

Modellierung und Analyse transaktionaler Geschäftsprozesse

Diplomarbeit

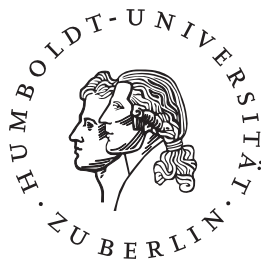
Carsten Frenkler

1. Juli 2005

Gutachter:

Dr. habil. Karsten Schmidt

Prof. Dr. Wolfgang Reisig



Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik

Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit zwei wichtigen Aspekten von Geschäftsprozessen. Das ist einerseits die Frage, ob die Funktionalität eines Prozesses genutzt werden kann und andererseits die Frage, ob die Transaktionen eines Prozesses vernünftig verarbeitet werden können. Für die Analyse der beiden Aspekte wird ein geeignetes Modell benötigt.

Hierfür werden die auf Petrinetzen basierenden Workflow-Module erweitert, indem transaktionale Eigenschaften in ein Petrinetz übersetzt werden und in das Modul integriert werden. Letztlich liefert die Analyse der Bedienbarkeit des erweiterten Moduls eine Antwort auf beide Fragen.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation	1
1.2	Ziel der Arbeit	2
1.3	Gliederung der Arbeit	3
2	Grundlagen	5
2.1	Anwendungsgebiet	5
2.2	Modellierungsmethode	6
2.3	Petrinetze / Workflow-Modul	8
2.4	Umgebung	10
2.5	Bedienbarkeit	13
3	Transaktionale Eigenschaften	17
3.1	Hintergrund	17
3.2	Transaktionsbegriff/Schedule	18
3.3	Schöne Schedules	19
3.3.1	Eigenschaften von Schedules in Bezug auf einzelne Transaktionen	19
3.3.2	Eigenschaften von Schedules bezüglich mehrerer Transaktionen	21
4	Transaktionale Workflow-Module	27
4.1	Einleitung	27
4.2	taWFM	28
4.3	Petrinetz-Muster	33
4.3.1	Konstruktion von Transaktionen	34
4.3.2	Synchronisation von Transaktionen	36
4.3.3	Synchronisierte taWFM	54
5	Analyse transaktionaler Workflow-Module	55
6	Zusammenfassung und Ausblick	65
6.1	Modellierung	65
6.2	Analyse	66
6.3	Ausblick	66
	Literaturverzeichnis	69

Abbildungsverzeichnis

2.1	Workflow-Module M1 und M2	10
2.2	Umgebungen U1, U2, U3	12
2.3	komponierte Systeme $M1 \oplus U1$ und $M2 \oplus U1$	14
2.4	Wissen der Umgebung U1	16
4.1	gefärbtes Workflow-Modul	29
4.2	Eigenschaftsnetze E1,E2,E3	31
4.3	Eigenschaftsnetz E4	32
4.4	Eigenschaftsnetz E_{ta_1}	35
4.5	Eigenschaftsnetz $E_{ta_1} \parallel E_{ta_2}$	36
4.6	Synchronisation Lesen von x	39
4.7	Synchronisation Schreib-/Schreibkonflikt	40
4.8	Synchronisation Lese-/Schreibkonflikt	41
4.9	Synchronisation Lesen neben Lesen/Schreiben	41
4.10	Synchronisation Lesen/Schreiben neben Schreiben	42
4.11	Synchronisation Lesen/Schreiben neben Lesen/Schreiben	42
4.12	Synchronisation von vier Transaktionen	43
4.13	Invariante vom Typ 1	46
4.14	Invariante vom Typ 2	48
4.15	Eigenschaft8-Synchronisation	51
4.16	Invariante vom Typ 3	53
4.17	taWFM Π_1	54
5.1	taWFM Π_2	57
5.2	Umgebung U2	57
5.3	komponiertes System $gM_2 \oplus U2$	58
5.4	komponiertes System $\Pi_2 \oplus U2$	58
5.5	Wissen $\Pi_2 \oplus U2$	60
5.6	taWFM Π_3	61
5.7	Umgebung U1	61
5.8	Wissen $gM_3 \oplus U1$	62
5.9	Wissen $\Pi_3 \oplus U1$	63

1 Einleitung

Diese Arbeit befaßt sich mit der Modellierung und Analyse von Geschäftsprozessen. Für einen Modellierer ist die Frage, ob die Funktionalität eines Prozesses von einem Benutzer genutzt werden kann, der Prozeß also bedienbar ist, von grundlegender Bedeutung. Für die technische Realisierung eines Prozesses besteht für einen Modellierer darüber hinaus die Notwendigkeit, nachzuweisen, daß die Datenzugriffe eines Prozesses von einem transaktionsverarbeitenden System abgearbeitet werden können. Wir werden uns mit beiden Problemen in dieser Arbeit beschäftigen und einem Modellierer ein Verfahren an die Hand geben, mit dem er beide Probleme gleichzeitig lösen kann.

1.1 Motivation

Der Erfolg eines Unternehmens hängt maßgeblich von den Produkten oder Dienstleistungen ab, die es am Markt anbietet und der Effizienz, mit der die Produkte und Dienstleistungen erstellt werden. Aufgrund der zunehmenden Globalisierung des Wettbewerbs ist in vielen Bereichen gerade die Effizienz der entscheidende Wettbewerbsvorteil eines Unternehmens. Damit ist die Art und Weise, wie ein Produkt erstellt wird und der Einsatz von Ressourcen ein wichtiges Kriterium, um konkurrenzfähig zu sein. Für viele Unternehmen sind deshalb die eigenen Geschäftsprozesse eine enorm wichtige Grundlage des Erfolges.

Globaler Wettbewerb bedeutet für viele Unternehmen, die Effizienz ihrer Abläufe weiter zu steigern. Daraus resultiert die Notwendigkeit, die eigenen Geschäftsprozesse enger mit den Prozessen der Partner zu verzahnen und Ressourcen und Dienstleistungen schnell und flexible in die eigenen Abläufe einbinden zu können. Infolge dessen geht der Trend seit einigen Jahren dahin, verteilt ablaufende lokale Geschäftsprozesse zu einem verteilten Geschäftsprozeß zu kombinieren. Dabei bietet ein Unternehmen seinen lokalen Geschäftsprozeß als ein Service zur Benutzung an.

Für die Umsetzung eines solchen Services auf der Basis von IT-Systemen setzt sich zunehmend die *Web-Service-Architektur* durch. Ein Web-Service ist eine lokal abgegrenzte Komponente mit einem definierten Interface. Über das Interface wird die Funktionalität eines Web-Service zur Benutzung bereitgestellt. Die Realisierung eines lokalen Geschäftsprozesses als Web-Service bietet einem Unternehmen die Möglichkeit, kombiniert mit anderen Web-Services, den lokalen Geschäftsprozeß in einen verteilten Geschäftsprozeß einzubinden oder andere Web-Services, für einzelne Aufgaben zu nutzen [Kre01].

Bei der Entwicklung eines Web-Services gibt es Problem, die es zu lösen gilt. Dazu zählt insbesondere die Frage, ob ein Service benutzt werden kann. Das heißt, gibt es eine *Umgebung*, die über das Interface den lokalen Prozeß so steuern kann, daß der Prozeß zu einem definierten Ende kommt, ohne daß dabei Nachrichten ignoriert werden? Auf der Basis, der in [Mar03] vorgestellten *Workflow-Module*, können Geschäftsprozesse mit den Ansätzen aus [Mar03], [Wei04] und [Sch04] auf die Existenz einer solchen Umgebung untersucht werden.

In allen drei Ansätzen wird jedoch von Zugriffen auf Daten abstrahiert. Unter Berücksichtigung von Datenzugriffen ergibt sich ein zweites grundlegendes Problem. Die unter dem Begriff der Transaktionsverarbeitung bekannten Konzepte zur Verarbeitung von Datenzugriffen auf Datenbanken dienen der Erhaltung der Integrität der Daten einer Datenbank. Dabei werden Zugriffe auf eine Datenbank nur ausgeführt, wenn sie in einer geeigneten Reihenfolge vorliegen. Damit hat die Verarbeitung von Datenzugriffen eines Prozesses Einfluß auf den Kontrollfluß. Der Einfluß kann so groß sein, daß die Funktionalität eines Geschäftsprozesses von einer Umgebung nicht mehr nutzbar ist. Da der Kontrollfluß direkten Einfluß auf die Bedienbarkeit hat, sollten beide Probleme gemeinsam analysiert werden. Wir werden deshalb den Einfluß der Transaktionsverarbeitung auf den Kontrollfluß eines Geschäftsprozesses untersuchen und als Erweiterung für Workflow-Module modellieren und das Gesamtsystem analysieren.

Diese Arbeit versteht sich als Einstieg in die Untersuchung der Bedienbarkeit unter Berücksichtigung von Transaktionskonzepten. Deshalb werden wir in dieser Arbeit die Konzepte der klassischen Transaktionsverarbeitung näher beleuchten, das heißt der Verarbeitung von Transaktionen nach dem *ACID-Prinzip* [GR94], [VGH93]. Die Konzepte der klassischen Transaktionsverarbeitung haben im Umfeld von Geschäftsprozessen auch ihre Grenzen. Deshalb existieren auch eine Reihe weiterer Konzepte, in denen die recht starren ACID-Prinzipien gelockert werden [GR94],[SABS02], [SAS99], [EL96]. Um jedoch komplexere Transaktionskonzepte in die Analyse der Bedienbarkeit mit einbeziehen zu können, ist die Untersuchung der grundlegenden Konzepte wichtig.

1.2 Ziel der Arbeit

Ziel dieser Arbeit ist es, ein Verfahren zu entwickeln, mit dem es gelingt, Bedienbarkeit und die Einhaltung transaktionaler Eigenschaften eines Geschäftsprozesses gemeinsam analysieren zu können. Wir wollen damit sicherstellen, daß die Funktionalität eines Prozesses auch unter Berücksichtigung von Datenzugriffen auf eine Datenbank genutzt werden kann und eine Datenbank in der Lage ist, alle Datenzugriffe ordnungsgemäß zu verarbeiten.

Für die Umsetzung dieses Ziels benötigen wir ein Modell zur Darstellung von transaktionalem Verhalten. Ein solches Modell muß die Modellierung von Transaktionen

und deren Synchronisation unterstützen. Dafür werden wir Eigenschaftsnetze auf der Basis von Petrinetzen einführen und Konstruktionsalgorithmen für die Erstellung von Eigenschaftsnetzen angeben.

Ein weiteres Teilziel besteht darin, Workflow-Module um transaktionales Verhalten zu erweitern. Dafür müssen Eigenschaftsnetze in Workflow-Module integriert werden. Die Integration soll genau so erfolgen, daß die Analyse der Bedienbarkeit auf das neue Modul, dem transaktionalen Workflow-Modul, anwendbar ist.

Wir geben einem Modellierer damit die Möglichkeit, mit einem Verfahren entscheiden zu können, ob sein Geschäftsprozeß bedienbar ist und mit einer Datenbank zusammenarbeiten kann. Sollten bei der Analyse Probleme in der Struktur eines Prozesses auftreten, so sind sie erkennbar, und der Prozeß kann entsprechend angepaßt und wiederum analysiert werden.

1.3 Gliederung der Arbeit

Die vorliegende Arbeit besteht neben dieser Einleitung aus fünf weiteren Kapiteln. Im zweiten Kapitel werden die notwendigen Grundlagen erklärt. Hierbei geht es neben einer Beschreibung des Umfeldes, um die Einführung des Petrinetzformalismus und deren Eignung für die Modellierung von Geschäftsprozessen. Außerdem wird die Modellierung mit Workflow-Modulen vorgestellt und deren Analyse auf Bedienbarkeit beschrieben. Im dritten Kapitel werden die Begriffe Transaktion, Schedule und transaktionale Eigenschaften von Schedules formal definiert. In Kapitel 4 wird beschrieben, wie transaktionale Eigenschaften, als internes Verhalten, in Workflow-Module integriert werden. Dafür werden gefärbte Workflow-Module, Eigenschaftsnetze und zuletzt transaktionale Workflow-Module eingeführt. Die im dritten Kapitel beschriebenen transaktionalen Eigenschaften werden in Kapitel 4 in Eigenschaftsnetze übersetzt, und die Einhaltung der geforderten transaktionalen Eigenschaften wird bewiesen. Im fünften Kapitel wird gezeigt, daß sich die Analyse der Bedienbarkeit auf transaktionale Workflow-Module anwenden läßt. Am Ende werden die Ergebnisse dieser Arbeit im Kapitel 6 zusammengefaßt und ein Ausblick auf Fragestellungen gegeben, die sich aus dieser Arbeit ergeben.

2 Grundlagen

In diesem Kapitel werden wir die Grundlagen dieser Arbeit erläutern. Dabei ist die Betrachtung des Anwendungsgebietes vorangestellt. Hier klären wir, was wir unter einem Geschäftsprozeß verstehen. In den darauf folgenden Ausführungen werden wir Petrinetze als Modellierungsmethode einführen. Danach zeigen wir, wie ein Geschäftsprozeß als Petrinetz in Form eines Workflow-Moduls modelliert wird. Zuletzt befassen wir uns in diesem Kapitel mit dem Begriff der Bedienbarkeit eines Workflow-Moduls. Workflow-Module sind Ausgangspunkt für deren Erweiterung um transaktionale Eigenschaften, und die Bedienbarkeit ist eine Eigenschaft eines Workflow-Moduls, die es auch nach einer Erweiterung eines Workflow-Moduls zu analysieren gilt.

2.1 Anwendungsgebiet

Ein Unternehmen verfolgt mit der Produktion einer Ware oder der Erstellung einer Dienstleistung einen Geschäftszweck. Ein Geschäftszweck ist meist, Gewinne durch Erbringen einer Leistung (Ware oder Dienstleistung) zu erwirtschaften. Für eine Leistung sind in der Regel eine Reihe von Arbeitsschritten notwendig, die zu einem Geschäftsprozeß zusammengefaßt werden. Um sich ein klares Bild von einem Geschäftsprozeß machen zu können, wird eine abstrakte *Ablaufbeschreibung* benötigt. Dafür wird eine geeignete *Modellierungsmethode* ausgewählt (z.B. Petrinetze, EPK) und ein Modell des Geschäftsprozesses erstellt. Ein solches Modell nennen wir *Geschäftsprozeßmodell*. Damit ein Unternehmen mit einer Leistung auch seinen Geschäftszweck erfüllt, müssen Geschäftsprozesse oft an veränderte Anforderungen (z.B. Kommunikation mit Partnern) angepaßt und optimiert werden. Hierfür können Eigenschaften eines Geschäftsprozesses durch Analyse nachgewiesen oder durch Simulation ermittelt werden. Im Ergebnis kann der Geschäftsprozeß angepaßt werden. Ein Unternehmen kann auf diese Weise seine betrieblichen Abläufe erfolgreich planen und Ressourcen effizient einsetzen.

Ein Geschäftsprozeß definiert, welche Aktivitäten in welcher Reihenfolge für die Erbringung einer Leistung ausgeführt werden müssen. Eine Aktivität ist der kleinste Baustein eines Geschäftsprozesses. Die Erbringung einer konkreten Leistung (z.B. die Produktion eines Fahrzeuges) ist ein *Geschäftsvorfall*. Ein Geschäftsvorfall entsteht meist durch einen Auftrag (z.B. Bestellung eines Fahrzeuges). Ein Auftrag stellt die notwendigen Informationen eines Geschäftsvorfalles zur Verfügung und initialisiert damit einen Geschäftsprozeß. Ein Geschäftsprozeß hat einen definierten Anfang und

ein definiertes Ende. Die Bestellung löst in unserem Fall den Geschäftsprozeß aus, und nach Abnahme des Fahrzeuges, durch den Auftraggeber, ist der Geschäftsprozeß beendet. Im Folgenden definieren wir die Begriffe Geschäftsprozeß und Workflow in Anlehnung an Definitionen in [Obe96].

Definition 2.1 (Geschäftsprozeß / Workflow). Ein *Geschäftsprozeß* ist eine Folge von Aktivitäten, die in einem logischen Zusammenhang stehen, inhaltlich abgeschlossen sind und unter Zuhilfenahme von Ressourcen und eingehenden Informationen durch Menschen und /oder Maschinen auf ein Unternehmensziel hin ausgeführt werden. Ein *Workflow* ist die informationstechnische Realisierung eines Geschäftsprozesses. *

Ein Geschäftsprozeß bildet die Grundlage eines Workflows, der wiederum durch eine Ablaufsteuerung gesteuert wird. Die Ablaufsteuerung ist Bestandteil eines *Workflow-Management-Systems*, das darüber hinaus auch die Planung, Modellierung, Kontrolle und Dokumentation eines Workflows ermöglicht.

Der herkömmliche Ansatz, einen Geschäftsprozeß als einen monolithischen Prozeß zu betrachten, wird zunehmend um den Ansatz eines *verteilten Geschäftsprozesses* ergänzt. Hierbei wird ein Geschäftsprozeß aus mehreren kleineren Geschäftsprozessen zusammengesetzt, die nicht zwingenderweise Geschäftsprozesse desselben Unternehmens sind. Die an einem solchen Geschäftsprozeß beteiligten Partner kommunizieren über Nachrichten miteinander. Für die Umsetzung dieses Ansatzes werden Geschäftsprozesse verstärkt als Web-Service implementiert.

Definition 2.2 (Web-Service). Ein Web-Service ist eine abgeschlossene Softwarekomponente, die mit einer wohldefinierten Schnittstelle zur Außenwelt versehen ist und über Nachrichtenaustausch mit der Außenwelt kommuniziert.

Ein Web-Service greift dabei auf Internettechnologien, wie z.B. HTTP, HTTPS, FTP, SMTP, TCP/IP usw., zurück. In dieser Arbeit wird von den technischen Details von Web-Services abstrahiert.

Wir werden um eine Schnittstelle erweiterte Geschäftsprozesse modellieren und analysieren. Diese Modelle können jederzeit in einen Web-Service übersetzt werden.

2.2 Modellierungsmethode

Grundlage aller weiteren Betrachtungen ist ein geeignetes Modell eines Geschäftsprozesses. Ein Modell ist geeignet, wenn es einen festgelegten Zweck erfüllt. Um den Zweck eines Geschäftsprozeßmodells zu erfüllen, werden Anforderungen an das Modell definiert. Solche Anforderungen können eine einfache und gut verständliche Darstellung, das Hervorheben oder Abstrahieren von Aspekten eines Geschäftsprozesses (Kontrollfluß, Daten, Ressourcen usw.) und die Verfügbarkeit von Analysemethoden zum Nachweis von Eigenschaften eines Geschäftsprozesses sein.

In der Praxis werden für die Modellierung von Geschäftsprozessen eine Reihe von Modellierungsmethoden eingesetzt. Ausgewählte Vertreter sind hier *ereignisgesteuerte Prozeßketten (EPK)* [Sch91], *Flow Definition Language* [Ley01], *UML* [OMG03] und *Petrinetze* [Aal96b]. Wir benutzen Petrinetze als Modellierungsmethode. Motivieren wir kurz diese Entscheidung. Petrinetze haben eine formale Semantik, damit ist ein Petrinetzmodell eines Geschäftsprozesses eine eindeutige Definition des Geschäftsprozesses. Ein Petrinetzmodell dient dadurch einem klaren Verständnis eines Prozesses. Wir konzentrieren uns in dieser Arbeit auf den Kontrollfluß, die externe Kommunikation und das Verhalten eines Geschäftsprozesses bei Zugriff auf Daten. Wir abstrahieren von Aspekten wie z.B. der Modellierung von Daten, Organisationsstrukturen und Implementationsdetails. Van der Aalst zeigt in [Aal96a],[Aal98], wie sich der Kontrollfluß eines Geschäftsprozesses mit Petrinetzen modellieren läßt. Es ist mit Petrinetzen möglich, alle notwendigen Kontrollflußstrukturen zu modellieren. Dabei besitzen Petrinetze eine einfache graphische Darstellung. Graphische Darstellungen haben oft das Problem, bei großen Systemen unübersichtlich zu werden. Dies ist bei Petrinetzen nicht anders. Hierfür existieren jedoch Konzepte, ein Modell hierarchisch aufbauen zu können und Petrinetze zu verfeinern [Peu01]. Dabei bleiben Eigenschaften über Hierarchiestufen hinweg erhalten. Sollen weitere Aspekte eines Geschäftsprozesses ausgedrückt werden, so ist dies mit Petrinetzen ebenfalls möglich [Aal98].

Petrinetze heben sich von den anderen oben genannten Modellierungsmethoden insbesondere durch ihr reichhaltiges Angebot an Analysemethoden ab, mit denen Eigenschaften eines Prozesses nachgewiesen werden können. Aufgrund ihrer formalen Semantik, lassen sich Eigenschaften in Petrinetzen berechnen. So lassen sich z.B. Invarianten und Deadlocks aus der Struktur eines Petrinetzes berechnen [Sta90]. Da bei Petrinetzen Zustand und Zustandsübergang klar definiert sind, lassen sich Methoden des Model-Checking auf Petrinetze gut anwenden. Damit können geforderte Eigenschaften eines Geschäftsprozesses, wie z.B. Erreichbarkeitsaussagen und Sicherheitseigenschaften, auch in größeren Prozessen nachgewiesen werden. Einige Eigenschaften, die ein Petrinetzmodell besitzen kann, sind im Umfeld der Geschäftsprozeßmodellierung geprägt worden. Exemplarisch seien hier Begriffe wie *soundness* [Aal98], *weak soundness* [Mar03] oder *relaxed soundness* [Deh03] genannt.

Neben der Analyse eines Petrinetzmodells kann eine Eigenschaft auch durch Simulation ermittelt werden. Dafür gibt es z.B. die Klasse der stochastischen Petrinetze mit deren Hilfe insbesondere Engpässe an bestimmten Stellen in einem Geschäftsprozeß durch Simulation ermittelt werden. Es gibt aber auch Klassen von Petrinetzen, bei denen eine Analyse schwer möglich ist, da hier viele Probleme nicht entscheidbar sind. Genannt seien hier z.B. *High-Level* Petrinetze. In dieser Arbeit spielen solche Netze allerdings keine Rolle.

Abschließend stellen wir demzufolge fest: Mit Petrinetzen läßt sich auf einem angemessenen Abstraktionsniveau ein Geschäftsprozeßmodell modellieren. Der in dieser Arbeit im Vordergrund stehende Kontrollflußaspekt ist sehr gut darstellbar und

gut analysierbar. Damit erfüllen Petrinetzmodelle den Zweck der Modellierung eines Geschäftsprozesses.

2.3 Petrinetze / Workflow-Modul

Wir werden in den weiteren Abschnitten dieses Kapitels die grundlegenden Definitionen einführen, die für die weitere Arbeit notwendig sind. Allen voran steht die Einführung der Petrinetze. Für eine grundlegende Einführung in die Theorie der Petrinetze verweisen wir auf [Rei85].

Definition 2.3 (Petrinetz). Ein *Petrinetz* $N = (P, T, F, m_0)$ besteht aus zwei disjunkten endlichen Mengen P , T und einer Relation $F \subseteq (P \times T) \cup (T \times P)$. Die Elemente von P , T und F heißen *Plätze*, *Transitionen* bzw. *Kanten* des Netzes. Eine Markierung ist eine Abbildung $m: P \rightarrow \mathbb{N} \cup \{0\}$ und m_0 die Anfangsmarkierung.

Für jedes Element $x \in (P \cup T)$ nennen wir die Menge $\bullet x = \{y \in (P \cup T) \mid [y, x] \in F\}$ den *Vorbereich* und die Menge $x^\bullet = \{y \in (P \cup T) \mid [x, y] \in F\}$ den *Nachbereich* des Elements.

Damit haben wir die Struktur eines Petrinetzes beschrieben. Diese läßt sich wie folgt graphisch darstellen: Wir stellen einen Platz durch einen Kreis, eine Transition durch ein Rechteck und eine Kante durch einen Pfeil dar. Dabei existieren Kanten nur zwischen Plätzen und Transitionen und umgekehrt. Auf einem Platz können *Marken* liegen. Wenn auf einem Platz mindestens eine Marke liegt, dann ist dieser Platz *markiert*. Wir werden in dieser Arbeit Marken nicht voneinander unterscheiden und stellen sie als schwarze Punkte dar. Eine Markierung eines Netzes N heißt auch *Zustand*.

Ein Petrinetz kann seinen Zustand ändern. Dafür muß sich die Markierung des Netzes ändern, indem mindestens eine Transition die Markierung eines oder mehrerer Plätze ändert.

Definition 2.4 (Verhalten von Petrinetzen). Eine Transition t ist *aktiviert* in der Markierung m , wenn für alle p mit $p \in \bullet t$ gilt $m(p) > 0$. Eine Transition t kann in der Markierung m in die Markierung m' *schalten*, wenn t in m aktiviert ist und für alle p gilt, $m'(p) = m(p) + W([t, p]) - W([p, t])$ mit $W([x, y]) = 1$ wenn $[x, y] \in F$, und $W([x, y]) = 0$ sonst. $R_N(m)$ bezeichnet die Menge von Markierungen, die von m durch schalten einer endlichen Menge von Transitionen erreicht werden kann. *

Wir benutzen in dieser Arbeit Petrinetze mit einem *Kantengewicht* von 1. Das heißt, wenn eine Kante im Netz existiert, dann “fließt“ über diese Kante maximal eine Marke, wenn die zur Kante inzidente Transition schaltet. Wenn eine Transition schaltet, dann geht eine Markierung m in eine Markierung m' über. Wir nennen diesen Vorgang einen *Schritt* und schreiben: $m \xrightarrow{t} m'$. Eine Folge von Schritten geschrieben $m \xrightarrow{*} m'$ heißt *Schaltsequenz*.

Wir werden in dieser Arbeit eine Menge von Petrinetzen zu einem Petrinetz vereinigen und dies *parallele Komposition* nennen.

Definition 2.5 (parallele Komposition). Die *parallele* Komposition zweier Petrinetze $N = (P, T, F)$ und $N' = (P', T', F')$ ist das Netz $N'' = (P \cup P', T \cup T', F \cup F')$. *

Die nun folgende Definition basiert auf einer eingeschränkten Klasse von Petrinetzen, den *Workflow-Netzen* [Aal98]. Diese wurden in [Mar03] um eine Schnittstelle für ein- und ausgehende Kommunikation ergänzt.

Definition 2.6 (Workflow-Modul). Ein Petrinetz M ist ein *Workflow-Modul*, wenn

- i. P enthält einen ausgezeichneten *Anfangsplatz* α und einen ausgezeichneten *Endplatz* ω ;
- ii. P ist die disjunkte Vereinigung von Mengen P_M (*interne Plätze*), P_I (*Input-Plätze*), und P_O (*Output-Plätze*);
- iii. $F \cap (P_O \times T) = \emptyset$, $F \cap (T \times P_I) = \emptyset$;
- iv. $m_{0_M}(\alpha) = 1$, $m_{0_M}(p) = 0$ für alle $p \neq \alpha$;
- v. F enthält keine Zyklen, d.h. die transitive Hülle von F ist irreflexiv.

*

Wie wir bereits erwähnt haben, besitzt ein Geschäftsprozeß einen definierten Anfang und ein definiertes Ende. Dies ist in einem Workflow-Modul durch die Plätze α und ω repräsentiert. Für die Situation bevor ein Geschäftsvorfall eintritt, steht der Anfangszustand $m_{0_M}(\alpha)$. Ist ein Geschäftsprozeß abgearbeitet, dann entspricht dies im Workflow-Modul der Markierung $m_{f_M}(\omega) = 1$ und $m_{f_M}(p) = 0$ für alle $p \neq \omega$. Wir nennen diese Markierung die *Endmarkierung des Moduls* M . Das interne Verhalten, also der eigentliche Geschäftsprozeß, wird durch die internen Netzelemente P_M , T und $f \in (P_M \times T) \cup (T \times P_M)$ modelliert.

Ein Input-Platz bzw. ein Output-Platz ist ein *Nachrichtenkanal*. Eine Marke auf einem Nachrichtenkanal entspricht einer *Nachricht*. Nachrichtenkanäle sind asynchron, d.h. die Reihenfolge, in der ein Modul Nachrichten verarbeitet, ist unabhängig von der Reihenfolge, in der die Nachrichten an ein Modul versandt wurden. Gleiches gilt auch umgekehrt, d.h. auch die Nachrichten, die ein Modul verschickt, werden unabhängig von der Reihenfolge des Sendens vom Empfänger verarbeitet. Wir werden in dieser Arbeit ausschließlich azyklische Workflow-Module betrachten.

Beispiel 2.1 (Workflow-Modul). Für ein Workflow-Modul vereinbaren wir folgende Notation: Einen internen Platz $p \in P_M$ beschriften wir mit einem kleinen griechischen Buchstaben, die Transitionen eines Moduls beschriften wir mit einem kleinen t , gefolgt von einer fortlaufenden Zahl. Input- bzw. Output-Plätze erhalten einen kleinen Buchstaben als Namen.

Die Abbildung 2.1 zeigt zwei Workflow-Module. Das Modul M1 kann eine Nachricht a empfangen. In der Markierung α hat M1 die nichtdeterministische Auswahl, $t1$ oder $t2$ zu schalten. Wenn $t1$ schaltet, dann ist β markiert, und M1 kann mit der Nachricht b antworten, sonst ist δ markiert und M1 antwortet dann mit der Nachricht c . Im Modul M2 sind lediglich die Richtungen der Kommunikation umgekehrt. M2 kann eine Nachricht a verschicken und die Nachrichten b bzw. c empfangen. Wir werden später sehen, daß M1 in einem noch zu definierenden Sinne vernünftig benutzbar ist und M2 nicht. *

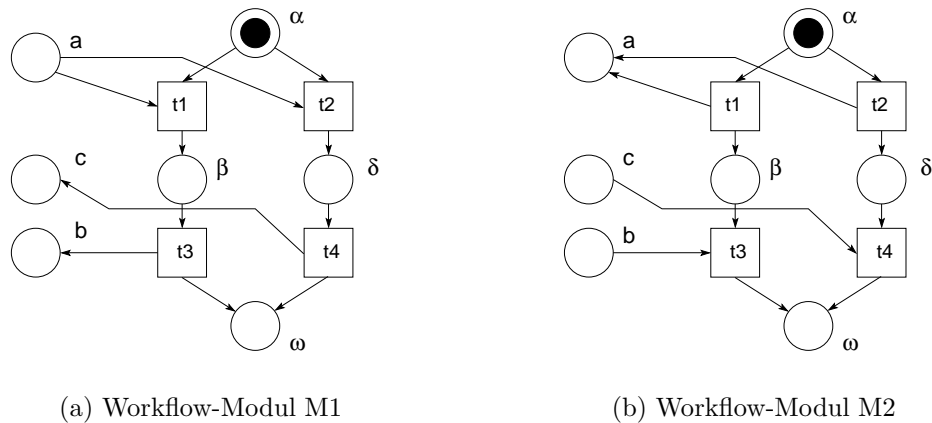


Abbildung 2.1: Zwei Workflow-Module

2.4 Umgebung

Die folgenden Definitionen gehen auf [Sch04] und [MS05] zurück. Ein gegebener Geschäftsprozeß in Form eines Workflow-Moduls soll natürlich auch benutzt werden. Dabei interagiert ein Benutzer mit einem Workflow-Modul mit dem Ziel, einen festgelegten Endzustand zu erreichen. Ein solcher Endzustand bedeutet für ein Modul das Erreichen seiner Endmarkierung. Für einen Benutzer heißt dies, mit dem Modul so zu kommunizieren, daß ein Modul immer in seine Endmarkierung gelangt. Es gilt herauszufinden, ob es ein System aus Benutzer und Workflow-Modul gibt, so daß die Endmarkierung des Moduls immer erreicht werden kann. Um diese Frage

beantworten zu können, muß demnach das System aus einem Benutzer und einem Workflow-Modul untersucht werden.

Klären wir also jetzt die Frage, was wir unter einem Benutzer eines Workflow-Moduls verstehen. Wir werden in dieser Arbeit für ein Workflow-Modul immer genau einen Benutzer annehmen. Informal können wir einen Benutzer eines Moduls als eine *Umgebung* des Moduls auffassen. Aus der Sicht eines Moduls erhält das Modul Nachrichten aus seiner Umgebung und versendet Nachrichten an seine Umgebung. Wir nehmen dabei an, daß eine Umgebung auf die Input- und Output-Plätze eines Moduls zugreifen kann.

Nähern wir uns nun einer formalen Definition des Begriffes der Umgebung. Die Interaktion von einer Umgebung mit einem Workflow-Modul entspricht einer Abfolge von Sende- und Empfangsereignissen, also dem Austausch von Nachrichten. Nehmen wir an, eine Umgebung ist bereit eine Nachricht zu empfangen, dann entspricht dies einem Zustand. Wenn die Umgebung in diesem Zustand eine Nachricht empfängt, dann ist sie in einem neuen Zustand. Eine Umgebung kann demnach als eine Menge von Zuständen und seine Zustandsübergänge aufgefaßt werden. Ein Zustandsübergang ist dabei entweder das Senden einer Nachricht an ein Modul oder das Empfangen einer Nachricht von einem Modul. Damit läßt sich eine Umgebung als Automat gut beschreiben.

Ein Workflow-Modul legt fest, welche Nachrichten es empfangen oder senden kann. Wir nehmen also an, daß eine Umgebung nur Nachrichten verarbeiten kann, die durch die Input-Plätze bzw. Output-Plätze eines Workflow-Moduls gegeben sind. Ein Alphabet A eines Automaten für eine Umgebung ist gegeben durch die Nachrichten, die ein Modul verarbeiten kann. Per Definition besitzen Workflow-Module endliches Verhalten, d.h. es gibt nur endlich viele Zustände, in denen sich ein Workflow-Modul befinden kann. Ein Modul kann somit nur endliche Folgen von Nachrichten erzeugen. Eine Umgebung eines Moduls braucht infolgedessen nur endliche Wörter verarbeiten. Aus diesem Grund kann eine Umgebung als endlicher Baum dargestellt werden. Für ein Alphabet A sei A^* die Menge der endlichen Wörter über A . Wenn eine Umgebung nur endliche Wörter verarbeiten muß, dann existiert auch ein längstes Wort. Dieses Wort habe die Länge l_m . Wir werden die Tiefe eines Baumes durch die Zahl l_m begrenzen. Das endliche Verhalten eines Moduls ermöglicht es, l_m zu berechnen und einen Algorithmus dafür anzugeben. Wir werden l_m als gegeben annehmen. Eine Abfolge von Nachrichten ist ein Wort über A . Eine einzelne Nachricht entspricht einem Buchstaben $a \in A$. Wollen wir einer Abfolge von Nachrichten ein Sende- oder Empfangsereignis hinzufügen, so ist dies eine Konkatenation von Wörtern. Deshalb bezeichne \underline{a} die Multimenge, die a genau einmal enthält und sonst kein Element. Die Multimenge \underline{a} ist ein Wort der Länge 1.

Definition 2.7 (Umgebung). Sei M ein Workflow-Modul und $A \subseteq (P_I \cup P_O)$. Eine mit A verbundene *Umgebung* ist ein Automat mit dem Alphabet A , einer Menge von Zuständen $Q \subseteq \{w \mid w \in A^*, \text{länge}(w) \leq l_M\}$, so daß Q enthält mit q alle Präfixe

von q , einer Übergangsfunktion δ mit $\delta(w, x) = \{wa\}$ wenn $wa \in Q$ und es ein a gibt mit $x = \underline{a}$, und $\delta(w, x) = \emptyset$ sonst, und λ (das leere Wort) als den Anfangszustand. *

Nach Definition 2.7 ist jeder Zustand einer Umgebung ein Wort, und für jedes Präfix q und ein \underline{a} existiert höchstens ein Folgezustand.

Beispiel 2.2 (Umgebungen). In der Abbildung 2.2 sind drei Umgebungen abgebildet. Die Module U1 und U2 sind jeweils Umgebungen der Workflow-Module M1 bzw. M2. Die Umgebung U3 ist keine Umgebung von M1 oder M2. U3 besitzt in seinem Alphabet den Buchstaben k . Damit ist A_{U3} keine Teilmenge der Input- bzw. Output-Plätze der Module M1 bzw. M2. Der Übersichtlichkeit halber sind die Zustände durchnummeriert. Jeder Zustandsübergang beschreibt eine Interaktion mit einem Modul, und die annotierte Multimenge steht für eine Nachricht an ein Modul bzw. von einem Modul. Wenn von einer Umgebung eine Nachricht an ein Modul gesendet wird, dann wird die Nachricht in einem Nachrichtenkanal abgelegt. Umgekehrt kann eine Umgebung eine Nachricht empfangen, wenn eine Nachricht in einem Nachrichtenkanal bereitliegt. *

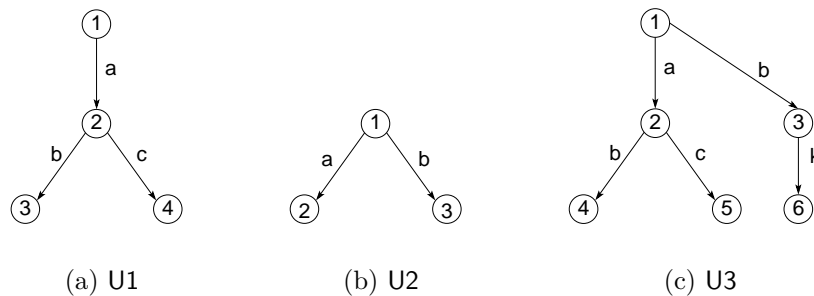


Abbildung 2.2: Drei Umgebungen

Ein Workflow-Modul bietet eine Menge von Interaktionsmöglichkeiten für eine Umgebung. Wir wollen bei der Analyse eines Moduls *alle* Möglichkeiten der Interaktion eines Systems aus einer Umgebung und einem Modul untersuchen. Deshalb werden wir im Folgenden von einer Umgebung fordern, daß sie über alle Nachrichtenkanäle mit einem Modul interagieren kann.

Definition 2.8 (zulässige Umgebung). Eine Umgebung U ist *zulässig* für ein Workflow-Modul M , wenn das Alphabet von U gleich dem Alphabet des Moduls ist. *

Damit ist U1 offensichtlich sowohl für M1 als auch für M2 eine zulässige Umgebung. Die Umgebung U2 ist nicht zulässig für M1 oder M2.

Wir haben jetzt geklärt, was wir unter einem Workflow-Modul und einer Umgebung verstehen. Für die Analyse müssen wir noch klären, was ein komponiertes System aus einem Workflow-Modul und einer Umgebung ist.

Definition 2.9 (Komponiertes System). Sei M ein Workflow-Modul und sei U eine Umgebung für M . Das *komponierte System* ist ein Transitionssystem mit $R_N(m_0) \times Q$ als die Menge von Zuständen und Kanten

- i. von $[m, q]$ nach $[m', q]$ wenn es eine Transition t in M gibt mit $m \xrightarrow{t} m'$;
- ii. von $[m, q]$ nach $[m', q']$ wenn es eine Multimenge B gibt, so daß in Q $q' \in \delta(q, B)$, für alle $p \in P_O$, $m(p) \geq B(p)$ und $m'(p) = m(p) - B(p)$, für alle $p \in P_I$ $m'(p) = m(p) + B(p)$, und für alle $p \in P_M$ gilt $m'(p) = m(p)$.

Der Initialzustand ist $[m_0, \lambda]$

*

Der Zustand eines komponierten Systems aus einem Modul und einer Umgebung ist ein Tupel, bestehend aus einer aus dem Anfangszustand des Moduls erreichbaren Markierung und einem Zustand der Umgebung. Ein Zustandsübergang im komponierten System ist entweder das Schalten einer Transition des Moduls oder das Empfangen oder Senden einer Nachricht der Umgebung.

Beispiel 2.3 (komponiertes System). Betrachten wir nun das komponierte System in Abbildung 2.3(a). Es ist die Komposition aus dem Modul **M1** und der Umgebung **U1**. Im Anfangszustand $[\alpha, 1]$ kann nur die Umgebung ein **a** senden. Die Umgebung geht dabei in den Zustand 2 über, und im Modul wird auf dem Input-Platz **a** eine Marke abgelegt. Das komponierte System befindet sich jetzt im Zustand $[\alpha, a, 2]$. Im Modul sind jetzt die Transitionen **t1** und **t2** aktiviert. Es kann also entweder **t1** oder **t2** schalten. Damit gibt es zwei mögliche Zustandsübergänge im komponierten System. Wir können sowohl in den Zustand $[\beta, 2]$, als auch in den Zustand $[\delta, 2]$ gelangen. Es schalte jetzt **t1**, dann ist im Zustand $[\beta, 2]$ die Transition **t3** im Modul aktiviert. Wenn **t3** geschaltet hat, dann sind ω und der Output-Platz **b** markiert. Jetzt kann die Umgebung die Nachricht empfangen, d.h. die Marke von **b** entfernen und selber in den Zustand 3 übergehen. Wir haben jetzt den Zustand $[\omega, 3]$ erreicht, in dem sich nichts mehr ändern kann. Analoges gilt, wenn im Zustand $[\alpha, a, 2]$ die Transition **t4** schaltet.

*

2.5 Bedienbarkeit

Wir wollen herausfinden, ob es eine Umgebung zu einem Modul gibt, so daß das Modul immer seine Endmarkierung erreichen kann. Dafür muß erst einmal klar sein, wie wir Umgebungen, mit denen ein Modul immer in seine Endmarkierung gelangen kann, von den Umgebungen trennen können, mit denen das Modul nicht dazu in der Lage

In einer Umgebung eines Workflow-Moduls repräsentiert jeder Zustand die gesamte Abfolge von Interaktionen mit dem Modul. Aus der Sicht der Umgebung ist es nicht möglich, aus der Interaktion mit dem Modul das interne Verhalten des Moduls herzuleiten. Wir haben nun folgende Situation. Von der Umgebung interessieren uns genau die Zustände, die zu einer Strategie gehören, wir können aber ohne Kenntnis des Verhaltens des Moduls die Zustände der Umgebung nicht klassifizieren. Das komponierte System jedoch beinhaltet das interne Verhalten des Moduls. Aus einem komponierten System sind für jeden Zustand der Umgebung alle möglichen Markierungen des Moduls ableitbar. Dieses Wissen über das interne Verhalten eines Moduls wird mit einer entsprechenden Umgebung wie folgt zusammengeführt:

Definition 2.12 (Wissen der Umgebung). Sei U eine mit A verbundene Umgebung und q ein Zustand von U . Sei S eine Menge von Zuständen des komponierten Systems aus U und M . Dann heißt $K(q) = \{m \mid \exists q : [m, q] \in S\}$ das Wissen der Umgebung U im Zustand q . *

Zu jedem Zustand einer Umgebung annotieren wir alle möglichen Markierungen des Moduls. Wie in Abbildung 2.4 zu sehen ist, werden die Markierungen an jeden Knoten angetragen. Jeder Zustand einer Umgebung gehört entweder einer Strategie oder keiner Strategie an. Um die Zugehörigkeit klären zu können, müssen die Zustände einer Umgebung klassifiziert werden. Dafür wird das Wissen in jedem Zustand ausgewertet. Das Wissen einer Umgebung sind Markierungen des Moduls, die ausgewertet werden müssen. Das bedeutet, wir klassifizieren Markierungen eines Moduls. Die Markierungen eines Workflow-Moduls lassen sich wie folgt klassifizieren: Es gibt Markierungen, in denen ein Schritt im Modul möglich ist, Markierungen in denen ein Schritt nur möglich ist, wenn die Umgebung mit dem Modul interagiert und Markierungen, in denen unter keinen Umständen ein Schritt stattfinden kann. Von besonderen Interesse sind natürlich die Markierungen, in denen die Umgebung selbst Einfluß auf das Schalten einer Transition hat oder in denen das Modul selbst verklemmt.

Definition 2.13 (Deadlocks). Sei M ein Workflow-Modul. Ein Deadlock von M ist eine Markierung, in der keine Transition aktiviert ist. Ein Deadlock m ist *intern*, wenn $m \neq m_f$, für jede Transition existiert ein $p \in P_M$ mit $[p, t] \in F$ und $m(p) = 0$ und alle Outputplätze unmarkiert sind. Ein Deadlock m ist *extern*, wenn er weder intern noch $m = m_f$ ist. *

Beispiel 2.5 (Wissen der Umgebung). Die Abbildung 2.4(a) repräsentiert das Wissen der Umgebung U1 für das Modul M1. Jedem Zustand sind die möglichen Markierungen des Moduls zugeordnet. Aus der Abbildung 2.3(a) ist leicht ersichtlich, wie das Wissen für einen Zustand abgeleitet wird. Wir wissen bereits, daß U1 eine Strategie für das Modul M1 ist. Für Knoten, die Bestandteil einer Strategie sind, vereinbaren wir als Notation einen schwarz gefüllten Kreis. In der Abbildung 2.4(b)

ist das Wissen von $U1$ für das Modul $M2$ dargestellt. Kein Knoten aus $U1$ ist in diesem Fall Teil einer Strategie. Wir stellen Knoten, die zu keiner Strategie gehören, durch einen schwarz umrandeten Kreis dar. In beiden Abbildungen sind die Deadlocks durch (e) bzw. (i) kenntlich gemacht. *

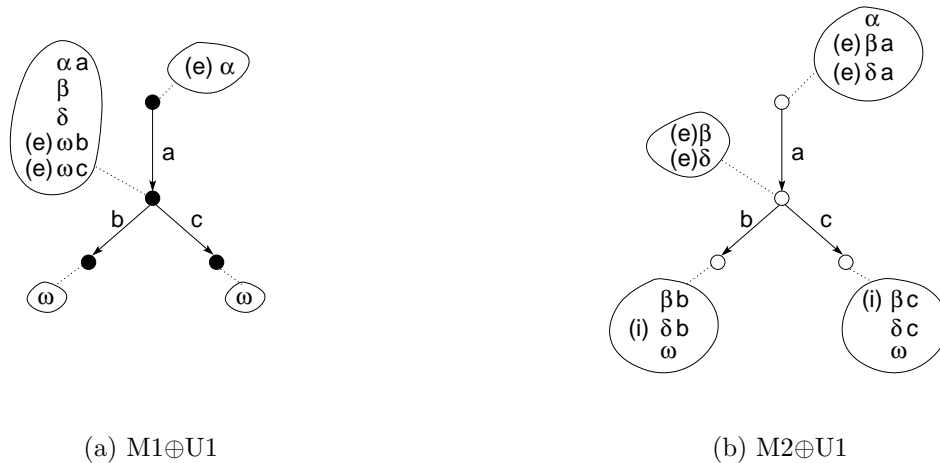


Abbildung 2.4: Wissen der Umgebung $U1$

Es ist jetzt möglich, jeden Zustand einer Umgebung eines Workflow-Moduls zu klassifizieren. Die Korrektheit der Klassifikation wird in [Sch04] über das folgende Theorem und dessen Beweis gezeigt. Das Theorem gibt das Verhältnis zwischen dem Wissen einer Umgebung und einer Strategie wieder.

Theorem 2.1 (bedienen). *Eine mit $(P_I \cup P_O)$ verbundene Umgebung U ist eine Strategie für das Modul M , gdw. U besitzt eine nichtleere Menge von Zuständen und für alle $q \in Q$ gilt:*

- i. $K(q)$ enthält keine internen Deadlocks;
- ii. für jeden externen Deadlock $m \in K(q)$ besitzt das komponierte System aus M und U eine von $[m, q]$ ausgehende Transition. *

Bislang kann nachgewiesen werden, ob eine mit $P_I \cup P_O$ verbundene Umgebung das Modul M bedient. Es ist demnach noch die Frage offen, ob es eine Umgebung U für ein gegebenes Modul M gibt, so daß gilt: U bedient M . Hierfür ist in [Sch04] eine Algorithmus für die Konstruktion einer Umgebung angegeben, aus der sich *alle* möglichen Strategien berechnen lassen. Für die Berechnung der Strategien ist ebenfalls ein Algorithmus aufgeführt.

Damit haben wir die notwendigen Begriffe für die weitere Arbeit eingeführt.

3 Transaktionale Eigenschaften

Wir werden in dieser Arbeit transaktionale Eigenschaften für Workflow-Module modellieren und analysieren. Die dafür notwendigen Begriffe werden im folgenden Kapitel eingeführt. Wir werden zunächst klären, was wir unter einem *Schedule* verstehen und den Begriff des Schedules formal definieren. Nachfolgend definieren wir eine Reihe von Eigenschaften bezüglich eines Schedules, die es uns ermöglichen, einen für die weitere Analyse *geeigneten* Schedule von einem hierfür *ungeeigneten* Schedule unterscheiden zu können.

3.1 Hintergrund

Die hier eingeführten Begriffe sind in Anlehnung an das klassische Transaktionskonzept [VGH93] definiert worden. Dieses Transaktionskonzept gilt im allgemeinen als eines der fundamentalen Merkmale eines Datenbanksystems [GR94]. Es soll mögliche Anomalien bei der gleichzeitigen Benutzung einer Datenbank durch mehrere Benutzer ausschließen. Zu diesem Zweck werden Benutzerzugriffe zu *Transaktionen* zusammengefaßt und ihre Abarbeitung genügt vier Eigenschaften, die als *ACID*-Eigenschaften bekannt sind [VGH93]:

- **Atomarität:** (Atomicity) Es werden entweder alle Benutzerzugriffe einer Transaktion ausgeführt oder kein Benutzerzugriff.
- **Konsistenz:** (Consistency) Für Datenbanken existieren Integritätsbedingungen, die definieren, wann eine Datenbank in einem konsistenten Zustand ist. Wenn eine Transaktion auf einem konsistenten Zustand gestartet wird, dann befindet sich die Datenbank auch nachdem die Transaktion ausgeführt wurde in einem konsistenten Zustand.
- **Isolation:** Eine Transaktion läuft isoliert zu anderen Transaktionen ab. Das bedeutet, wenn eine Transaktion parallel zu anderen Transaktionen ausgeführt wird, dann ist das Ergebnis gleich dem Ergebnis einer alleinigen Ausführung der Transaktion. Es werden einer Transaktion nur Daten eines konsistenten Zustandes zur Verarbeitung übergeben.

- **Persistenz:** (Durability) Wenn eine Transaktion vollständig abgearbeitet wurde, dann sind alle Effekte dieser Transaktion Bestandteil eines konsistenten Zustands und überdauern alle möglichen Fehler des Datenbanksystems.

Wir werden in dieser Arbeit von Datenbanksystemen und ihrer Synchronisationssteuerung und Fehlerbehandlung abstrahieren und lediglich fordern, wann immer unser Modell in einem als gut definierten Zustand ist, dann werden die ACID-Eigenschaften nicht verletzt.

3.2 Transaktionsbegriff/Schedule

Ein geeignetes Modell, von Zugriffen auf eine Datenbank zu abstrahieren, besteht darin, Zugriffe als Lese- und Schreibaktionen auf Variablen zu beschreiben:

Definition 3.1 (Aktion). Sei Var eine Menge von Variablen. Eine Aktion auf einer Variablen $x \in Var$ ist entweder *Lesen* oder *Schreiben* von x und wird notiert durch $r(x)$, bzw. $w(x)$ *

Im Gegensatz zum klassischen Transaktionsmodell, das auch unter der Bezeichnung *Read-Write-Modell* geläufig ist, werden wir keine Folgen von Aktionen zu einer *Transaktion* zusammenfassen, sondern eine Menge von Aktionen.

Definition 3.2 (Transaktion). Eine Transaktion über $V \subseteq Var$ ist eine Menge von Aktionen auf Variablen $x \in V$ und wird bezeichnet mit ta . *

Wir legen für Transaktionen keine weiteren Anforderungen fest und trennen damit Eigenschaften, wie zum Beispiel das Lesen einer Variablen vor dem Schreiben dergleichen, vom Begriff der Transaktion und werden diese Eigenschaften später als Eigenschaften eines *Schedules* definieren. Wie wir später sehen werden, ermöglicht uns diese Herangehensweise, Transaktionen ohne große Schwierigkeiten in unser Modell zu integrieren.

Definition 3.3 (Schedule). Ein Schedule S einer Menge von Transaktionen TA ist eine Sequenz $S = a_1 \dots a_m$ von Aktionen über dem Alphabet $\bigcup TA$. *

Beispiel 3.1. Sei die Menge von Transaktionen TA wie folgt gegeben

$$\begin{aligned} ta_1 &= \{r_1(x), r_1(y), w_1(x)\}, \\ ta_2 &= \{r_2(z), w_2(x)\}, \\ ta_3 &= \{w_3(u)\}. \end{aligned}$$

Dann sind

$$\begin{aligned}S_1 &= w_2(x)r_1(x)r_2(z)w_1(x)r_1(y), \\S_2 &= r_1(x)r_2(z)r_1(x)\end{aligned}$$

zwei Schedules über TA . In einem Schedule können Schreib- und Leseaktionen in beliebiger Reihenfolge vorkommen. Des Weiteren ist es möglich, daß einzelne Aktionen beliebig häufig in einem Schedule auftreten. So tritt in S_2 die Aktion $r_1(x)$ zweimal auf. In S_1 und S_2 ist $w_3(u)$ nicht enthalten, und dies wird für einen Schedule über TA nicht gefordert. *

Definition 3.4 (Vorkommen einer Transaktion). Sei $S = a_1 \dots a_m$ ein Schedule über TA . Eine Transaktion $ta \in TA$ kommt in S vor, gdw. es ein i mit $a_i \in ta$ gibt. *

Betrachten wir nochmal das Beispiel 3.1, dann kommen sowohl ta_1 als auch ta_2 in den beiden Schedules S_1 und S_2 und ta_3 in keinem vor.

3.3 Schöne Schedules

Es gibt in der Menge von Schedules eine Teilmenge von Schedules mit Eigenschaften, die wir nutzen werden und eben solche, die diese Eigenschaften nicht besitzen. Wie wir diese Mengen auseinanderhalten können, werden wir jetzt definieren.

Für die weiteren Ausführungen werden wir einen Schedule als *schön* bezüglich einer Menge von Eigenschaften bezeichnen, wenn er jede dieser Eigenschaften hat.

3.3.1 Eigenschaften von Schedules in Bezug auf einzelne Transaktionen

Eigenschaft 1: vollständiges Vorkommen

Stellen wir uns einen Schedule $S = a_1 \dots a_m$ über TA vor, in dem eine Transaktion ta vorkommt, es aber eine Aktion $a \in ta$ gibt, die nicht in diesem Schedule enthalten ist, d.h. $a \notin \{a_1, \dots, a_m\}$. Wir wissen, eine Transaktion soll entweder vollständig oder gar nicht abgearbeitet werden. Deshalb wird die Transaktion ta nicht bearbeitet, selbst wenn einzelne Aktionen $b \in ta_i$ im Schedule stehen. Es stellt sich nun die Frage, ob wir einen Schedule, in dem eine solche Transaktion vorkommt, als schön betrachten wollen. Für die Integrität einer Datenbank stellt diese Situation kein Problem dar. Für Aktionen, die nur deshalb nicht ausgeführt werden, weil eine andere zur Transaktion gehörige Aktion nicht in einem Schedule steht, bedeutet das, daß Aktivitäten im Geschäftsprozeß nicht bearbeitet werden können, und das ist unangemessen. Wenn jedoch eine ganze Transaktion aus TA nicht in einem Schedule über TA vorkommt, dann werden wir dies als zulässig erachten. Ein mögliches Auftreten dieses Falles

ist die alternative Ausführung zweier Transaktionen als Folge einer Entscheidung in einem Geschäftsprozeß.

Definition 3.5 (vollständiges Vorkommen). Sei $S = a_1 \dots a_m$ ein Schedule über TA . S hat die Eigenschaft E_1 , gdw. gilt: Wenn $ta \in TA$ in S vorkommt, dann $ta \subseteq \{a_1, \dots, a_m\}$. *

Ein Schedule über TA erfüllt die Eigenschaft 1, wenn entweder alle Aktionen einer Transaktion in dem Schedule enthalten sind oder keine Aktion einer Transaktion in dem Schedule zu finden ist. In unserem Beispiel 3.1 kommt die Transaktion ta_2 im Schedule S_1 vollständig vor. Aus der Transaktion ta_1 fehlt die Aktion $w_1(y)$ im Schedule S_1 , so daß ta_1 in S_1 nicht vollständig vorkommt. Der Schedule S_1 besitzt die Eigenschaft 1 nicht. Im Schedule S_2 kommt keine der beiden vorkommenden Transaktionen vollständig vor, so daß auch dieser Schedule die Eigenschaft 1 nicht besitzt. Ein in diesem Sinne schöner Schedule über $\{ta_1, ta_2\}$ ist der Schedule

$$S_3 = w_2(x)r_1(x)r_2(z)w_1(x)r_1(y)r_1(x)r_2(z).$$

Eigenschaft 2: Lesen vor Schreiben

Nach der *ACID*-Eigenschaft Isolation werden an eine Transaktion nur Daten eines konsistenten Zustandes übergeben. Das bedeutet, eine Transaktion liest lediglich Variablenwerte, die existierten, bevor die Transaktion selbst gestartet ist. Im Schedule S_3 wird die Variable x durch die Aktion $w_1(x)$ geschrieben. Damit steht ein $r_1(x)$ vor der Aktion $w_1(x)$ und ein $r_1(x)$ nach $w_1(x)$. Beide Leseaktionen dürfen nur den gleichen Wert für x lesen, da die eigene Transaktion (ta_1) noch nicht beendet ist. Allerdings wird x geschrieben, bevor die zweite Leseaktion x auslesen will. Es gilt diese Situation zu vermeiden.

Definition 3.6 (Lesen vor Schreiben). Sei $S = a_1 \dots a_m$ ein Schedule über TA . In S gilt die Eigenschaft *Lesen vor Schreiben*, gdw. für alle ta gilt: Wenn $a_i, a_j \in ta$ dann gilt für alle $x \in V$, wenn $a_i = r(x)$ und $a_j = w(x)$, dann $i < j$. *

Die Schedules S_1 und S_2 haben damit die Eigenschaft 2. Der Schedule S_3 besitzt sie nicht, da die Aktion $w_1(x)$ vor einer Aktion $r_1(x)$ steht.

Eigenschaft 3: Einmaligkeit

Per Definition kann ein Schedule unendlich lang sein, wenn z.B. mindestens eine Aktion unendlich oft im Schedule auftritt. Was verstehen wir jetzt unter Atomarität in Bezug auf die Transaktion, dessen Aktion unendlich oft in einem Schedule zu finden ist? Die Beantwortung dieser Frage ist nicht Bestandteil dieser Arbeit, weshalb wir uns darauf beschränken werden, endliche Schedules zu fordern. Des Weiteren wollen

wir, daß in einem schönen Schedule eine Aktion höchstens einmal auftritt. Diese Einschränkung hat sich in der Praxis bewährt und wird in dieser Arbeit vorausgesetzt. Eine Erweiterung auf Schedules, in denen Aktionen endlich oft in einem Schedule auftreten können, kann und muß in weiterführenden Arbeiten untersucht werden.

Definition 3.7 (Einmaligkeit). Sei $S = a_1 \dots a_m$ ein Schedule über TA . S hat die Eigenschaft E_3 , gdw. $\text{card}(\{a_1, \dots, a_m\}) = m$ *

Wir werden in dieser Arbeit Schedules als Grundlage verwenden, die jeweils alle drei dieser Eigenschaften erfüllen. Wir werden einen Schedule S *korrekt* nennen, wenn S die Eigenschaften 1-3 erfüllt.

Der Schedule S_3 besitzt mit den Aktionen $r_1(x)$ und $r_2(z)$ zwei Aktionen, die jeweils zwei mal in S_3 stehen und damit hat S_3 die Eigenschaft 3 nicht. S_1 und S_2 besitzen die Eigenschaft 3. Keiner der Schedules $S_1 - S_3$ ist damit ein korrekter Schedule. Über der Menge $\{ta_1, ta_2\}$ ist der Schedule

$$S_4 = w_2(x)r_1(x)w_1(x)r_1(y)r_2(z)$$

ein korrekter Schedule.

3.3.2 Eigenschaften von Schedules bezüglich mehrerer Transaktionen

Die Eigenschaften 1-3 klären die Frage, wie eine Transaktion in einem Schedule repräsentiert sein muß, damit der Schedule für die späteren Betrachtungen geeignet ist. Jetzt werden wir Bedingungen beschreiben, die zwischen den Transaktionen in einem Schedule gelten müssen.

Eigenschaft 4: Konfliktserialisierbarkeit

Konsistenz wird im klassischen Transaktionskonzept dadurch erreicht, daß verlangt wird, daß Transaktionen serialisierbar sind. Konsistenz wird anschaulich wie folgt erhalten: Unter der Voraussetzung, daß jede Transaktion für sich die Konsistenz von Daten erhält, bleibt diese auch bei Hintereinanderausführung aller Transaktionen (serieller Schedule) erhalten. Da aber Transaktionen in der Regel nicht streng hintereinander auftreten, sondern zeitlich ineinander verzahnt sind, erhält nur ein Schedule, der äquivalent zu einem seriellen Schedule ist, die Konsistenz der Daten. In der Literatur werden verschiedene Definition für Äquivalenz zweier Schedules angeführt (z.B. Final-State, View-, Konfliktserialisierbarkeit)[VGH93], die im Ergebnis zu unterschiedlichen Versionen von Serialisierbarkeit führen. Wir werden in dieser Arbeit die in der Praxis meist gebrauchte Version, die Konfliktserialisierbarkeit, betrachten.

Die Konfliktserialisierbarkeit basiert auf dem Begriff des Konfliktes zwischen zwei Aktionen. Hierbei stehen zwei Aktionen zueinander in Konflikt, wenn mindestens eine von beiden schreibt. Für einen Schedule über TA formalisieren wir die Menge aller Konflikte im Schedule wie folgt:

Definition 3.8 (Konfliktrelation). Sei $S = a_1 \dots a_m$ ein Schedule über TA , $ta_1, ta_2 \in TA$ kommen in S vor und $ta_1 \neq ta_2$. Zwei Aktionen a_i, a_j mit $a_i \in ta_1$ und $a_j \in ta_2$ stehen in S in Konflikt, gdw. a_i und a_j Aktionen auf derselben Variable x sind und mindestens eine der beiden Aktionen schreibt. $C(S) := \{[a_i, a_j] \mid a_i, a_j \text{ stehen in } S \text{ in Konflikt und } i < j \text{ in } S\}$ heißt *Konfliktrelation* von S . *

In einem Schedule stehen nur Aktionen unterschiedlicher Transaktionen zueinander im Konflikt und eine Konfliktrelation ist irreflexiv. Für den folgenden Beispielschedule S_5 und S_6 haben wir die Konfliktrelation $C(S_5)$ und $C(S_6)$ angegeben.

$$S_5 = r_3(z)w_1(z)w_1(x)r_2(x)r_2(y)w_3(y)r_1(u)r_3(u)$$

$$C(S_5) = \{[r_3(z), w_1(z)], [w_1(x), r_2(x)], [r_2(y), w_3(y)]\}$$

$$S_6 = r_2(y)r_3(z)w_1(x)r_1(u)w_1(z)r_2(x)w_3(y)r_3(u)$$

$$C(S_6) = \{[r_3(z), w_1(z)], [w_1(x), r_2(x)], [r_2(y), w_3(y)]\}$$

Aufbauend auf dem Begriff des Konfliktes wollen wir Äquivalenz zwischen einem Schedule über TA und einem seriellen Schedule über TA nachweisen. Dafür muß die Frage beantwortet werden, wann zwei Schedules zueinander äquivalent sind.

Definition 3.9 (Konfliktäquivalenz). Seien $S_1 = a_1 \dots a_n, S_2 = b_1 \dots b_m$ Schedules über TA . S_1, S_2 heißen *konfliktäquivalent*, gdw. $\{a_1, \dots, a_n\} = \{b_1, \dots, b_m\}$ und $C(S_1) = C(S_2)$. *

Zwei Schedules, in denen dieselben Aktionen auftreten, sind konfliktäquivalent, wenn ihre Konflikte in dem jeweiligen Schedule in gleicher Reihenfolge stehen. Betrachten wir dazu die Schedules S_5 und S_6 . Die beiden Schedules sind konfliktäquivalent, da S_5 und S_6 aus den gleichen Aktionen bestehen und $C(S_5) = C(S_6)$ gilt.

Um zu zeigen, daß ein bestimmter Schedule konsistenzhaltend ist, muß er konfliktäquivalent mit einem Schedule sein, in dem alle Transaktionen sequentiell vorkommen. Formalisieren wir deshalb den bereits informell benutzten Begriff des *seriellen Schedules*.

Definition 3.10 (serieller Schedule). Sei $S = a_1 \dots a_m$ ein Schedule über TA . S heißt *serieller Schedule*, gdw. für alle $ta \in TA$ und alle $i, j, k \in \{1, \dots, m\}$ mit $i < j < k$ gilt: Wenn $a_i \in ta$ und $a_k \in ta$, so ist auch $a_j \in ta$.

*

Der Schedule

$$S_7 = w_1(x)r_1(u)w_1(z)r_2(x)r_2(y)r_3(z)w_3(y)r_3(u)$$

ist offensichtlich ein serieller Schedule.

Damit können jetzt Konsistenz erhaltende Schedules und Konsistenz nicht erhaltende Schedules unterschieden werden. Es muß für einen beliebigen Schedule jetzt noch die Existenz eines äquivalenten seriellen Schedules nachgewiesen werden.

Definition 3.11 (Konfliktserialisierbarkeit). Sei $S_1 = a_1 \dots a_m$ ein Schedule über TA . S heißt *konfliktserialisierbar*, gdw. ein serieller Schedule S_2 existiert, so daß gilt: S_1, S_2 sind konfliktäquivalent.

*

Fragen wir uns, ob S_6 konfliktserialisierbar ist. Dafür muß es mindestens einen Schedule geben, der zu S_6 konfliktäquivalent ist.

S_7 besteht aus den selben Aktionen wie S_6 , und S_7 ist seriell. S_6 ist nicht konfliktäquivalent zu S_7 , wie anhand der Konfliktrelationen leicht zu prüfen ist.

$$C(S_7) = \{[w_1(x), r_2(x)], [w_1(z), r_3(z)], [r_2(y), w_3(y)]\}$$

S_6 ist ein Schedule über drei Transaktionen. Damit existieren genau sechs serielle Schedules über den Transaktionen. Zu keinem dieser sechs Schedules ist S_6 konfliktäquivalent. Damit ist S_6 nicht serialisierbar. Wir verzichten an dieser Stelle auf die vollständige Aufzählung aller Schedules und deren Konfliktrelationen.

Die Konfliktserialisierbarkeit kann effizienter nachgewiesen werden, als im schlechtesten Fall über den Vergleich mit allen seriellen Schedules. In [VGH93] ist ein Algorithmus angegeben, mit dem der Nachweis mit polynomiellen Aufwand über der Länge eines Schedules gelingt.

Eigenschaft 5: Striktheit

Die ACID-Eigenschaft Isolation bedeutet, das Ergebnis einer alleinigen Ausführung einer Transaktion ist gleich dem Ergebnis einer Ausführung parallel zu beliebig vielen anderen Transaktionen. Was bedeutet das? Wenn in einer Transaktion eine Variable geschrieben wird, dann hängt das zu schreibende Ergebnis meist von den vorher durch die Transaktion gelesenen Variablenwerten ab. Wenn aber die zu lesenden Variablenwerte verändert werden, solange die Transaktion noch aktiv ist, kann es sein,

daß andere Variablenwerte als bei Alleinausführung gelesen werden. Wird dadurch das Ergebnis verändert, ist dies nicht hinnehmbar. Ein weiteres Problem tritt im Falle eines Fehlers auf. Fehler können z.B. ein Systemausfall (Hardware) oder das Zurückweisen einer Transaktion bei der Verarbeitung eines Schedules sein. Dafür betrachten wir das nun folgende Beispiel:

$$S_8 = w_1(x)r_2(x)w_2(x)w_1(y).$$

Im Schedule S_8 wird die Variable x durch ta_1 geschrieben. Die Transaktion ta_1 ist aber noch nicht beendet, da die Variable y noch geschrieben werden soll. Die Transaktion ta_2 liest und schreibt x und ist dann beendet, d.h. das Ergebnis von ta_1 ist jetzt persistent. Nehmen wir jetzt an, die Verarbeitung dieses Schedules bricht infolge eines Fehlers ab. Dann muß das von ta_1 geschriebene Ergebnis zurückgesetzt werden, da die Transaktion noch nicht vollständig abgearbeitet war. Damit hat aber ta_2 ein persistentes Ergebnis auf der Basis eines nie gültigen Wertes von x produziert. Das heißt: Stünde die ta_1 nicht in dem Schedule, wäre das Ergebnis u.U. ein anderes, und das gilt es ja gerade zu vermeiden.

Die Isolation ergibt sich nun dadurch, daß jede Transaktion nur Endergebnisse der anderen lesen kann. Eine Klasse von Schedules, die Isolation sicherstellen, sind *strikte* Schedules.

Definition 3.12 (Striktheit). Sei $S_1 = a_1 \dots a_m$ ein Schedule über TA . S heißt *strikt*, gdw. für alle k und i gilt: Wenn $a_i \in ta_k$ mit $a_i \in \{r(y), w(y)\}$ und $a_n = w(y)$, $a_n \in ta_l$, $n < i$ und $l \neq k$, dann existiert kein $a_j \in ta_l$ mit $i < j$. *

Ein Beispiel für einen strikten Schedule ist S_9 .

$$S_9 = r_3(z)w_1(z)w_1(x)r_2(x)r_2(y)w_3(y)$$

In S_9 steht $r_2(x)$ hinter der Schreibaktion $w_1(x)$. Da nach $r_2(x)$ keine Aktion aus ta_1 mehr folgt und des Weiteren keine Transaktion nach $w_1(z)$ auf die Variable z und keine Transaktion nach $w_3(y)$ auf die Variable y zugreift, ist S_9 strikt.

In welchem Verhältnis stehen nun aber Serialisierbarkeit und Striktheit zueinander? Beide Mengen von Schedules stehen zueinander in keiner Teilmengenbeziehung. Wir verdeutlichen dies anhand zweier einfacher Beispiele.

Die Menge aller strikten Schedules über TA ist keine Teilmenge aller konfliktserialisierbaren Schedules über TA . Ein entsprechendes Beispiel ist der Schedule S_9 . Wir wissen, S_9 ist strikt. Zu S_9 läßt sich aber kein konfliktäquivalenter Schedule finden. Zur Begründung betrachten wir uns die Konfliktrelation des S_9 etwas genauer.

$$C(S_9) = \{[r_3(z), w_1(z)], [w_1(x), r_2(x)], [r_2(y)w_3(y)]\}$$

Die ersten beiden Konflikte in $C(S_9)$ bedeuten für einen konfliktäquivalenten seriellen Schedule, daß ta_3 vor ta_1 steht, ta_1 wiederum vor ta_3 zu finden ist. Damit muß in einem konfliktäquivalenten seriellen Schedule auch ta_3 vor ta_2 stehen. Der dritte Konflikt in $C(S_9)$ impliziert aber ta_2 vor ta_3 . Es kann also keinen konfliktäquivalenten seriellen Schedule für S_9 geben.

Umgekehrt gilt offensichtlich auch, daß die Menge aller konfliktserialisierbaren Schedules über TA keine Teilmenge aller strikten Schedules über TA ist. Dafür betrachten wir den Schedule S_{10} . S_{10} ist ein abgewandelter Schedule von S_9 , in dem die Aktion $r_3(z)$ nicht an erster, sondern an zweiter Stelle im Schedule steht.

$$S_{10} = w_1(z)r_3(z)w_1(x)r_2(x)r_2(y)w_3(y)$$

S_{10} ist nicht strikt, da jetzt hinter $r_3(z)$ mit der Aktion $w_1(x)$ eine Aktion aus ta_1 liegt.

$$S_{11} = w_1(z)w_1(x)r_3(z)w_3(y)r_2(x)r_2(y)$$

$$\begin{aligned} C(S_{10}) &= \{[w_1(z), r_3(z)], [w_1(x), r_2(x)], [r_2(y), w_3(y)]\} \\ C(S_{11}) &= \{[w_1(z), r_3(z)], [w_1(x), r_2(x)], [r_2(y), w_3(y)]\} \end{aligned}$$

Zu S_{10} ist S_{11} offensichtlich ein konfliktäquivalenter serieller Schedule, so daß S_{10} die Eigenschaft 4 besitzt.

Eigenschaft 6: Rigorosität

Eine Verschärfung der Striktheit ist die *Rigorosität*.

Definition 3.13 (Rigorosität). Sei $S_1 = a_1 \dots a_m$ ein Schedule über TA. S heißt *rigoros*, gdw. S ist strikt und für alle $ta_k, ta_l \in TA$ und $k \neq l$ gilt: Wenn $a_i \in ta_k$ und $a_i = r(x)$ und $a_j \in ta_l$ und $a_j = w(x)$ und $i < j$, dann existiert kein $a_n \in ta_k$ mit $j < n$. *

Für rigorose Schedules wird zusätzlich zur Striktheit gefordert, wann immer eine Variable x geschrieben werden soll, müssen alle Transaktionen, die x gelesen haben, beendet sein. Die Menge der rigorosen Schedules über TA ist jeweils eine Teilmenge der konfliktserialisierbaren Schedules über TA bzw. strikten Schedules über TA.

Eigenschaften 7 und 8: Sichere Schedules

Wir können jetzt festlegen, welche Schedules für unsere weitere Arbeit geeignet sind und welche eben nicht. Wir werden im Weiteren immer korrekte Schedules fordern, die zudem die Eigenschaft 4 besitzen und eine der beiden Eigenschaften 5 oder 6.

Definition 3.14 (Sicherer Schedule). Ein Schedule S über TA heißt *sicherer Schedule*, gdw. S ist korrekt und konfliktserialisierbar und mindestens strikt.

Ein sicherer Schedule, der strikt ist, besitzt die Eigenschaft 7. Wenn ein sicherer Schedule rigoros ist, dann hat er die Eigenschaft 8. Die Menge der Schedules über TA mit der Eigenschaft 8 ist offensichtlich eine Teilmenge der Menge der Schedules über TA mit der Eigenschaft 7. Die Schedules S_9 bzw. S_{10} erfüllen jeweils nur eine Eigenschaft aus E4-E6. Betrachten wir den Schedule S_{11} , in dem die Aktion $r_3(z)$ am Ende des Schedules liegt:

$$S_{11} = w_1(z)w_1(x)r_2(x)r_2(y)w_3(y)r_3(z).$$

Offensichtlich ist S_{11} konfliktserialisierbar, da es sich bereits um einen seriellen Schedule handelt. Des Weiteren ist S_{11} strikt, so daß S_{11} die Eigenschaft 7 erfüllt. Darüber hinaus ist er sogar rigoros und damit ein Schedule mit der Eigenschaft 8.

4 Transaktionale Workflow-Module

Im zweiten Kapitel haben wir Workflow-Module eingeführt und im dritten Kapitel dargelegt, welche Eigenschaften schöne Schedules besitzen. Wir führen in diesem Kapitel Workflow-Module und transaktionale Eigenschaften zusammen und definieren *transaktionale Workflow-Module*. Dafür definieren wir zunächst *gefärbte Workflow-Module* und übersetzen anschließend transaktionale Eigenschaften in Petrinetze. Die Komposition aus einem gefärbten Workflow-Modul und übersetzter transaktionaler Eigenschaft nennen wir *transaktionales Workflow-Modul* und kürzen es im Folgenden mit taWFM ab.

4.1 Einleitung

Ein Workflow-Modul gibt das interne Verhalten eines Prozesses wieder und zeigt, wie mit ihm interagiert werden kann. Aus der Struktur eines Moduls kann bereits zur Designzeit auf das Verhalten zur Laufzeit geschlossen werden. Wir haben Mengen von erreichbaren Zuständen benutzt, um Bedienbarkeit zu entscheiden. Wir können also anhand von Zuständen Bedienbarkeit nachweisen.

Ein Schedule spiegelt hingegen eine Historie wider, also eine Abfolge von bereits zurückliegenden Variablenzugriffen. Ein Schedule entsteht zur Laufzeit eines Systems. Aus einem Schedule lassen sich keine Rückschlüsse auf den Zustand eines Systems ziehen. Bei der Verarbeitung eines Schedules können in der Regel auch keine Annahmen über noch eintreffende Variablenzugriffe gemacht werden. Damit muß für jeden neu eintreffenden Variablenzugriff entschieden werden, ob ein Schedule die geforderten (ACID)-Eigenschaften besitzt. Bei den in der Praxis zur Sicherstellung der Eigenschaften eingesetzten Verfahren kann der Fall eintreten, daß ein Schedule nicht bearbeitet wird, selbst wenn er die geforderten Eigenschaften besitzt. Hier sei insbesondere das weit verbreitete *2-Phase-Locking (2PL)* genannt, das nicht alle serialisierbaren Schedules verarbeiten kann [VGH93].

Unser Ziel ist es, transaktionale Eigenschaften als internes Verhalten eines Workflow-Moduls zu repräsentieren. Aus der Struktur eines Workflow-Moduls soll die Einhaltung einer transaktionalen Eigenschaft über die Auswertung von erreichbaren Zuständen möglich sein. Wir werden transaktionale Eigenschaften als Petrinetze modellieren und in Workflow-Module integrieren. Damit besitzen wir dann ein vollständig auf Petrinetzen basierendes Modell eines Workflows, auf das die bisherigen Analysen angewandt werden können. Wir sind damit unabhängig von konkreten Verfahren zur Verarbeitung von Schedules, und die Analyse eines Moduls schließt transaktionale

Eigenschaften mit ein.

4.2 Transaktionale Workflow-Module

Um Transaktionen und transaktionale Eigenschaften modellieren zu können, muß das Modell eines Workflow-Moduls erweitert werden. Dabei gehen wir wie folgt vor: Wir nehmen an, daß zu einem Workflow-Modul die Menge der Transaktionen, um die es im Weiteren gehen wird, gegeben ist. Wir definieren, wie wir sowohl Aktionen als auch Transaktionen in einem Workflow-Modul darstellen werden. Außerdem nehmen wir als gegeben an, welche transaktionalen Eigenschaften ein Schedule über den Transaktionen erfüllen muß. Wir definieren eine Klasse von Petrinetzen, mit denen wir die geforderten Eigenschaften modellieren werden.

Definieren wir nun Workflow-Module mit Transaktionen:

Definition 4.1 (gefärbtes Workflow-Modul). Ein Petrinetz N ist ein *gefärbtes Workflow-Modul (gWFM)*, wenn

- i. N ist ein Workflow-Modul,
- ii. T ist die disjunkte Vereinigung von Mengen T_K (Kommunikationstransitionen), T_A (Aktionstransitionen), und T_M (interne Transitionen),
- iii. $F \cap (P_O \times T_A) = \emptyset$, $F \cap (T_A \times P_O) = \emptyset$, $F \cap (P_I \times T_A) = \emptyset$, $F \cap (T_A \times P_I) = \emptyset$,
- iv. T_A ist die disjunkte Vereinigung von Mengen T_{ta} (Transaktionen).

*

Aus einem Workflow-Modul wird ein gefärbtes Workflow-Modul, indem die Menge der Transitionen partitioniert wird.

Beispiel 4.1. Die Abbildung 4.1 zeigt ein gefärbtes Workflow-Modul. Es ist ein gWFM über der Menge $TA = \{ta_1, ta_2, ta_3\}$ mit $ta_1 = \{r_1(x), w_1(x), w_1(y)\}$, $ta_2 = \{r_2(x)\}$, und $ta_3 = \{w_3(y), w_3(z)\}$ zu sehen. Für ein gefärbtes Workflow-Modul vereinbaren wir folgende Notation: Wir annotieren zu einer Aktionstransition $r(x)$ oder $w(x)$ und interpretieren es als Lesen oder Schreiben einer Variablen x . Im Weiteren benutzen wir Indizes für die Färbung. Eine Aktionstransition $t \in T_{ta_3}$ wird annotiert mit $w_3(z)$, wenn z geschrieben werden soll. Transitionen, die zu einer Transaktion gehören, werden in der Darstellung mit der gleichen Farbe versehen.

Wir berücksichtigen nach Definition 4.1 ausschließlich Aktionen, die Element einer Transaktion sind. In einem Workflow kann es aber Aktionen geben, die keiner Transaktion angehören. Diese Aktionen unterliegen dann aber nicht den an die Transaktionen gestellten Anforderungen. Sie haben deshalb keinen Einfluß auf die Einhaltung

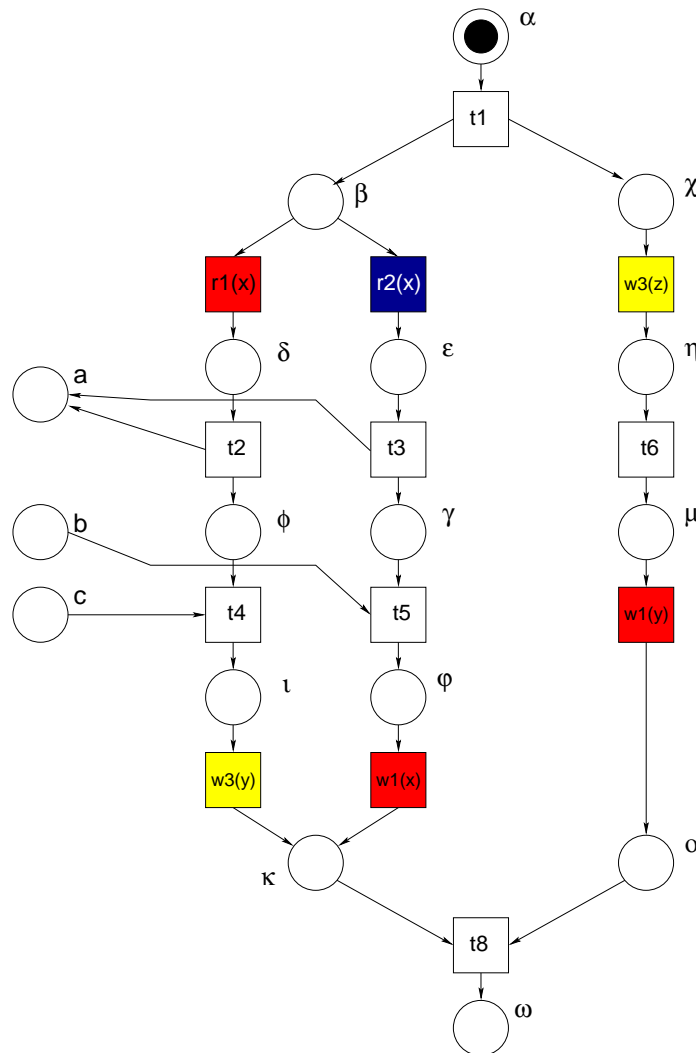


Abbildung 4.1: gefärbtes Workflow-Modul

transaktionaler Eigenschaften. In einem Workflow-Modul werden sie deshalb nicht annotiert und entsprechende Transitionen sind damit interne Transitionen, in Abbildung 4.1 z.B. die Transition t_6 . In der Definition 3.2 haben wir Transaktionen als eine Menge von Aktionen definiert. Folglich ist die Lage einer Aktion in einem Modul an keine Bedingungen geknüpft.

Die Struktur eines gefärbten Workflow-Moduls ist damit beschrieben. Wir werden jetzt klären, wie wir für die Transaktionen eines Moduls die geforderten Eigenschaften sicherstellen werden. Dafür betrachten wir erst einmal das komponierte System aus einem gefärbten Workflow-Modul und einer Umgebung. Ein solches System definiert eine Menge von sequentiellen Abläufen. Ein jeder dieser Abläufe entspricht einer Reihenfolge von Annotationen des Moduls. Wir interpretieren eine derartige

Reihenfolge als einen Schedule über TA . Wir werden einen durch einen Ablauf eines Moduls erzeugten Schedule, einen *Schedule des Moduls* nennen. Aus der Menge dieser Schedules interessieren uns die Schedules, die die geforderten Eigenschaften besitzen.

Aus einem gefärbten Workflow-Modul läßt sich die Einhaltung der transaktionalen Eigenschaften nur anhand der erzeugten Schedules nachweisen. Das heißt, wenn in einem komponierten System aus einem gefärbten Workflow-Modul und einer Umgebung das Modul seine Endmarkierung erreicht hat, muß der erzeugte Schedule untersucht werden. Wenn dieser Schedule eine Eigenschaft nicht besitzt, z.B. nicht serialisierbar ist, dann läßt sich das nicht an einem Zustand des Moduls ablesen. Ziel ist es aber, ein Modul zu besitzen, in dem anhand von Zuständen klar wird, ob ein Modul schöne Schedules erzeugt. Wir werden jetzt eine Klasse von Petrinetzen vorstellen, mit denen wir für eine gegebene Menge von Transaktionen transaktionale Eigenschaften modellieren werden.

Definition 4.2 (Eigenschaftsnetz). Ein Petrinetz E ist ein Eigenschaftsnetz wenn,

- i. P enthält eine Menge ausgezeichneter *Initialplätze* I und eine Menge ausgezeichneter *Endplätze* O ;
- ii. $m_{0_E}(i) = 1$ für alle $i \in I$;
- iii. T ist die disjunkte Vereinigung von Mengen T_A (Aktionstransitionen), und T_E (interne Transitionen),
- iv. T_A ist die disjunkte Vereinigung von Mengen T_{ta} (Transaktionen),
- v. für alle $t \in T$ gilt: t kommt in jedem Ablauf höchstens einmal vor. *

In einem Eigenschaftsnetz ist M_{f_E} eine Menge von Markierungen und für alle $m_{f_E} \in M_{f_E}$ und für alle $p \in O$ gilt $m_{f_E}(p) = 1$. Wir nennen m_{f_E} eine *Endmarkierung des Eigenschaftsnetzes* E . Für Eigenschaftsnetze vereinbaren wir folgende Notation: Ein Initialplatz wird mit einem kleinen i gekennzeichnet. Bei Bedarf wird dieses i ergänzt um eine Zahl und einen weiteren Buchstaben. Die Zahl interpretieren wir als Index einer Transaktion und den zweiten Buchstaben als Variablennamen. Ein Endplatz erhält ein o als Namen, der bei Bedarf um einen Transaktionsindex ergänzt wird. Eine Aktionstransition wird gemäß den Vereinbarungen für gefärbte Workflow-Netze gekennzeichnet und interpretiert.

Wir werden in dieser Arbeit nur Eigenschaftsnetze betrachten, denen es auch möglich ist, mindestens eine Endmarkierung zu erreichen.

Definition 4.3 (Gültiger Ablauf). Sei E ein Eigenschaftsnetz. Ein sequentieller Ablauf $m_{0_E} \xrightarrow{*} m_{f_E}$ mit $m_{f_E} \in M_{f_E}$ heißt *gültiger Ablauf*.

Mit dem Begriff des gültigen Ablaufes können wir Eigenschaftsnetze klassifizieren.

Definition 4.4 (Gültiges Eigenschaftsnetz). Sei E ein Eigenschaftsnetz. E heißt *gültiges* Eigenschaftsnetz, falls E einen gültigen Ablauf besitzt. *

Ziel ist es, transaktionale Eigenschaften in ein gültiges Eigenschaftsnetz zu übersetzen.

Beispiel 4.2. In der Abbildung 4.2 sind drei Eigenschaftsnetze abgebildet. $E1$ ist ein Eigenschaftsnetz über der leeren Menge von Transaktionen und $E2$ und $E3$ sind Eigenschaftsnetze über den beiden Transaktionen $ta_1 = \{r_1(x), w_1(x)\}$ und $ta_2 = \{r_2(x)\}$. Die beiden Netze $E1$ und $E2$ sind keine gültigen Eigenschaftsnetze, da sie jeweils keinen gültigen Ablauf besitzen. In $E1$ kann $t3$ nie schalten und $E1$ erreicht folgerichtig nie seine Endmarkierung. Im Netz $E2$ existieren genau zwei Abläufe. Im ersten Ablauf schalten die Transitionen $r_1(x)$ und $w_1(x)$. Im zweiten Ablauf schaltet die Transition $r_2(x)$. $E2$ erreicht nie eine Endmarkierung, da beide Endplätze nie gleichzeitig markiert sind.

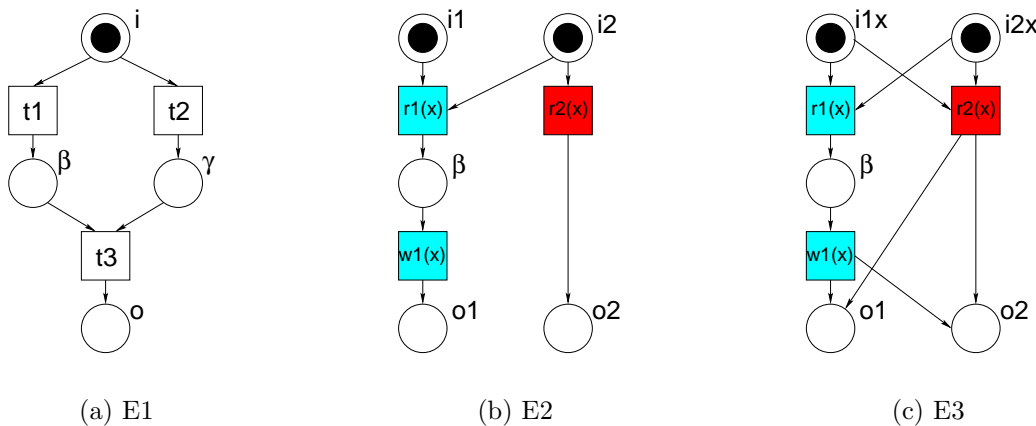


Abbildung 4.2: Eigenschaftsnetze

Ein gültiges Eigenschaftsnetz ist das Netz $E3$. In jedem Ablauf wird eine Endmarkierung erreicht, d.h. sowohl o_1 als auch o_2 sind dann markiert. *

Wir wollen die Gültigkeit einer transaktionalen Eigenschaft in einem Eigenschaftsnetz anhand einer Endmarkierung feststellen. Das gelingt nur, wenn alle Abläufe die in eine Endmarkierung führen, Schedules erzeugen, die die Eigenschaft besitzen.

Definition 4.5 (Schönes Eigenschaftsnetz). Ein Eigenschaftsnetz ist *schön* bezüglich einer Eigenschaft, wenn jeder gültige Ablauf des Eigenschaftsnetzes die Eigenschaft besitzt. *

Schöne Eigenschaftsnetze besitzen eine ausgezeichnete Menge von Zuständen, von denen wir wissen, daß ein Zustand aus dieser Menge nur über Abläufe erreicht werden kann, die schöne Schedules erzeugen. Wir können damit die Gültigkeit einer Eigenschaft anhand eines Zustandes erkennen. Wir werden im Folgenden die Formulierung, “ein Eigenschaftsnetz hat eine Eigenschaft“, anstelle von, “ein Eigenschaftsnetz ist schön bezüglich einer Eigenschaft“, synonym benutzen.

Beispiel 4.3. Betrachten wir abermals das Netz **E3** aus der Abbildung 4.2(c). **E3** besitzt, wie leicht zu sehen ist, die Eigenschaften 1-3. Das Netz enthält dennoch eine Modellierungsschwäche, denn es ist anhand der Endmarkierung nicht erkennbar, welche Transaktionen in einem erzeugten Schedule vorkommen. In **E3** kommt entweder ta_1 oder ta_2 in einem Schedule vor. Für die Analyse wird es später wichtig sein, anhand der Markierung von Initial- und Endplätzen, festzustellen, ob eine Transaktion in einem Schedule vorkommt oder nicht. Im Netz **E4**, dargestellt in Abbildung 4.3, läßt sich das offensichtlich ermitteln. Die Markierung $[i_{1x}, o_2]$ bedeutet, daß in dem dazugehörigen Schedule nur die Transaktion 2 vorkommt, und in der Markierung $[o_1, o_2]$ kommen beide Transaktionen im Schedule vor. In **E4** läßt sich das Vorkommen einer Transaktion aus einem Zustand des Eigenschaftsnetzes ermitteln.

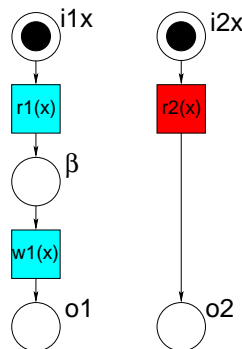


Abbildung 4.3: E4

Wir werden bei der Konstruktion von Eigenschaftsnetzen sicherstellen, daß das Vorkommen einer Transaktion aus einem Zustand ermittelt werden kann.

Ein schönes Eigenschaftsnetz über TA beschreibt mit seinen Endmarkierungen, daß alle gültigen Abläufe schöne Schedules erzeugen. Oft ist es wichtig, daß *alle möglichen* schönen Schedules über TA durch ein Eigenschaftsnetz erzeugt werden können.

Definition 4.6 (Vollständig schönes Eigenschaftsnetz). Sei E ein bezüglich einer Eigenschaft X schönes Eigenschaftsnetz und G_E die Menge aller der Schedules, die

durch alle gültigen Abläufe erzeugt werden. Sei G_S die Menge aller möglichen Schedules über TA , die die Eigenschaft X besitzen. Ein Eigenschaftsnetz E ist *vollständig schön* bezüglich einer Eigenschaft X , wenn $G_E = G_S$ ist. *

Wir werden jetzt gefärbte Workflow-Module und Eigenschaftsnetze komponieren. Wann ein gefärbtes Workflow-Modul und ein Eigenschaftsnetz komponiert werden können, zeigt die folgende Definition.

Definition 4.7 (Kompatibilität). Seien $M = (P, T, F)$ ein Workflow-Modul und $E = (P', T', F')$ ein Eigenschaftsnetz. Wir nennen M und E *kompatibel*, wenn

- i. $P \cap P' = F \cap F' = \emptyset$,
- ii. $T_A = T'_A$. *

Wenn ein gefärbtes Workflow-Modul und ein Eigenschaftsnetz komponiert werden sollen, dann besitzen sie die gleichen Transaktionen. Das komponierte System aus einem Workflow-Modul und einem Eigenschaftsnetz nennen wir *transaktionales Workflow-Modul*.

Definition 4.8 (transaktionales Workflow-Modul). Seien $M = (P, T, F)$ ein Workflow-Modul und $E = (P', T', F')$ ein Eigenschaftsnetz und seien M und E zueinander kompatibel. Das komponierte System von M und E , geschrieben $\Pi = M \oplus E$, heißt *transaktionales Workflowmodul*, wenn gilt:

- i. $P_\Pi = P \cup P'$,
- ii. $T_\Pi = T_K \cup T_M \cup (T_A \cap T'_A) \cup T_E$,
- iii. $F_\Pi = F \cup F'$. *

Ein gefärbtes Workflow-Modul und ein Eigenschaftsnetz werden komponiert, indem ihre gemeinsamen Transitionen, die Aktionstransitionen, miteinander verschmolzen werden. In einem transaktionalen Workflow-Modul werden wir die Aktionstransitionen eines gefärbten Workflow-Moduls durch ein Eigenschaftsnetz so synchronisieren, daß wann immer ein Workflow-Modul ω erreichen kann und ein schöner Schedule existiert, ein Zustand existiert, an dem beides ablesbar ist.

4.3 Petrinetz-Muster

Für die Bedienbarkeit eines Workflow-Moduls ist es wichtig, daß es eine Strategie für das Modul gibt. Hierfür muß ein Modul im komponierten System immer seine Endmarkierung erreichen können. Für ein transaktionales Workflow-Modul soll diese Entscheidung ebenfalls über Endmarkierungen getroffen werden können. Eine

Voraussetzung dafür sind schöne Eigenschaftsnetze. Damit bei der späteren Analyse kein schöner Schedule unberücksichtigt bleibt, werden transaktionale Eigenschaften in vollständig schöne Eigenschaftsnetze übersetzt.

In dieser Arbeit wird die Modellierung von transaktionalen Workflow-Modulen und deren Analyse anhand von Modulen gezeigt, die entweder die Eigenschaft E7 oder die Eigenschaft E8 besitzen. Die dafür notwendigen Eigenschaftsnetze werden für eine gegebene Menge TA konstruiert. Dabei wird wie folgt vorgegangen: Wir konstruieren Transaktionen und komponieren sie parallel. Dann konstruieren wir eine geeignete Synchronisation für die Transaktionen, so daß die geforderten Eigenschaften gelten. Das fertige Eigenschaftsnetz besitzt dann per Konstruktion die Eigenschaften.

4.3.1 Konstruktion von Transaktionen

Beginnen wir mit der Konstruktion einer Transaktion. Eine Transaktion wird aus dem leeren Eigenschaftsnetz wie folgt konstruiert:

Definition 4.9 (Konstruktion einer E1-E3 synchronisierten Transaktion).

Sei TA eine Menge von Transaktionen über $V \subseteq Var$ und sei $ta_k \in TA$ eine Transaktion. Das E1-E3 synchronisierte Eigenschaftsnetz der Transaktion ta_k wird wie folgt aus dem leeren Netz $E = (P, T, F)$, mit $P = \emptyset$, $T = \emptyset$ und $F = \emptyset$, konstruiert.

Für alle $x \in V$ gilt:

- i. Wenn es eine Aktion $a \in \{r_k(x), w_k(x)\}$ gibt, dann füge E den Platz mit dem Label i_{k_x} hinzu. In der Anfangsmarkierung gilt $m_{0_E}(i_{k_x}) = 1$.
- ii. Wenn es eine Aktion $r_k(x) \in ta_k$ gibt, dann füge E den Platz l_{k_x} und die Transition $r_k(x)$ und die Kanten $[i_{k_x}, r_k(x)]$ und $[r_k(x), l_{k_x}]$ hinzu.
- iii. Wenn es eine Aktion $w_k(x) \in ta_k$ gibt, dann füge E den Platz s_{k_x} und die Transition $w_k(x)$ und die Kante $[w_k(x), s_{k_x}]$ hinzu. Wenn es $r_k(x) \in ta_k$ gibt, füge E die Kante $[l_{k_x}, w_k(x)]$, bzw. $[i_{k_x}, w_k(x)]$ sonst hinzu.
- iv. Füge E die Transition t_{o_k} , den Platz o_k und die Kante $[t_{o_k}, o_k]$ hinzu.
- v. Für alle Aktionen $a = w_k(x)$ füge zu E die Kante $[s_{k_x}, t_{o_k}]$ hinzu. Existiert eine Aktion $a = r_k(x)$ und keine Aktion $a = w_k(x)$, dann füge die Kante $[l_{k_x}, t_{o_k}]$ hinzu.

*

Jedes Eigenschaftsnetz einer Transaktion besitzt für jede Variable genau einen Initialplatz. Für einen Initialplatz schreiben wir i_{x_y} und interpretieren diesen Namen als Initialplatz der Variablen y der Transaktion mit dem Index x . Jedes Eigenschaftsnetz einer Transaktion hat genau einen Endplatz, den wir mit o_x notieren. In der

Anfangsmarkierung sind nur die Initialplätze markiert. Eine Transaktion besitzt genau eine Endmarkierung, in der gilt $m_{f_E}(o_x) = 1$ und $m_{f_E}(p) = 0$ sonst. Zusätzlich zur Notation von Initial- und Endplätzen werden wir für den Platz im Nachbereich einer Lesetransition l_{x_y} und für den Platz im Nachbereich einer Schreibtransition s_{x_y} schreiben. Damit haben wir eine Konstruktionsvorschrift, um jede gegebene Transaktion in ein Eigenschaftsnetz, mit den Eigenschaften E1-E3 zu übersetzen.

Beispiel 4.4 (Transaktion). In der Abbildung 4.4 ist das Eigenschaftsnetz der Transaktion $ta_1 = \{r_1(x), r_1(y), w_1(y)\}$ abgebildet.

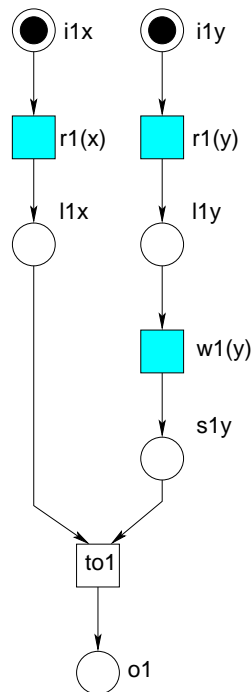


Abbildung 4.4: Eigenschaftsnetz E_{ta_1}

Das Netz besitzt die Eigenschaft E8. Jeder gültige Ablauf des Netzes erzeugt offensichtlich einen korrekten Schedule, d.h. jede Aktion steht genau einmal im Schedule, und per Konstruktion wird eine Variable gelesen, bevor sie geschrieben wird. Außerdem ist die Endmarkierung immer erreichbar, und in der Endmarkierung kommt die Transaktion vollständig in einem Schedule vor. Die Eigenschaften E4 und E5 sind für eine einzelne Transaktion nicht relevant. Damit ist das Eigenschaftsnetz vollständig bezüglich E8.

Im nächsten Schritt werden alle konstruierten Transaktionen parallel komponiert. Die Abbildung 4.5 zeigt die Komposition von E_{ta_1} und dem Eigenschaftsnetz E_{ta_2} der Transaktion $ta_2 = \{w_2(x), r_2(y), w_2(y)\}$.

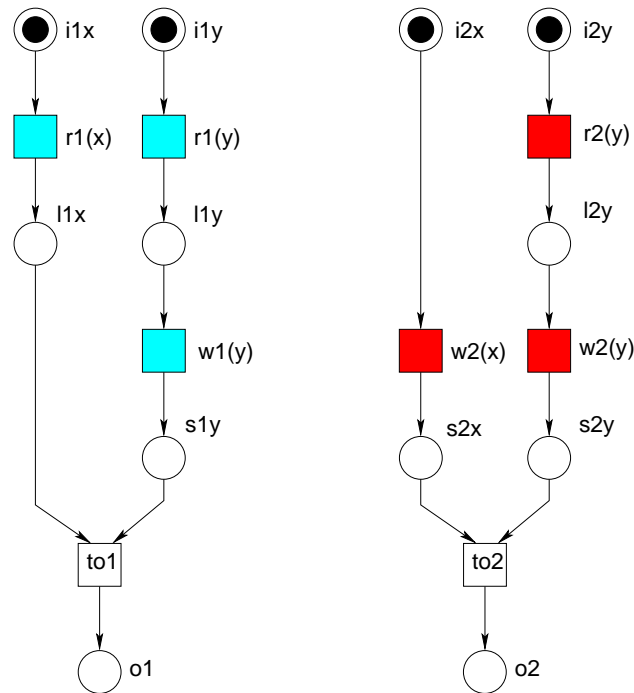


Abbildung 4.5: $E_{ta_1} \parallel E_{ta_2}$

Wenn E1-E3 synchronisierte Transaktionen parallel komponiert werden, dann besitzt das entstandene Netz offensichtlich ebenfalls wieder die Eigenschaften E1-E3. Im Netz $E_{ta_1} \parallel E_{ta_2}$ aus der Abbildung 4.5 ist die Endmarkierung der erreichbare Zustand $[o_1, o_2]$. Im Zustand $[o_1, o_2]$ haben alle Transitionen geschaltet, es wurde in jeder Transaktion gelesen, bevor geschrieben wurde, und jede Transition hat nur einmal geschaltet. Die Menge aller Schedules des Eigenschaftsnetzes enthält Schedules, die die Eigenschaft E7 oder E8 nicht besitzen. So ist z.B. der Schedule $S = w_2(x)r_1(y)r_2(y)r_1(x)w_1(y)w_2(y)$ ein durch einen gültigen Ablauf erzeugter Schedule des Netzes $E_{ta_1} \parallel E_{ta_2}$. S ist nicht serialisierbar und nicht strikt. Die Aktionstransitionen müssen deshalb geeignet synchronisiert werden.

4.3.2 Synchronisation von Transaktionen

Wir werden jetzt die Aktionstransitionen per Konstruktion so synchronisieren, daß die Menge der Schedules, die von einem Eigenschaftsnetz über TA erzeugt werden können, genau die Menge der Schedules über TA sind, die die Eigenschaft E7 oder E8 besitzen. Wir werden damit für eine gegebene Menge von Transaktionen konfliktserialisierbare und mindestens strikte Schedules für alle gültigen Abläufe sicherstellen.

Eigenschaft 7: Konfliktserialisierbarkeit und Striktheit

Beginnen wir mit der Eigenschaft E7. Für eine gegebene Menge von Transaktionen werden die Transaktionen gemäß der Definition 4.9 konstruiert und parallel komponiert. Ausgehend von einem so erzeugten Netz werden jetzt Plätze und Kanten in das Netz eingefügt und Verhalten des Netzes Schritt für Schritt eingeschränkt, bis lediglich gültige Abläufe übrig bleiben, die schöne Schedules erzeugen. Grundlage für jeden Konstruktionsschritt ist die Klassifikation möglicher Konflikte zwischen Aktionen, also Konflikten zwischen Aktionstransitionen im Netz.

Definition 4.10 (Konfliktrelationen von Transaktionen). Sei TA eine Menge von Transaktionen über $V \subseteq Var$, die jeweils gemäß Definition 4.9 in ein Eigenschaftsnetz E_{ta_i} modelliert wurden. Dann ist $E_0 = (P, T, F)$ mit $P = \bigcup_i P_i$, $T = \bigcup_i T_i$, $F = \bigcup_i F_i$ das aus den Transaktionen komponierte Eigenschaftsnetz.

- i. Zwei Transitionen $t \in T_{ta_i}$ und $t' \in T_{ta_j}$ mit $i \neq j$ stehen in E_0 zueinander in *Schreibrelation* $t\mathcal{R}_x^1 t'$ bezüglich der Variablen x mit $x \in V$, gdw. für t gilt $w_i(x)$ und für t' gilt $\{r_j(x), w_j(x)\}$.
- ii. Zwei Transitionen $t \in T_{ta_i}$ und $t' \in T_{ta_j}$ mit $i \neq j$ stehen in E_0 zueinander in *Schreib-/Leserelation* $t\mathcal{R}_x^2 t'$ bezüglich der Variablen x mit $x \in V$, gdw. für t gilt $r_i(x)$ und für t' gilt $w_j(x)$. *

Mit den Relationen \mathcal{R}_x^1 und \mathcal{R}_x^2 werden alle möglichen Konflikte über einer Menge TA beschrieben. Ein Element $[t, t'] \in \mathcal{R}_x^1$ oder $[t, t'] \in \mathcal{R}_x^2$, bedeutet in einem Schedule S über TA , daß die entsprechende Aktion $a = t$ vor $a' = t'$ steht und $[a, a'] \in C(S)$ gilt.

Konstruktion eines E7-synchronisierten Eigenschaftsnetzes

Wir können jetzt, mit Hilfe der in Definition 4.10 eingeführten Konfliktrelationen, Eigenschaftsnetze konstruieren, die die Eigenschaft E7 besitzen.

Definition 4.11 (Konstruktion eines E7-synchronisierten Eigenschaftsnetzes).

Sei $E_0 = (P, T, F)$ mit $P = \bigcup_i P_i$, $T = \bigcup_i T_i$, $F = \bigcup_i F_i$ ein gemäß Def. 4.10 konstruiertes Eigenschaftsnetz. Das E7-synchronisierte Eigenschaftsnetz über TA wird aus E_0 wie folgt konstruiert:

- i. Für alle Variablen $x \in V$ gilt, wenn die Relation \mathcal{R}_x^1 nicht leer ist, dann entsteht das Netz E_1 aus E_0 indem E_0 der Platz mit dem Label x und die folgenden Kanten hinzugefügt werden: $[x, t]$ und $[t_{o_i}, x]$. Für alle Variablen x gilt in der Anfangsmarkierung $m_{0_E}(x) = 1$.

- ii. Für alle Variablen $x \in V$ gilt, wenn die Relation \mathcal{R}_x^2 nicht leer ist, dann entsteht das Netz E_2 aus E_1 indem für jedes Element aus \mathcal{R}_x^2 , dem Netz E_1 ein Platz mit dem Label K_{i,j_x} hinzugefügt wird. Die Relation \mathcal{R}^2 ist offensichtlich asymmetrisch, so daß $K_{i,j_x} \neq K_{j,i_x}$ ist. Für alle Variablen $x \in V$ gilt, wenn $[t, t'] \in \mathcal{R}_x^2$ dann werden dem Netz folgende Kanten hinzugefügt: $[x, t]$, $[t, x]$, $[K_{i,j_x}, t]$, $[t_{o_i}, K_{i,j_x}]$, $[K_{i,j_x}, t_{o_j}]$, $[t_{o_j}, K_{i,j_x}]$. Für alle Variablen x gilt in der Anfangsmarkierung $m_{0_E}(K_{i,j_x}) = 1$.

Die Menge \mathcal{K} ist die Menge aller per Konstruktion eingefügten Plätze. *

In einem Schedule werden Konflikte nicht unterschieden. Das heißt, ein Konflikt zwischen zwei Schreibaktionen wird nicht von einem Konflikt zwischen einer Leseaktion und einer Schreibaktion unterschieden. Beide Konfliktarten werden in $C(S)$ zusammengefaßt. Beide Konflikte unterscheiden sich aber durchaus. Das Schreiben einer Variablen ändert eine globale Ressource, während das Lesen sie erst einmal nur benutzt. Wenn also eine Transaktion eine Variable schreibt, dann geht die Transaktion in eine kritische Phase und keine Aktion darf mehr auf die Variable zugreifen. Wir können im Netz E_1 aus Definition 4.11 jeden hinzugefügten Platz als Semaphore einer Variable auffassen. Der Semaphore wird zurückgegeben, wenn die Transaktion beendet wird. Das Lesen einer Variablen bedeutet, solange die Transaktion nicht beendet ist, können schreibende Zugriffe auf die Variable zur Verletzung der Eigenschaft E7 führen. Da durch das Lesen der Zugriff auf eine Variable nicht gesperrt wird, muß allen die Variable schreibenden Transaktionen signalisiert werden, von welcher noch nicht beendeten Transaktion die Variable gelesen wurde. Im Netz E_2 steht ein Platz K_{i,j_x} für einen paarweisen Konflikt zwischen ta_i und ta_j . Über die Markierung des Platzes K_{i,j_x} zeigt ta_i der Transaktion ta_j an, ob x gelesen wurde und ta_i noch nicht beendet ist.

Wir werden jetzt anhand der Synchronisation der Zugriffe auf eine Variable durch zwei Transaktionen, die Synchronisation von Aktionstransitionen erläutern.

Fall 1: Lesen neben Lesen

Der einfachste Fall liegt vor, wenn keine Transaktion eine Variable x schreibt. Wenn zwei Transaktionen x lesen, dann müssen die entsprechenden Transitionen nicht synchronisiert werden. Die Abbildung 4.6 zeigt zwei Transaktionen ta_1 und ta_2 , die x lesen. Die anderen Variablenzugriffe der beiden Transaktionen sind abgedeckt worden. Die in t_{o_1} bzw. t_{o_2} eingehenden Kanten deuten jedoch ihre Existenz an. Die beiden Lesetransitionen $r_1(x)$ und $r_2(x)$ sind nicht synchronisiert. Zwei per Konstruktion nicht synchronisierte Lesetransitionen implizieren nach Definition 4.11, daß es keine andere Transaktion $ta_i \in TA$ gibt, die x schreibt. Es gibt also kein $t \in T$, so daß $r_1(x)$ oder $r_2(x)$ in einem Element aus \mathcal{R}_x^1 oder \mathcal{R}_x^2 vorkommen.

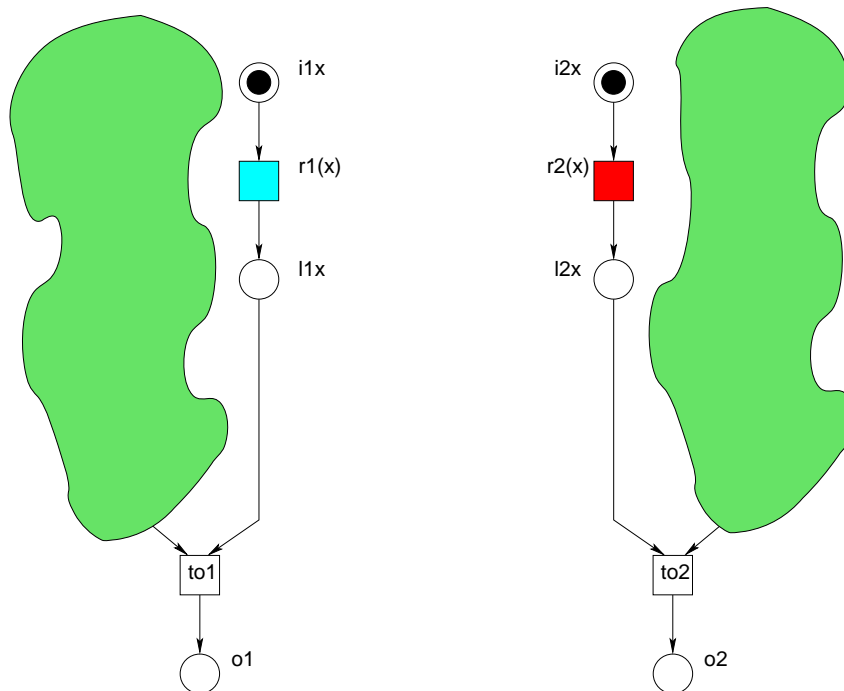


Abbildung 4.6: Synchronisation Lesen von x

Fall 2: Schreiben neben Schreiben

Im zweiten Fall (siehe Abb. 4.7) schreiben zwei Transaktionen die Variable x und keine von beiden liest x . Beide Transaktionen besitzen damit einen möglichen Scheibkonflikt. Da die Relation \mathcal{R}_x^1 für den Fall, daß zwei Transaktionen schreiben, symmetrisch ist, sind $[w_1(x), w_2(x)] \in \mathcal{R}_x^1$ und $[w_2(x), w_1(x)] \in \mathcal{R}_x^1$. Der Platz x ist Semaphore und das Schreiben von x wird verhindert, solange die Transaktion, die geschrieben hat, nicht beendet ist. Wenn eine Transaktion beendet wird, dann schaltet die Transition t_{o_i} und die Transaktion verläßt damit ihre kritische Phase. Im Beispiel kann z.B. t_{o_2} schalten, wenn in ta_2 $w_2(x)$ und $r_2(y)$ geschaltet haben, d.h. alle Variablenzugriffe von ta_2 erfolgreich waren. Wenn t_{o_2} geschaltet hat, ist der Platz x wieder markiert und der Zugriff auf die Variable x ist wieder freigegeben.

Fall 3: Lesen neben Schreiben

Im dritten Fall (siehe Abb. 4.8) stehen die Transitionen $r_1(x)$ und $w_2(x)$ in Konflikt. Es müssen hier zwei mögliche Konfliktsituationen modelliert werden. Die eine Konfliktsituation ist der Schreibkonflikt $[w_2(x), r_1(x)] \in \mathcal{R}_x^1$, und die andere Konfliktsituation ist der Lese-/Schreibkonflikt $[r_1(x), w_2(x)] \in \mathcal{R}_x^2$. Um eine Variable lesen zu können, muß der Platz x markiert sein. In der Abbildung 4.8 kann x durch $r_1(x)$

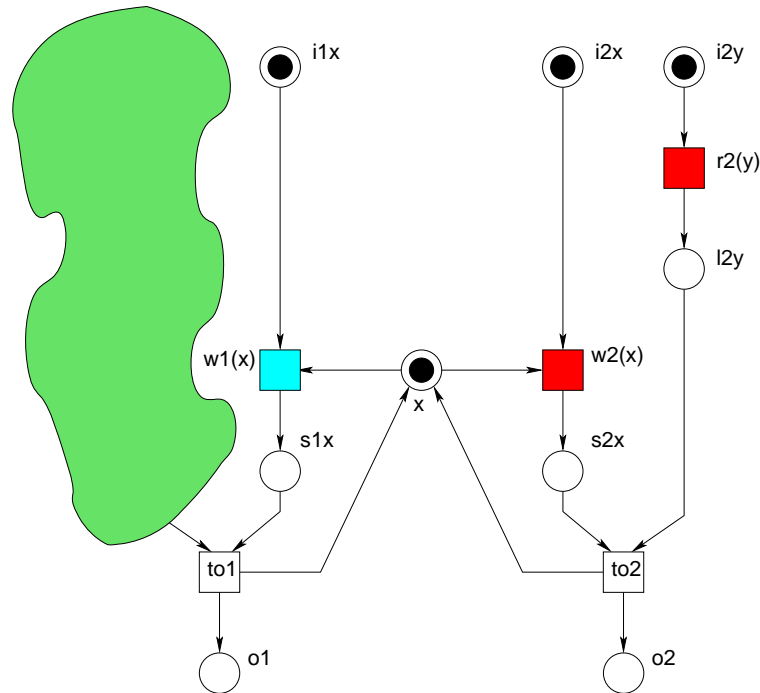


Abbildung 4.7: Synchronisation Schreib-/Schreibkonflikt

nicht gelesen werden, wenn in der Transaktion ta_2 die Transition $w_2(x)$ geschaltet hat und ta_2 noch nicht beendet wurde, also o_2 nicht markiert ist. Im Allgemeinen kann eine Transition $r_i(x)$ nicht schalten, wenn es eine andere Transaktion gibt, die x geschrieben hat und noch in ihrer kritischen Phase ist. Wenn die Transaktion ta_1 x liest, dann wird die Marke von $K_{1,2_x}$ entfernt und damit ta_2 signalisiert, daß x gelesen wurde. Da $K_{1,2_x}$ erst nach Beendigung von ta_1 wieder markiert ist, ist das Schalten von t_{o_2} damit abhängig vom Erfolg der Aktionen $r_1(y)$ und $r_1(x)$.

Fall 4: Lesen neben Lesen/Schreiben

Der vierte Fall (siehe Abb. 4.9) ist analog zum dritten Fall, nur daß hier die Transaktion ta_1 die Variable x liest, bevor sie geschrieben wird. Wenn es keine weitere Transaktion gibt, die x schreibt, existiert kein $t \in T$, so daß $r_2(x)$ in keinem Element aus \mathcal{R}_x^1 oder \mathcal{R}_x^2 vorkommt, so daß $r_2(x)$ nicht synchronisiert wird.

Fall 5/6: Lesen/Schreiben neben Schreiben oder Lesen/Schreiben

Die beiden letzten Fälle sind Kombinationen der Fälle 2 und 3 und sind in den Abbildungen 4.10 und 4.11 dargestellt.

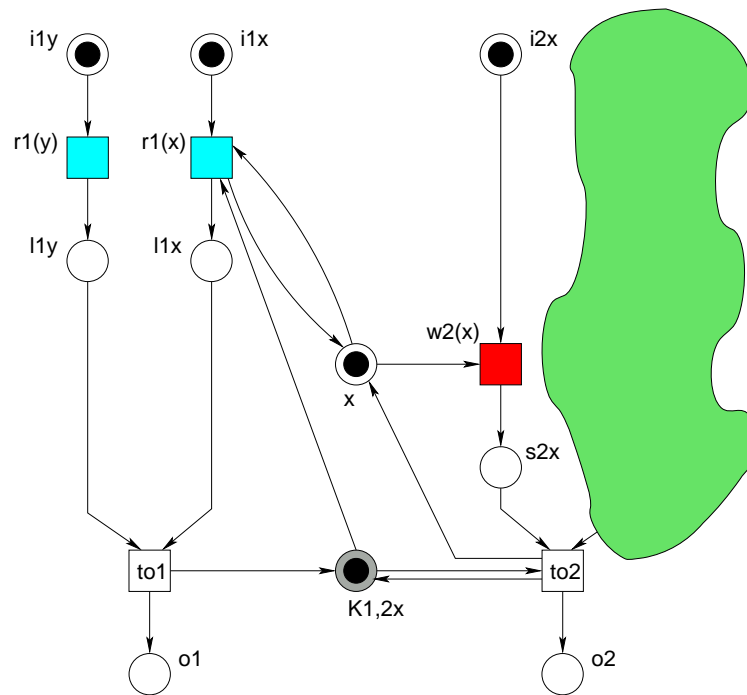


Abbildung 4.8: Synchronisation Lese-/Schreibkonflikt

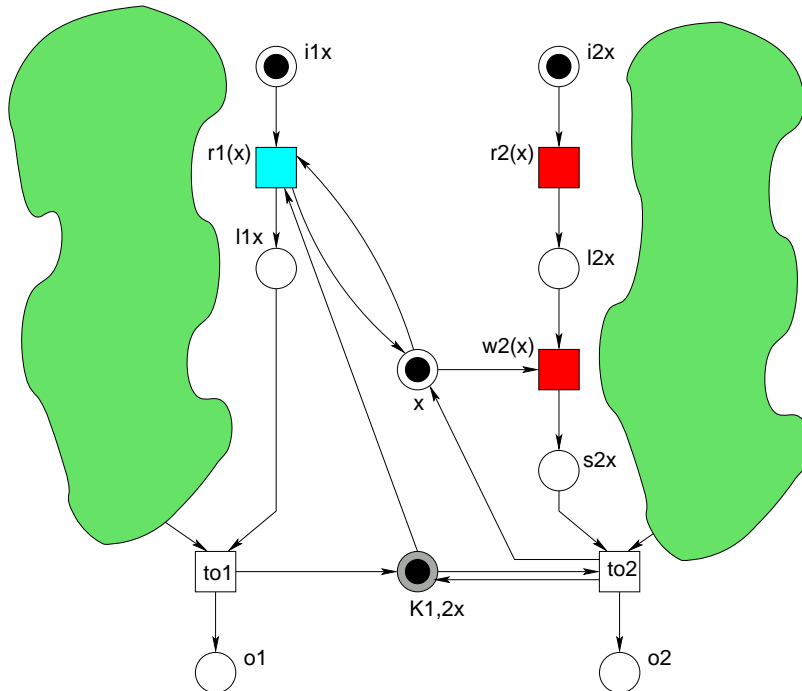


Abbildung 4.9: Synchronisation Lesen neben Lesen/Schreiben von x

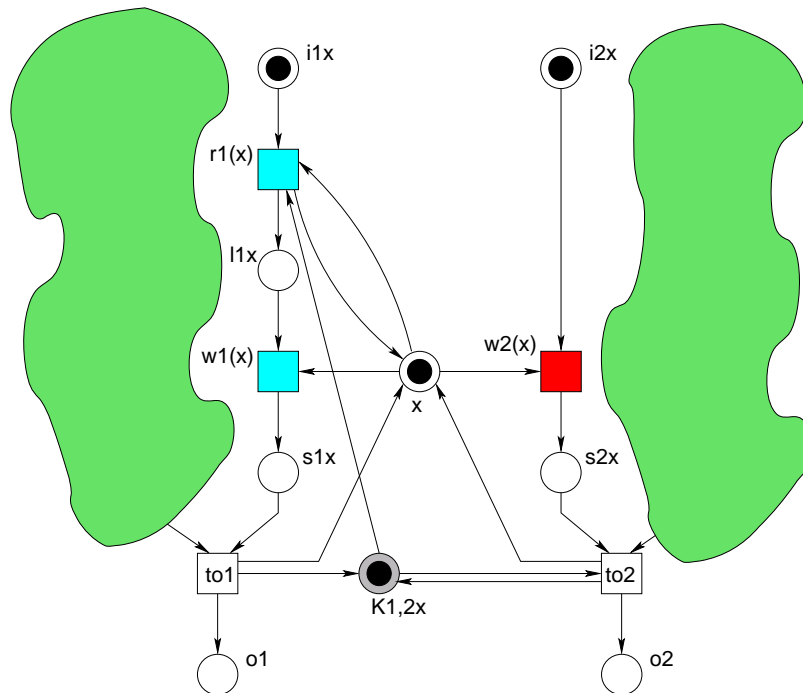


Abbildung 4.10: Synchronisation Lesen/Schreiben neben Schreiben von x

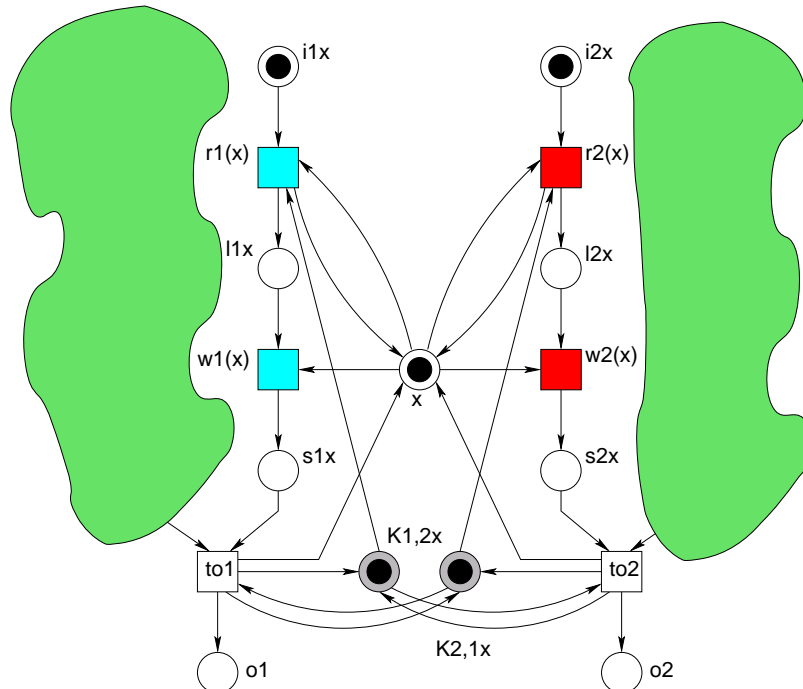


Abbildung 4.11: Synchronisation Lesen/Schreiben neben Lesen/Schreiben von x

Synchronisation ist aufwendig, so daß große Netze entstehen. Die Abbildung 4.12 zeigt die Synchronisation der Zugriffe von vier Transaktionen auf eine Variable und deutet den konstruktiven Aufwand an, der nötig ist, um ein Eigenschaftsnetz zu konstruieren. Aus Gründen der Übersichtlichkeit wurden die Label an den meisten Netzelementen weggelassen.

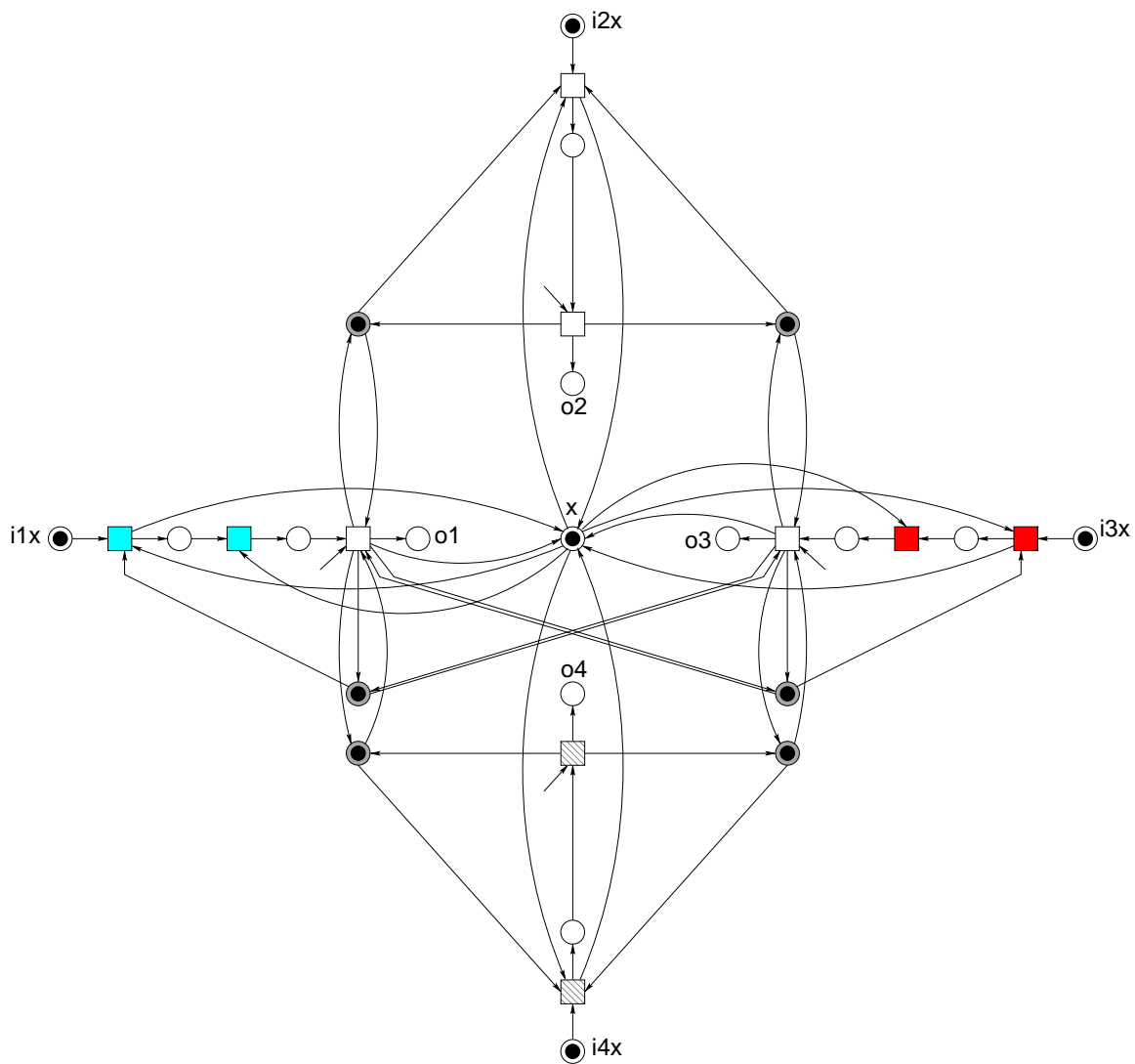


Abbildung 4.12: Synchronisation der Zugriffe auf x von vier Transaktionen

Wir werden jetzt die Korrektheit von E7-synchronisierten Eigenschaftsnetzen nachweisen. Es ist zu zeigen, daß ein E7-synchronisiertes Eigenschaftsnetz ein Eigenschaftsnetz gemäß der Definition 4.2 ist und das Netz die Eigenschaft E7 besitzt.

Satz 4.1. Ein E7-synchronisiertes Netz ist ein Eigenschaftsnetz.

Beweis (Satz 4.1): Sei E ein beliebiges E7-synchronisiertes Eigenschaftsnetz über TA . Die Punkte i)- iv) der Definition 4.2 gelten offensichtlich per Konstruktion. Es gilt 4.2 v) zu zeigen, d.h. in einem E7-synchronisierten Eigenschaftsnetz gilt für alle $t \in T$: t kommt in jedem Ablauf höchstens einmal vor. Wir zeigen: 4.2 v) gilt per Konstruktion.

Nach Definition 4.10 haben wir E aus dem Netz $E_0 = (P, T, F)$ mit $P = \bigcup_i P_i$, $T = \bigcup_i T_i$, $F = \bigcup_i F_i$ konstruiert. E_0 ist die parallele Komposition aller Transaktionen $ta_i \in TA$. Jedes Eigenschaftsnetz einer Transaktion ist per Konstruktion azyklisch (siehe Def.: 4.9). Damit ist auch die parallele Komposition azyklisch, d.h. jede Transition schaltet höchstens einmal, und kein Platz wird mehrmals markiert. Nach Definition 4.11 fügen wir in jedem Konstruktionschritt dem Netz Plätze und Kanten hinzu. Es wird in einem Konstruktionschritt ein Kantenzug von einer Transition zu einem eingefügten Platz, bzw. von einem eingefügten Platz zu einer Transition eingefügt. Es gibt damit im Netz E keinen Platz aus E_0 mit einer per Konstruktion in E_0 eingefügten eingehenden Kante. Da in E_0 kein Platz mehrmals markiert werden konnte, gilt dies damit per Konstruktion in E ebenfalls. Jede Transition $t \in T$ besitzt in E einen Platz aus E_0 in seinem Vorbereich. Damit besitzt jede Transition $t \in T$ einen Platz im Vorbereich, der höchstens einmal markiert ist. Damit sind alle Transitionen höchstens einmal aktiviert. Jede Transition $t \in T$ kann höchstens einmal schalten und kommt so höchstens einmal in jedem Ablauf vor. \square

Ziel der Konstruktion eines E7-synchronisierten Eigenschaftsnetzes ist es, ein Eigenschaftsnetz zu besitzen, in dem alle gültigen Abläufe schöne Schedules erzeugen. Es gilt, den folgenden Satz zu beweisen.

Satz 4.2. Ein E7-synchronisiertes Eigenschaftsnetz hat die Eigenschaft E7.

Für den Nachweis der Eigenschaft E7 müssen alle durch gültige Abläufe erzeugte Schedules korrekt, konfliktserialisierbar und strikt sein. Beginnen wir mit dem Nachweis der Striktheit.

Lemma 4.2.1 (Striktheit eines E7-synchronisierten Eigenschaftsnetzes). Ein E7-synchronisiertes Eigenschaftsnetz besitzt die Eigenschaft E5.

Beweis (Lemma 4.2.1): Sei $E = (P, T, F)$ ein beliebiges E7-synchronisiertes Eigenschaftsnetz. Zu zeigen ist, jeder gültige Abläufe in E ist strikt.

Wenn in E gilt, \mathcal{R}_x^1 ist die leere Schreibrelation bezüglich der Variablen x , dann gibt es keine zwei Transitionen $t \in T_{ta_i}$ und $t' \in T_{ta_j}$ und $i \neq j$ deren Aktionen

zueinander in Konflikt stehen. Damit kann es keinen Ablauf von E geben, in dem die erzeugte Reihenfolge von Aktionen auf x die Eigenschaft E5 verletzt.

Nach Def. 4.11 gilt für jede Variable $x \in V$: Wenn die Relation \mathcal{R}_x^1 nicht leer ist, dann existiert ein Platz $x \in P$. Des Weiteren gilt nach den Definitionen 4.9 und 4.11 per Konstruktion für eine Variable x und für alle t^* für die es $t^{*'}$ mit $[t^*, t^{*'}] \in \mathcal{R}_x^1$ gibt: Wenn $t^* \in T_{ta_i}$, dann gibt es nach Def. 4.9 und 4.11 per Konstruktion die Kanten $[x, t^*] \in F$ und $[t_{o_i}, x] \in F$ und $m_{0_E}(x) = 1$.

Seien ta_1 und ta_2 zwei Transaktionen in E und $\mathcal{R}_x^1 = \{[w_1(x), r_2(x)], [w_1(x), w_2(x)], [w_2(x), r_1(x)], [w_2(x), w_1(x)]\}$. Per Konstruktion gibt es für jeden Platz s_{i_x} genau eine Transition t_{o_i} im Nachbereich von s_{i_x} . Damit gilt die Invariante $x + s_{1_x} + s_{2_x} = 1$ (siehe Abbildung 4.13). Für alle Transaktionen ta_i , für die es t^* und $t^{*'}$ gibt mit $[t^*, t^{*'}] \in \mathcal{R}_x^1$, gilt somit per Konstruktion für alle i die Invariante $x + s_{1_x} + \dots + s_{i_x} + \dots + s_{n_x} = 1$. Sei jetzt $m \in R_E(m_0)$ eine von m_0 erreichbare Markierung in E und gelte $m(s_{i_x}) = 1$, d.h. die Schreibtransition $w_i(x)$ hat geschaltet. In dieser Markierung gilt $m(x) = 0$. Nach Def. 4.11 gilt per Konstruktion x liegt im Vorbereich aller der Transitionen $t^{*'}$ für die gilt, $[t^*, t^{*'}] \in \mathcal{R}_x^1$. Das bedeutet es kann keine Transition in E geben, die in m eine Lese- oder Schreibaktion auf x ausführen, d.h. schalten kann. Eine Markierung m' mit $m'(x) = 1$ kann von der Markierung m nur erreicht werden, wenn die Transition t_{o_i} schalten kann. Die Transition t_{o_i} kann offensichtlich nur schalten, wenn alle Aktionstransitionen $t'' \in T_{ta_i}$ geschaltet haben. Damit gilt in m' , $m'(o_i) = 1$. Daraus folgt, wenn $w_i(x)$ stattgefunden hat, kann es keine Aktion $\{r_j(x), w_j(x)\}$ mit $i \neq j$ geben, bevor die Transaktion ta_i beendet ist. Da dies für alle $x \in V$ gilt, gilt die Eigenschaft E5 für alle gültigen Abläufe. \square

Wir werden jede Invariante eines Eigenschaftsnetzes E , die wie im Beweis 4.2.1 aufgebaut sind, im Weiteren eine *Invariante vom Typ 1* nennen.

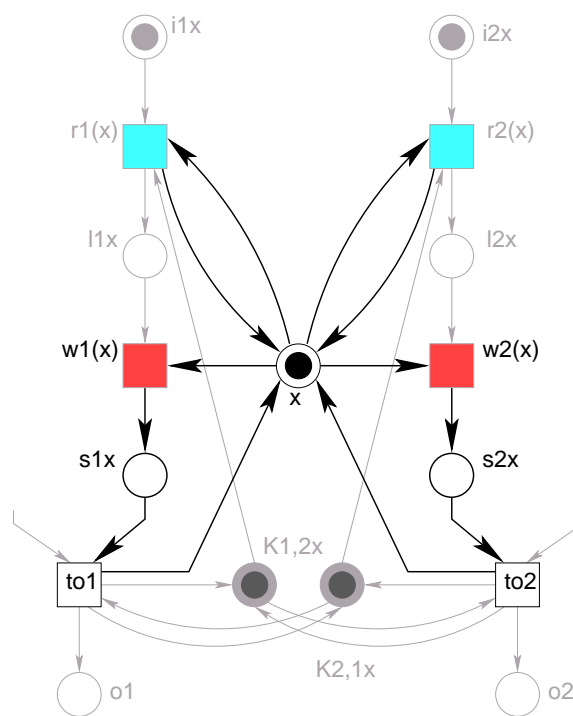


Abbildung 4.13: Invariante vom Typ 1

Lemma 4.2.2 (Konfliktserialisierbarkeit eines E7-synchronisierten Netzes).
Ein E7-synchronisiertes Eigenschaftsnetz hat Eigenschaft E4.

Beweis (Lemma 4.2.2): Sei $E = (P, T, F)$ ein beliebiges E7-synchronisiertes Eigenschaftsnetz. Seien $\mathcal{R}_E^1 = \bigcup_{x \in V} \mathcal{R}_x^1$ und $\mathcal{R}_E^2 = \bigcup_{x \in V} \mathcal{R}_x^2$ Mengen von Konflikten zwischen Aktionen aus E , dann ist $\mathcal{R}_E^{12} = \mathcal{R}_E^1 \cup \mathcal{R}_E^2$ die Menge *aller* Konflikte zwischen Aktionen aus E . Sei $\mathcal{R}^3 = (\mathbb{N} \times \mathbb{N})$ die Menge aller Paare von Indizes, dann ist $f : \mathcal{R}_E^{12} \rightarrow \mathcal{R}^3$ die Abbildung, die jedem $[a_i, a_j] \in \mathcal{R}_E^{12}$ ein Indexpaar zuordnet und $f([a_i, a_j]) = [i, j]$. Die Menge $\mathcal{R}_E^3 = f(\mathcal{R}_E^{12})$ ist die Relation der Konflikte zwischen den Transaktionen des E. Die transitive Hülle \mathcal{R}_E^{3+} ist die Menge konfliktbehafteter Transaktionen. Wenn die Relation \mathcal{R}_E^{3+} nicht irreflexiv ist, d.h. es gibt $[i, i] \in \mathcal{R}_E^{3+}$, dann gibt es mindestens einen Zyklus von Konflikten zwischen Transaktionen. Kommt in einem Schedule über TA ein solcher Zyklus vor, dann ist der Schedule nicht konfliktserialisierbar [VGH93]. Es gilt zu zeigen: Es gibt keinen gültigen Ablauf, in dem ein solcher Zyklus auftritt.

Nehmen wir an, es gibt ein i_0 mit $i_0 \mathcal{R}_E^{3+} i_0$. Dann gibt es eine Menge von Elementen $Z \subseteq \mathcal{R}_E^3$ mit $i_0 \mathcal{R}_E^3 i_1, i_1 \mathcal{R}_E^3 i_2, \dots, i_n \mathcal{R}_E^3 i_0$. Von jedem $z \in Z$ wissen wir, es gibt mindestens ein Element aus der Menge von Konflikten zwischen Aktionen \mathcal{R}_E^{12} , d.h. es gibt $[a_{i_k}, a_{i_{k+1}}] \in \mathcal{R}_E^1$ mit $f([a_{i_k}, a_{i_{k+1}}]) = z$ oder $[a_{i_k}, a_{i_{k+1}}] \in \mathcal{R}_E^2$ mit $f([a_{i_k}, a_{i_{k+1}}]) = z$, für alle $1 \leq k \leq n$. Sei nun G eine beliebige Menge von Konflikten und für alle $g \in G$ gilt: $f(g) \in Z$, d.h. G ist eine Menge von Konflikten die einen Zyklus von Konflikten zwischen Transaktionen erzeugt.

Für alle $g \in G$ gilt eines der beiden folgenden Szenarien:

1. Wenn es Transitionen $t, t' \in T$ mit $[t, t'] \in G \cap \mathcal{R}_E^1$ gibt und $t = w_k(x)$, dann bedeutet das für die beiden beteiligten Transaktionen: Wenn t in einer von m_0 erreichbaren Markierung m nach m' schalten kann, dann führt das Schalten der Transition t dazu, daß die Marke von x abgezogen wird (siehe Abb. 4.13). Wegen der Invariante vom Typ 1 gilt jetzt in m' , $m'(x) = 0$ und t' kann erst schalten, wenn die Transaktion ta_k beendet ist, d.h. $m^*(o_k) = 1$.
2. Analoges gilt, wenn es Transitionen $t'', t''' \in T$ mit $[t'', t'''] \in G \cap \mathcal{R}_E^2$ gibt und $t'' = r_k(y)$. In diesem Fall existiert nach den Definitionen 4.9, 4.11 per Konstruktion eine der folgenden Invarianten: Wenn die Transaktion ta_k die Variable y nur liest, dann gilt die Invariante $K_{k, k+1_y} + l_{k_y} = 1$. Wenn die Transaktion ta_k die Variable y liest und schreibt, dann gilt die Invariante $K_{k, k+1_y} + l_{k_y} + s_{k_y} = 1$. In beiden Fällen gilt: Nachdem $r_k(y)$ geschaltet hat, ist $K_{k, k+1_y}$ erst wieder markiert, wenn die Transaktion ta_k beendet ist (siehe Abb. 4.14 für $k = 1$).

Für den Zyklus von Konflikten der Transaktionen $i_0 - i_n$ bedeutet das: Für $[i, i+1] \in Z$ und $f([a_i, a_{i+1}]) = [i, i+1]$ gilt: Wenn Aktionstransition a_i geschaltet hat, dann ist die Aktionstransition a_{i+1} solange nicht aktiviert bis die Transaktion ta_i beendet ist. Das gilt paarweise für jedes Element $z \in Z$. Damit kann jede Transaktion des Zyklus

erst beendet werden, wenn die Vorgängertransaktion beendet ist. Da die Transaktion ta_i sowohl Vorgänger als auch Nachfolger von ta_n ist, kann die Transition t_{o_i} nie schalten. Da dies für alle Zyklen gilt, sind alle gültigen Abläufe konfliktserialisierbar. \square

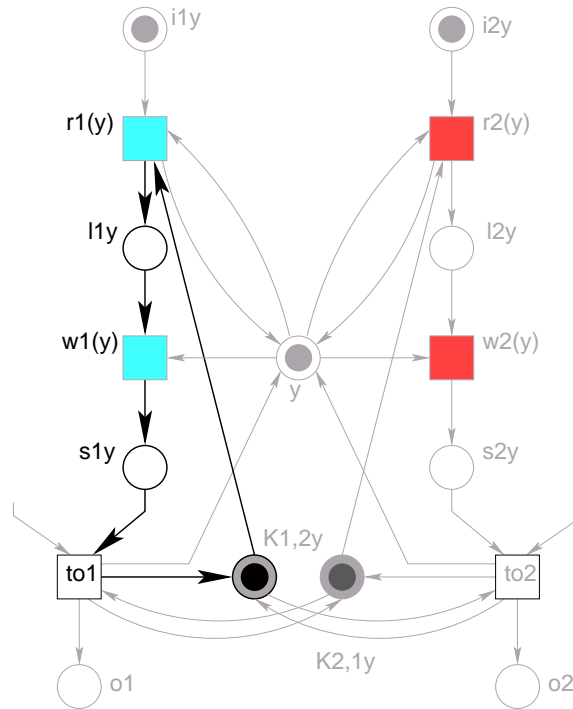


Abbildung 4.14: Invariante vom Typ 2

Wir müssen für den Nachweis des Satzes 4.2 jetzt noch Korrektheit der Schedules nachweisen.

Beweis (Satz 4.2): Sei $E = (P, T, F)$ ein beliebiges E7-synchronisiertes Eigenschaftsnetz. Es ist zu zeigen, daß alle gültigen Abläufe des E korrekte Schedules sind. Mit den bewiesenen Lemmata 4.2.1 und 4.2.2 gilt dann der Satz 4.2. Wir argumentieren analog zum Beweis 4.3.1.

Nach Definition 4.10 haben wir E aus dem Netz $E_0 = (P, T, F)$ mit $P = \bigcup_i P_i$, $T = \bigcup_i T_i$, $F = \bigcup_i F_i$ konstruiert. E_0 ist die parallele Komposition aller Transaktionen $ta_i \in TA$. Jedes Eigenschaftsnetz einer Transaktion erzeugt per Konstruktion korrekte Schedules (Beispiel 4.4). Damit erzeugt auch jeder gültige Ablauf von E_0 korrekte Schedules, d.h. in jedem gültigen Ablauf gelten die Eigenschaften $E1 - E3$. Für einen gültigen Ablauf gilt für alle $ta_i \in TA$, $m(o_i) = 1$ in der Endmarkierung. Das heißt für E , es haben alle Transitionen geschaltet. Daraus folgt, jede Transaktion kommt in dem erzeugten Schedule vollständig vor (Eigenschaft 1). Nach Beweis 4.3.1 kann jede Transition höchstens einmal schalten und damit gilt Eigenschaft 3. Die Eigenschaft 2, daß heißt lesen vor schreiben, gilt nach Definition 4.9 per Konstruktion.

Jeder gültige Ablauf des E erzeugt damit einen korrekten Schedule. Zusammen mit den Lemmata 4.2.1 und 4.2.2 gilt: E besitzt die Eigenschaft $E7$.

Satz 4.3 (Vollständigkeit bezüglich E7). Ein E7-synchronisiertes Eigenschaftsnetz ist vollständig bezüglich der Eigenschaft E7.

Beweis (Satz 4.3): Sei S ein beliebiger Schedule über TA und S besitze die Eigenschaft E7. Sei $E = (P, T, F)$ das E7-synchronisierte Eigenschaftsnetz über TA . Wir werden zeigen: Wenn S die Eigenschaft E7 besitzt, dann existiert in E ein gültiger Ablauf.

Sei $a_j \in \{r_k(x), w_k(x)\}$ eine beliebige Aktion in S . Es gilt zu zeigen: Wenn S die Eigenschaft E7 besitzt, dann kann die Aktionstransition $t \in T_{ta_k}$ mit $t = a_j$ schalten. Nehmen wir an, in E haben alle Aktionstransitionen geschaltet, die den Schedule bis zur Aktion a_j erzeugen.

1. **Fall:** Es gibt keine Aktionstransition t' mit $[t, t'] \in \mathcal{R}_E^{12}$ oder $[t', t] \in \mathcal{R}_E^{12}$. Das heißt, in S kann es keine Aktion geben, die mit a_j in Konflikt steht. Wenn $t = r_k(x)$ ist, dann ist nach Definition 4.11 per Konstruktion $\bullet t = \{i_{k_x}\}$ und $m_{0_E}(i_{k_x}) = 1$. Damit kann t schalten. Wenn $t = w_k(x)$ ist, dann ist nach Definition 4.11 per Konstruktion $\bullet t = \{i_{k_x}\}$ oder $\bullet t = \{l_{k_x}\}$. Wenn $\bullet t = \{i_{k_x}\}$ ist, dann kann t , wegen $m_{0_E}(i_{k_x}) = 1$, schalten. Wenn $\bullet t = \{l_{k_x}\}$ ist, dann muß die Transition $r_k(x)$ geschaltet haben. Da S die Eigenschaften $E1 - E3$ erfüllt, wissen wir $r_k(x)$ steht in S vor $w_k(x)$ und damit steht auch $r_k(x)$ vor $w_k(x)$ in einem Ablauf in E . Die Transition $r_k(x)$ hat geschaltet und damit ist $m(l_{k_x}) = 1$ und t kann schalten.
2. **Fall:** Es gibt Aktionstransitionen t^* mit $[t, t^*] \in \mathcal{R}_E^{12}$. Das heißt, es gibt mindestens einen möglichen Konflikt zweier Aktionen. Wir unterscheiden jetzt die Fälle $[t, t^*] \in \mathcal{R}_x^1$ und $[t, t^*] \in \mathcal{R}_x^2$.

- a) Betrachten wir zuerst den Fall $[t, t^*] \in \mathcal{R}_x^2$, das heißt $t = r_k(x)$. Da S die Eigenschaft E7 besitzt, sind alle Transaktionen, die die Variable x geschrieben haben, beendet. Aufgrund der Invariante vom Typ 1, für alle i gilt $x + s_{1_x} + \dots + s_{i_x} + \dots + s_{n_x} = 1$, gilt: Wenn alle Transaktionen, die x geschrieben haben, beendet sind, dann liegt auf keinem Platz s_{i_x} mit $1 \leq i \leq n$, eine Marke und damit gilt $m(x) = 1$. Per Konstruktion gilt, solange t nicht geschaltet hat, $m(i_{k_x}) = 1$. Für alle t^* mit $[t, t^*] \in \mathcal{R}_x^2$ gilt: Für alle Transaktionen $ta_l \in TA$ mit $t^* \in ta_l$ gilt die Invariante vom Typ 2, das heißt $K_{k,l_x} + l_{k_x} + s_{k_x} = 1$ oder $K_{k,l_x} + l_{k_x} = 1$. Da t noch nicht geschaltet hat sind l_{k_x} und s_{k_x} unmarkiert, so daß $m(K_{k,l_x}) = 1$ für alle l gilt. Damit kann t schalten.

- b) Betrachten wir nun den Fall $[t, t^*] \in \mathcal{R}_x^2$, das heißt $t = w_k(x)$. Analog zu 2a gilt $m(x) = 1$. Wenn es die Aktion $r_k(x) \in ta_k$ gibt, dann steht $r_k(x)$ vor $w_k(x)$ in S . In E hat demzufolge auch die Aktionstransition $r_k(x)$ geschaltet und der Vorbereich von $w_k(x)$ ist vollständig markiert. Wenn $r_k(x) \notin ta_k$ gilt, dann ist $i_{k_x} \in \bullet w_k(x)$ und per Konstruktion gilt $m(i_{k_x}) = 1$. Auch in diesem Fall ist der Vorbereich vollständig markiert. Damit kann t schalten.

Es können damit alle Aktionstransitionen schalten. Wenn alle Aktionstransitionen einer Transaktion k geschaltet haben, dann kann die Transition t_{o_k} schalten, wenn alle Plätze $K_{l, k_x \in V} \in \bullet t_{o_k}$ markiert sind. Wenn es mindestens einen Platz $K_{l, k_x \in V} \in \bullet t_{o_k}$ gibt, der nie mehr markiert wird, dann gibt es einen Zyklus von Konflikten, und S ist nicht serialisierbar (siehe Beweis 4.3.3). Da S serialisierbar ist, gibt es eine Markierung, in der t_{o_k} schalten kann. Da dies für alle $ta_k \in TA$ gilt, kann E seine Endmarkierung erreichen und S besitzt einen gültigen Ablauf in E . \square

Eigenschaft 8 Konfliktserialisierbarkeit und Rigorosität

Striktheit bedeutete, daß der Zugriff auf eine Variable gesperrt wird, wenn eine Transaktion die Variable schreibt und die Sperre erst wieder aufgehoben ist, wenn die Transaktion beendet ist. Wenn eine Transaktion eine Variable liest, wird sie nicht gesperrt und eine andere Transaktion darf die Variable schreiben. Die Striktheit fordert jetzt, daß auch das Lesen einer Variablen zu einer Sperre führt. Das heißt, wenn eine Variable in einer Transaktion gelesen wird, darf keine andere Transaktion die Variable schreiben. Die Menge der rigorosen Schedules über TA ist eine Teilmenge der strikten Schedules über TA und damit muß das Verhalten eines E7-synchronisierten Eigenschaftsnetzes über TA weiter eingeschränkt werden.

Konstruktion eines E8-synchronisierten Eigenschaftsnetzes

Definition 4.12 (Konstruktion eines E8-synchronisierten Eigenschaftsnetzes).

Sei $E_0 = (P, T, F)$ mit $P = \bigcup_i P_i$, $T = \bigcup_i T_i$, $F = \bigcup_i F_i$ ein gemäß Def. 4.10 konstruiertes Eigenschaftsnetz. Das E8-synchronisierte Eigenschaftsnetz über TA wird aus E_0 wie folgt konstruiert:

- i. Für alle Variablen $x \in V$ gilt, wenn die Relation \mathcal{R}_x^1 nicht leer ist, dann entsteht das Netz E_1 aus E_0 , indem E_0 ein Platz mit dem Label x und die folgenden Kanten hinzugefügt werden: $[x, t]$ und $[t_{o_i}, x]$. Für alle Variablen x gilt in der Anfangsmarkierung $m_{0_E}(x) = 1$.
- ii. Für alle Variablen $x \in V$ gilt, wenn die Relation \mathcal{R}_x^2 nicht leer ist, dann entsteht das Netz E_2 aus E_1 indem für jedes Element aus \mathcal{R}_x^2 , dem Netz E_1 ein Platz mit dem Label K_{i, j_x} hinzugefügt wird. Für alle Variablen $x \in V$ gilt,

wenn $[t, t'] \in \mathcal{R}_x^2$, dann werden dem Netz folgende Kanten hinzugefügt: $[x, t]$, $[t, x]$, $[K_{i,j_x}, t]$, $[t_{o_i}, K_{i,j_x}]$, $[K_{i,j_x}, t']$, $[t_{o_j}, K_{i,j_x}]$. Für alle Variablen x gilt in der Anfangsmarkierung $m_{0_E}(K_{i,j_x}) = 1$. *

Die Konstruktion eines E8-synchronisierten Eigenschaftsnetzes entspricht bis auf eine kleine Änderung bei den einzufügenden Kanten, der Konstruktion eines E7-synchronisierten Eigenschaftsnetzes. Der Unterschied wird im folgenden Beispiel verdeutlicht.

Beispiel 4.5. Wir beschreiben jetzt lediglich den Unterschied in der Konstruktion zwischen E7- und E8-synchronisierten Eigenschaftsnetzen. In der Abbildung 4.15 liest die Transaktion ta_1 die Variable x und ta_2 schreibt x . Betrachten wir jetzt den Platz $K_{1,2_x}$. In einem E7-synchronisierten Eigenschaftsnetz wird bei dem Leseschreibekonflikt $[r_1(x), w_2(x)] \in \mathcal{R}_x^2$ die Kante $[K_{1,2_x}, t_{o_2}]$ eingefügt (siehe Abb. 4.8). Diese Kante wird in einem E8-synchronisierten Eigenschaftsnetz durch die Kante $[K_{1,2_x}, w_2(x)]$ ersetzt. In der Abbildung 4.15 ist die neue Kante als fetter gestrichelter Pfeil dargestellt.

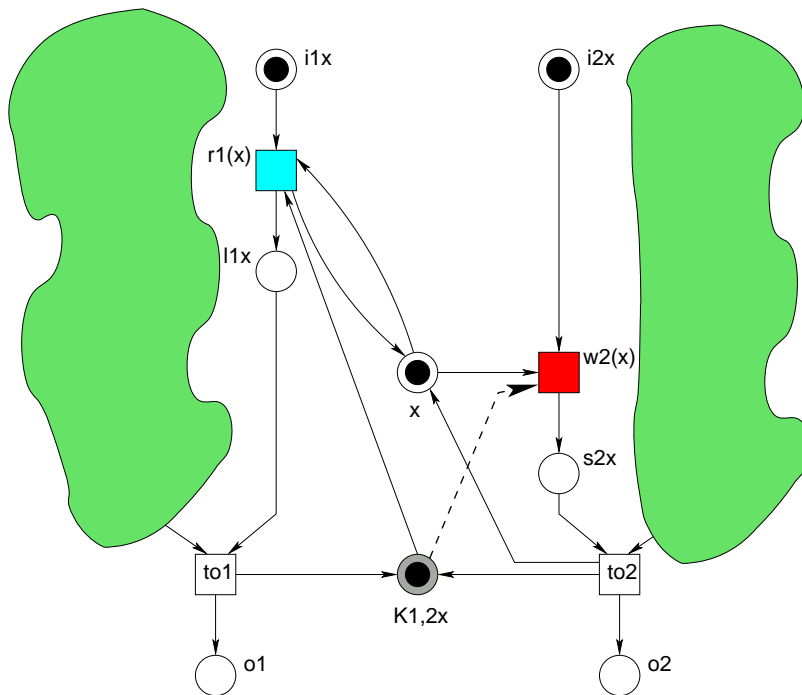


Abbildung 4.15: E8-Synchronisation der Zugriffe auf x für Lesen neben Schreiben

Satz 4.4. Ein E8-synchronisiertes Eigenschaftsnetz besitzt die Eigenschaft E8 und ist vollständig bezüglich E8.

Beweis (Satz 4.4): Sei $E = (P, T, F)$ ein beliebiges E7-synchronisiertes Eigenschaftsnetz über TA und sei E' das Netz, das durch die Ersetzung aller Kanten $[K_{k,l_{x \in V}}, t_{o_l}]$ mit $ta_k, ta_l \in TA$ durch die Kanten $[K_{k,l_{x \in V}}, w_l(x)]$ entstanden ist. Damit ist E' nach Definition 4.12 ein E8-synchronisiertes Eigenschaftsnetz. Es gilt zu zeigen, daß ein E8-synchronisiertes Eigenschaftsnetz sowohl konfliktserialisierbar als auch rigoros ist. Wir zeigen, daß die Menge aller gültigen Abläufe des Netzes E' genau die Menge der rigorosen Abläufe des E7-synchronisierten Eigenschaftsnetzes E über TA ist.

Das Netz E besitzt die Eigenschaft E7, das heißt, alle gültigen Abläufe sind konfliktserialisierbar und strikt. Seien $ta_1 \in TA$ und $ta_2 \in TA$ zwei Transaktionen und es gibt $t \in T_{ta_k}$ und $t' \in T_{ta_l}$ mit $[t, t'] \in \mathcal{R}_x^2$. Dann gilt in E die Invariante $K_{1,2_x} + l_{1_x} + s_{1_x} = 1$ bzw. wenn die Transaktion 1 die Variable x nicht schreibt $K_{1,2_x} + l_{1_x} = 1$. In E' wurde die Kante $[K_{1,2_x}, to_1]$ durch die Kante $[K_{1,2_x}, w_2(x)]$ ersetzt. Damit gilt in E' jetzt die Invariante vom Typ 3 (siehe Abb.4.16) $K_{1,2_x} + l_{1_x} + s_{1_x} + s_{2_x} = 1$ bzw. wenn die Transaktion 1 die Variable x nicht schreibt $K_{1,2_x} + l_{1_x} + s_{2_x} = 1$. Aufgrund der Invarianten vom Typ 1 und 2 kann es in E gültige Abläufe geben, in denen sowohl l_{1_x} als auch s_{2_x} gleichzeitig markiert sind. Dieser Fall tritt ein, wenn $r_1(x)$ vor $w_2(x)$ und $w_2(x)$ vor to_1 schaltet. Die Invariante vom Typ 3 drückt aus, daß dieser Fall in E' unmöglich ist. Es kann keinen Ablauf des E' geben, in dem l_{1_x} und s_{2_x} gleichzeitig markiert sind. Aus der Menge der gültigen Abläufe des Netzes E wurden damit in E' alle Abläufe ausgeschlossen, in denen die Transaktion 2 noch schreiben konnte, nachdem die Transaktion 1 bereits gelesen hat und noch nicht beendet wurde. Damit kann es keinen Ablauf von E' geben, in dem eine Variable $x \in V$ geschrieben wird, bevor alle Transaktionen, die diese Variable gelesen haben, beendet sind. Da in E bereits Striktheit galt und in E' lediglich Abläufe des E ausgeschlossen wurden, sind damit alle gültigen Abläufe des E' rigoros.

Der Nachweis der Konfliktserialisierbarkeit folgt der Argumentation des Beweises des Lemma 4.2.2. Damit besitzt das Eigenschaftsnetz E' die Eigenschaft E8.

Per Konstruktion wurden in E' genau die Abläufe ausgeschlossen, in denen Rigo-rosität verletzt wurden. Da E vollständig bezüglich der Eigenschaft E7 ist und in E' genau die nicht rigorosen Abläufe ausgeschlossen sind, ist E' vollständig bezüglich E8. □

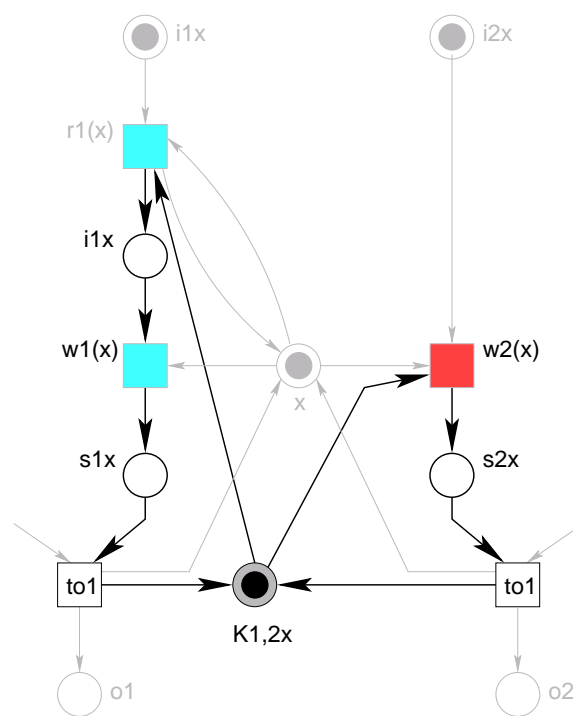


Abbildung 4.16: Invariante vom Typ 3

4.3.3 Synchronisierte taWFM

Mit den Konstruktionsvorschriften für E7- und E8-synchronisierte Eigenschaftsnetze sind wir jetzt in der Lage, gefärbte Workflow-Module geeignet zu synchronisieren.

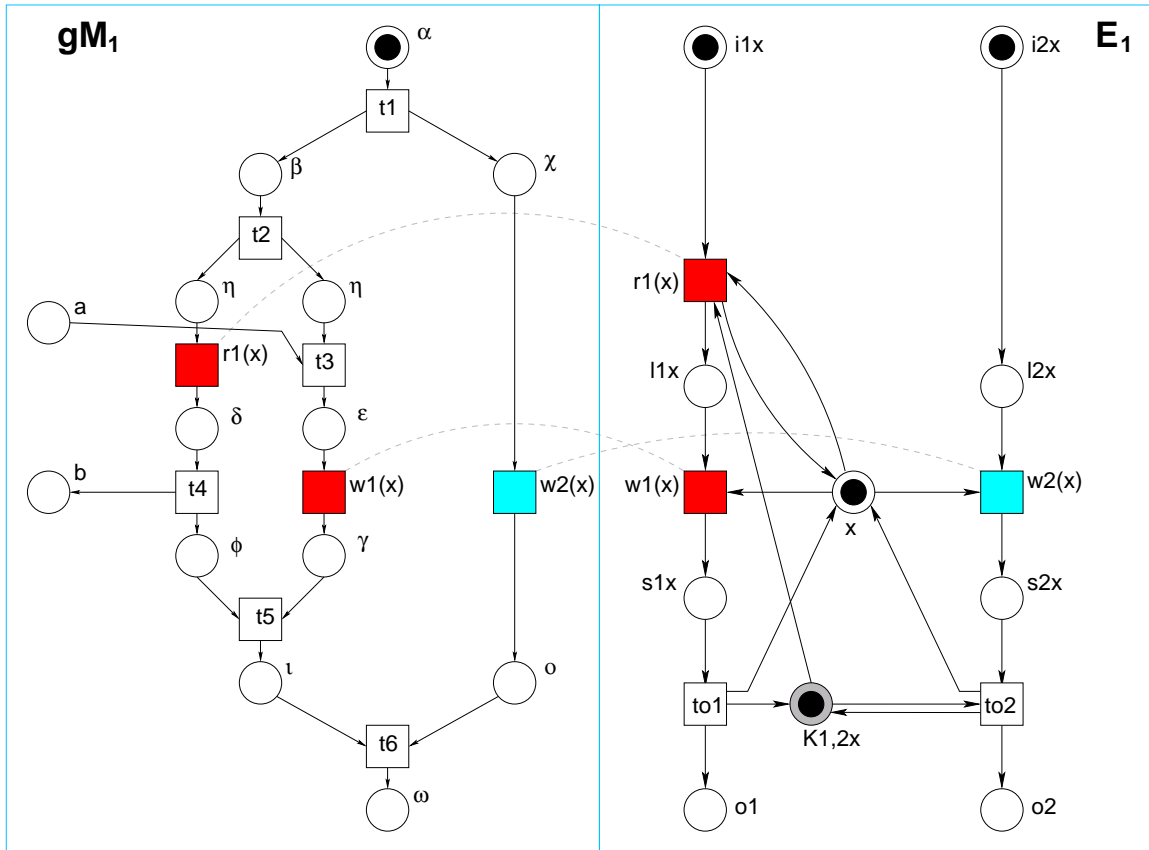


Abbildung 4.17: transaktionales Workflow-Modul Π_1

Die Abbildung 4.17 zeigt das transaktionale Workflow-Modul Π_1 , bestehend aus dem gefärbten Workflow-Modul gM_1 und dem E7-synchronisierten Eigenschaftsnetz E_1 . Das Modul gM_1 besitzt die Transaktionen $ta_1 = \{r(x), w(x)\}$ und $ta_2 = \{w(x)\}$. Die gestrichelten Linien zwischen einigen Transitionen verdeutlichen, daß es sich bei den Transitionen an den Enden einer Linie um ein und dieselbe Transition handelt.

Die Anfangsmarkierung eines taWFM $m_{0\Pi}$ setzt sich zusammen aus der Anfangsmarkierung eines gefärbten Workflow-Moduls $m_{0_{gM}}$ und der Anfangsmarkierung des komponierten Eigenschaftsnetzes m_{0_E} . Die in der Markierung $m_{0\Pi}$ markierten Plätze fassen wir zusammen zur Menge $A_\Pi = \{\alpha\} \cup I \cup \mathcal{K}$. In $m_{0\Pi}$ gilt für alle $p \in A_\Pi$ $m_{0\Pi}(p) = 1$ und $m_{0\Pi}(p) = 0$ sonst.

5 Analyse transaktionaler Workflow-Module

Wir haben im letzten Kapitel den Einfluß von transaktionalen Eigenschaften auf den Kontrollfluß eines Workflow-Moduls anhand der Eigenschaften E7 und E8 beschrieben und modelliert. Dabei haben wir transaktionale Eigenschaften als internes Verhalten eines Workflow-Moduls repräsentiert. Die Einhaltung einer transaktionalen Eigenschaft kann an einem Zustand eines Moduls abgelesen werden. Da die Bedienbarkeitsanalyse eines Moduls auf der Auswertung von erreichbaren Zuständen eines Moduls, in einem komponierten System aus einem Modul und einer Umgebung basiert, kann diese Analyse auf transaktionale Workflow-Module angewandt werden. Hierfür sind lediglich kleine Anpassungen vorzunehmen, die wir jetzt beschreiben.

Eigenschaftsnetze sind ein Modell, um für eine gegebene Menge von Transaktionen, transaktionales Verhalten für alle Transaktionen abzubilden. Deshalb besitzt ein Eigenschaftsnetz eine Eigenschaft genau dann, wenn es alle Transaktionen abarbeiten kann, das heißt, eine Endmarkierung erreicht. In einem transaktionalen Workflow-Modul bedeutet diese Annahme jedoch, daß alle Aktionstransitionen in einem Ablauf des Moduls vorkommen müssen. Stellen wir uns folgende Situation vor. Wenn in einem Modul zwei Transaktionen auf alternativen Pfaden liegen, dann kann höchstens eine von beiden ausgeführt werden. Damit kann in einem Ablauf des Moduls mindestens eine Transaktion nicht in seine Endmarkierung gelangen. Vielmehr verbleibt diese Transaktion in seiner Initialmarkierung. Für die Analyse wollen wir unterscheiden können, ob eine Transaktion in einem Ablauf eines transaktionalen Workflow-Moduls vorkommt oder nicht. Dafür benötigen wir einerseits Eigenschaftsnetze, in denen anhand einer Markierung das Vorkommen einer Transaktion entschieden werden kann (siehe Beispiel 4.3) und andererseits eine geeignete Endmarkierung eines transaktionalen Workflow-Moduls. In E7/E8-synchronisierten transaktionalen Workflow-Modulen ist das Vorkommen einer Transaktion in einem Schedule eindeutig an einer Markierung des Moduls erkennbar. Die Endmarkierung m_{f_M} eines Workflow-Moduls M ergänzen wir für transaktionale Workflow-Module wie folgt:

Definition 5.1 (Endmarkierung eines E7/E8-synchronisierten taWFM). Sei Π ein E7/E8-synchronisiertes transaktionales Workflow-Modul über TA . Die Menge M_{f_Π} ist die Menge von Endmarkierungen von Π und für jede Markierung $m_{f_\Pi} \in M_{f_\Pi}$ gilt: $m_{f_\Pi}(\omega) = 1$ und $m_{f_\Pi}(p) = 1$ für alle $p \in \mathcal{K}$ und für alle Transaktionen $ta_k \in TA$ ist $m(o_k) = 1$ oder wenn $m(o_k) = 0$, dann ist $m(i_{k_x}) = 1$ für alle $x \in V$ und $m_{f_\Pi}(p) = 0$ sonst.

Ein Endzustand ist eine Markierung, die über einen Ablauf erreichbar ist, in der eine Transaktion entweder vollständig oder gar nicht vorkommt. In einer Endmarkierung sind alle Input- und Outputplätze unmarkiert. In einer Endmarkierung ist kein Platz $p \in P_M$ markiert mit Ausnahme von ω .

Die Definitionen für Umgebungen (2.7), zulässige Umgebung (2.8) und komponiertes System (2.9) ergeben sich kanonisch für transaktionale Workflow-Module, so daß wir an dieser Stelle auf deren Ausführung verzichten. Die Definition einer Strategie jedoch muß für transaktionale Workflow-Module angepaßt werden.

Definition 5.2 (Strategie eines transaktionalen Workflow-Moduls). Sei Π ein transaktionales Workflow-Modul. Eine Umgebung ist eine *Strategie* für Π , wenn im komponierten System von jedem aus dem Anfangszustand erreichbaren Zustand, ein Zustand erreichbar ist, dessen erste Komponente $m_{f_\Pi} \in M_{f_\Pi}$ ist. *

Für den Nachweis, ob eine gegebene Umgebung eine Strategie für ein Workflow-Modul ist, genügt es für das komponierte System zu zeigen, daß das Modul immer die Markierung $[\omega]$ erreicht. Wie das folgende Beispiel zeigt, reicht diese Annahme in einem taWFM nicht aus.

Beispiel 5.1. Die Abbildung 5.1 zeigt das taWFM Π_2 , bestehend aus dem gefärbten Workflow-Modul \mathbf{gM}_2 und dem Eigenschaftsnetz \mathbf{E}_2 . Das Modul kann die beiden Nachrichten \mathbf{a} und \mathbf{b} empfangen und besitzt die Transaktion $ta_1 = \{r(x), w(x)\}$. Um den Unterschied zwischen einer Strategie für ein Workflow-Modul und einer Strategie für ein taWFM zu verdeutlichen, werden wir jetzt sowohl das komponierte System aus \mathbf{gM}_2 und der zulässigen Umgebung $\mathbf{U2}$ (Abb. 5.2) als auch das komponierte System aus Π_2 und $\mathbf{U2}$ untersuchen.

Beginnen wir mit dem komponierten System $\mathbf{gM}_2 \oplus \mathbf{U2}$, das in der Abbildung 5.3 dargestellt ist. Aus dem System geht hervor, daß $\mathbf{U2}$ eine Strategie für \mathbf{gM}_2 ist. Damit bedient $\mathbf{U2}$ das Modul \mathbf{gM}_2 .

Betrachten wir nun das komponierte System aus Π_2 und $\mathbf{U2}$ aus Abbildung 5.4. Auch hier gilt offenbar, daß aus jedem erreichbaren Zustand des Systems ein Zustand erreichbar ist, in dem in \mathbf{gM}_2 der Platz ω markiert ist. Dennoch ist $\mathbf{U2}$ keine Strategie für Π_2 . Damit $\mathbf{U2}$ eine Strategie ist, muß die erste Komponente im komponierten System eine Endmarkierung des Π_2 sein. Das Modul Π_2 besitzt mit $[\omega, \mathbf{o}_1]$ genau eine Endmarkierung. Aus dem Zustand $[\omega, \mathbf{1x}, \mathbf{2}]$ ist kein weiterer Zustand erreichbar, und es liegt offensichtlich keine Endmarkierung des Moduls Π_2 vor. Die Umgebung $\mathbf{U2}$ bedient damit Π_2 nicht.

Dafür gibt es folgende Ursache: Wenn die Umgebung an Π_2 eine Nachricht \mathbf{a} sendet und $\mathbf{t2}$ schaltet, dann kann die Transition $w_1(x)$ nicht schalten. Damit kommt ta_1 in dem vom Modul erzeugten Schedule nicht vollständig vor. Dies ist ersichtlich in der Markierung $[\omega, \mathbf{1x}]$.

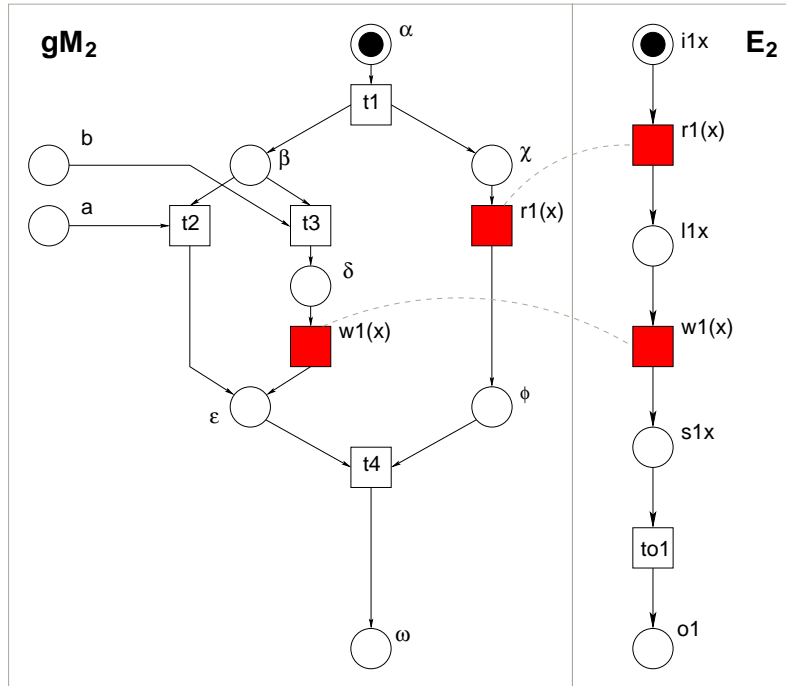


Abbildung 5.1: taWFM Π_2

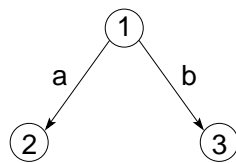


Abbildung 5.2: Umgebung U_2

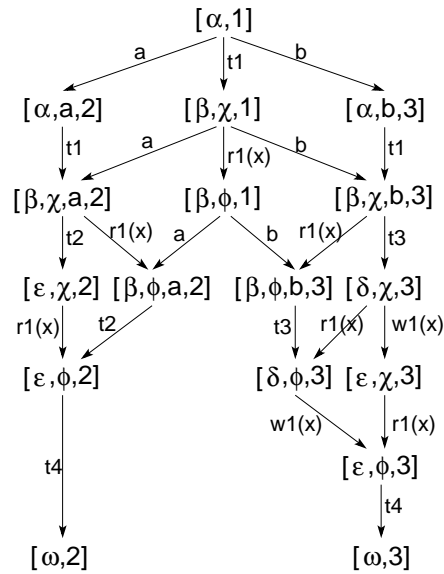


Abbildung 5.3: komponiertes System $gM_2 \oplus U_2$

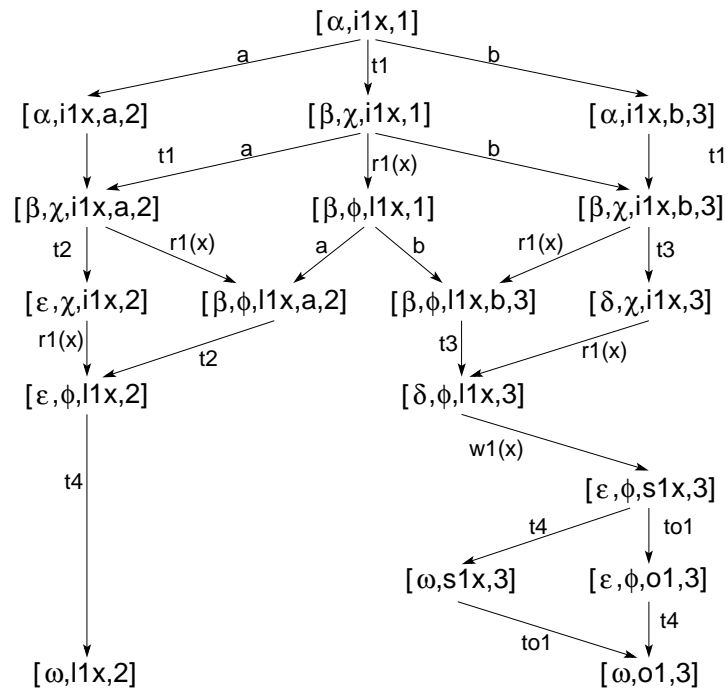


Abbildung 5.4: komponiertes System $\Pi_2 \oplus U_2$

Das Beispiel 5.1 zeigt, daß eine bedienende Umgebung eines (gefärbten) Workflow-Moduls, bei Annahme transaktionaler Eigenschaften für das entsprechende transaktionale Workflow-Modul, ungeeignet sein kann.

Wir wissen jetzt, wie ein Endzustand eines taWFM aufgebaut sein muß, um eine Umgebung als eine Strategie für ein taWFM klassifizieren zu können. Das Wissen einer Umgebung für ein taWFM läßt sich kanonisch aus der Definition 2.12 ableiten. Damit können wir jetzt analog zur Definition 2.13, Deadlocks von transaktionalen Workflow-Modulen definieren.

Definition 5.3 (Deadlocks eines transaktionalen Workflow-Modul). Sei Π ein transaktionales Workflow-Modul. Ein Deadlock von Π ist eine Markierung, in der keine Transition aktiviert ist. Ein Deadlock ist intern, wenn $m \notin M_{f_\Pi}$, für jede Transition existiert ein $p \in P_\Pi$ mit $[p, t] \in F_\Pi$ und $m(p) = 0$ und alle Outputplätze unmarkiert sind. Ein Deadlock ist extern, wenn er weder intern noch $m \in M_{f_\Pi}$ ist. *

Da das Theorem 2.1 auch für transaktionale Workflow-Module gültig bleibt, gelingt es mit Hilfe des Wissens einer Umgebung und Deadlocks, auch für taWFM die Knoten einer Umgebung zu klassifizieren. Die Begründung für den Nachweis der Gültigkeit des Theorems folgt dem Beweis des Theorems in [Sch04].

Wir haben im Beispiel 5.1 festgestellt, daß die Umgebung U2 das Modul Π_2 nicht bedient. Dennoch gibt es eine Umgebung, die Π_2 bedient. Wie diese berechnet werden kann, zeigen wir jetzt.

Wie bereits in Kapitel 2 erwähnt wurde, ist in [Sch04] ein Algorithmus angegeben worden, mit dem sich aus einer konstruierten Umgebung alle möglichen Strategien berechnen lassen. Die Funktionsweise des Algorithmus werden wir jetzt informell kurz beschreiben. Sei U eine gegebene Umgebung für ein Modul M . Streiche aus U genau die Zustände, für die $K(q)$ einen internen Deadlock enthält. Des Weiteren streiche die Zustände, für die $K(q)$ einen externen Deadlock enthält und es für diesen Deadlock keine Transition im komponierten System gibt. Wiederhole das Streichen von Knoten so lange, bis keine Knoten mehr gestrichen werden können. Wenn eine nichtleere Menge von Zuständen übrig bleibt, dann ist diese Umgebung eine Strategie für M .

Beispiel 5.2. Wir werden den gerade beschriebenen Algorithmus auf unser Beispiel 5.1 anwenden. Die Abbildung 5.5 zeigt das Wissen von U2. $K(q)$ enthält im Zustand 2 einen internen Deadlock, so daß dieser Zustand gestrichen wird. Der Zustand 1 enthält einen externen Deadlock, für den es im komponierten System die Transition b gibt. Im Zustand 3 besitzt $K(q)$ weder einen internen noch einen externen Deadlock. Damit ist die Umgebung, bestehend aus den Zuständen 1 und 3 eine Strategie für das Modul Π .

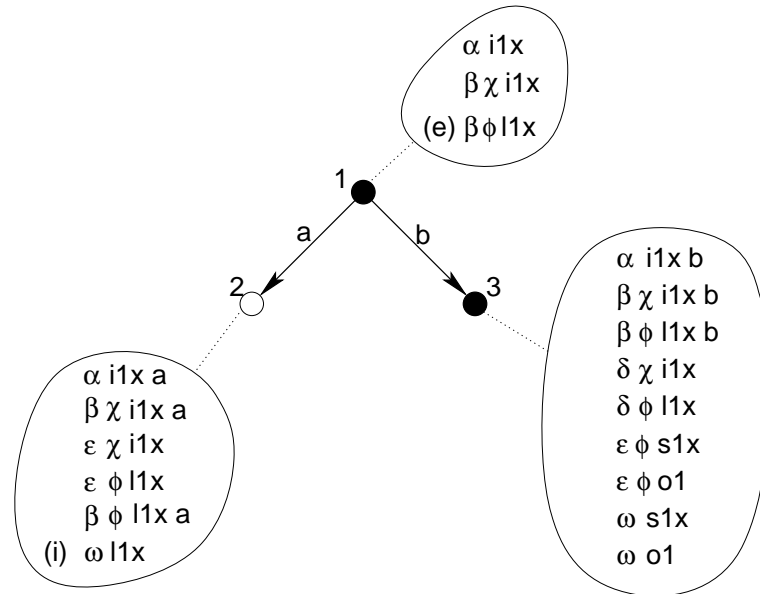


Abbildung 5.5: Wissen $\Pi_2 \oplus U_2$

Die Beispiele 5.1 und 5.2 zeigen, daß sich die Menge der Strategien für ein Workflow-Modul bei Annahme von transaktionalen Eigenschaften verändern kann. Im äußersten Fall kann es sein, daß ein bedienbares Workflow-Modul bei Annahme von transaktionalen Eigenschaften nicht mehr bedienbar ist.

Es gibt aber auch nicht bedienbare Workflow-Module, die unter Annahme transaktionaler Eigenschaften bedienbar sind. Auch hier gelingt es, Bedienbarkeit anhand der Analyse eines taWFM zu zeigen. Das folgende Beispiel zeigt ein nicht bedienbares gefärbtes Workflow-Modul, dessen E7-synchronisiertes taWFM bedienbar ist.

Beispiel 5.3. Das in Abbildung 5.6 dargestellte taWFM Π_3 ist die Komposition des Moduls gM_3 und dem E7-synchronisierten Eigenschaftsnetz E_3 über den Transaktionen des gM_3 . Als Grundlage für gM_3 diente das nicht bedienbare Modul M_2 aus Abbildung 2.1(b). M_2 wurde um die Transaktionen $ta_1 = \{w(x), w(z)\}$, $ta_2 = \{r(x), w(x)\}$ und $ta_3 = \{r(y), w(y)\}$ ergänzt.

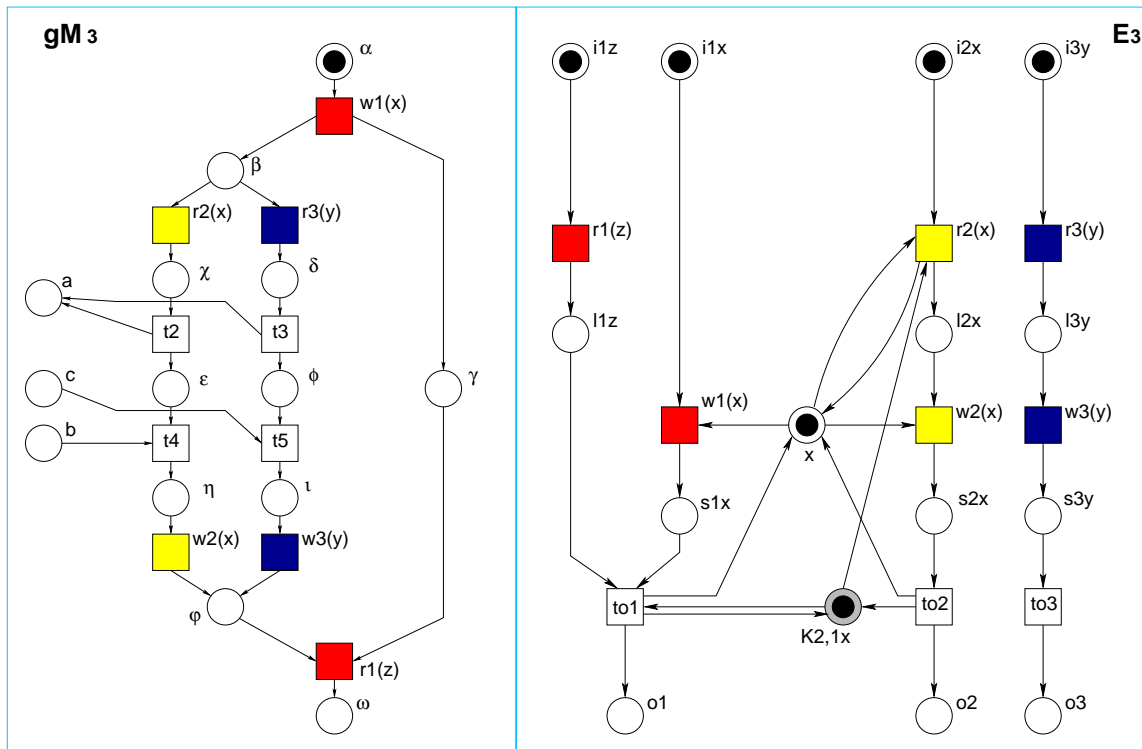


Abbildung 5.6: taWFM Π_3

Wir untersuchen jetzt das komponierte System aus gM_3 und der Umgebung U_1 aus Abbildung 5.7.

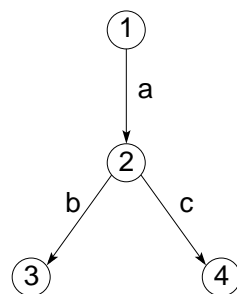


Abbildung 5.7: Umgebung U_1

Die Abbildung 5.8 zeigt für die Zustände der Umgebung U1 Ausschnitte des Wissens im jeweiligen Zustand. Die Zustände 3 und 4 enthalten jeweils einen internen Deadlock. Damit ist U1 keine Strategie für das Modul gM_3 . Es ist auch keine Teilmenge von Zuständen von U1 eine Strategie für gM_3 , denn nach Streichen der Zustände 3 und 4, können die externen Deadlocks im Zustand 2 nicht mehr aufgelöst werden, so daß der Zustand 2 gestrichen wird. Damit existiert in der Umgebung kein Zustandsübergang mehr, und die beiden Deadlocks im Zustand 1 werden nicht aufgelöst. Es bleibt demzufolge die leere Menge von Zuständen übrig.

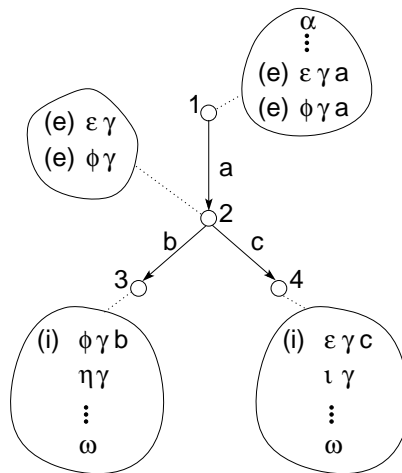


Abbildung 5.8: Wissen $gM_3 \oplus U1$

Darüber hinaus existiert für das Modul gM_3 keine Strategie, so daß gM_3 nicht bedienbar ist. Die Ursache liegt in der nichtdeterministischen Auswahl, die das Modul in der Markierung $[\beta, \gamma]$ treffen kann. Das bedeutet, gM_3 versendet die Nachricht a entweder über t_2 oder über t_3 . Die Umgebung kann aus der Nachricht a nicht ableiten, in welchem Zustand sich das Modul befindet und welche Nachricht das Modul erwartet. Dieses Problem ist auch als Non-Local-Choice-Problem bekannt.

Für das komponierte System $\Pi_3 \oplus U1$ dagegen können wir eine bedienende Umgebung finden. Die Abbildung 5.9 zeigt ausschnittsweise das Wissen der Umgebung in seinen Zuständen. Im Zustand 3 besitzt $K(q)$ einen internen Deadlock und dieser Zustand kann nicht Bestandteil einer Strategie sein. Das Wissen im Zustand 4 besitzt keinen externen und keinen internen Deadlock und mit der Markierung $[\omega, o_1, i_2, o_3, K_{2,1_x}, x]$ die Endmarkierung des taWFM Π_3 . Aus dieser Markierung ist ersichtlich, daß gM_3 und die Transaktionen ta_1 und ta_3 vollständig abgearbeitet wurden. Die Transaktion ta_2 hingegen ist in ihrer Initialmarkierung verblieben. Wenn diese Markierung erreicht wird, dann ist ein Schedule mit der Eigenschaft E7 erzeugt wor-

den. Offensichtlich läßt sich aus der Umgebung U1 jetzt eine Strategie herausrechnen. Die Umgebung bestehend aus den Zuständen 1, 2 und 4 bedient das taWFM Π_3 .

Die Ursache für die Bedienbarkeit von Π_3 ist die Tatsache, daß die Transition $r_2(x)$ nie schalten kann. Die Markierung, die das Schalten von $r_2(x)$ verhindert, ist in Abbildung 5.9 grau dargestellt. Das Schalten von $r_2(x)$ muß verhindert werden, da sonst Striktheit verletzt ist.

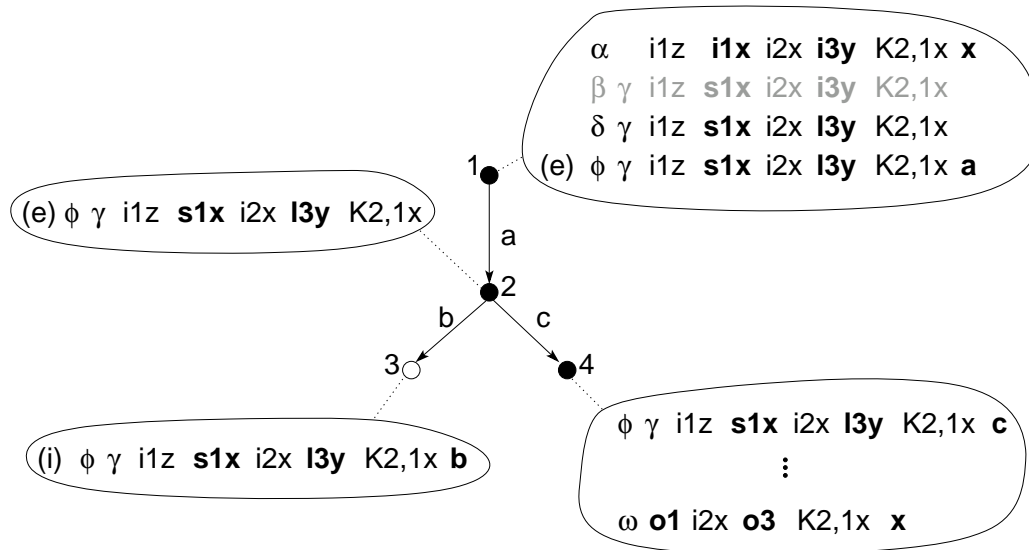


Abbildung 5.9: Wissen $\Pi_3 \oplus U1$

Die Beispiele haben gezeigt, daß sich Bedienbarkeit unter dem Einfluß transaktionaler Eigenschaften verändern kann. Für eine gegebene Umgebung ist der Nachweis exemplarisch gezeigt worden. Aufgrund der Gültigkeit des Theorems 2.1 kann der Nachweis, ob es eine Strategie für ein taWFM gibt, mit Hilfe des Algorithmus aus [Sch04] erbracht werden.

Durch den Nachweis der Bedienbarkeit eines taWFM wird gleichzeitig die Einhaltung einer transaktionalen Eigenschaft sichergestellt.

6 Zusammenfassung und Ausblick

Wir haben uns in der vorliegenden Arbeit mit zwei Aspekten von Geschäftsprozessen beschäftigt, der Bedienbarkeit und der Modellierung und Analyse von transaktionalem Verhalten eines Prozesses. Dabei verfolgten wir zwei Ziele: Das erste Ziel bestand darin, die auf Petrinetzen basierenden Workflow-Module, um transaktionales Verhalten zu erweitern. Die Erweiterung sollte dabei so gestaltet sein, daß das zweite Ziel, die Analyse der Bedienbarkeit und der Nachweis von korrektem transaktionalem Verhalten, durch die Anwendung der Bedienbarkeitsanalyse auf ein erweitertes Workflow-Modul gleichzeitig gelingt. Mit den Ergebnissen dieser Arbeit sind beide Ziele erreicht worden.

6.1 Modellierung

Wir haben in dieser Arbeit eine Möglichkeit geschaffen, Workflow-Module um transaktionales Verhalten zu erweitern. Dafür wurde die Modellierung von Transaktionen in Workflow-Modulen ermöglicht (gefärbtes Workflow-Modul). Ein Ablauf eines gefärbten Workflow-Moduls erzeugt einen Schedule. Ob dieser Schedule schön ist, hängt davon ab, ob er eine geforderte transaktionale Eigenschaft besitzt. In dieser Arbeit ist ein Schedule schön, wenn er konfliktserialisierbar und strikt oder rigoros ist. Damit ein gefärbtes Workflow-Modul nur Schedules erzeugt, die schön sind, müssen die Datenzugriffe des Moduls synchronisiert werden. Für die Synchronisation haben wir Petrinetze definiert, mit denen transaktionale Eigenschaften modelliert werden - die Eigenschaftsnetze. Mit Eigenschaftsnetzen werden für eine gegebene Menge von Transaktionen, sowohl die Transaktionen als auch die Synchronisation ihrer Datenzugriffe modelliert. Wir haben für die hier geforderten Eigenschaften die Konstruktionsvorschriften der Eigenschaftsnetze angegeben und deren Gültigkeit gezeigt. Das heißt, jeder Ablauf in einen Endzustand eines Eigenschaftsnetzes besitzt die geforderte transaktionale Eigenschaft. Die Gültigkeit einer transaktionalen Eigenschaft in einem erzeugten Schedule kann an Zuständen eines Eigenschaftsnetzes abgelesen werden. Ein Eigenschaftsnetz wird mit einem kompatiblen gefärbten Workflow-Modul zu einem transaktionalen Workflow-Modul komponiert. Das modellierte transaktionale Verhalten eines Eigenschaftsnetzes wird in einem taWFM ausgenutzt, um die Datenzugriffe des taWFM zu synchronisieren. Damit ist es uns gelungen, transaktionale Eigenschaften als internes Verhalten eines Moduls zu modellieren.

6.2 Analyse

Die Analyse der Bedienbarkeit basiert darauf, ein komponiertes System aus einer Umgebung und einem Workflow-Modul zu untersuchen. Dabei geht es insbesondere um die Untersuchung erreichbarer Zustände eines Moduls. So ist ein Workflow-Modul bedienbar, wenn aus jedem erreichbaren Zustand eines komponierten Systems ein Zustand erreichbar ist, in dem das Workflow-Modul selbst in seiner Endmarkierung ist. Für taWFM ist es gelungen, Zustände analog zu Workflow-Modulen zu klassifizieren. Jedes taWFM besitzt damit eine Anfangsmarkierung und eine Menge von Endmarkierungen. Jeder Ablauf in einem taWFM, der in eine solchen Endmarkierung führt, erzeugt einen Schedule, der die geforderte transaktionale Eigenschaft besitzt. Damit konnten wir die Analyse der Bedienbarkeit für taWFM übernehmen. Wir haben gezeigt, wie sich transaktionales Verhalten auf die Bedienbarkeit eines Moduls auswirkt. Es gibt bedienbare Workflow-Module, die als taWFM nicht bedienbar sind oder in denen nur noch ein Teil der Funktionalität genutzt werden kann. Wir konnten ebenfalls zeigen, daß es nicht bedienbare Workflow-Module gibt, die als taWFM bedienbar sind.

6.3 Ausblick

Aus dieser Arbeit ergeben sich weitere mögliche Forschungsthemen auf die wir jetzt kurz eingehen werden.

Verteilte Bedienbarkeit In dieser Arbeit wurde die Bedienbarkeit eines taWFM mit genau einer Umgebung betrachtet. In [Sch04] ist bereits ein Ansatz vorgestellt worden, in dem die Umgebung aus mehreren voneinander unabhängigen Partnern zusammengesetzt ist. Unter dem Begriff der verteilten Bedienbarkeit kann untersucht werden, ob mehrere unabhängige Partner ein Workflow-Modul bedienen. Es ist zu vermuten, daß dieser Ansatz auf taWFM übertragbar ist. Damit kann dann der Einfluß von Transaktionskonzepten auf verteilte Bedienbarkeit untersucht werden.

Zyklische Workflow-Module Die hier vorgestellten Ergebnisse beruhen auf azyklischen Workflow-Modulen. An der Analyse der Bedienbarkeit für zyklische Workflow-Module wird bereits an unserem Lehrstuhl gearbeitet. Ein zyklisches (gefärbtes) Workflow-Modul in einem taWFM kann bedeuten, daß Transaktionen komplett oder teilweise wiederholt werden. In einem nächsten Schritt gilt es, Eigenschaftsnetze diesbezüglich anzupassen und zyklische taWFM zu analysieren.

Transaktionale Eigenschaften Der Fall der zyklischen Workflow-Module zeigt, daß es Anwendungsfälle gibt, in denen die in dieser Arbeit modellierten transaktionalen

Eigenschaften zu rigide sind. In weiteren Arbeiten sollten deshalb taWFM für weitere Transaktionskonzepte modelliert und analysiert werden. Insbesondere Transaktionskonzepte mit Fehlerbehandlung und Kompensationsmöglichkeiten sind im Umfeld von Geschäftsprozessen sehr wichtig. Eine Analyse von taWFM, die Kompensation und Fehlerbehandlung unterstützen, hätte große praktische Relevanz.

Strukturanalyse und Reduktion In [Mar03] werden Aussagen zur Bedienbarkeit eines Workflow-Moduls, auf der Basis der Analyse der Struktur des Moduls, getroffen. Martens benennt Strukturen, sogenannte *Anti-Pattern*, die auf eine fehlerhafte Modellierung hindeuten. Analog dazu existieren ähnliche Muster im Kontext der Transaktionsverarbeitung auch in transaktionalen Workflow-Modulen. So macht z.B. eine Transaktion, die eine Aktion in einem Pfad und eine andere Aktion in einem alternativen Pfad besitzt, keinen Sinn, da die gesamte Transaktion nie vollständig ausgeführt werden kann. Damit findet aber keine Aktion dieser Transaktion jemals statt. Es ist zu vermuten, daß es eine Reihe von Mustern in taWFM gibt, die auf eine fehlerhafte Modellierung hindeuten.

Die Konstruktion eines vollständigen Eigenschaftsnetzes ist aufwendig. Nicht jedes Netzelement in einem Eigenschaftsnetz ist jedoch notwendig, um transaktionale Eigenschaften sicherzustellen. Im Beispiel 5.3 haben wir z.B. für das Modul \mathbf{gM}_3 , das vollständige Eigenschaftsnetz E_3 modelliert. Bei genauerer Betrachtung des taWFM Π_3 können wir allerdings feststellen, daß die Netzelemente zwischen $i_{3,y}$ und o_3 nicht benötigt werden. Die Strukturanalyse könnte demnach angewandt werden, um Modellierungsfehler aufzudecken und die erzeugten Netze klein zu halten.

Literaturverzeichnis

- [AAEA⁺96] Gustavo Alonso, Divyakant Agrawal, Amr El El-Abbadi, Mohan Kamath, Roger Günthör, and C. Mohan. Advanced transaction models in workflow contexts. In *Proceedings of the 12th International Conference on Data Engineering (ICDE '96)*, pages 574–583, Washington - Brussels - Tokyo, February 1996. IEEE Computer Society.
- [Aal96a] W.M.P. van der Aalst. Structural Characterizations of Sound Workflow Nets. Computing Science Reports 96/23, Eindhoven University of Technology, Eindhoven, 1996.
- [Aal96b] W.M.P. van der Aalst. Three Good reasons for Using a Petri-net-based Workflow Management System. In S. Navathe and T. Wakayama, editors, *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, pages 179–201, Cambridge, Massachusetts, 1996.
- [Aal98] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
- [Deh03] Juliane Dehnert. *A Methodology for Workflow Modeling - From business process modeling towards sound workflow specification*. Dissertation, Technische Universität Berlin, 2003.
- [EL96] Johann Eder and Walter Liebhart. Workflow recovery. In *Conference on Cooperative Information Systems*, pages 124–134, 1996.
- [Fre04] Carsten Frenkler. BPEL-Boxen. Ein Modell zur Integration von Transaktionskonzepten in Geschäftsprozesse mit Petrinetzen. Studienarbeit, Humboldt-Universität zu Berlin, 2004.
- [GR94] Jim Gray and Andreas Reuter. *Transaction processing : concepts and techniques*. Morgan Kaufmann, San Mateo, Calif., 1994.
- [JBS99] Stefan Jablonski, Markus Böhm, and Wolfgang Schulze. *Workflow-Management: Entwicklung von Anwendungen und Systemen*. dpunkt-Verlag für digitale Technologie GmbH, Heidelberg, 1999.

- [Kin01] Ekkart Kindler. *Systematische Spezifikation und Verifikation von Konsistenzprotokollen*. Habilitationsschrift, Mathematisch-Naturwissenschaftliche Fakultät, Humboldt-Universität zu Berlin, 2001.
- [Kre01] Heather Kreger. *Web Services Conceptual Architecture (WSCA)*. <http://www.ibm.com/software/solutions/webservices/pdf/WSCA.pdf>, May 2001.
- [Ley01] Frank Leymann. *Web Services Flow Language (WSFL 1.0)*. <http://www.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>, May 2001.
- [Mar03] Axel Martens. *Verteilte Geschäftsprozesse – Modellierung und Verifikation mit Hilfe von Web Services*. Phd thesis, Humboldt-Universität zu Berlin, 2003.
- [MS05] Peter Massuthe and Karsten Schmidt. Operating Guidelines - an Alternative to Public View. Technischer Bericht 189, Humboldt-Universität zu Berlin, 2005.
- [Obe96] Andreas Oberweis. *Modellierung und Ausführung von Workflows mit Petri-Netzen*. B.G. Teubner Verlagsgesellschaft, Leipzig, 1996.
- [OMG03] OMG. Unified Modelling Language (UML). Spezifikation 1.5, Objekt Management Group, 2003.
- [Peu01] Sibylle Peuker. *Halbordnungsbasierte Verfeinerung zur Verifikation verteilter Algorithmen*. Dissertation, Humboldt-Universität zu Berlin, 2001.
- [Rei82] Wolfgang Reisig. *Petrinetze - Eine Einführung*. Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, 1982.
- [Rei85] W. Reisig. *Petri Nets*. Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, eacts monographs on theoretical computer science edition, 1985.
- [Rei98] W. Reisig. *Elements of Distributed Algorithms: Modeling and Analysis with Petri Nets*. Springer-Verlag, 1998.
- [SABS02] Heiko Schuldt, Gustavo Alonso, Catriel Beerl, and Hans-Jorg Schek. Atomicity and isolation for transactional processes. *Database Systems*, 27(1):63–116, 2002.
- [SAS99] Heiko Schuldt, Gustavo Alonso, and Hans-Jörg Schek. Concurrency control and recovery in transactional process management. In *Proceedings of the Eighteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '99)*, pages 316–326, New York, May 1999. Association for Computing Machinery.

- [Sch91] August-Wilhelm Scheer. *ARIS, Modellierungsmethoden, Metamodelle, Anwendungen*. Springer-Verlag, Berlin, Heidelberg, New York, Tokyo, 1991.
- [Sch04] Karsten Schmidt. Distributed usability of web services. In *11th Workshop on Algorithms and Tools for Petri Nets (AWPN '04)*, pages 19–24, 2004.
- [Sta90] Peter H. Starke. *Analyse von Petri-Netz-Modellen*. B.G. Teubner-Verlag, Stuttgart, 1990.
- [Sta04] Christian Stahl. Transformation von BPEL4WS in Petrinetze. Diplomarbeit, Humboldt-Universität zu Berlin, Institut für Informatik, April 2004.
- [VGH93] G. Vossen and M. Gross-Hardt. *Grundlagen der Transaktionsverarbeitung*. Addison-Wesley, Bonn, 1993.
- [Wei04] Daniela Weinberg. Analyse der Bedienbarkeit. Diplomarbeit, Humboldt-Universität zu Berlin, Institut für Informatik, Oktober 2004.
- [WWV⁺97] Michael Weber, Rolf Walter, Hagen Völzer, Tobias Vesper, Wolfgang Reisig, Sibylle Peuker, Ekkard Kindler, Jörn Freiheit, and Jörg Desel. DAWN: Petrinetzmodelle zur Verifikation Verteilter Algorithmen. Informatik-Bericht 88, Humboldt-Universität zu Berlin, Institut für Informatik, Unter den Linden 6, D-10099 Berlin, December 1997.

Erklärung

Hiermit erkläre ich, die vorliegende Diplomarbeit „Modellierung und Analyse transaktionaler Geschäftsprozesse“ selbständig und ohne fremde Hilfe verfasst zu haben und nur die angegebenen Hilfsmittel verwendet zu haben.

Berlin, den 1. Juli 2005