

# Synchrone und asynchrone Kommunikation in offenen Workflownetzen

Manja Wolf  
Humboldt-Universität zu Berlin  
Institut für Informatik  
Unter den Linden 6  
D-10099 Berlin  
wolf@informatik.hu-berlin.de

3. Mai 2007

Es werden Probleme der Steuerung einer Klasse von Petrinetzen untersucht. Diese Netze werden durch asynchrone Nachrichten und durch synchrone Aktionen gesteuert. Ein wichtiges Kriterium für solche Netze ist die Bedienbarkeit, die dann gegeben ist, wenn für das Netz ein Controller existiert, der gemeinsam mit dem Netz von jedem erreichbaren Zustand aus zu einem Endzustand gelangt. Die Ergebnisse der Untersuchung der zentralen und dezentralen Steuerung solcher Netze werden beschrieben und erläutert. Die Verbindung zu praktischen Anwendungen wird hergestellt, insbesondere für verteilte Geschäftsprozesse.

## 1 Einleitung

Unternehmen haben ein großes Interesse, die Effizienz ihrer Geschäftsprozesse zu erhöhen. Der Einsatz theoretisch fundierter Methoden zur Analyse dieser Prozesse ist daher für praktische Anwendungen besonders interessant. Ein Ansatz, der sich als besonders geeignet erwiesen hat, ist die Modellierung der Geschäftsprozesse mittels Workflownetzen.

*Workflownetze*, definiert in [3], sind eine Klasse von Petrinetzen. Sie haben ausgewiesene Anfangs- und Endzustände. Für diese Netze wurde der Begriff der Soundness charakterisiert [3]. Soundness fordert, daß ein Netz von jedem erreichbaren Zustand aus in einen Endzustand kommen kann. Darüber hinaus wird gefordert, daß keine Transition tot ist, d. h. jede Transition erreichbar ist.

Für die Modellierung der immer häufigeren interagierenden Prozesse wurden in [6, 9] *offene Workflownetze* vorgeschlagen. Offene Workflownetze basieren auf dem reinen Workflownetzmodell und wurden um Interfaces zum Nachrichtenaustausch erweitert. Sie haben ausgewiesene Anfangszustände und eine Menge von Endzuständen. Ein wichtiges Kriterium für offene Workflownetze ist die *Bedienbarkeit*. Der Begriff der Soundness war Grundlage für den Bedienbarkeitsbegriff. Im Unterschied zur Soundness genügt es hierbei, daß das Workflownetz gemeinsam mit wenigstens einem über das Interface mit dem offenen Workflownetz verbundenen Prozeß (Controller) von jedem erreichbaren Zustand aus zu einem oder mehreren Endzuständen gelangen kann.

Das Ziel der Forschung auf dem Gebiet der offenen Workflownetze ist es, den Anwendern die erforderlichen theoretisch fundierten Methoden zur Analyse der Eigenschaften realer Prozesse in die Hand zu geben. Darüber hinaus wird an der automatischen Steuerung des Zusammenspiels mehrerer Prozesse gearbeitet.

Derzeit sind offene Workflownetze untersucht, die azyklisch sind und bei denen Nachrichten ausschließlich über asynchrone Kommunikation ausgetauscht werden. Die Forschungsergebnisse für diese Netze wurden in [11] veröffentlicht. Es wurde für die zentrale Bedienbarkeit gezeigt, daß ein Controller für das offene Workflownetz existiert, der mit allen Interfaceplätzen verbunden ist und das Netz bedient (Strategie). Von der liberalsten Strategie spricht man, wenn alle anderen Controller, die das Netz ebenfalls bedienen, aus einer Teilmenge der Zustände dieses Controllers bestehen. Für die Berechnung der Strategie dieses Controllers wurde ein Algorithmus angegeben. Für die dezentrale Bedienbarkeit wurde gezeigt, daß nicht in jedem Falle eine liberalste Strategie existiert. Es wird nach einer speziellen zentralen Strategie gesucht. Auch hierfür wurde ein Algorithmus angegeben.

Reale Prozesse laufen häufig auch mit synchroner Kommunikation ab. Aus diesem Grund wurde nun das bestehende Konzept um synchrone Kommunikation erweitert und die in [11] veröffentlichten Aussagen unter den neuen Gesichtspunkten überprüft. Mit den in dieser Arbeit beschriebenen Ergebnissen kann nun das Konzept der offenen Workflownetze um einen neuen Baustein der Betrachtung erweitert werden. Damit ist man der Betrachtung des gesamten Problemraumes einen weiteren Schritt näher gekommen.

In der vorliegenden Arbeit werden die modifizierten Konzepte für offene Workflownetze, ihre Controller und deren Interaktion vorgestellt. Es wird die zentrale Bedienbarkeit untersucht. Dabei wird gezeigt, daß auch unter Berücksichtigung der synchronen Kommunikation eine liberalste Strategie existiert und der Algorithmus aus der vorangegangenen Veröffentlichung [11] auf die modifizierten Konzepte anwendbar ist. Danach werden die Ergebnisse der Untersuchung der dezentralen Bedienbarkeit beschrieben und es wird ein modifizierter Algorithmus für die Entscheidung der dezentralen Bedienbarkeit vorgestellt.

## 2 Grundlegende Definitionen

Die in dieser Arbeit untersuchten Workflownetze kommunizieren sowohl synchron als auch asynchron mit ihrer Umgebung. Hierbei können mehrere Nachrichten ggf. auch des gleichen Typs gleichzeitig verarbeitet werden. Dies wird über Multimengen abgebildet.

**Definition 1 (Multimenge)** Sei  $Z$  eine Menge (Trägermenge). Eine Multimenge  $A$  ist eine Abbildung  $A$  über  $Z$  mit  $A: Z \rightarrow \mathbb{N}$ . Sei

- $a \in A$  gdw.  $A(a) > 0$ ,
- $A = \emptyset$  gdw. für alle  $a$  gilt  $A(a) = 0$ ,
- $A_1 \sqcup A_2$  diejenige Multimenge mit  $(A_1 \sqcup A_2)(a) = A_1(a) + A_2(a)$ ,
- $\underline{a}$  diejenige Multimenge mit  $\underline{a}(a) = 1$  und  $\underline{a}(x) = 0$  für  $x \neq a$ ,
- $A_1 \setminus A_2$  diejenige Multimenge mit  $(A_1 \setminus A_2)(a) = A_1(a) - A_2(a)$ , falls  $A_1(a) \geq A_2(a)$ , sonst sei  $(A_1 \setminus A_2)(a) = 0$ .

$\text{bags}(Z)$  bezeichnet die Menge der Multimengen über der Menge  $Z$ .

Multimengen werden als Auflistung der Elemente entsprechend ihrer Vielfachheit in eckigen Klammern geschrieben. Beispiel:  $A = [a, a, a, b, b]$  ist diejenige Multimenge über der Menge  $Z = \{a, b, c\}$  mit  $A(a) = 3$ ,  $A(b) = 2$ ,  $A(c) = 0$

Offene Workflownetze (oWFN) sind eine Klasse von Petrinetzen. Daher werden nun die Begriffe *Petrinetz* und *Verhalten von Petrinetzen* definiert.

**Definition 2 (Petrinetz)** Ein Petrinetz  $N$  besteht aus zwei endlichen, disjunkten Mengen  $P$  (Plätze) und  $T$  (Transitionen), einer Relation  $F$  ( $F \subseteq (T \times P) \cup (P \times T)$ ) und einer Anfangsmarkierung  $m_0$ . Eine Markierung ist eine Abbildung  $m: P \rightarrow \mathbb{N}$ . Der Vektor  $t^+$  ist derjenige Vektor mit  $t^+(p) = 1$  falls  $[t, p] \in F$  und  $t^+(p) = 0$  sonst. Der Vektor  $t^-$  ist derjenige Vektor mit  $t^-(p) = 1$  falls  $[p, t] \in F$  und  $t^-(p) = 0$  sonst.

**Definition 3 (Verhalten von Petrinetzen)** Eine Transition  $t$  ist in einer Markierung  $m$  aktiviert, wenn für alle Plätze  $p$  mit  $[p, t] \in F$  gilt  $m(p) > 0$ . Eine Transition  $t$  kann von einer Markierung  $m$  in eine Markierung  $m'$  schalten ( $m \xrightarrow{t} m'$ ), wenn  $t$  in  $m$  aktiviert ist und  $m' = m + t^+ - t^-$ . Eine Multimenge  $U$  über  $T$  heißt Schritt.  $U$  ist ausführbar, wenn  $\sum_{t \in T} U(t) * t^- \leq m$ .  $U$  schaltet von einer Markierung  $m$  in eine Markierung  $m'$  ( $m \xrightarrow{U} m'$ ), wenn  $U$  ausführbar ist und wenn  $m' = m + \sum_{t \in T} U(t) * (t^+ - t^-)$ . Die Menge der Markierungen, die durch Schalten einer endlichen Zahl von Transitionen von  $m$  aus erreichbar ist, wird mit  $R_N(m)$  bezeichnet.

Auf dieser Grundlage werden Workflownetze mit asynchronen und synchronen Ein-, Ausgabekanälen nun als spezielle Klasse von Petrinetzen definiert.

**Definition 4 (Workflownetz)** Ein Workflownetz  $M = (P, T, L, F, F_L, m_0, \Omega)$  mit synchronen und asynchronen Kommunikationskanälen ist eine Erweiterung eines Petrinetzes  $N$ , mit

- $P$  einer disjunkten Vereinigung der Mengen  $P_I$  (Inputplätze) und  $P_O$  (Outputplätze) für asynchrone Kommunikation und  $P_N$  (interne Plätze);
- $T$  einer Menge von Transitionen;
- $L$  einer Menge von Labeln für synchrone Kommunikation;
- $F \subseteq (T \times P) \cup (P \times T)$ ;
- $F_L \subseteq L \times T$  die Verbindung von Labeln mit Transitionen;
- $F \cap (P_O \times T) = \emptyset, F \cap (T \times P_I) = \emptyset$  (asynchrone Kommunikation);
- einer Anfangsmarkierung  $m_0$ ;
- $\Omega$  einer Menge von Endmarkierungen  $m$  mit der Eigenschaft  $m(p) = 0$  für  $p \in P_I \cup P_O$ ;
- $F$  enthält keine Zyklen (Die transitive Hülle von  $F$  ist irreflexiv).

Da im folgenden immer über Workflownetze gemäß Definition 4 gesprochen wird, werden diese „Workflownetze mit asynchronen und synchronen Kommunikationskanälen“ als „offene Workflownetze“ oder „oWFN“ bezeichnet.

In dieser Arbeit werden nur azyklische Systeme betrachtet.

Abbildung 1 zeigt ein Beispiel für ein oWFN mit asynchroner und synchroner Kommunikation. Die asynchrone Kommunikation wird durch Interfaceplätze und die synchrone Kommunikation durch Label dargestellt. Die Label für die synchronen Aktionen werden mit schmalen Balken dargestellt, die durch gestrichelte Linien mit den jeweiligen Transitionen verbunden sind. In der Abbildung ist vereinfacht ein Prozeß zur Onlinebuchung eines Fluges dargestellt. Zuerst stellt ein Kunde an das Reiseportal (das oWFN) eine Anfrage zu den verfügbaren Flügen ( $F$ ). Das Reiseportal leitet diese Anfrage an den Fluganbieter weiter ( $A$ ). Der Fluganbieter erstellt nun eine Liste der verfügbaren Flüge ( $La$ ) und das Reiseportal stellt diese sofort dem Kunden zur Verfügung ( $Lk$ ). Nun wählt der Kunde einen Flug aus und reserviert diesen ( $R$ ). Die Reservierung erfolgt synchron und löst im Netz unmittelbar die Buchung aus. Deshalb wird sowohl dem Kunde als auch dem Fluganbieter zeitgleich die Buchungsbestätigung ( $Bk, Ba$ ) zur Verfügung gestellt.

**Definition 5 (Port)** Sei  $M$  ein offenes Workflownetz. Eine Menge  $\Pi \subseteq P_I \cup P_O \cup L$  von Interfaceplätzen und Labeln ist ein Port  $\Pi$  von  $M$ .

Ein Grund für die Nutzung von Petrinetzen zur Modellierung von Geschäftsprozessen besteht in der Eleganz der Modellierung asynchroner Kommunikation mittels Interfaceplätzen und synchroner Kommunikation über Label.

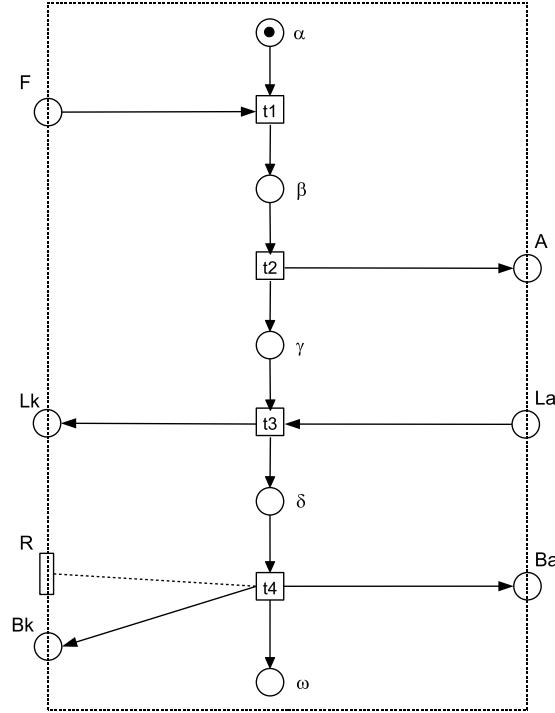


Abbildung 1: vereinfachte Darstellung eines Prozesses zur Onlinebuchung eines Fluges

Controller werden als klassische Automaten modelliert. Die Interfaceplätze und Label werden von den Controllern mitgenutzt. Daher ist es nicht erforderlich, sie auch bei den Controllern zu modellieren. Deshalb ist die Begründung der Modellierung von oWFN mittels Petrinetzen bei den Controllern nicht gegeben. Natürlich wäre es möglich, auch die Controller als Petrinetze zu modellieren. Die hier vorgeschlagenen Konstruktionen für die Kommunikation zwischen Netz und Controller basieren jedoch hauptsächlich auf dem Konzept der Zustände. Diese lassen sich mit anderen Modellen besser abbilden. Es ist zwar möglich, ein Petrinetz aus einem Zustandsraum heraus zu konstruieren, entweder direkt als State Machine oder verfeinert durch das Anwenden von Region Theory [4, 8]. Diese Konstruktionen sind aber bereits gut untersucht. Daher wird hier auf die Kernkonstruktion zurückgegriffen und Controller werden als klassische Automaten modelliert.

**Definition 6 (Controller)** Sei  $M$  ein offenes Workflownetz und  $I \subseteq (P_I \cup P_O \cup L)$ . Ein Controller, der mit  $I$  verbunden ist, ist ein Automat mit einem Alphabet  $\text{bags}(I)$ , einer Menge von Zuständen  $Q$ , einer Schrittrelation  $\delta : Q \times \text{bags}(I) \rightarrow \wp(Q)$ , einem Anfangszustand  $q_0$  und einer Menge von Endzuständen  $\Omega_C$ .

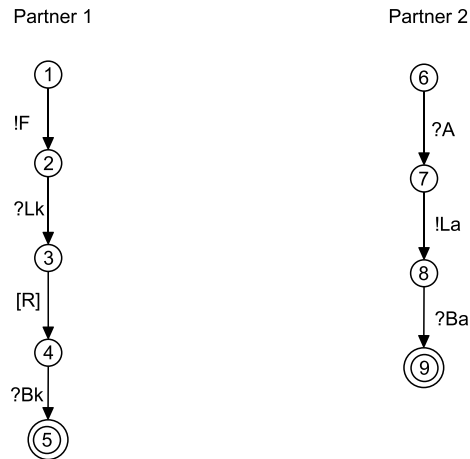


Abbildung 2: eine Menge Controller für das oWFN in Abbildung 1

Abbildung 2 zeigt zwei Controller. Ein Schritt im Controller beschreibt verschlüsselt über das Alphabet eine Interaktion mit dem offenen Workflownetz. Die Multimenge von Interfaceplätzen und Labeln, die an dem Schritt beteiligt ist, steht für die asynchron gesendeten bzw. empfangenen Nachrichten und die synchronen Interaktionen. Asynchron gesendete Nachrichten sind mit einem Ausrufezeichen z.B.  $!F$  gekennzeichnet, asynchron empfangene Nachrichten mit einem Fragezeichen z.B.  $?Lk$ . Synchroner Aktionen sind von einer eckigen Klammer umschlossen, z.B.  $[R]$ . Die Endzustände sind durch Kreise mit doppelten Linien dargestellt.

Nur Nachrichten, die auf den Outputplätzen des oWFN liegen, können von einem Controller konsumiert werden. Zwischen dem Absenden einer Nachricht durch das offene Workflownetz oder einen Controller und ihrer Verarbeitung im Netz bzw. durch den Controller kommt es hierbei zu einer zeitlichen Verzögerung.

Eine synchrone Interaktion aktiviert und schaltet unmittelbar eine der hinter dem jeweiligen Label befindliche Transition. Es können mehrere Label gleichzeitig aktiviert werden. Die synchrone Kommunikation kann als Drücken von Knöpfen an einem Gerät betrachtet werden. In dieser Arbeit wird das gleichzeitige Drücken mehrerer Knöpfe zugelassen.

Die Kommunikation mit dem offenen Workflownetz kann mit mehreren Controllern stattfinden. Die Menge der Controller muß dabei alle Interfaceplätze und Label des Netzes umfassen. Die Aufteilung der Interfaceplätze und Label unter den Controllern muß disjunkt sein.

Definition 8 (komponiertes System) beschreibt das Zusammenwirken eines offenen Workflownetzes mit einer *passenden* Menge von Controllern.

**Definition 7 (passend)** Eine Menge Controller  $\{C_1, \dots, C_n\}$  ist passend

für ein offenes Workflownetz  $M$  mit einer gegebenen Menge von Ports, wenn mit jedem Port von  $M$  genau einer der Controller verbunden ist. Die disjunkte Vereinigung der Ports muß gleich der Menge  $P_I \cup P_O \cup L$  sein.

Die Menge der beiden Controller in Abbildung 2 ist für das offene Workflownetz in Abbildung 1 passend.

Für die Beschreibung des gemeinsamen Verhaltens des offenen Workflownetzes mit einer Menge der passenden Controller wird im folgenden der Begriff des *komponierten Systems* definiert. Da die synchrone und asynchrone Kommunikation möglichst wenig eingeschränkt werden soll, wird die Definition des komponierten Systems sehr komplex. Für die Übersichtlichkeit dieser Definition werden daher vorab einige Begriffe eingeführt.

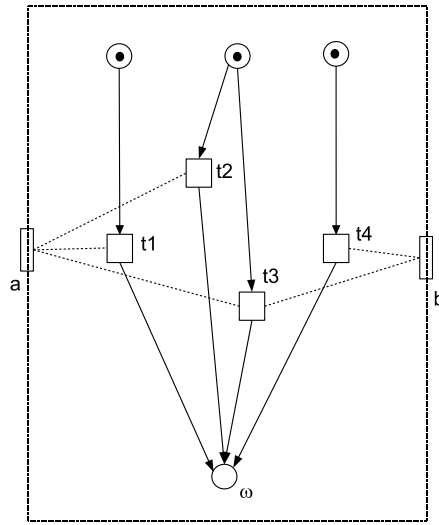


Abbildung 3: Wenn a und b gleichzeitig stattfinden [a,b], darf nicht allein Transition t3 schalten. Es könnten aber z.B. t1 und t3 schalten. Wenn (a) zweimal gleichzeitig stattfindet [a,a], müssen zwei verschiedene Transitionen schalten z.B. t1 und t3

Da in einem Zustandsübergang mehrere Nachrichten eines Typs (eines Interfaceplatzes bzw. Labels) beteiligt sein können, wird hierfür eine Multimenge  $B$  über  $P_I \cup P_O \cup L$  genutzt. Die auf  $P_I$  eingeschränkte Multimenge  $B|_{P_I}$  wird als  $in(B)$  bezeichnet, die auf  $P_O$  eingeschränkte Multimenge  $B|_{P_O}$  als  $out(B)$  und die auf die Label eingeschränkte Multimenge  $B|_L$  als  $lab(B)$ .

Eine Menge  $T_S \subseteq T$  von Transitionen *entspricht* der Menge  $lab(B)$ , falls  $T_S$  die disjunkte Vereinigung von Mengen  $T_{S_k}$  ist für alle  $k \in L$  wobei  $|T_{S_k}| = B(k)$  und für alle  $t \in T_{S_k}$  gilt  $(t, k) \in F_L$ .

Der Vektor  $\sum_{t \in T_S} t^-$  wird als  $T_S^-$  bezeichnet, der Vektor  $\sum_{t \in T_S} t^+$  als  $T_S^+$ .

Mit dem Begriff  $T_S$  entspricht  $lab(B)$  wird für die synchrone Kommunikation sichergestellt, daß in einem Schritt mit jedem aktivierten Label immer genau eine Transition schaltet. Wenn ein Label in einem Schritt mehrfach aktiviert wird, schalten jeweils verschiedene Transitionen. Werden mehrere verschiedene Label gleichzeitig gedrückt, sind niemals zwei oder mehr dieser Label ein und derselben Transition zugeordnet. Abbildung 3 zeigt ein oWFN, in dem sowohl mehrere Label mit einer Transition verbunden sind, als auch ein Label mit mehreren Transitionen.

**Definition 8 (komponiertes System)** Sei  $M$  ein offenes Workflownetz und

$\{C_1, \dots, C_n\}$  eine passende Menge von Controllern. Dann ist das komponierte System  $M \oplus C_1 \oplus \dots \oplus C_n$  ein Transitionssystem mit  $R_N(m_0) \times Q_1 \times \dots \times Q_n$  als seine Menge von Zuständen und Kanten

- von  $[m, q_1, \dots, q_n]$  zu  $[m', q_1, \dots, q_n]$ , wenn es in  $M$  eine Transition  $t \in T$  mit  $(l, t) \notin F_L$  für alle  $l \in L$  gibt, mit  $m \xrightarrow{t} m'$  (interne Abläufe),
- von  $[m, q_1, \dots, q_n]$  zu  $[m', q'_1, \dots, q'_n]$ , wenn es
  - eine Multimenge  $B$  und
  - eine Menge  $T_S$ , die  $lab(B)$  entspricht,

gibt, für die

- der Schritt  $T_S$  in  $m$  ausführbar ist und
- für alle  $p \in P_O$  ist  $m(p) \geq B(p)$  und
- für alle  $i, i = 1, \dots, n$ :

Sei  $\Pi_i$  der Port mit dem  $C_i$  verbunden ist.

Sei  $B_i$  die Multimenge für die für alle  $b \in I$  gilt: wenn  $b \in \Pi_i$  dann ist  $B_i(b) = B(b)$  sonst ist  $B_i(b) = 0$ .

Wenn  $B_i \neq \emptyset$ , dann ist in  $C_i$   $q'_i \in \delta_i(q_i, B')$ ,

sonst ist  $q'_i = q_i$  und

- $m' = m - (T_S^- + out(B)) + (T_S^+ + in(B))$ .

(synchrone und asynchrone Kommunikation)

Durch synchrone Kommunikation werden unmittelbar Transitionen im Netz geschaltet und damit neue Markierungen im Netz erreicht. Der Übergang mit einem Label führt also direkt zu einer Aktion im Netz. Wie schon erwähnt, ist es möglich, in einem Schritt mehrere Label gleichzeitig zu aktivieren und damit mehrere Transitionen gleichzeitig im Netz zu schalten. Für den Fortgang im Netz kann es nicht nur erlaubt, sondern sogar erforderlich sein, mehrere Label gleichzeitig zu aktivieren. Abbildung 4 zeigt ein Netz mit ausschließlich synchroner Kommunikation, das nur korrekt bedient werden kann, wenn beide Label gleichzeitig aktiviert werden.

Bei der asynchronen Kommunikation werden Nachrichten von Controllern an das Netz gesendet oder vom Netz empfangen. Beim Empfang einer Nachricht durch einen Controller wird dieser eine Marke von einem Outputplatz

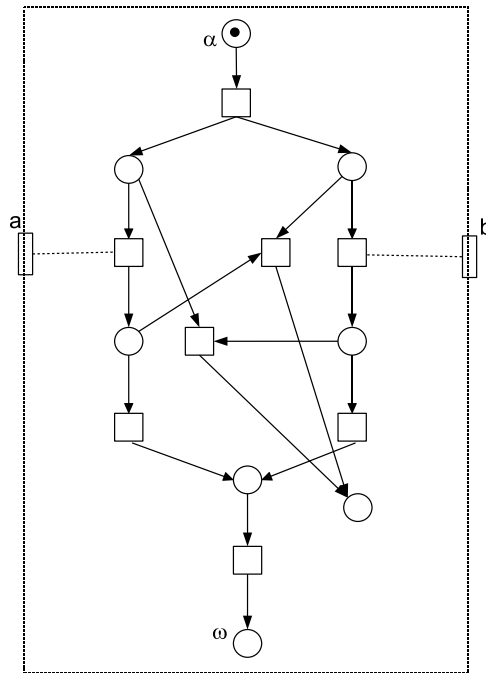


Abbildung 4: Beispiel für ein Netz mit ausschließlich synchroner Kommunikation bei dem gleichzeitiges Schalten der beiden Label für das Erreichen des Endzustandes erforderlich ist.

konsumieren. Diese Nachricht wird durch eine netzinterne Aktion vorab in Form einer Marke auf dem Outputplatz generiert. Beim Senden einer Nachricht durch den Controller legt dieser eine Marke auf einen Inputplatz. Das Netz kann diese Marke durch eine interne Aktion konsumieren. Wie der Begriff asynchrone Kommunikation nahelegt, finden das Senden und das Konsumieren einer Nachricht zeitlich unabhängig voneinander statt. In einem Schritt können daher mehrere Nachrichten gleichen Typs (gleiche Input- bzw. Outputplätze) gleichzeitig gesendet bzw. empfangen werden. Ggf. verbleiben Marken (bis zum Konsumieren durch das Netz oder einen Controller) auf den Interfaceplätzen.

Sofern synchrone und asynchrone Kommunikation gemeinsam in einem Netz auftreten, haben die synchronen Vorgänge direkte Auswirkungen auf die Interfaceplätze, die mit den gelabelten Transitionen verbunden sind.

Abbildung 5 zeigt zwei Netze  $N_1$  und  $N_2$ , die dies veranschaulichen: Sind Inputplätze (asynchrone Kommunikation) mit gelabelten Transitionen verbunden, müssen zur Ausführung des Schrittes nicht nur die Label aktiviert werden, sondern zum Zeitpunkt der Aktivierung der Label bereits ausreichend Marken auf den mit den Labeln verbundenen Inputplätzen liegen. In  $N_1$  (Inputplatz an gelabelter Transition  $t_1$ ) muß ein Controller erst eine Mar-

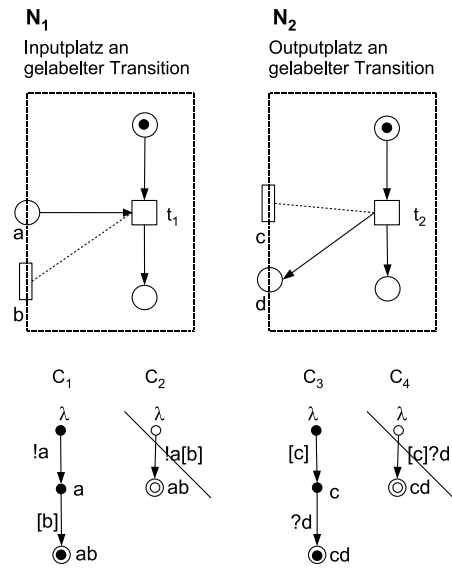


Abbildung 5: Beispiele für gemischte, synchron asynchrone Kommunikation und den Folgen für die Schaltfolge. Die Controller 1 und 3 bedienen die jeweils darüber dargestellten Netze, die Controller 2 und 4 nicht.

ke auf den Platz  $a$  gelegt haben, damit die synchrone Aktion stattfinden kann ( $C_1$ ). Die von Controller  $C_2$  in einem Schritt auf den Inputplatz  $a$  gelegte Marke kann nicht im gleichen Schritt durch das Netz konsumiert werden.

Sind andererseits Outputplätze mit gelabelten Transitionen verbunden, werden durch Schalten des Schrittes Marken auf die mit den Labeln verbundenen Outputplätze gelegt. Marken aus asynchronen Nachrichten können nicht im gleichen Schritt mit synchronen Aktionen konsumiert werden. Dies resultiert aus der zeitlichen Differenz zwischen Nachrichtenversand und -konsum bei asynchroner Kommunikation. D. h. die in  $N_2$  durch die synchrone Aktion  $c$  generierte und auf den Outputplatz  $d$  gelegte Marke kann erst in einem späteren Schritt konsumiert werden. Der Controller  $C_4$  kann nicht in *einem* Schritt eine Marke auf  $d$  legen und gleichzeitig wieder konsumieren.

Abbildung 6 zeigt das komponierte System aus dem offenen Workflownetz in Abbildung 1 und den Controllern aus Abbildung 2. Das Ziel der Bedienung ist die korrekte Terminierung des offenen Workflownetzes. Das Netz muß dann in einer Markierung aus  $\Omega$  und die Controller jeweils in einem Zustand aus  $\Omega_C$  sein.

**Definition 9 (Strategie)** Sei  $M$  ein offenes Workflownetz. Eine Menge passender Controller ist dann eine Strategie für  $M$ , wenn im komponierten System von jedem Zustand, der vom Anfangszustand erreichbar ist, ein Zustand erreichbar ist, dessen Komponenten jeweils Endmarkierungen von

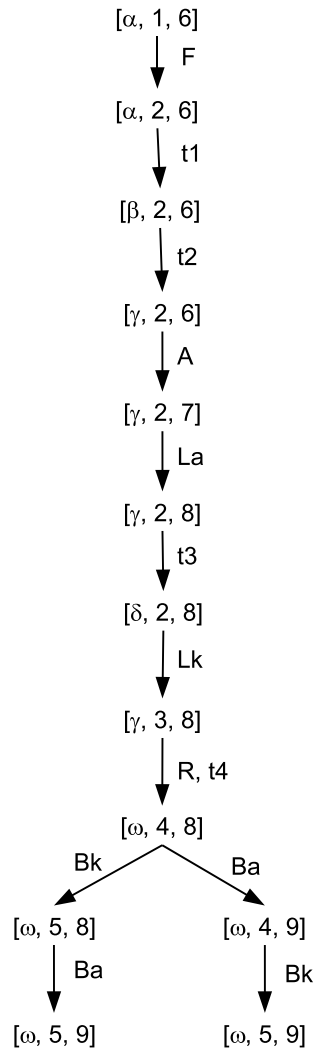


Abbildung 6: Das komponierte System zum oWFN aus Abb. 1 und den Controllern aus Abb. 2.

*M und aller mit M verbundener Controller sind.*

**Beobachtung** Controller wurden hier sehr liberal definiert (nichtdeterministisch, mit der Möglichkeit der gleichzeitigen Ausführung mehrerer Nachrichten, mit internen Übergängen (leere Multimenge), möglicherweise unbeschränkt). Nicht alle der Möglichkeiten, die diese Definition gestattet, sind jedoch für die Steuerung eines offenen Workflownetzes notwendig. Im Weiteren werden Controller daher auf *Baumcontroller* eingeschränkt und es wird gezeigt, daß wann immer ein bedienender Controller existiert, auch ein bedienender Baumcontroller existiert. In einem Baumcontroller über einem be-

beliebigen Alphabet  $A$  werden die Zustände eindeutig durch ihren Weg durch den Controller benannt und als Worte über  $A$  dargestellt. Sei  $A^*$  die Menge der endlichen Worte über  $A$ . Die *maximale Länge* eines Ablaufes in einem bedienenden Controller wird mit  $l_M$  bezeichnet. Ein solches  $l_M$  muß es wegen der geforderten Azyklizität von  $M$  geben. Die Anzahl der sinnvollen Nachrichten, die zwischen Controller und offenem Workflownetz ausgetauscht werden können, ist dadurch beschränkt. Diese Schranke kann berechnet werden. Ein Algorithmus für diese Berechnung ist hier nicht Gegenstand der Betrachtung. Ein Vorteil von Baumcontrollern besteht darin, daß sie durch ihre Zustandsmenge vollständig bestimmt sind. Nachfolgend wird der Begriff *Baumcontroller* definiert.

**Definition 10 (Baumcontroller)** *Ein mit  $I \subseteq P_I \cup P_O \cup L$  verbundener Controller heißt Baumcontroller  $B$ , wenn*

- *sein Alphabet  $I_B \subseteq ((P_I \cup P_O) \cap I) \cup \text{bags}(L \cap I)$  ist,*
- *$Q_B \subseteq \{w \mid w \in I_B^*, \text{length}(w) \leq l_M\}$ ,*
- *$Q_B \neq \emptyset$*
- *$Q_B$  mit  $q$  alle Präfixe von  $q$  enthält,*
- *$\delta : Q_B \times I_B \rightarrow \wp(Q_B)$ , so daß  $\delta(w, x) = \{wx\}$  wenn  $wx \in Q_B$  mit  $x \in I_B$  und sonst  $\delta(w, x) = \emptyset$  und*
- *$\lambda$  (das leere Wort) der Anfangszustand ist.*

Die hier geforderte Präfixabgeschlossenheit ist sinnvoll und hinreichend für die Erreichbarkeit aller Zustände ausgehend vom Anfangszustand. Die vorgeschlagenen Restriktionen vereinfachen nachfolgende Betrachtungen ohne Einschränkung der Allgemeinheit.

**Beobachtung** Da im Netz mehrere synchrone Aktionen gleichzeitig stattfinden können, wird die Menge der möglichen Worte über dem Alphabet  $A$  größer als bei reiner asynchroner Kommunikation. Anders als bei der asynchronen Kommunikation verbleiben hier keine Nachrichten im Netz (als Marken auf Interfaceplätzen) sondern es werden alle durch Label aktivierten Transitionen sofort geschaltet. Somit kann an einem Bogen des Baumcontrollers (einem Zustandsübergang) die Multimenge der am Schritt beteiligten Label und Interfaceplätze stehen. Das Alphabet ändert sich also im Vergleich zu rein asynchroner Kommunikation. Das Gesamtverhalten des Systems bleibt jedoch weiterhin beschränkt, da die Anzahl der Plätze und Label endlich ist und damit nur endlich viele zusätzliche Zustände (Worte) erreicht werden können.

In einem Baumcontroller ist gegenüber einem beliebigen Controller das Alphabet kleiner. Z. B. kommen Schritte nicht vor, die sowohl asynchrone als auch synchrone Aktionen enthalten. Dennoch ist diese Restriktion keine Einschränkung für Bedienbarkeit.

**Satz 1** Wenn  $\{C_1, \dots, C_n\}$  eine Strategie für  $M$  ist und  $C_i$  mit  $I$  verbunden ist, dann gibt es einen mit  $I$  verbundenen Baumcontroller  $C'_i$ , so daß  $\{C_1, \dots, C'_i, \dots, C_n\}$  eine Strategie für  $M$  ist.

**Beweisidee**

Ein gemischter sequentieller Schritt wird sequentialisiert, indem erst Empfangs-, dann synchrone und zum Schluß Sendeaktionen stattfinden. Hierfür wird ein Controller generiert, der mit je einem internen Posteingangs- und Postausgangskorb ausgestattet ist. Bei asynchronen Interaktionen zwischen Netz und Controller werden diese Nachrichten nun in den Postkörben zwischengespeichert. Sofern Label über Transitionen mit Inputplätzen verbunden sind, sind diese Inputplätze Vorbedingung für die synchrone Aktion. Die als Vorbedingung erforderlichen Marken können zu einem beliebigen Zeitpunkt *vor* dem Schalten dieser synchronen Aktion in den internen Postkorb gelegt werden. Beim Schalten der synchronen Aktion werden diese Marken durch eine interne Aktion aus dem Posteingang genommen. Die durch die synchrone Aktion generierten Marken für Outputplätze des Netzes werden durch eine interne Aktion in den Postausgang gelegt. Von dort können sie danach zu einem beliebigen Zeitpunkt an das Netz geschickt werden.

In Abbildung 7 wird dies an einem Beispiel veranschaulicht.

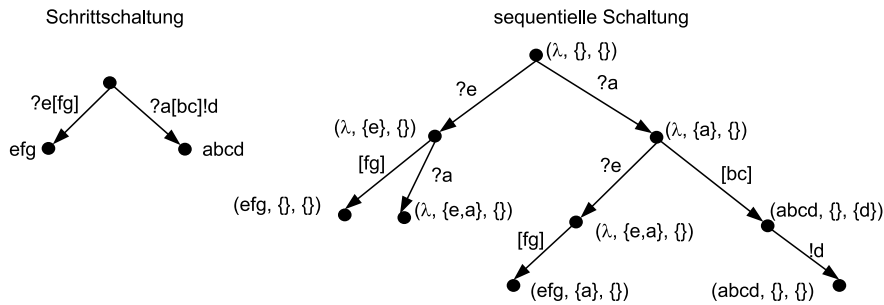


Abbildung 7: Beispiel zur Veranschaulichung der Beweisidee von Satz 1. Bei dem Controller rechts, der sequentiell schaltet, bestehen die Zustände nun aus drei Komponenten. Die erste Komponente ist der Zustand des Controllers, die zweite die Multimenge der Posteingangsnachrichten und die dritte die Multimenge der Postausgangsnachrichten.

Diese Argumentation wird im folgenden Beweis für Satz 1 formalisiert:

**Beweis**

Für den Beweis der Aussage aus Satz 1 wird in einem ersten Schritt aus dem Ursprungscontroller  $C_i$  ein Automat  $C^*$  konstruiert, der die asynchronen Aktionen einzeln und die synchronen Aktionen in einem Schritt schaltet.

Der Automat  $C^*$  hat das Alphabet  $((P_I \cup P_O) \cap I) \cup \text{bags}(L \cap I)$  der Menge der asynchronen Nachrichtenplätze des Controllers  $C_i$  und einer Multimenge der Label von  $C_i$  sowie die Menge von Zuständen  $Q \times \text{bags}(P_O \cap I) \times \text{bags}(P_I \cap I)$ . Die zweite Komponente eines Zustandes ein Posteingangskorb des Controllers, in dem die eingehenden Nachrichten aus dem offenen Workflownetz  $M$  zwischengespeichert werden. Analog dazu werden in der dritten Komponente die vom Controller an das Netz  $M$  gesendeten Nachrichten zwischengespeichert. Die Menge der Endzustände  $\Omega_C^*$  wird daher als  $\Omega \times \{\square\} \times \{\square\}$  festgelegt. D.h. Endzustände des Controllers  $C^*$  können nur solche Zustände sein, die Endzustände im Ursprungscontroller  $C_i$  sind und in denen alle zwischengespeicherten Nachrichten abgerufen wurden. Die Zustandsübergänge im Controller  $C^*$  werden wie folgt definiert:

Falls im Controller  $C_i$  ein Zustandsübergang  $q \xrightarrow{M_I + M_S + M_O} q'$  stattfindet, mit  $M_I$  der Multimenge der asynchronen eingehenden Nachrichten,  $M_S$  der Multimenge der synchronen Aktionen und  $M_O$  der Multimenge der asynchronen gesendeten Nachrichten, finden in  $C^*$  bei einem Zustand  $[q, M_1, M_2]$  folgende Zustandsübergänge statt:

- $[q, M_1, M_2] \xrightarrow{a} [q, M_1 + a, M_2]$  mit  $a \in P_O$  (\*)
- $[q, M_1, M_2] \xrightarrow{M_S} [q', M_1 - M_I, M_2 + M_O]$  falls  $M_1 \geq M_I$  (\*\*)
- und  $[q, M_1, M_2] \xrightarrow{b} [q, M_1, M_2 - b]$  mit  $b \in P_I$  und  $M_2(b) \geq 0$  (\*\*\*)

Der Zustandsübergang (\*\*) hat als Kommunikation mit dem offenen Workflownetz lediglich die Multimenge der synchronen Aktionen  $M_S$ . Die im ursprünglichen Schritt aus  $C$  gesendeten und empfangenen asynchronen Nachrichten  $M_I$  und  $M_O$  tauchen hier lediglich als Zustandsänderung im Automaten auf, nicht jedoch nach außen an das offene Workflownetz. Bei diesem Zustandsübergang werden intern (im Automaten  $C^*$ ) die eingehenden Nachrichten  $M_I$  vom „Posteingang“ abgerufen und die an das Netz gesendeten Nachrichten  $M_O$  in den „Postausgang“ gelegt. Der Zustandsübergang (\*) füllt den „Posteingang“ des Automaten mit Nachrichten, die das offene Workflownetz  $M$  an den Controller  $C^*$  sendet. Der Zustandsübergang (\*\*\*) ist dementsprechend der Nachrichtenversand an das Netz  $M$ .

Es ist nun zu zeigen, daß  $\{C_1, \dots, C^*, \dots, C_n\}$  eine Strategie von  $M$  ist.

Wenn ein Schritt im Ursprungscontroller  $C_i$  stattfinden kann, so können äquivalent dazu im Controller  $C^*$  erst die asynchronen eingehenden Nachrichten von  $M$  empfangen werden (\*), dann in einem Schritt die Multimenge der synchronen Aktionen schalten (\*\*) und danach die asynchronen Nachrichten an  $M$  gesendet werden (\*\*\*)

Angenommen in  $C^*$  findet ein Übergang der Form (\*\*) statt. Dies ist nur möglich, wenn der interne Posteingang  $M_1$  bereits ausreichend gefüllt ist, also vom offenen Workflownetz  $M$  die für den Schritt notwendigen asynchronen Nachrichten  $M_I$  bereits gesendet wurden und somit  $M_1 \geq M_I$ . Damit kann der entsprechende Schritt auch in  $C_i$  stattfinden.

Wie leicht zu sehen ist, kann der Controller  $C^*$  in einen Baumcontroller  $C^{**}$  gemäß Definition 10 (Baumcontroller) überführt werden.

Wenn keine synchronen Aktionen im Ursprungsschritt enthalten sind, ist dieser Baumcontroller  $C^{**}$  nichtdeterministisch. Der Zustandsübergang (\*\*\*) reduziert sich damit zu einer internen Aktion. Diese Aktion ruft Nachrichten aus dem Posteingang ab oder legt Nachrichten in den Postausgangskorb. Nach außen ist der Zustandsübergang ein  $\tau$ -Übergang. Aus dem nichtdeterministischen Baumcontroller wird durch Weglassen dieser  $\tau$ -Übergänge ein deterministischer erstellt. Das Weglassen der  $\tau$ -Übergänge kann erfolgen, weil dies zu keiner Einschränkung des Verhaltens führt. Ohne die internen Übergänge werden dagegen die nachfolgenden Aktionen vorgezogen und in der Verzweigung, in denen sonst die internen Übergänge stattfinden, kann nun noch zwischen diesen nachfolgenden Aktionen gewählt werden. Daher ist der entstehende Baumcontroller erst recht Teil einer Strategie für  $M$ . q.e.d.

In Vorbereitung auf die Beschreibung eines Algorithmus zur Generierung eines Controllers, der das Netz korrekt bedient, folgen nun weitere Definitionen. Eingabe für den Algorithmus ist ein Baumcontroller mit unbeschränktem Verhalten. Ein solcher Baumcontroller wird als *rauschend* bezeichnet und wie folgt definiert:

**Definition 11 (Rauschen)** Sei  $I \subseteq P_I \cup P_O \cup L$ . Dann wird der Baumcontroller  $B$  mit dem Alphabet  $I_B \subseteq ((P_I \cup P_O) \cap I) \cup \text{bags}(L \cap I)$ , den Zuständen  $Q = \{w \mid w \in I_B^*, \text{length}(w) \leq l_M\}$  und den Endzuständen  $\Omega_R$  als rauschend bzgl.  $I_B$  bezeichnet. Man schreibt hierfür  $\text{rauschen}(I_B)$ .

Der Controller  $\text{rauschen}(I_B)$  hat als Zustandsmenge die Menge aller möglichen Zustände für ein gegebenes komponiertes System  $M \oplus C$ . Der Algorithmus generiert durch geeignetes Streichen von Zuständen einen Controller, der das Netz korrekt bedient. Zur Erläuterung dieses „geeigneten Streichens“ von Zuständen werden die Begriffe *Wissen eines Controllers*, *interne* und *externe Deadlocks* benötigt und vorab beschrieben und definiert.

In einem mit  $I$  verbundenen Baumcontroller  $C$  verschlüsseln die Zustände in Form von Wörtern die gesamten Informationen über die bisher stattgefundene Interaktion mit dem offenen Workflownetz  $M$ . Es wird davon ausgegangen, daß dem Controller  $C$  zum Zeitpunkt der Durchführung seiner Aktionen das Verhalten des Netzes  $M$  vollständig verborgen ist. Hiervon ausgenommen ist das Wissen über die Outputplätze in  $I \cap P_O$  und das Wissen über geschaltete synchrone Aktionen (wobei dem Controller hier auch nicht bekannt ist, welche netzinternen Transitionen geschaltet haben). Für die Konstruktion eines korrekt bedienenden Controllers wird das Wissen des Controllers im Hinblick auf das gesamte komponierte System betrachtet. In Kenntnis von  $M$  und  $C$  ist es daher möglich darauf zu schließen, bei welcher

Markierung (in welchem Zustand) sich  $M$  möglicherweise befindet, wenn  $C$  in einem gegebenen Zustand  $q$  ist. Dieses Wissen, das ein Zustand von  $C$  über die Markierung von  $M$  enthält, formalisieren wir über eine Abbildung  $K$  wie folgt:

**Definition 12 (Wissen eines Controllers)** *Sei  $C$  ein mit  $I$  verbundener Controller und  $q$  ein Zustand von  $C$ . Sei  $S$  die Menge der erreichbaren Zustände des komponierten Systems  $M \oplus C \oplus \text{rauschen}(((P_I \cup P_O) \setminus I) \cup \text{bags}(L \setminus I))$  wobei für  $[m, q, q'] \in S$ ,  $m$  die Markierung von  $M$  und  $q'$  der Zustand von  $\text{rauschen}(((P_I \cup P_O) \setminus I) \cup \text{bags}(L \setminus I))$  ist. Dann ist das Wissen von  $C$  im Zustand  $q$   $K(q) := \{m \mid \exists q' : [m, q, q'] \in S\}$ .*

Die Wissensfunktion enthält Kenntnisse über das komponierte System  $M \oplus C$ , insbesondere über die Wirkung der Aktivitäten des Controllers im Netz. Daraus kann gefolgert werden, welche Aktionen eines Controllers zugelassen werden und welche nicht. Durch die Ergänzung des Verhaltens von  $C$  um  $\text{rauschen}(((P_I \cup P_O) \setminus I) \cup \text{bags}(L \setminus I))$  wird diese Definition unabhängig vom Verhalten der anderen mit  $(P_I \cup P_O \cup L) \setminus I$  verbundenen Controller, da hierdurch zu dem Verhalten von  $C$  die Gesamtheit der möglichen Aktionen dieser anderen beteiligten Controller hinzugefügt wird.

Als letzte Vorbereitung für die Beschreibung eines Algorithmus zur Konstruktion eines geeigneten Controllers werden zusätzlich zu  $\text{rauschen}(I)$  und zur Wissensfunktion  $K$  Deadlocks für offene Workflownetze  $M$  klassifiziert.

**Definition 13 (Deadlock)** *Sei  $M$  ein offenes Workflownetz. Ein Deadlock von  $M$  ist eine Markierung, die keine ungelabelte Transition aktiviert. Ein Deadlock  $m$  ist intern, wenn  $m \notin \Omega$ ,  $m(p_O) = 0$  für alle  $p_O \in P_O$  und wenn es für jede gelabelte wie ungelabelte Transition ein  $p \in P_N$  gibt, mit  $[p, t] \in F$  und  $m(p) = 0$ . Ein Deadlock  $m'$  ist extern, wenn es nicht intern ist und wenn  $m' \notin \Omega$ .*

Interne Deadlocks lassen sich nicht auflösen und sind damit endgültig. D.h. kein Controller kann ein Netz aus einem internen Deadlock befreien. Externe Deadlocks können durch geeignete Controlleraktivitäten verlassen werden, indem Marken auf die unmarkierten Inputplätze gelegt werden bzw. Label aktiviert werden.

Abbildung 9 zeigt Werte von  $K$  für einen mit dem oWFN in Abbildung 8 verbundenen Controller. Die Deadlocksituationen sind durch (e) externer Deadlock bzw. (i) interner Deadlock gekennzeichnet. Die unterschiedlich gefärbten Punkte werden zu einem späteren Zeitpunkt erläutert.

Nun wird beschrieben, wie diese Funktion  $K$  und die Kenntnis über Deadlocksituationen im Netz  $M$  zur Konstruktion geeigneter Controller genutzt wird.

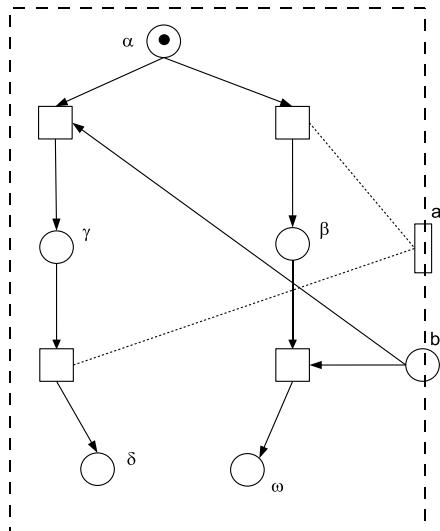


Abbildung 8: oWFN mit synchroner und asynchroner Kommunikation. Das Netz kann nur korrekt bedient werden, wenn erst die synchrone Aktion erfolgt und danach die asynchrone Nachricht empfangen wird.

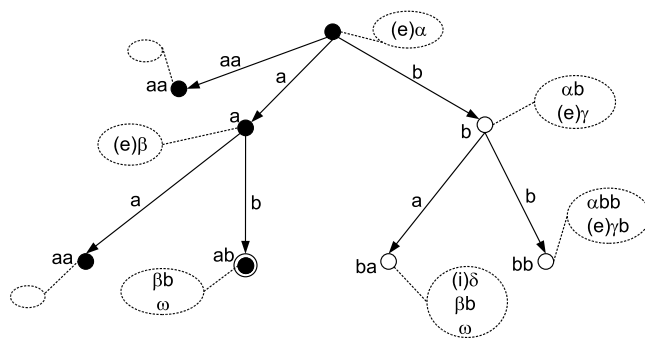


Abbildung 9: Werte von  $K$  für das oWFN aus Abbildung 8

### 3 Zentrale Bedienbarkeit

In diesem Abschnitt wird zentrale Bedienbarkeit untersucht. Es werden Strategien gesucht für oWFN in denen ein einzelner Controller mit dem kompletten Interface  $P_I \cup P_O \cup L$  verbunden ist. Zur Vereinfachung der Notation wird für die Einermenge  $\{C\}$  dieses einen Controllers in diesem Abschnitt immer  $C$  geschrieben.

**Definition 14 (zentrale Strategie)** Eine einelementige Strategie für ein oWFN  $M$  ist eine zentrale Strategie für  $M$ .

**Definition 15 (zentrale Bedienbarkeit)** Ein *oWFN*  $M$  ist zentral bedienbar, wenn es eine mit  $P_I \cup P_O \cup L$  verbundene Strategie  $C$  gibt.

**Definition 16 (liberalste Strategie)** Ein Baumcontroller  $C$  mit der Menge von Zuständen  $Q$  ist eine liberalste Strategie für  $M$ , wenn er eine Strategie ist und jeder Baumcontroller, der ebenfalls eine Strategie für  $M$  ist, eine Menge von Zuständen hat, die in  $Q$  enthalten ist.

Aus der Teilmengenbeziehung der Zustände kann insbesondere abgelesen werden, daß jede Strategie als eine Einschränkung des Verhaltens der liberalsten Strategie angesehen werden kann.

Der folgende Ansatz für zentrale Bedienbarkeit basiert auf der in [11] für asynchrone Kommunikation vorgestellten Charakterisierung von zentralen Strategien:

**Satz 2** Ein Baumcontroller  $C$ , der mit  $P_I \cup P_O \cup L$  verbunden ist, ist genau dann eine Strategie für ein *oWFN*  $M$ , wenn er eine nichtleere Menge  $Q$  von Zuständen hat und die folgenden beiden Bedingungen für alle  $q \in Q$  erfüllt:

- $K(q)$  enthält keine internen Deadlocks,
- für jeden externen Deadlock  $m \in K(q)$  hat das komponierte System  $M \oplus C$  einen Zustandsübergang, der in  $[m, q]$  beginnt.

Der im zweiten Punkt geforderte Zustandsübergang des komponierten Systems muß ein Zustandswechsel von  $C$  oder ein synchroner Schritt sein, da  $m$  ein Deadlock von  $M$  ist.

### Beweis

*Implikation* Sei  $C$  eine Strategie für  $M$  und  $q$  ein Zustand von  $C$ . Wenn  $m \in K(q)$  dann ist  $[m, q]$  im komponierten System erreichbar. Dann kann  $K(q)$  keinen internen Deadlock  $m$  enthalten, denn wenn  $m$  ein interner Deadlock wäre, wäre es unmöglich von  $[m, q]$  aus einen Zielzustand im komponierten System zu erreichen. Damit wäre  $C$  keine Strategie für  $M$  (Widerspruch zur Voraussetzung). Wenn  $K(q)$  einen externen Deadlock  $m$  enthält, dann muß  $[m, q]$  einen Nachfolger im komponierten System haben, da  $C$  sonst keine Strategie für  $M$  wäre.

*Replikation* Sei  $C$  ein Baumcontroller, der die beiden genannten Bedingungen erfüllt. Da  $M$  azyklisch ist und  $C$  ein Baumcontroller, enthält das komponierte System keine unendlichen Pfade. Deshalb führt jede Sequenz irgendwann zu einem Zustand  $[m^*, q^*]$  ohne Nachfolger im komponierten System. Wenn  $m^*$  ein Deadlock in  $M$  wäre, könnte  $m^*$  daher kein interner Deadlock sein, weil  $K(q^*)$  keinen internen Deadlock enthält (nach Voraussetzung Punkt 1).  $m^*$  kann auch kein externer Deadlock sein, weil  $[m^*, q^*]$  keinen Nachfolgezustand hat. Folglich ist  $m^*$  ein Deadlock, der weder intern noch extern ist, also eine der Endmarkierungen  $m^* \in \Omega$ .

q.e.d.

Mit der Charakterisierung aus Satz 2 kann nun eine liberalste Strategie für  $M$  konstruiert werden. Beginnend mit  $rauschen(P_I \cup P_O \cup bags(L))$  werden alle Zustände gestrichen, die zu internen Deadlocks führen. Darüber hinaus werden danach auch Zustände gestrichen, in denen externe Deadlocks keine Nachfolger haben. Das Streichen von  $q$  erfordert auch das Streichen aller  $qw$ , weil solche Zustände dann nicht mehr vom Anfangszustand aus erreichbar sind. Im Algorithmus 1 wird ein Zustandsübergang  $[q, a, q']$  des Controllers in einer Markierung  $m$  von  $M$  *möglich* genannt, wenn  $a$  entweder ein Inputplatz oder ein Outputplatz ist und sofern  $a$  ein Outputplatz ist  $m(a) > 0$  oder wenn  $a$  eine Multimenge synchroner Aktionen ist, für die eine entsprechende Menge aktivierter Transitionen in  $M$  existiert.

### Algorithmus 1

1.  $Q := \{w | w \in (P_I \cup P_O \cup bags(L))^*, length(w) \leq l_M\} \setminus \{qw | K(q) \text{ enthält interne Deadlocks, } w \in (P_I \cup P_O \cup bags(L))^*\};$
2. *WHILE*  
 $\exists q: K(q) \text{ enthält einen externen Deadlock } m \text{ und für alle } a, \text{ die in } m \text{ möglich sind, gilt } qa \notin Q$   
*DO*  
 $Q := Q \setminus \{qw | w \in (P_I \cup P_O \cup bags(L))^*\};$
3.  $\Omega_B := \{q | K(q) \cap \Omega \neq \emptyset\}$

Abbildung 9 zeigt ein Beispiel für die Anwendung dieses Algorithmus. Beginnend mit dem Automaten  $rauschen$  wird erst  $ba$  entfernt, weil  $K(ba)$  einen internen Deadlock enthält (mit (i) markiert). Weil  $ba$  einen internen Deadlock enthält, werden seine Nachfolger nicht mehr dargestellt. Im nächsten Schritt, während des ersten Durchlaufens der WHILE-Schleife, wird  $bb$  entfernt, weil  $K(bb)$  einen externen Deadlock enthält in dem in  $bb$  kein Zustandswechsel möglich ist, insbesondere weil hier überhaupt kein Zustandswechsel mehr möglich ist. Beim zweiten Durchlaufen der Schleife wird dann  $b$  entfernt, weil nach Entfernung des  $bb$  kein Zustandswechsel für den in  $K(b)$  enthaltenen externen Deadlock mehr möglich ist. Es verbleibt die Menge der Zustände  $\{\lambda, a, ab, aa\}$ , die tatsächlich die Menge der Zustände für eine Strategie für das Netz aus Abbildung 8 ist. Die Menge der Endzustände  $\Omega_B$  wird im letzten Schritt mit  $\{ab\}$  berechnet.

Abbildung 11 zeigt ein weiteres Beispiel für die Berechnung einer liberalsten Strategie. Hier gibt es wie schon in Abbildung 9 mehrere Zustände  $q$  mit  $K(q) = \emptyset$ . Es ist für die Berechnung der liberalsten Strategie insbesondere für die verteilte Bedienbarkeit wichtig, diese Zustände nicht zu löschen. Die Gründe hierfür werden im Abschnitt 4 (Dezentrale Bedienbarkeit) klar.

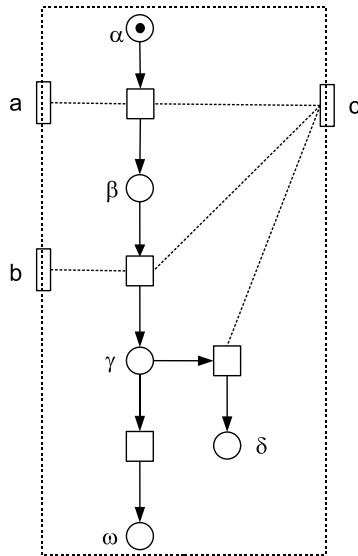


Abbildung 10: oWFN mit rein synchroner Kommunikation.

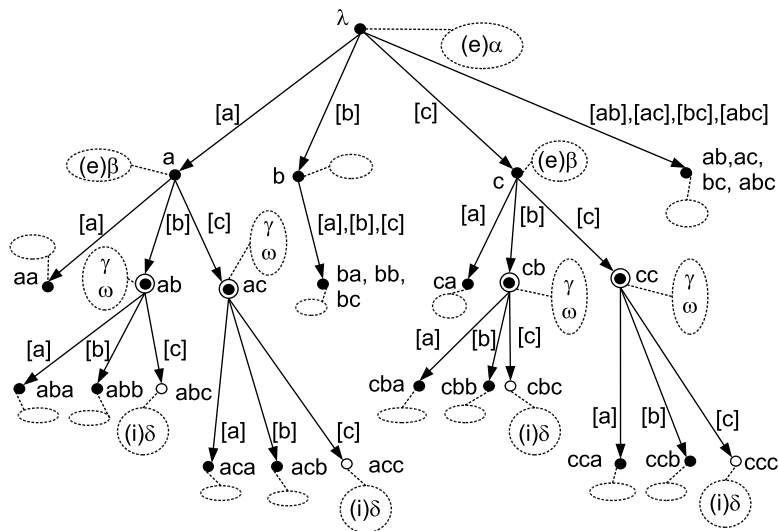


Abbildung 11: Die Abbildung  $\text{Rauschen}(\{a,b,c\})$  für das oWFN in Abb. 10. Zur besseren Übersicht wurden Schritte mehrerer gemeinsamer Aktionen und der Aktionen, die auf Aktion  $b$  folgen, zusammengefaßt dargestellt. Die Wissensfunktion ist hier jeweils leer, weil die Aktionen so nicht stattfinden können.

**Satz 3** *Algorithmus 1 berechnet entweder die leere Menge von Zuständen, dann hat  $M$  keine Strategie. Oder er berechnet eine liberalste Strategie für  $M$ .*

### Beweis

Wegen Definition 10 (Baumcontroller) hat jeder Baumcontroller für  $M$  eine Menge von Zuständen, die in  $\{w|w \in (P_I \cup P_O \cup \text{bags}(L))^*, \text{length}(w) \leq l_M\}$  enthalten ist. Keiner der Zustände, der durch Algorithmus 1 entfernt wurde, ist in der Menge der Zustände einer der Strategien enthalten, weil dies Satz 2 verletzen würde. Deshalb hat jede Strategie, die ein Baumcontroller ist, eine Menge von Zuständen, die in der von Algorithmus 1 berechneten Menge enthalten ist. In der Konsequenz folgt aus der Berechnung der leeren Menge von Zuständen, daß keine Strategie für  $M$  existiert. Wenn eine nichtleere Menge von Zuständen berechnet wird, ist es die Menge von Zuständen einer Strategie, da Satz 2 erfüllt ist, und es ist damit insbesondere die Zustandsmenge einer liberalsten Strategie.  
q.e.d.

**Korollar 1** *Wenn ein Netz zentral bedienbar ist, dann gibt es eine liberalste Strategie für  $M$ .*

## 4 Dezentrale Bedienbarkeit

Im folgenden Abschnitt wird eine Partition  $U = I_1, \dots, I_n$  der Menge von Interfaceplätzen und Label  $P_I \cup P_O \cup L$  von  $M$  vorausgesetzt. Jede Strategie gemäß Definition 9 (Strategie) bezeichnen wir im Folgenden als *dezentrale Strategie*. Diese Definition unterstellt, daß die an der dezentralen Strategie beteiligten Controller nicht untereinander, sondern ausschließlich mit dem Netz kommunizieren. Wenn die Kommunikation zwischen Controllern erlaubt würde, wären diese in der Lage, jede zentrale Strategie umzusetzen. Dezentrale Bedienbarkeit könnte so nicht ernsthaft untersucht werden. Die Situation, in der verschiedene Parteien mit einem Prozeß kommunizieren, entspricht auch den typischen Anwendungen von Geschäftsprozessen. Ein Beispiel hierfür ist ein Reiseportal bei dem die Kunden mit Flug- und Hotelbuchungssystemen kommunizieren, ohne daß diese untereinander kommunizieren (siehe Beispiel in Abbildung 1).

Wie schon im Abschnitt 3 (zentrale Bedienbarkeit) erwähnt, kann es für die korrekte Bedienung eines oWFN erforderlich sein, daß mehrere synchrone Aktionen gleichzeitig stattfinden müssen (siehe Abbildung 4). Die Partition muß in diesem Falle so vorgegeben sein, daß sich alle an diesen Aktionen beteiligten Label bei *einem* Controller (in einer Klasse der Partition) befinden. Andernfalls kann nicht sichergestellt werden, daß die Aktionen tatsächlich gleichzeitig stattfinden. Es muß darüber hinaus sichergestellt sein, daß im Falle eines zufälligen gemeinsamen Schaltens mehrerer synchroner Aktionen verschiedener Controller kein unerwünschtes Verhalten im Netz  $M$  auftritt. Sofern dies möglich ist, müssen sich auch diese Label in einer Klasse (bei einem Controller) befinden. Die Controller müssen so gestaltet sein, daß die

Reihenfolge der Aktionen verschiedener Controller keine Rolle für ein korrektes Bedienen des Netzes spielt.

Wenn eine Menge von Controllern eine dezentrale Strategie bildet, dann bildet deren parallele Komposition eine zentrale Strategie. Für die Definition der *parallelen Komposition* wird der Begriff *Projektion* benötigt und wie folgt definiert:

**Definition 17 (Projektion)**

Sei  $w$  ein Wort über dem Alphabet  $P_I \cup P_O \cup \text{bags}(L)$ . Für eine Teilmenge  $I \subseteq P_I \cup P_O \cup L$  sei die Projektion  $w_I$  von  $w$  auf  $I$  das Wort

- *Induktionsanfang:*  $\lambda_I = \lambda$
- *Induktionsschritt:* Das Wort  $wx_I$  ist
  - $w_I$ , wenn  $x \in P_I \cup P_O$  und  $x \notin I$
  - $w_Ix$ , wenn  $x \in P_I \cup P_O$  und  $x \in I$
  - $w_I$ , wenn  $x \in \text{bags}(L)$  und  $x(a) = 0$  für alle  $a \in I \cap L$
  - $w_Ix'$ , wenn  $x \in \text{bags}(L)$  und  $x'(a) = x(a)$  wenn  $a \in I \cap L$ , sonst ist  $x'(a) = 0$

Die Definition der *Projektion* ist induktiv über die Länge des Wortes aufgebaut. Es wird auf den Teil  $I$  des gesamten Interfaces projiziert, mit dem der betrachtete Controller verbunden ist. Sofern ein Buchstabe des alten Wortes aus einer asynchronen Nachricht resultiert und dieser Buchstabe (dieser Interfaceplatz) in  $I$  enthalten ist, wird er an das neu zu bildende Wort angehängt. Entspricht der Buchstabe einer asynchronen Nachricht, die aber nicht in  $I$  ist, wird dieser Buchstabe gestrichen. Wenn der Buchstabe einer Multimenge von synchronen Nachrichten entspricht, gibt es wieder zwei Möglichkeiten: Entweder gehört keiner der in der Multimenge enthaltenen Buchstaben zu  $I$ . Dann wird die gesamte Multimenge gestrichen. Oder die Multimenge wird auf die Buchstaben reduziert, die in  $I$  enthalten sind. Diese Multimenge wird als neuer Buchstabe an das zu bildende Wort gehängt. Durch die Projektion werden also aus dem Wort alle Buchstaben von asynchronen Nachrichten gestrichen, die nicht zu  $I$  gehören und aus den Multimengen von synchronen Nachrichten ebenfalls alle Buchstaben entfernt, die nicht zu  $I$  gehören.

Die *parallele Komposition von Baumcontrollern* wird nun wie folgt definiert:

**Definition 18 (parallele Komposition von Baumcontrollern)**

Seien  $C_1, \dots, C_n$  Baumcontroller, wobei  $C_i$  eine Menge von Zuständen  $Q_i$  hat und mit  $I_i$  verbunden ist. Dann ist die *parallele Komposition* von  $C_1, \dots, C_n$  der mit  $\bigcup_{i=1}^n I_i$  verbundene Controller  $C$ , der  $Q$  als Menge seiner Zustände hat, mit  $q \in Q$  gdw. für alle  $i$  gilt  $q_{I_i}$  ist die Projektion von  $q$  auf  $I_i$  und

$q_i \in Q_i$ . Die übrigen Bestandteile der parallelen Komposition ergeben sich aus der Definition Baumcontroller.

In Abbildung 12 ist am Beispiel des Netzes aus Abbildung 10 die parallele Komposition von zwei Controllern veranschaulicht.

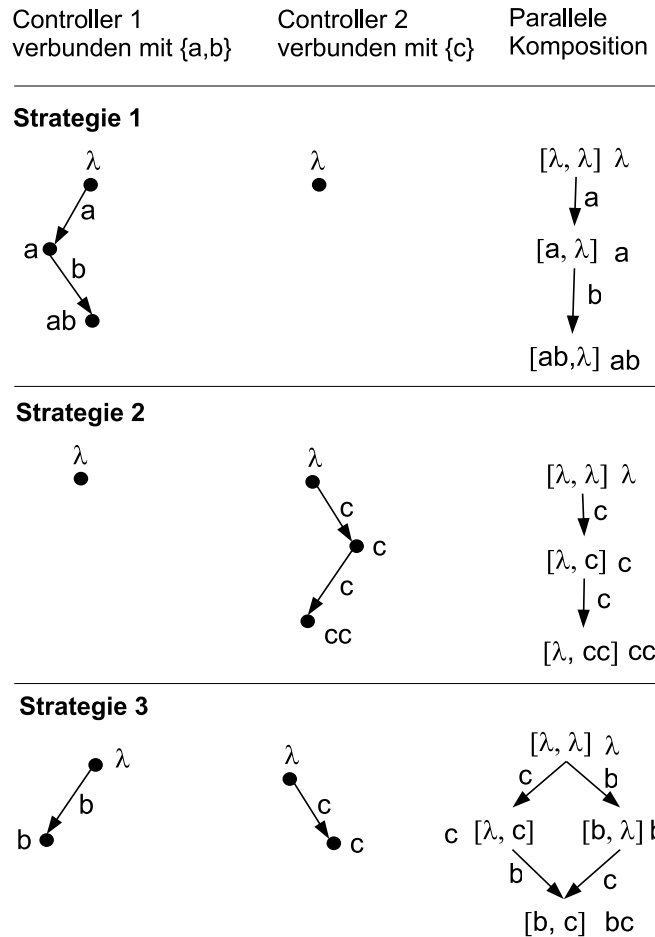


Abbildung 12: Drei Strategien bestehend aus je zwei Controllern zum oWFN aus Abbildung 10 mit den drei einzig möglichen Strategien und ihre parallele Komposition.

Mit dieser Definition ist auch die Forderung erfüllt, daß Label bei einem Controller vereint sind, die gleichzeitig schalten müssen. Beim Schalten mehrerer Label verschiedener Controller in einem Schritt befindet sich in der parallelen Komposition insbesondere der Zustand mit der Multimenge, in der alle geschalteten Label enthalten sind. Es sind darüber hinaus aber auch die Zustände in der parallelen Komposition enthalten, bei denen die synchronen Aktionen auf mehrere Multimengen aufgeteilt sind und nacheinander aus-

geführt werden und zwar in jeglicher möglicher Reihenfolge. Sofern also ein ausschließlich gemeinsames Schalten der Label gefordert ist, kann dies nur dadurch gesichert werden, daß sich diese Label bei *einem* Controller befinden.

Es wird nun eine Aussage über den Zusammenhang zwischen komponiertem System und der parallelen Komposition aufgestellt. Für den Beweis dieser Aussage wird die Bisimulation genutzt, die wie folgt definiert ist:

**Definition 19 (Simulation, Bisimulation)** *Seien  $TS_1$  und  $TS_2$  Transitionssysteme mit  $TS_1 = [S_1, E_1, s_{1_0}]$  und  $TS_2 = [S_2, E_2, s_{2_0}]$  wobei  $S_1$  ( $S_2$ ) die Menge der Zustände,  $E_1$  ( $E_2$ ) die Menge der Zustandsübergänge und  $s_{1_0}$  ( $s_{2_0}$ ) der Anfangszustand von  $TS_1$  ( $TS_2$ ) ist. Eine Simulation ist eine binäre Relation  $R$  mit  $R \subseteq S_1 \times S_2$  wobei:*

- $(s_{1_0}, s_{2_0}) \in R$ ;
- Für alle  $s \in S_1$  und  $q \in S_2$  gilt: Wenn  $(s, q) \in R$  so gilt für alle  $t_1 \in E_1$ :  
Gibt es ein  $s' \in S_1$  mit  $s \xrightarrow{t_1} s'$ , so gibt es ein  $q' \in S_2$  und ein  $t_2 \in E_2$  mit  $q \xrightarrow{t_2} q'$  und  $(s', q') \in R$ .
- Eine Bisimulation ist eine Simulation  $R$ , für die es auch die inverse Relation  $R^{-1}$  gibt, d.h. es gibt für jedes  $q' \in S_2$  mit  $q \xrightarrow{t_2} q'$  ein  $s' \in S_1$  mit  $s \xrightarrow{t_1} s'$  und  $(s', q') \in R^{-1}$ .

**Satz 4** *Wenn  $\{C_1, \dots, C_n\}$  eine dezentrale Strategie für ein oWFN  $M$  ist, dann ist ihre parallele Komposition  $C$  eine zentrale Strategie für  $M$ .*

### Beweis

Es wird gezeigt, daß das komponierte System  $M$  und  $C$  bisimilar zu dem aus  $M \oplus C_1 \oplus \dots \oplus C_n$  ist. Die komponierten Systeme  $M \oplus C$  und  $M \oplus C_1 \oplus \dots \oplus C_n$  sind Transitionssysteme. Sei  $R$  diejenige Relation mit  $([m, q], [m, q_1, \dots, q_n]) \in R$  gdw. für alle  $i = 1, \dots, n : q_i = q_{I_i}$ .

Für die Anfangsmarkierung  $m_0$  gilt offensichtlich  $([m_0, \lambda], [m_0, \lambda, \dots, \lambda]) \in R$  ( $\lambda$  ist das leere Wort).

Es ist nun zu zeigen, daß  $R$  eine Simulation ist.

Sei  $[m, q] \xrightarrow{x} [m, q']$  ein Zustandsübergang in  $M \oplus C$  und sei  $([m, q], [m, q_1, \dots, q_n]) \in R$ . Es ist zu zeigen, daß es einen Zustandsübergang  $[m, q_1, \dots, q_n] \xrightarrow{x} [m', q'_1, \dots, q'_n]$  in  $M \oplus C_1 \oplus \dots \oplus C_n$  gibt und  $([m', q'], [m', q'_1, \dots, q'_n]) \in R$ .

Gemäß Definition 8 (komponiertes System) sind folgende Zustandsübergänge in  $M \oplus C$  möglich:

- mit  $m \xrightarrow{t} m'$ , falls der Zustandsübergang intern in  $M$  stattfindet. Hier ändert sich insbesondere nichts an den Zuständen  $q \in Q$  bzw.  $q_i \in Q_i$ .
- mit  $q' = qx$ , falls der Zustandsübergang im Baumcontroller stattfindet, wobei  $x$  eine asynchrone gesendete oder empfangene Nachricht ist und in das Alphabet der Komponente  $C_i$  gehört. D. h. insbesondere, die

Komponente  $C_i$  kann auch den Zustandsübergang mit  $x$  ausführen, also  $q'_i = q_i x$ . Das Netz hat also entweder ein  $x$  gesendet oder erhalten. Für die weiteren Komponenten  $C_j$   $j \neq i$  ist  $q'_j = q_j$ . Der Effekt im Netz selbst ist daher gleich und  $([m', qx], [m', q'_1, \dots, q'_n]) \in R$ .

- mit  $q' = qy$ , falls der Zustandsübergang im Baumcontroller stattfindet, wobei  $y$  eine Multimenge von synchronen Aktionen ist und somit  $y$  die Label der Komponenten  $C_i$  enthält, die an dem Schritt beteiligt sind. D. h. es gibt für jedes  $i$  eine Multimenge  $y_i$  mit  $y_i(a) = y(a)$  falls  $a \in I \cap L$  und sonst  $y_i(a) = 0$ , mit der im Controller  $C_i$  der Zustandsübergang stattfinden kann, also  $q'_i = q_i y'$ . Sofern die Multimenge  $y_i = \emptyset$  ist  $q'_i = q_i$ . Damit können in allen, an dem Schritt beteiligten Komponenten, die Zustandsübergänge gleichzeitig stattfinden und der Effekt im Netz ist auch hier gleich. Somit ist auch  $([m', qy], [m', q'_1, \dots, q'_n]) \in R$ .

Somit ist  $R$  eine Simulation.

Es bleibt zu zeigen, daß  $R^{-1}$  eine Simulation ist.

Sei  $[m, q_1, \dots, q_n] \rightarrow [m', q'_1, \dots, q'_n]$  ein Zustandsübergang im komponierten System  $M \oplus C_1 \oplus \dots \oplus C_n$  und sei  $([m, q], [m, q_1, \dots, q_n]) \in R$ . Es ist zu zeigen, daß es einen Zustandsübergang  $[m, q] \rightarrow [m', q']$  im komponierten System  $M \oplus C$  gibt und  $([m', q'], [m', q'_1, \dots, q'_n]) \in R$ .

Auch in diesem Fall ändert sich durch einen internen Übergang nichts an den Zuständen  $q \in Q$  bzw.  $q_i \in Q_i$  und  $([m', q'], [m', q'_1, \dots, q'_n]) \in R$ . Wenn eine asynchrone Nachricht  $x$  empfangen oder gesendet wird, so entsteht ein Zustand  $q'_i = q_i x$ . Dann gibt es im parallelen komponierten System  $M \oplus C$  einen Zustand  $q$  in dem mit  $x$  ein Zustandsübergang zu einem Zustand  $q' = qx$  stattfinden kann. Dieser Zustand  $qx$  ist gemäß der Definition 18 (parallele Komposition von Baumcontrollern) insbesondere genau der Zustand  $(qx)_{I_i} = q_i x$ . Wenn  $x$  eine Multimenge von synchronen Aktionen ist, so entsteht für jedes  $i$ ,  $i = 1, \dots, n$  mit der Multimenge  $x_i$  (mit  $x_i(a) = x(a)$  falls  $a \in I \cap L$  und sonst  $x_i(a) = 0$ ) ein Zustand  $q'_i = q_i x_i$  falls  $x_i \neq \emptyset$ , sonst ist  $q'_i = q_i$ . Im parallelen komponierten System  $M \oplus C$  kann der Zustand  $q$  mit der Multimenge  $x$  in einen Zustand  $q' = qx$  übergehen. Für  $qx$  gilt gemäß der Definition 18 (parallele Komposition von Baumcontrollern) insbesondere  $(qx)_{I_i} = q_i x$ . Der Effekt im Netz bleibt somit auch hier für beide komponierten Systeme gleich. Damit gilt insbesondere, daß  $([m', q'], [m', q'_1, \dots, q'_1, \dots, q'_n]) \in R$ .

Somit ist auch  $R^{-1}$  eine Simulation und  $R$  eine Bisimulation zwischen den beiden Transitionssystemen.

q.e.d.

Die Controller in Abbildung 12 bilden jeweils eine dezentrale Strategie für das offene Workflownetz in Abbildung 10. Ihre parallele Komposition ist eine zentrale Strategie und, wie Satz 3 besagt, in der liberalsten Strategie

enthalten.

Satz 4 legt nahe, eine dezentrale Strategie zu finden, indem eine zentrale Strategie gesucht wird, die entsprechend der gegebenen Partition des Interfaces zerlegt werden kann. Eine solche Strategie muß sicherstellen, daß Zustandswechsel, die verschiedene Klassen der Interfacepartition betreffen, *unabhängig* voneinander und *synchron verteilbar* sind.

**Definition 20 (Unabhängigkeit)** *Seien  $a, b \in P_I \cup P_O \cup \text{bags}(L)$ ,  $C$  ein Baumcontroller mit einer Menge von Zuständen  $Q$  und  $q \in Q$ .*

- $a$  aktiviert  $b$  in  $q$ , wenn  $qb \notin Q$  und  $qa, qab \in Q$
- $a$  deaktiviert  $b$  in  $q$ , wenn  $qb, qa \in Q$  und  $qab \notin Q$
- zwei Zustände  $q, q'$  sind äquivalent, wenn für alle  $w : qw \in Q$  gdw.  $q'w \in Q$
- $a$  und  $b$  sind unabhängig, wenn für alle Zustände  $q : a$  und  $b$  aktivieren einander nicht und deaktivieren einander nicht und wenn  $qab, qba \in Q$ , so sind  $qab$  und  $qba$  äquivalent.

**Definition 21 (synchroner Verteilbarkeit)** *Sei  $C$  ein Baumcontroller mit einer Menge von Zuständen  $Q$ ,  $q \in Q$ ,  $\{I_1, \dots, I_k\} = U$  eine Partition des Interfaces und  $K \in \text{bags}(L)$  eine Multimenge von synchronen Aktionen.  $C$  ist synchron verteilbar, wenn Folgendes gilt:*

- falls für ein  $U' \subseteq U$  für jedes  $I_j \in U'$  eine Aktion  $K_j$  aus  $\text{bags}(I_j \cap L)$  in  $q$  aktiviert ist, dann auch  $\sum_{j: I_j \in U'} K_j$  und
- falls  $K \in \text{bags}(L)$  in  $q$  aktiviert ist, so auch jede Aktion  $K_{I_j}$  für alle  $j$  mit  $K_{I_j} \neq \emptyset$ .
- Die Zustände  $qK$  und  $qK_{I_1} \dots K_{I_k}$  sind äquivalent.

Ein parallel komponierter Baumcontroller ist demnach *synchron verteilbar*, wenn synchrone Aktionen mehrerer Partitionsklassen sowohl einzeln in beliebiger Reihenfolge als auch gleichzeitig schalten können und wenn die erreichten Zustände äquivalent sind.

Abbildung 13 zeigt eine weitere Strategie zum oWFN aus Abbildung 10. Für diese Strategie ist die Unabhängigkeit und die synchrone Verteilbarkeit verletzt. Sie ist daher keine verteilbare Strategie für die gegebene Partition.

**Satz 5** *Sei  $C$  ein Controller, der mit  $I$  verbunden ist und  $I$  partitioniert in  $I_1, \dots, I_n$ . Dann sind folgende zwei Aussagen äquivalent:*

1. *Es existieren Controller  $C_1, \dots, C_n$  ( $C_i$  verbunden mit  $I_i$ ) so, daß  $C$  die parallele Komposition von  $C_1, \dots, C_n$  ist.*
2.  *$C$  ist synchron verteilbar und für alle Zustände  $q$  von  $C$ , für alle  $j$  und alle  $a, b \in I$  mit  $a \in I_j$  und  $b \notin I_j$  folgt, daß  $a$  und  $b$  unabhängig sind.*

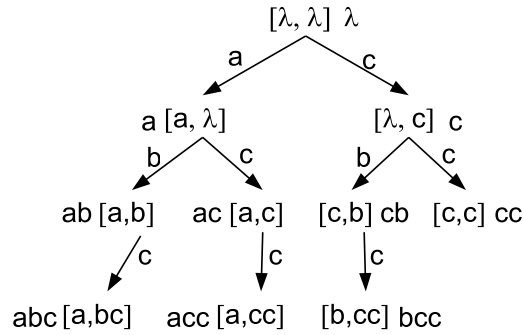


Abbildung 13: Die Abbildung zeigt eine zentrale Strategie für das Netz in Abbildung 10, die bzgl. der Ports  $\{a,b\}$  und  $\{c\}$  nicht verteilbar ist. Es ist keine Unabhängigkeit gegeben, denn im Zustand  $a$  wird  $a$  von  $c$  deaktiviert und im Zustand  $\lambda$  aktiviert  $c$   $b$ . Darüber hinaus ist die synchrone Verteilbarkeit nicht gegeben, denn die Aktionen der verschiedenen Controller können zwar nacheinander aber weder in beliebiger Reihenfolge noch gemeinsam stattfinden.

Der Beweis des Satzes 5 unterteilt sich in die Implikation und die Replikation. Zur besseren Veranschaulichung der Replikation wird vorab die in Replikation benutzte Konstruktion mit den dabei durchgeführten Umwandlungsschritten an einem **Beispiel** demonstriert.

Es werden zwei Controller  $C_1$  und  $C_2$  mit den Zustandsmengen  $Q_1 = \{q_{I_1} | q \in Q\}$  und  $Q_2 = \{q_{I_2} | q \in Q\}$  betrachtet, die durch Projektion aus einem unabhängigen, synchron verteilbaren Controller erstellt wurden. Die Ports der beiden Controller seien  $I_1 = \{a, b\}$  und  $I_2 = \{c, d, e\}$  wobei  $a$  und  $c$  asynchrone Nachrichten sind und  $b, d, e$  synchrone Aktionen. Der Controller  $C^*$  ist der Controller, der durch parallele Komposition der beiden Controller entsteht. Es wird nun ein  $q^* \in Q^*$  mit  $q^* = a[db]c[deb][b]$  gewählt. Für dieses  $q^* \in Q^*$  wird nun die Zugehörigkeit zu  $Q$  gezeigt. Die Projektionen von  $q^*$  auf  $I_1$  und  $I_2$  sind:  $q_{I_1}^* = a[b][b][b]$  mit  $q_{I_1}^* \in Q_1$  und  $q_{I_2}^* = [d]c[de]$  mit  $q_{I_2}^* \in Q_2$ . Das Vorgehen für den Zwischenschritt wird nun am Beispiel von  $q_{I_2}^*$  gezeigt. Weil  $q^* \in Q^*$  muß es einen Zustand  $q' \in Q$  geben, so daß  $q_{I_2}^*$  in dem Zustand enthalten ist:  $q'$  sei  $a[d]c[deb]aa$ . Aufgrund der synchronen Verteilbarkeit kann nun die Multimenge  $[deb]$  auf die einzelnen Ports verteilt werden. Es entsteht das Wort  $q'' = a[d]c[de][b]aa$ . Wegen der Unabhängigkeit können nun die Aktionen von  $C_2$  nach vorn verschoben werden. Es entsteht  $\{[d]c[de]a[b]aa\} \in Q$ . Wegen der Präfixabgeschlossenheit ist dann auch  $\{[d]c[de]\} = q_{I_2}^*$  in  $Q$  enthalten. Hier ist der Zwischenbeweis abgeschlossen. Nun wird noch veranschaulicht, warum das gesamte Wort  $q^*$  in  $Q$  enthalten ist. Wenn die einzelnen  $q_{I_j}^*$  in  $Q$  enthalten sind so auch deren Hintereinanderausführung, in unserem Beispiel also auch das Wort  $a[b][b][b][d]c[de]$ . Wie-

derum wegen der Unabhängigkeit, aufgrund derer ja alle Durchmischungen in  $Q$  enthalten sind, ist auch das Wort  $a[d][b]c[de][b][b]$  in  $Q$  enthalten. Wegen der synchronen Verteilbarkeit ist dann auch das Wort  $q^* = a[db]c[deb][b]$  in  $Q$ .

Abbildung 14 zeigt das Beispiel noch einmal in der Übersicht.

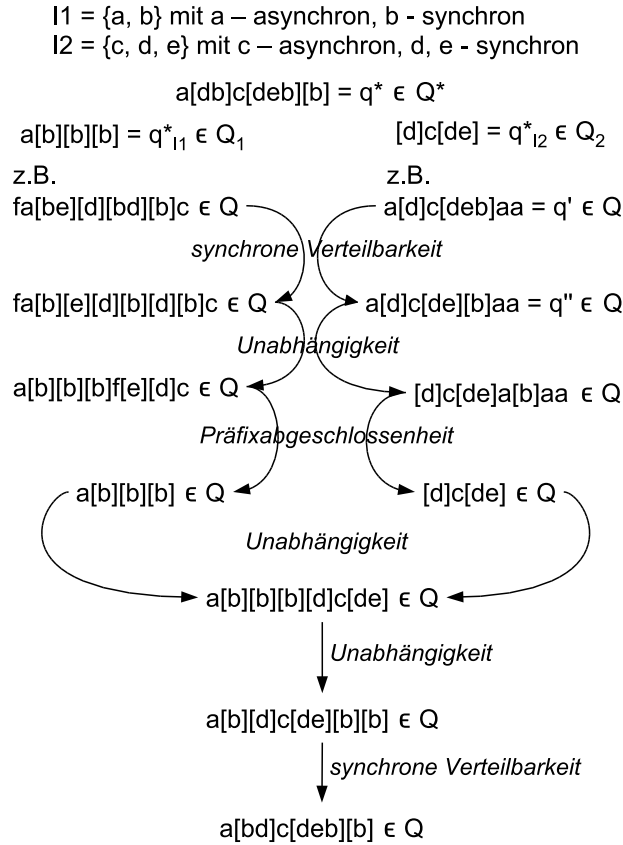


Abbildung 14: Veranschaulichung des Beispiels zum Beweis von Satz 5

### Beweis

*Implikation* Sei  $C$  die parallele Komposition der  $C_i$ . Sei  $q$  ein Zustand von  $C$ . Sei  $a \in I_j$  und  $b \in I_k$  ( $j \neq k$ ). Dann folgt aus  $qa, qab \in Q$ , daß  $qb \in Q$ , weil  $qab_{I_k} = qb_{I_k}$ . Also wird  $b$  nicht von  $a$  aktiviert.

Aus  $qa, qb \in Q$  folgt  $qab \in Q$ , weil gemäß Definition 17 (Projektion)  $qab_{I_j} = qa_{I_j}$ . Also wird  $b$  nicht von  $a$  aktiviert.

Für alle Sequenzen  $w$  und alle  $h, h \in 1, \dots, n$ , gilt darüber hinaus, daß  $qabw_{I_h} = qbaw_{I_h}$ .  $qab, qba$  sind somit äquivalent.

Daraus folgt, daß  $a$  und  $b$  unabhängig sind.

Sei  $U' \subseteq U$ ,  $U = \{I_1, \dots, I_n\}$  und sei für jedes  $I_j \in U'$  eine Multimenge  $K_j$  von synchronen Aktionen in  $q$  aktiviert. Dann ist nach Definition 18 (parallele

Komposition) auch  $\sum_{j:I_j \in U'} K_j$  in  $q$  aktiviert.

Sei  $K \in \text{bags}(L)$  in  $q$  aktiviert. Dann folgt ebenfalls nach Definition 18 (parallele Komposition) unmittelbar, daß jede Multimenge  $K_{I_j}$  für alle  $j$  mit  $K_{I_j} \neq \emptyset$  in  $q$  aktiviert ist.

Damit ist  $C$  synchron verteilbar.

*Replikation* Sei  $C$  ein Controller, für den Unabhängigkeit und synchrone Verteilbarkeit gilt,  $Q$  die Menge seiner Zustände. Seien  $C_j$  die Controller mit  $Q_j = \{q_{I_j} | q \in Q\}$  und  $Q^*$  die Zustandsmenge der parallelen Komposition  $C^*$  der  $C_1, \dots, C_n$ . Es ist zu zeigen, daß  $Q^* = Q$ .

1. Es ist zu zeigen, daß  $Q^* \subseteq Q$ .

Sei  $q \in Q$ . Dann ist zu zeigen, daß  $q \in Q^*$ . Nach Annahme ist  $q_{I_1} \in Q_1, \dots, q_{I_n} \in Q_n$ . Nach Definition 18 (parallele Komposition) ist dann auch  $q \in Q^*$ .

2. Es ist weiterhin zu zeigen, daß  $Q \subseteq Q^*$ .

Sei  $q^* \in Q^*$ . Dann ist zu zeigen, daß  $q^* \in Q$ . Wenn  $q^* \in Q^*$ , dann gilt für alle  $j$ , daß  $q_{I_j}^* \in Q_j$ . Als Zwischenziel ist nun zu zeigen, daß  $q_{I_j}^* \in Q$  ist.

Beweis: Es gilt  $q_{I_j}^* \in Q_j$ . Also gibt es ein  $q' \in Q$  mit  $q_{I_j}^* = q'_{I_j}$ . Aus  $q'$  kann man wegen der synchronen Verteilbarkeit ein  $q''$  generieren, in dem alle *synchronen* Aktionen jeweils Multimengen über *einem* Port sind. Wegen der Unabhängigkeit sind alle Durchmischungen in  $Q$  enthalten. Somit kann man alle Aktionen von  $I_j$  nach vorn tauschen. D.h. alle Aktionen dieses Controllers werden im parallelen System vor denen der anderen Controller ausgeführt. Dieser Zustand hat dann als Präfix  $q_{I_j}^*$ . Wegen der Präfixabgeschlossenheit ist dieser Zustand  $q_{I_j}^*$  dann auch in  $Q$  enthalten.

Nun bleibt zu zeigen, daß  $q^* \in Q$ . Da die Zustände  $q_{I_j}^* \in Q$  für alle  $j$ , ist wegen der Unabhängigkeit auch der Zustand  $q_{I_1}^* \dots q_{I_n}^* \in Q$ . Wiederum wegen der Unabhängigkeit ist auch derjenige Zustand  $q^{**}$  in  $Q$ , der sich von  $q^*$  lediglich darin unterscheidet, daß synchrone Aktionen, die in  $q^*$  stattfinden, sich in  $q^{**}$  als Sequenz ihrer Projektionen auf die einzelnen  $I_j$  wiederfinden. Aufgrund der synchronen Verteilbarkeit ist dann auch  $q^* \in Q$ .

q.e.d.

Dezentrale Strategien können damit durch eine Erweiterung des Algorithmus 1 berechnet werden. In der Schleife werden hierfür nun nicht nur die Zustände  $q$  mit externen Deadlocks entfernt, sondern auch die Zustände, die für eine Verletzung der Unabhängigkeit und der synchronen Verteilbarkeit verantwortlich sind.

## Algorithmus 2

1.  $Q := \{w | w \in (P_I \cup P_O \cup \text{bags}(L))^*, \text{length}(w) \leq l_M\} \setminus \{qw | K(q) \text{ enthält interne Deadlocks, } w \in (P_I \cup P_O \cup \text{bags}(L))^*\}$   
 $U := \{I_1, \dots, I_n\};$
2. REPEAT UNTIL  $Q$  bleibt unverändert
  - a) IF  $\exists q: K(q)$  enthält einen externen Deadlock  $m$  und für alle  $a$ , die in  $m$  möglich sind, gilt  $qa \notin Q$  THEN lösche  $q$
  - b) IF  $\exists(q, a, b), a \in I_k, b \in I_j, j \neq k: a$  aktiviert  $b$  in  $q$  THEN lösche  $qab$
  - c) IF  $\exists(q, a, b), a \in I_k, b \in I_j, j \neq k: a$  deaktiviert  $b$  in  $q$  THEN lösche  $qa$  oder lösche  $qb$
  - d) IF  $\exists(q, a, b), a \in I_k, b \in I_j, j \neq k: qab, qba \in Q$  und  $qab, qba$  sind nicht äquivalent THEN lösche alle  $qabw$  für die  $qbaw \notin Q$  und lösche alle  $qbaw$  für die  $qabw \notin Q$
  - e) IF  $\exists(q, a_1, \dots, a_n): a_j \in \text{bags}(L \cap I_j), 1 \leq j \leq n$  und entweder  $a_j = \emptyset$  oder  $qa_j \in Q$  und  $q\sum_{j=1}^n a_j \notin Q$  THEN wähle ein  $j$  mit  $a_j \neq \emptyset$  und lösche  $qa_j$
  - f) IF  $\exists(q, a), a \in \text{bags}(L): \exists j$  mit  $qa|_{I_j} \notin Q$  THEN lösche  $qa$
  - g) IF  $\exists(q, a_1, \dots, a_n): a_j \in \text{bags}(L \cap I_j), 1 \leq j \leq n$  und entweder  $a_j = \emptyset$  oder  $qa_j \in Q$  und  $q\sum_{j=1}^n a_j, qa_1 \dots a_n$  sind nicht äquivalent THEN lösche  $qa$  und wähle ein  $j$  mit  $a_j \neq \emptyset$  und lösche  $qa_j$
3. WHERE lösche  $q = Q := Q \setminus \{qw | w \in (P_I \cup P_O \cup \text{bags}(L))^*\};$
4.  $\Omega_B := \{q | K(q) \cap \Omega \neq \emptyset\}.$

**Beobachtung** Die Schritte, die das Deaktivieren („lösche  $qa$  oder lösche  $qb$ “) und die synchrone Verteilbarkeit („wähle ein  $j$  mit  $a_j \neq \emptyset$  und lösche  $qa_j$ “) betreffen, enthalten Nichtdeterminismus. Wie Abbildung 12 veranschaulicht, ist dieser Nichtdeterminismus (bzw. Backtracking) erforderlich. Es ist, wie in [11] dargestellt, eine der wenigen Methoden, Symmetrie zu durchbrechen und damit von einer symmetrischen Ausgangssituation zu den einzigen (nichtsynchronen) Lösungen zu kommen.

**Satz 6** Ein offenes Workflownetz  $M$  ist dezentral bedienbar bzgl. einer Partition  $\{I_1, \dots, I_n\}$  eines Ports, genau dann, wenn Algorithmus 2 mindestens eine nichtleere Menge von Zuständen zurückgibt.

### Beweis

Wegen Satz 2 und 4 ist jede Menge von Zuständen, für die keine der Bedingungen innerhalb der Schleife anwendbar ist, eine zentrale Strategie und

parallele Komposition einer Menge von Controllern, die infolgedessen eine dezentrale Strategie ist. Wenn eine Menge von Zuständen eine der Situationen der IF-Anweisungen enthält, können die gelöschten Zustände (im Falle der Deaktivierung und der synchronen Verteilbarkeit nur einer der möglichen) in keiner Teilmenge von  $Q$  auftreten, die sowohl Strategie als auch dekomponierbar ist.  
q.e.d.

Die in Abbildung 12 dargestellten Strategien sind die einzigen Strategien für die gegebene Partition. Keine der drei ist liberaler als die beiden anderen. Somit zeigt das Beispiel einen der Fälle, in denen es nicht eine einzige liberalste dezentrale Strategie gibt. Durch Backtracking (für Punkt 2. (c) Deaktivieren) wird bei dem vorgestellten Algorithmus eine solche Menge von Strategien berechnet, daß für jede überhaupt existierende Strategie gilt, daß eine der berechneten mindestens so liberal ist wie diese.

## 5 Verwandte Arbeiten

Für reine Workflownetze ohne Interaktion mit der Umgebung wurde der Begriff der *Soundness* charakterisiert [3]. Soundness fordert, daß ein Netz von jedem erreichbaren Zustand aus in einen Endzustand kommen kann und darüber hinaus wird gefordert, daß keine Transition tot ist, d. h. jede Transition erreichbar ist. Der Begriff der Soundness ist die Grundlage für den hier vorgestellten Bedienbarkeitsbegriff.

Das Bedienbarkeitsproblem kann auch als Modelcheckingproblem für alternatingtime Temporallogik formuliert werden [2]. Das eher generellere Modelcheckingproblem für diese Logik mit unvollständigen Informationen ist allerdings unentscheidbar.

Bei supervisory control, die unter anderem Petrinetzmodelle als Netze nutzt, basiert die Steuerung eher auf der Auswahl steuerbarer Transitionen als auf Nachrichtenaustausch. Dort werden vollständig beobachtbare Petrinetze [17, 7] unterstellt, Klassen von Petrinetzen betrachtet, die mit den hier untersuchten Netzen nicht vergleichbar sind [20, 13, 10, 1, 18] oder andere Ziele für Steuerung aufgestellt [5, 13, 1, 14].

Controllerprobleme für generelle diskrete Systeme oder generell Petrinetze wurden untersucht durch [16, 17, 15, 12]. Soweit sie in die hier untersuchten Rahmenbedingungen passen, betrachten sie wesentlich allgemeinere Situationen und erhalten damit schwächere Resultate.

## 6 Zusammenfassung

Es wurden Probleme der Steuerung einer speziellen Klasse von Petrinetzen untersucht. Sowohl den Netzen als auch den Steuerungsproblemen liegen

mögliche praktische Anwendungen verteilter Geschäftsprozesse zugrunde.

Es wurde gezeigt, daß auch für Netze mit synchroner Kommunikation eine liberalste zentrale Strategie existiert, sofern überhaupt eine Strategie existiert. Auch die in [11] vorgestellte Lösung des Problems der dezentralen Bedienbarkeit konnte unter Berücksichtigung zusätzlicher Bedingungen bestätigt werden. Es wird nach einer speziellen zentralen Strategie gesucht. Es wurde auch hier bestätigt, daß nicht immer eine liberalste Strategie existiert. Im Unterschied zur rein asynchroner Kommunikation können synchrone Aktionen nicht zerlegt und einzeln betrachtet werden.

Die vorliegenden Resultate sind zum jetzigen Zeitpunkt noch nicht implementiert. So kann nicht untersucht werden, ob die Ansätze für praktische Anwendungen effizient genug implementierbar sind.

## 7 Ausstehende Arbeiten

Zur Vervollständigung der vorliegenden Untersuchungen sollten für die hier vorgestellten Netze die *autonome Bedienbarkeit* und *Bedienungsanleitungen* untersucht werden.

Für die autonome Bedienbarkeit wird für den dezentralen Fall ein Controller gesucht, der keine Kenntnisse über die restlichen mit dem Netz verbundenen Controller hat und Teil einer Strategie ist. In den bereits vorliegenden Arbeiten wurde für rein asynchron kommunizierende Netze gezeigt, daß jede Zusammenstellung solcher als *kooperativ* charakterisierten Controller ein offenes Workflownetz bedienen. Es ist nun zu untersuchen, ob bzw. unter welchen Bedingungen dies auch bei synchroner und gemischt synchron asynchroner Kommunikation gilt.

Eine Bedienungsanleitung ist eine Charakterisierung aller Strategien für ein Netz. Für azyklische oWFN besteht eine solche Bedienungsanleitung aus der liberalsten Strategie angereichert um zusätzlichen Annotationen. Mit den Bedienungsanleitungen soll es einem Geschäftspartner möglich sein, einen bedienenden Controller für ein gegebenes offenes Workflownetz (einen gegebenen Geschäftsprozeß) zu generieren, ohne daß er Kenntnisse über die internen Abläufe im offenen Workflownetz selbst hat. Dieses Konzept hat eine hohe Praxisrelevanz, da damit gesichert wird, daß solche im allgemeinen schützenswerten Unternehmensinterna der Arbeitsabläufe des Unternehmens nicht veröffentlicht werden und trotzdem für potentielle Geschäftspartner die Bedingungen für eine Zusammenarbeit klar sind.

Eine weitere offene Problemstellung ist die Untersuchung zyklischer oWFN. Hierzu liegen bisher noch keine Veröffentlichungen vor.

Für die Implementation der Algorithmen für rein asynchrone Kommunikation wurden Reduktionstechniken entwickelt [19]. Ähnliche Reduktionen können auch für die Netze mit synchroner und gemischter Kommunikation entwickelt werden.

## Literatur

- [1] A. Ghaffari, N. Rezg und X. Xie. Feedback control logic for forbidden state problem of marked graphs. *IEEE Transactions on Automatic Control* 48, pages 18–29, 2003.
- [2] A. Martens, H. Schlingloff und K. Schmidt. Modeling and Model Checking Web Services. *Electronic Notes in Theoretical Computer Science* 126, pages 3–26, 2005.
- [3] W. M. P. van der Aalst. The application of Petri Nets to workflow management. *Journal of Circuits, Systems and Computers* 8(1), pages 21–66, 1998.
- [4] E. Badouel and P. Darondeau. Theory of regions. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 529–586, 1998.
- [5] K. Yamalidou, J. Moody, M. Lemmon und P. Antsaklis. Feedback control of PNs based on place invariants. *Automatica* 32(1), pages 15–28, 1996.
- [6] Ekkard Kindler. A Compositional Partial Order Semantics for Petri Net Components. *Application and Theory of Petri Nets, LNCS 1248*, 1997.
- [7] L. Holloway, X. Guan und L. Zhang. A generalization of state avoidance policies for controlled PN. *IEEE Transactions on Automatic Control* 39, pages 512–531, 1996.
- [8] M. Nielsen, G. Rozenberg und P.S. Thiagarajan. Elementary transition systems. *Theoretical computer science* 96, pages 3–33, 1992.
- [9] Peter Massuthe, Wolfgang Reisig, and Karsten Schmidt. An Operating Guideline Approach to the SOA. *Annals of Mathematics, Computing & Teleinformatics*, 1(3):35–43, 2005.
- [10] R.K. Boel, B. Bordbar und G. Stremersch. A min-plus polynomial approach to forbidden state control for general PNs. *Proc. WODES*, pages 79–84, 1998.
- [11] Karsten Schmidt. Controllability of Open Workflow Nets. In Jörg Dessel and Ulrich Frank, editors, *Enterprise Modelling and Information Systems Architectures*, volume P-75 of *Lecture Notes in Informatics (LNI)*, pages 236–249, 2005.
- [12] G. Stremersch. Linear algebraic design of supervisors for partially observed PN. *Proc. CSD Bratislava*, pages 281–286, 2000.
- [13] L. Holloway und B. Krogh. Synthesis of feedback control logic for a class of controlled PNs. *IEEE Transactions on Automatic Control* 35, pages 514–523, 1990.
- [14] A. Giua und C. Seatzu. Observability of place/transition nets. *Transactions on Automatic Control* 47, pages 1424–1437, 2002.

- [15] J.O. Moody und P.J. Antsaklis. PN supervisors for DES with uncontrollable and unobservable transitions. Tech. Report ISIS-99-04, Fevrier, 1999.
- [16] P.J. Ramadge und W.M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control and Optimization* 25(1), pages 206–230, 1987.
- [17] Y. Li und W.M. Wonham. Control of vector discrete event systems II - controller synthesis. *IEEE Transactions on Automatic Control* 39, pages 512–531, 1994.
- [18] P. Darondeau und X. Xie. Linear control of live marked graphs. *Automatica* 39(3), page 429/440, 2003.
- [19] Daniela Weinberg. Reduction Rules for Interaction Graphs. Informatik-Berichte 198, Humboldt-Universität zu Berlin, February 2006.
- [20] Z. Achour, N. Rezg und X. Xie. Supervisory control of partially observable marked graphs. *IEEE Transactions on Automatic Control* 49(11), pages 2007–2011, 2004.