, dass die Daten im Ausgangskanal `q1` in der gleichen Reihenfolge erscheinen wie im Eingangskanal `p1`. Darüberhinaus wird für das Protokoll gefordert, dass nicht eher ein $n+1$-ter Wert in `p1` erscheint als in `q1` der Wert n beobachtet wurde. Andere Protokolle erlauben hier eine beschränkte Überlappung. Dies ist eine Weiterentwicklung, die zur Vereinfachung hier

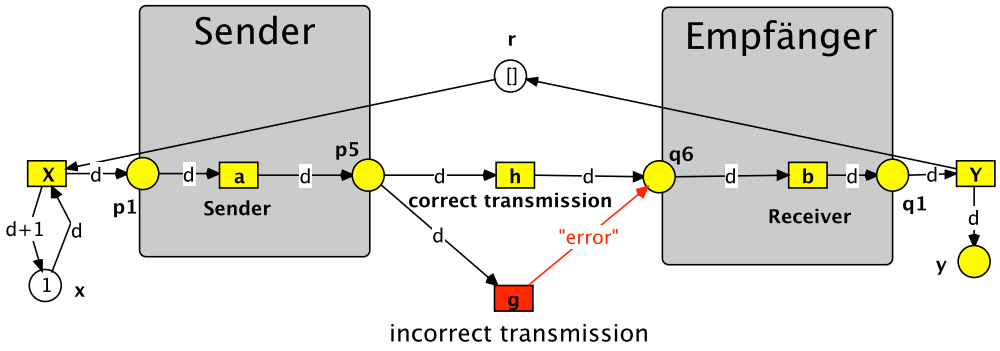

Abbildung 8.41: Übermittlung mit Fehlern: Netz abp-2

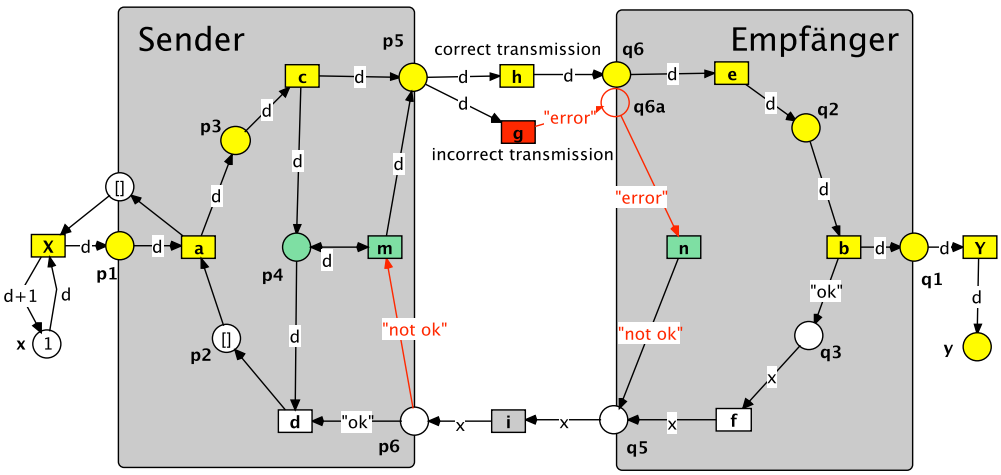

Abbildung 8.42: Übermittlung mit Quittung: Netz abp-3

nicht betrachtet wird. Das Netz abp-1 kann also als Spezifikation der Aufgabenstellung angesehen werden.

Im Netz abp-2 (Abb. 8.41) kann der Kanal Daten fehlerhaft übermitteln. Dies wird durch das alternative Schalten der Transition `g` modelliert, die die Fehlermeldung `"error"` erzeugt. Dies muss in Anwendungen durch unterliegende Protokollschichten implementiert werden, bzw. durch ein *timeout* im Fall von Datenverlust. Das Netz abp-2 hat ein von der Spezifikation abp-1 verschiedenes Verhalten.

Ein der Spezifikation entsprechendes Verhalten wird durch das Netz abp-3 durch die Rücksendung einer positiven (`"ok"`) oder negativen (`"not ok"`) Quittung erreicht. Bei einer negativen Quittung wird das im Platz `p4` gespeicherte Datum solange wieder versandt, bis eine positive Quittung erfolgt. Das Netz erfüllt jedoch nicht die Spezifikation, wenn auch die Rücksendung über die Transition `i` fehlerhaft ist.

Zur Vorbereitung der endgültigen Lösung wird mit abp-4 ein zu abp-3 verhaltensäquivalentes Protokoll eingeführt. Dem Datum `d` wird durch die Transition `a` ein

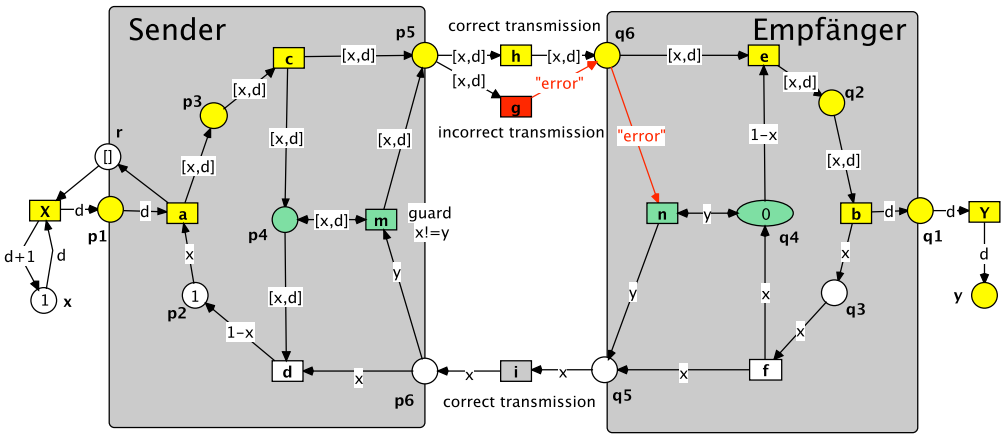

Abbildung 8.43: Übermittlung mit Quittung durch Alternierbit: Netz abp-4

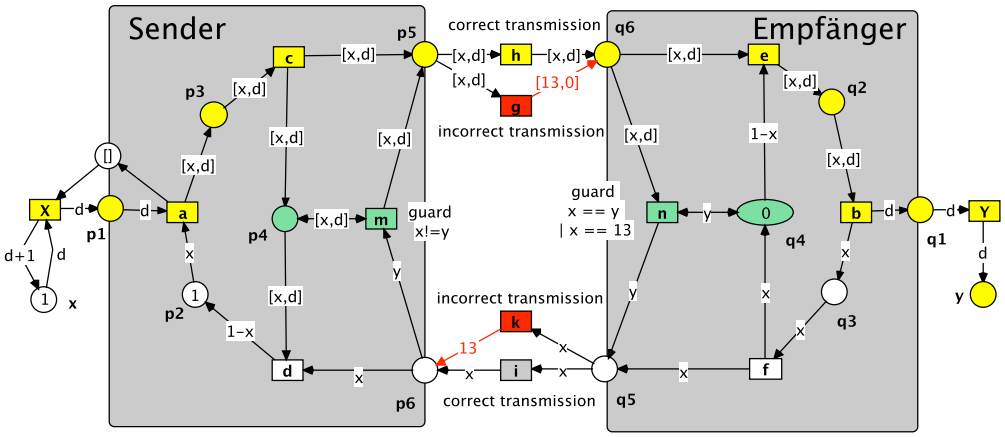

Abbildung 8.44: Das Alternierbitprotokoll abp-5

Bit `x` beigefügt, das anfangs den Wert 1 hat. Bei fehlerfreier Übermittlung schaltet die Transition `e`, da `q4` den komplementären Wert $1 - x$ (anfangs 0) enthält. Danach wird durch die Transition `b` das Bit wieder gelöscht und das Datum `d` wie vorher abgeliefert. Die Quittung erfolgt durch das (nicht geänderte) Bit `x`. Im Fehlerfall wird jedoch durch die Transition `n` mit dem komplementären Bit (anfangs 0) quittiert, wodurch die erneute Sendung eingeleitet wird. Dies erkennt der Sender dadurch, dass das Quittungs-Bit `y` von dem gespeicherten Bit `x` verschieden ist (`guard x!=y`). Um die nächste Datenübermittlung einzuleiten, wird das Ausgangsbit komplementiert nach `p2` gelegt. Der Vorgang wiederholt sich mit dem jeweils komplementären Bitwert.

Nun ist es nur noch ein kleiner Schritt zur endgültigen Lösung. Im Netz abp-5 (Abb. 8.44) ist nun auch im Quittungs-Kanal eine Störung möglich. Um den Integer-Test `guard x!=y` beibehalten zu können, wurde statt der Fehlermeldung `"error"` die (Unglücks-)Zahl `13` gewählt, d.h. auch bei einem Fehler in diesem Kanal wird das Datum `d` erneut

verschickt. Nun tritt aber folgendes Problem auf. Falls das Datum vorher störungsfrei übermittelt wurde, würde in diesem Fall eine verdopplete Ankunft beim Empfänger erfolgen. Dies kann der Empfänger jedoch daran erkennen, dass das Bit nicht gewechselt hat. Die verdopplte Sendung wird durch den Guard `x==y|x==13` (d.h. $x = y \vee x = 13$) in der Transition `n` gelöscht.
Als Besonderheit kommt dieses Protokoll zur Quittierung mit nur einem Bit `x` aus ([BS69]) - daher der Name *Alternierbitprotokoll*. Ein Beweis dafür, dass die Lösung korrekt ist, kann z.B. dadurch erfolgen, dass das Netz abp-5 als verhaltensäquivalent zu seiner Spezifikation abp-1 bewiesen wird. Verhaltensäquivalenz kann z.B. durch Bisimulation formalisiert werden. Dies wird im Rahmen der Prozessalgebra durchgeführt. Eine alternative Verifikationsmethode ist das *Model-Checking*. Hierbei wird die Spezifikation anhand des Erreichbarkeitsgraphen untersucht, der im vorliegenden Fall etwa 70 Zustände hat. Es wurden auf diese Weise bereits Systeme mit 10^{50} Zuständen verfiziert.