Dienstbeschreibungsnetze als Modellierungstechnik für dienstbasierte Geschäftsprozesse

Dissertation

zur Erlangung des akademischen Grades

Dr. rer. nat.

an der Fakultät für Mathematik, Informatik und Naturwissenschaften
der Universität Hamburg
eingereicht beim Department Informatik von

Jan F. Ortmann

aus Bremen

Februar 2010

Betreuer: Prof. Dr. Rüdiger Valk

und Dr. Daniel Moldt

Genehmigt von der MIN-Fakultät, Department Informatik, der Universität Hamburg
auf Antrag von
Dr. Daniel Moldt,
Prof. Dr. Rüdiger Valk,
Prof. Dr. Lars M. Kristensen und
Prof. Dr. Winfried Lamersdorf

Hamburg, den 18. Juni 2010

Prof. Dr. H. Oberquelle
Leiter Department Informatik

Für meine Eltern

Danksagung

Ich danke allen, die mich beim Verfassen dieser Arbeit unterstützt haben. Insbesondere gilt mein Dank meinen Betreuern Dr. Daniel Moldt und Prof. Dr. Rüdiger Valk für die Vielzahl spannender Diskussionen und für die daraus resultierenden Denkanstöße sowie für die Betreuung und dafür, dass ich diese Arbeit am Arbeitsbereich TGI schreiben konnte. Ich danke des Weiteren den beiden genannten Betreuern und Herrn Prof. Dr. Lars M. Kristensen für die Begutachtung der Arbeit.

Neben meinen Betreuern danke ich Herrn Dr. Michael Köhler-Bußmeier für seine Unterstützung bei der Erstellung der Arbeit und für seine Bereitschaft, Teile dieser Arbeit Korrektur zu lesen. Des Weiteren danke ich Frank Heitmann und Georg Zetzsche dafür, dass sie ebenfalls Teile dieser Arbeit Korrektur gelesen haben, und für die daraus resultierenden hilfreichen Anregungen. Auch danke ich allen Mitgliedern des Arbeitsbereichs TGI des Departments Informatik für die vielen anregenden Gespräche, deren Ergebnisse vielfach in diese Arbeit eingeflossen sind. Speziell bedanke ich mich bei meinen zuvor noch nicht erwähnten ehemaligen Kollegen Sven Offermann, Dr. Christine Reese, Dr. Michael Duvigneau und Dr. Lawrence Cabac für die Zusammenarbeit bei verschiedenen Veröffentlichungen und für die Kooperation am Arbeitsbereich TGI.

Für ihre kontinuierliche seelische und moralische Unterstützung danke ich vor allem meinen Eltern und meiner Schwester, die stets für mich da waren. Nicht zuletzt danke ich all meinen Freunden, speziell Kay Fiolka und Michael Looft, für ihre Unterstützung während des Erstellens dieser Arbeit.

Zusammenfassung

Die Modellierung komplexer dienstbasierter Systeme gewinnt mit zunehmender Verbreitung solcher Systeme stark an Bedeutung. Gerade die Komplexität solcher Systeme und die Vielzahl an der Entwicklung beteiligter Partner machen eine präzise Modellierung notwendig. Ein abstraktes Modell, das ein technisches System zwar konzeptionell abbildet, dieses aber nicht technisch umsetzt, birgt die Gefahr, nicht der technischen Implementation zu entsprechen oder schnell zu veralten.

Die Forderung nach einem präzisen Ausdrucksmittel auf der einen Seite und nach einem ausführbaren Modell auf der anderen Seite führt zwangsläufig zu einer formalen Modellierungstechnik. Diesem Umstand wird noch zusätzlich Nachdruck dadurch verliehen, dass Werkzeuge in der Lage sein sollten festzustellen, ob ein Modell grob fehlerhaft ist. Petrinetze und speziell Workflownetze besitzen bereits eine große Verbreitung bei der Modellierung von Workflows und Abläufen in Unternehmen. Da dienstbasierte Geschäftsprozesse ebenfalls solche Abläufe abbilden, liegt es nahe, auf den etablierten Konzepten der Workflownetze aufbauend einen Formalismus zur Modellierung dienstbasierter Systeme zur Umsetzung von Geschäftsprozessen zu erstellen. Dies erfolgt im Rahmen dieser Arbeit durch die eingeführten Dienstbeschreibungsnetze, die die bei der Modellierung von Workflows eingesetzten Workflownetze um die Möglichkeit erweitern, Daten und ihre Manipulation im Netz zu modellieren, so dass hierdurch das Verhalten von Diensten in einer realen Umwelt dargestellt werden kann.

Die Darstellung von Daten und Nebenläufigkeit in Abläufen führt häufig dazu, dass die Analysierbarkeit der Abläufe nicht mehr gewährleistet ist. Aus diesem Grund wird die Ausdrucksmächtigkeit von Dienstbeschreibungsnetzen auf elementare Dienstbeschreibungsnetze mit Free-Choice-Eigenschaft eingeschränkt, die einen Kompromiss zwischen Modellierungsmächtigkeit und Analysierbarkeit darstellen. Für diese Netzklasse lässt sich dann zeigen, dass sie beschränkt ist und dass die so modellierten Dienste bei einem Aufruf stets mit einer Antwort terminieren. Eine weitergehende Analyse erfordert dann jedoch die Betrachtung der Veränderungen auf den Daten. Selbst für Dienstbeschreibungsnetze, in denen jede mögliche Schaltfolge endlich ist, führen die hieraus resultierenden Fragestellung im Allgemeinen schnell zu sehr großen und praktisch nicht mehr beherrschbaren Zustandsräumen. Um diese Zustandsräume zu reduzieren, werden elementare Dienstbeschreibungsnetze sequentialisiert, was einer Elimination der Nebenläufigkeit entspricht. Der resultierende Zustandsraum ist dabei häufig deutlich kleiner.

Die praktische Verwendung von Dienstbeschreibungsnetzen rundet deren Betrachtung in dieser Arbeit ab. Hierbei wird der Frage nachgegangen, wie Netze zur Modellierung von dienstbasierten Systemen eingesetzt werden können und welche Ergebnisse sich hieraus ergeben. Unter Bezugnahme auf bestehende Modellierungsansätze und Vorgehensweisen wird die Verwendung von Dienstbeschreibungsnetzen diskutiert.

Abstract

Modeling complex service-based systems is becoming increasingly important as those system are becoming widely used. The complexity of such systems together with the variety of partners involved make a precise modelling necessary. An abstract model, however, that reflects a technical system conceptually yet lacks the potential to directly implement the system, bears the risk not to reflect the technical implementation or to become quickly obsolete.

The demand for precise means of expression on the one hand and an executable model on the other hand necessarily leads to a formal modelling technique. This is emphasised by the fact that tools should be able to detect major flaws in a model. Petri nets in general and Workflow nets in particular are widely used for modelling workflows und processes in organisations. As service-based business processes represent the same kind of processes, it is a logical consequence to base a new formalism for the modelling of service-based systems on the well estabilshed concepts of Workflow nets. This is achieved in the course of this thesis by the introduction of service description nets. They extend the expressiveness of Workflow nets used for modelling workflows by means to model data and its manipulation within the net allowing for the direct representation of real world service behavior by the net.

Allowing for the representation of data together with the possibility to model concurrency often leads to reduced analyzability of processes. For this reason the expressiveness of ordinary service description nets is reduced to elementary service description nets with the Free-Choice property. They represent a compromise between the expressiveness of the modelling technique und its analyzability. For this class of nets it can be shown that it is bounded and that each call to a service modelled by such a net returns eventually with a response. A further analysis, however, requires an examination of the change that occurs on the data. Even for service description nets where each firing sequence is finite the resulting questions will most likely lead to state spaces that are very large and often no longer practically manageable. To reduce these state spaces elementary service description nets are sequentialized resulting in an elimination of their concurrency. The resulting state space is in most cases considerably smaller.

The practical use of service description nets completes the treatment of service description nets in this thesis. Here, it will be examined how nets can be used to model service-based systems and which findings may result from this. In reference to existing approaches to model service-based systems, the use of service description nets will be studied.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Organisationen werden in zunehmendem Maße aus prozessorientierter Sicht betrachtet. Neben einer besseren Analyse der Dynamik innerhalb der Organisation bieten sich insbesondere aus betriebswirtschaftlicher Sicht zahlreiche Vorteile hierdurch, wie beispielsweise von Ahlrichs und Knuppertz (2006) dargestellt wird. Für die Informationstechnologie bedeutet dies jedoch, dass Organisationen dynamischer werden, was in einer entsprechenden Konzeption reflektiert werden muss. Den derzeit verbreitetsten Ansatz, dieser Dynamik und Komplexität Herr zu werden, stellen dienstorientierte Architekturen dar, die sich in den vergangenen Jahren zu einem wesentlichen Stützpfeiler der Implementation von Geschäftsprozessen in Unternehmen entwickelt haben. Angetreten, um die Komplexität organisationaler und interorganisationaler Prozesse besser strukturieren und warten zu können, haben diese Architekturen in einer Vielzahl von Unternehmen Einzug erhalten. Auch wenn Konsens über den grundlegenden Aufbau einer dienstorientierten Architektur besteht, gibt es verschiedene Vorgehensmodelle und Herangehensweisen zur Umsetzung einer solchen.

Die vorliegende Arbeit stellt einen graphischen Formalismus zur Modellierung dienstbasierter Systeme vor, der auf höheren Petrinetzen basiert. Diese erlauben es, Daten und ihre Veränderung direkt im Netz zu beschreiben, so dass sich das Modell zu einem ausführbaren System verfeinern lässt.

Im weiteren Verlauf dieses Kapitels wird in Abschnitt 1.1 die vorliegende Arbeit thematisch eingeordnet. Abschnitt 1.2 beschreibt die sich in diesem Zusammenhang ergebende Fragestellung und erläutert die Ausgangsthesen sowie die Zielstellung der Arbeit, aus der sich dann das in Abschnitt 1.3 geschilderte Vorgehen zu dessen Erreichung ergibt. Abschnitt 1.4 beschreibt verwandte Forschungsfragen bevor Abschnitt 1.5 abschließend den Aufbau der Arbeit erläutert.

1.1 Motivation und Einordnung der Arbeit

In dienstorientierten Architekturen (Service Oriented Architectures – SOAs) kapseln Dienste grob granular Funktionalität und stellen diese Funktionalität verschiedenen Dienstnutzern bereit, wobei organisationale Rahmenbedingungen wie etwa die Zugriffsrechte des Aufrufers durch den Dienst überprüft werden. Die Modellierung dienstorientierter Architekturen besitzt verschiedene Dimensionen, deren wichtigste die prozessorientierte Modellierung der Dienstkoordination ist. Dieser

starke Fokus auf Abläufen legt es nahe, einen etablierten Formalismus zur Prozessmodellierung auch für die Modellierung von Abläufen im Dienstkontext zu verwenden. Viele der bekannten Notationen zur Darstellung dienstbasierter Systeme wie die Business Process Modelling Notation (BPMN) (vgl. BPMN, 2008), die ereignisgesteuerten Prozessketten (EPK) (vgl. Keller u. a., 1992; Huth und Wieland, 2008) und die UML-Aktivitätsdiagramme (UML-AD) (Booch u. a., 2006; Dumas und ter Hofstede, 2001) orientieren sich daher an bestehenden Formalismen wie Petrinetzen und Kalkülen.

Insbesondere Petrinetze bringen zahlreiche Vorteile bei der Modellierung von Abläufen mit sich. Hierzu zählen sowohl eine fundierte formale Basis als auch die explizite Darstellung von Zuständen, wodurch die Einbindung von Daten und Datentypen ermöglicht wird, wie dies etwa in den von Jensen (1981, 1994) vorgestellten gefärbten Petrinetzen oder in den algebraischen Petrinetzen von Reisig (1991a) erfolgt ist. Im praktischen Einsatz hat sich gezeigt, dass sich mit Hilfe von Petrinetzen sowohl Abläufe in Unternehmen, wie von van der Aalst und van Hee (2002) beschrieben, als auch komplexe agentenbasierte Systeme, wie etwa von Rölke (2004) und Cabac u. a. (2008) dargestellt, modellieren und ausführen lassen. Der Entwurf komplexer agentenbasierter Systeme mittels der von Kummer (2002) vorgestellten Java-Referenznetze im Paose-Ansatz (Moldt, 2006; Cabac, 2010) stellt zudem eine Nähe zu modellgetriebenen Ansätzen der Softwareentwicklung (vgl. Stahl u. a., 2007) dar. Dies ist eine weitere Motivation für den Einsatz petrinetzbasierter Techniken, da Modelle hierdurch direkt ausführbar werden und die vielfach vorhandene Lücke zwischen Modell und Implementation geschlossen werden kann. Dies ist so mit bestehenden Notationen wie der BPMN, den EPKs oder den UML-Aktivitätsdiagrammen nicht möglich, was die Verwendung von Petrinetzen nahelegt.

Um die Ausführbarkeit eines Modells zu gewährleisten, muss dieses Modell neben der Ablaufbeschreibung auch die Beschreibung von Daten und die Beschreibung ihrer Veränderung umfassen. Andernfalls wäre das Verhalten des Modells unabhängig von den Daten und würde die häufig datengetriebenen Geschäftsprozesse und Workflows in Unternehmen nur unzureichend reflektieren. Der erfolgreiche Einsatz der von Kummer (2002) vorgestellten Java-Referenznetze zur Modellierung von Agentensystemen zeigt die Möglichkeiten der Ausführbarkeit petrinetzbasierter Modelle auf. Aufgrund der Verwendung der Programmiersprache Java als Anschriftensprache ist die Analysierbarkeit der Modelle jedoch nur eingeschränkt gegeben. Analyse und Ausführbarkeit petrinetzbasierter Modelle zu gewährleisten ist Gegenstand der im Rahmen dieser Arbeit vorgestellten Dienstbeschreibungsnetze als Modellierungstechnik für dienstbasierte Geschäftsprozesse.

1.2 Fragestellung und Ziel der Arbeit

Vor dem Hintergrund ausführbarer Modelle ist die zentrale Fragestellung, wann Abläufe vernünftig modelliert sind. Dieser Fragestellung wird bei der reinen Ablaufmodellierung durch die Korrektheit von Workflownetzen nachgegangen. Bei der Modellierung von Diensten stellt sich zudem die Frage, ob das Verhalten zweier Dienste miteinander kompatibel ist. Dienste werden hierbei als offene Workflows (vgl. Schmidt, 2005) betrachtet, die um weitere Dienste ergänzt werden.

In dieser Arbeit sollen neben der reinen Ablaufbetrachtung auch Daten und Funktionalität betrachtet werden. Dies wirft eine Reihe neuer Fragestellungen auf. Einmal stellt sich die Frage, wie ein Netzformalismus aussehen kann, der Dienste anhand von Netzen beschreibt. Ein solcher neuer Formalismus ist zudem nur sinnvoll, wenn er Vorteile gegenüber bestehenden Notationen bietet, so dass sich Fragen bezüglich der Analysierbarkeit dieses Netzformalismus anschließen. Diese Fragen werden im Rahmen dieser Arbeit beleuchtet. Im Vordergrund stehen dabei die Modellierung der Abläufe und die Modellierung der Funktionalität. Letztere wird auch von Papazoglou u. a. (2006) als entscheidend für die Komposition und Modellierung von Diensten angesehen. Die Komposition von Diensten wird hierbei als Top-Down-Prozess verstanden. In vielen praktischen Szenarien existieren zwar bereits Altsysteme, die durch Dienste gekapselt werden, jedoch erfolgt ihre Komposition und Koordination dann meist wiederum anhand eines Top-Down-Vorgehens.

Betrachtet man Workflows in Organisationen so treten sehr häufig Typänderungen bei den modellierten Datenobjekten auf. Auch werden verschiedene Zustände im Lebenszyklus eines Objektes häufig durch spezielle Typen modelliert. So kann eine Person zu einem Kunden werden, ohne dass es sich dabei um ein neues Objekt handelt. Auch kann ein Auftrag verschiedene Zustände wie *bearbeitet*, *ausgeliefert* oder *bezahlt* besitzen, wobei das Auftragsobjekt stets dasselbe ist. Ein intuitives Modell sollte diese Art von Typänderungen unterstützen und darüber hinaus eine Analyse zulassen, ob die modellierten Abläufe mit den modellierten Objektzuständen und ihren Abhängigkeiten übereinstimmen.

Zentrale Fragestellungen dieser Arbeit

Aus den Überlegungen zur Modellierung dienstbasierter Geschäftsprozesse ergeben sich drei zentrale Fragestellungen, die im Rahmen dieser Arbeit betrachtet werden. Einmal muss entschieden werden, wie ein netzbasierter Formalismus zur Darstellung von Dienstverhalten aufgebaut ist, und wie sich ein so modelliertes Modell ausführen lässt. Interessant ist vor diesem Hintergrund auch, wie sich die Verwendung dynamischer Konzepte zur intuitiven Modellierung umsetzen lässt. Dies erfolgt durch die Einführung von Dienstbeschreibungsnetzen und durch die Angabe der Schaltregel dieser Netzklasse.

Neben der Ausführbarkeit ist die Einführung eines formalen Modells stark durch

die Analysierbarkeit des Modells motiviert. Folglich stellt sich die Frage, welche Eigenschaften des Modells überprüft werden können und wie dies geschehen kann. In Anlehnung an die Korrektheitskriterien von Workflownetzen werden daher Korrektheitskriterien für Dienstbeschreibungsnetze definiert, die dann untersucht werden. Da sich herausstellt, dass viele Fragestellungen für Dienstbeschreibungsnetze im Allgemeinen unentscheidbar sind, wird eine Subklasse von Dienstbeschreibungsnetzen eingeführt, die die Entscheidbarkeit zentraler Fragen erlaubt.

Abschließend stellt sich die Frage, wie die eingeführte Netzklasse praktisch verwendet werden kann und wie sich diese Netzklasse in die bestehenden Modellierungstechniken einbettet. Somit ist es sinnvoll bestehende Vorgehensmodelle zu betrachten und die praktische Verwendung von Dienstbeschreibungsnetzen zu skizzieren.

Ziel dieser Arbeit

Diese Arbeit hat sich zum Ziel gesetzt, eine Netzklasse zur Modellierung dienstbasierter Systeme vorzustellen, die ausführbar und analysierbar ist. Dienstbeschreibungsnetze bilden diese Netzklasse und ermöglichen die Modellierung und die Ausführbarkeit dienstbasierter Systeme. Die Analyse kann dann für eine Subklasse der Dienstbeschreibungsnetze erfolgen, die als elementare Dienstbeschreibungsnetze charakterisiert werden.

Umsetzung des Ziels dieser Arbeit

Ausgehend von den bestehenden Formalismen der algebraischen Petrinetze (vgl. Reisig, 1991b) auf der einen Seite und den abstrakten Zustandsmaschinen (vgl. Börger und Stärk, 2003; Gurevich, 1991) auf der anderen Seite wird der Netzformalismus der Dienstbeschreibungsnetze vorgestellt, der die Manipulation der Daten direkt im Netz darstellt. Durch die Verwendung von Petrinetzen können Aussagen aus dem Feld der Petrinetzforschung und speziell aus dem Feld der Workflownetzforschung auf die als Dienstbeschreibungsnetz modellierten Dienste übertragen werden. Aufbauend auf vorhandenen Ergebnissen lässt sich dann analysieren, ob Teile eines Netzes nebenläufig schalten können. Dies ist insbesondere im Zusammenhang mit Daten interessant, da hierdurch überprüft werden kann, ob verschiedene, bei derselben Eingabe mögliche Schaltfolgen von einem initialen zu einem finalen Zustand in einem Dienstbeschreibungsnetz zum selben Ergebnis führen. Dies stellt in der Regel eine wünschenswerte Eigenschaft für Dienste dar.

Zentrales Element bei der Beschreibung dienstbasierter Systeme ist neben dem Ablauf auch die Funktionalität. Eine Subklasse von Dienstbeschreibungsnetzen, die elementaren Dienstbeschreibungsnetze, können zur Analyse der Funktionalität so in Zustandsmaschinen überführt werden, dass der Zustandsraum beim Testen, beim Model Checking oder bei der symbolischen Analyse entsprechend kleiner ausfällt.

Wann ein Netz in diese Klasse von Netzen fällt und wie sich diese Klasse analysieren lässt wird in diesem Zusammenhang detaillierter beschrieben.

Durch die von den algebraischen Netzen übernommene Typisierung der Variablen und Stellen des Netzes wird die Modellierung korrekter Modelle bereits auf syntaktischer Ebene unterstützt. Gleichzeitig erlaubt die von Dienstbeschreibungsnetzen unterstützte dynamische Typisierung, die über die Modellierungsmöglichkeiten herkömmlicher algebraischer Netze hinausgeht, eine dynamische Ergänzung möglicher Attribute eines Fachobjektes im Netz. Neben einer intuitiveren Modellierung eröffnet die dynamische Typisierung zudem die Möglichkeit einer weitergehenden Analyse. So kann überprüft werden, ob ein modellierter Ablauf mit den Zuständen eines Objektes konform ist. Beispielsweise kann überprüft werden, dass in jedem Ablauf nur Bestellungen ausgeliefert werden, wenn sie zuvor bezahlt worden sind.

Der Fokus der Betrachtung liegt im Rahmen dieser Arbeit weniger auf der intuitiven graphischen Darstellung der Modelle, sondern vielmehr auf der Möglichkeit ihrer Ausführbarkeit und ihrer Analyse. Geeignete Vereinfachungen der Notation oder gar eine völlig neue intuitivere Notation, die auf Dienstbeschreibungsnetze abbildbar ist, werden daher im Rahmen dieser Arbeit nur sehr eingeschränkt thematisiert. Die Zielgruppe der Notation sind somit Modellierer mit theoretischem Hintergrund.

1.3 Einführung des Formalismus und Darstellung des Vorgehens

Um diese Arbeit besser motivieren zu können, soll hier der Formalismus der Dienstbeschreibungsnetze kurz informell eingeführt werden. Eine detailliertere Einführung findet sich dann in Kapitel 4.

Modellierungsansatz

Die Modellierung von Daten in Dienstbeschreibungsnetzen erfordert zunächst eine Analyse, was zu modellieren ist und welche Granularität das Modell besitzen soll. Die Arbeit orientiert sich dabei an typischen Szenarien in Organisationen, die durch Workflows modelliert werden sollen. Die notwendigen Operationen beinhalten einmal die typischen CRUD-Anweisungen (Create, Read, Update, Delete) datenbankzentrierter Systeme (vgl. Brandon, 2002), die vielfach auch in Client-Server-Anwendungen Verwendung finden (vgl. (Kriha und Schmitz, 2009, S. 266ff) oder (Thuraisingham, 1998, S. 426ff)). Des Weiteren werden häufig Typveränderungen vorgenommen, deren Modellierung ebenfalls durch den Formalismus abgedeckt sein sollte.

Abstrakte Zustandsmaschinen beschreiben die aktuelle Interpretation eines Operatorsymbols durch Lokationen, die den Wert des Operators bei Anwendung auf

eine Eingabe angeben. Zur Modellierung der Datenveränderung verwenden abstrakte Zustandsmaschinen dann Update-Anweisungen, die den Wert einer Lokation neu definieren. Das Lesen von Daten erfolgt durch die Auswertung von Termen. Diese Auswertung kann dann Variablen zugewiesen werden. Als Kontrollkonstrukte erlauben abstrakte Zustandsmaschinen Schleifen, Verzweigungsanweisungen und Sequenzen. Zudem besitzen sie die Möglichkeit Prozeduren aufzurufen (vgl. Börger und Stärk, 2003). Während die Kontrollkonstrukte im hier gewählten Ansatz graphisch durch Petrinetze dargestellt werden, werden die Update-Anweisungen und die Zuweisungsanweisungen der abstrakten Zustandsmaschinen in veränderter Form übernommen. Sie werden dabei um die Möglichkeit ergänzt, neue Objekte zu erzeugen und die Typisierung bestehender Objekte zu verändern.

Die Granularität des Modells wird durch den Einsatzkontext bestimmt. Da es sich bei der Modellierung von Diensten um eine eher grobgranulare Modellierung von Organisationen und ihren Prozessen handelt, ist es nicht sinnvoll, detaillierte Berechnungen im Modell unterzubringen. Zudem verhindert der Aufbau vieler detaillierter Berechnungen die Analyse, da die Verwendung von Schleifen die Termination im Allgemeinen unentscheidbar macht. Berechnungen anhand von Funktionen werden daher als gegebene Elemente der Infrastruktur angesehen, deren Verhalten separat getestet oder verifiziert werden muss. Dies erlaubt es, das Modell von Detaillimplementationen frei zu halten, ohne dass die Ausführbarkeit des Modells deswegen eingeschränkt wäre. Aus diesem Grund wird auch von der sehr detaillierten Darstellung von Verhalten, wie sie für die Verifikation von Programmen notwendig ist, abstrahiert. Stattdessen wird das Verhalten auf einer den Diensten angemessenen Ebene modelliert.

Dienstbeschreibungsnetze

Die sich aus diesen Überlegungen ergebende Netzklasse sind die Dienstbeschreibungsnetze. Ein Exemplar dieser Netzklasse ist in Abbildung 1.1 beispielhaft dargestellt. Das Netz in Abbildung 1.1 modelliert eine einfache Behandlung eines Auftragseingangs und abstrahiert von der Ausnahmebehandlung oder von Sonderfällen. Die Subworkflows des Bezahlens und des Auslieferns sind durch eigene Dienstbeschreibungsnetze modelliert. Durch die Ersetzung der Transitionen im Hauptworkflow durch die jeweiligen Subworkflows ergibt sich dann der ausführbare Gesamtworkflow, wobei die Shipment-Transition durch das Shipment-Netz und die Invoicing-Transition ebenfalls durch das entsprechende Invoicing-Netz ersetzt werden. In jedem der Netze sind die Hauptabläufe gelb unterlegt. Die Aktivitäten, die durch Dritte ausgeübt werden, sind hingegen blau unterlegt. Die Kommunikation erfolgt dabei über Stellen, die bei Subworkflows verschmolzen werden.

Bei Eingang eines Kaufauftrags (*PurchaseOrder (PO)*) wird die Rechnungslegung (*Invoicing*) und die Auslieferung (*Shipment*) des Auftrags veranlasst und abschließend wird der Auftrag als vollständig gekennzeichnet. Im Subworkflow *Shipment*

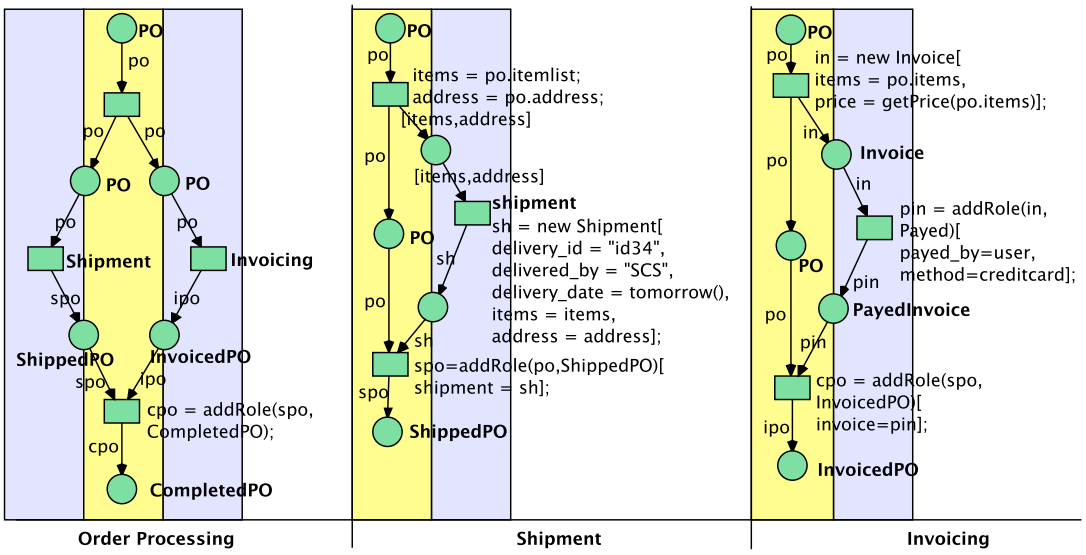


Abbildung 1.1: Das Dienstbeschreibungsnetz einer Auftragsannahme

wird die Liste der Bestellungen und die Adresse an eine spezielle Transition übergeben, die dann das Versenden veranlasst. Die Transition muss bei der Realisierung entsprechend ihrer Aufgabe implementiert werden. In diesem Fall wäre etwa denkbar, dass ein Mitarbeiter die bestellten Objekte aus dem Lager holt und dann an die Adresse versendet.

Die Rechnungslegung ist ebenfalls durch einen Subworkflow modelliert worden. In diesem Fall wird zunächst eine Rechnung erstellt, die dann durch den Kunden beglichen werden muss. In diesem Fall wäre es möglich, dass der Kunde die Rechnung direkt bezahlt, beispielsweise über die Nutzung einer Kreditkarte in einem Online-Versandhandel. Die Bezahlung ist dabei wiederum als Transition modelliert, die für die Ausführung implementiert werden muss.

Im dargestellten Netz treten drei Arten von Veränderungen der Objekte auf. Es werden über eine *new*-Anweisung neue Objekte erzeugt, es werden über *addRole*-Anweisungen die Typen bestehender Objekte verändert, und es werden Attribute initialisiert. Die Initialisierung von Attributen erfolgt über die Kurznotation in eckigen Klammern, die angibt, welche Attribute welchen Wert erhalten. Zudem werden Variablen über eine einfache Zuweisung Werte zugewiesen. Die Notation in Abbildung 1.1 stellt eine vereinfachte Notation dar, die eine bessere Lesbarkeit bietet. In der formalen Definition werden alle Änderungen über *assign*-Anweisungen für Zuweisungen und über *update*-Anweisungen für Attributveränderungen eingeleitet.

Der zugrundeliegende Formalismus dieser Notation besteht aus einer Algebra, die die Daten und die Datentypen repräsentiert, und aus einem Workflownetz, das den Ablauf darstellt. Zudem dienen die *update*- und *addRole*- beziehungsweise *re-*

moveRole-Anweisungen der Veränderung der Algebra im Netz. Anders als für abstrakte Zustandsmaschinen und algebraische Netze im Allgemeinen üblich, werden ordnungssortierte Algebren verwendet, die eine Ordnung auf den Sorten zulassen. Dies ermöglicht später eine direkte Abbildung der Konzepte einer Ontologie auf die Sorten der Algebra. In Kapitel 3 werden daher ordnungssortierte Algebren und Petrinetze eingeführt, wobei die bestehenden Ergebnisse aus dem Forschungsfeld der Petrinetze und der Workflownetze rekapituliert werden. Diese Ergebnisse werden später für die Analyse der Netze genutzt.

Analyse von Dienstbeschreibungsnetzen

Die Analyse der Netze wird einmal durch die Verhaltensanalyse und einmal durch die funktionale Analyse der Netze bestimmt. Während erstere das Verhalten des Dienstbeschreibungsnetzes im Zusammenspiel mit anderen Dienstbeschreibungsnetzen analysiert, widmet sich letztere der Funktionalität, die durch die Veränderungen auf den Daten zum Ausdruck kommt. Diese beiden Sichtweisen auf die Analyse lassen sich jedoch nicht vollständig voneinander trennen, da das Verhalten maßgeblich durch die Funktionalität mitbestimmt wird.

Ein zentrales Element der Verhaltensanalyse ist die Korrektheit eines Dienstbeschreibungsnetzes, die eng verwandt ist mit der Korrektheit seines zugrundeliegenden Workflownetzes. Die für Workflownetze definierte Korrektheit lässt sich auf Dienstbeschreibungsnetze ausweiten, indem ähnliche Kriterien unter Berücksichtigung der Daten gefordert werden, so dass stets gewährleistet ist, dass der modellierte Dienst in einen finalen Zustand übergeht. Zwar ist die Korrektheit des Workflownetzes, das einem Dienstbeschreibungsnetz zugrundeliegt, nicht immer ein notwendiges Kriterium für die Korrektheit des Dienstbeschreibungsnetzes, da durch die Betrachtung von Daten Schaltfolgen verhindert werden können, jedoch ist sie meist gewollt, da hierdurch das Modell intuitiver wird.

Hierüber hinaus lässt sich das Verhalten eines Dienstbeschreibungsnetzes im Zusammenspiel mit den verwendeten Objekten analysieren. Nimmt man wiederum die in Abbildung 1.1 modellierten Abläufe, so wäre es denkbar, dass für das PO-Objekt definiert wird, dass der Auslieferung stets die Zahlung vorausgehen soll. In diesem Fall wäre der Workflow auf der linken Seite von Abbildung 1.1 schlecht modelliert, da er eine nebenläufige Ausführung suggeriert, die jedoch praktisch niemals vorkommen kann, da das PO-Objekt zunächst in den Objektzustand *InvoicedPO* wechseln muss, bevor die Auslieferung beginnen kann. Darüber hinaus gibt es Fälle, in denen die Ausführung eines Workflows durch die möglichen Objektzustände gänzlich unterbunden wird. Dies sollte durch eine Analyse erkannt werden. Auch für Dienste können Reihenfolgen definiert sein, in denen sie aufgerufen werden müssen. Beispielsweise könnte der Ablauf in Abbildung 1.1 fehlerhaft sein, weil der Rechnungslegung keine Prüfung der Rechnung vorausgegangen ist. Die Einhaltung solcher Abhängigkeiten zwischen Diensten sollte ebenfalls analysierbar sein.

Die funktionale Analyse geht über die Modellierung von Objektzuständen hinaus. So ist es beispielsweise notwendig sicherzustellen, dass für jeden versendeten Auftrag auch eine Rechnung gestellt und bezahlt wurde, ohne dass es dabei zu einer Ausnahmesituation kam. Auch sollte sichergestellt sein, dass keine Aufträge verlorengehen. Dies lässt sich zum Teil durch eine symbolische Analyse überprüfen. Diese Analyse stößt jedoch bei unendlichen Datentypen und Schleifen zwangsweise an ihre Grenzen. Dennoch lässt sich durch eine Sequentialisierung des modellierten Netzes diese Art der Analyse auf einer Zustandsmaschine effizienter durchführen, da keine Nebenläufigkeit berücksichtigt werden muss.

1.4 Verwandte Forschungsfelder

Die im Rahmen dieser Arbeit vorgestellten Ansätze schneiden verschiedene Forschungsfelder an. Im Bereich der Formalismen zur Analyse von Abläufen sind die Bereiche der abstrakten Zustandsmaschinen von Börger und Stärk (2003) und der algebraischen Petrinetze von Reisig (1991b) direkt verwandt mit dem hier vorgestellten Ansatz der Dienstbeschreibungsnetze. Der Vorteil der hier dargestellten Herangehensweise gegenüber abstrakten Zustandsmaschinen liegt in der graphischen Notation, in der Möglichkeit, Nebenläufigkeit direkt in das Modell integrieren zu können, und in der vergröberten Darstellung von Funktionalität, die sich besser für dienstbasierte Systeme eignet. Zudem erlaubt die dynamische Änderung der Typisierung eine bessere Abbildung der Realität im Modell.

Gegenüber den algebraischen Netzen bieten Dienstbeschreibungsnetze die Vorteile einer kompakteren Notation und einer leichteren Übertragung eines realen Systems auf das Modell. Durch die Ausführbarkeit kann ein überprüftes Modell zudem auch als Implemention interpretiert werden. Elementare Dienstbeschreibungsnetze sind darüber hinaus effizienter analysierbar als algebraische Netze im Allgemeinen. Des Weiteren eröffnet die Verwendung dynamischer Konzepte in Dienstbeschreibungsnetzen die Möglichkeit intuitiverer Modelle.

Die dynamische Typänderung wäre zwar auch durch die Verwendung spezieller Programmiersprachen wie Smalltalk (vgl. Ingalls, 1981) oder Javascript (vgl. ISO 16262, 2002) möglich, jedoch bieten diese nicht die formalen Analysemöglichkeiten wie die Verwendung der Konzepte abstrakter Zustandsmaschinen. Auch gibt es Ansätze zur Darstellung verteilter abstrakter Zustandsmaschinen (DASMs) wie etwa von Glässer u. a. (2004). Diese besitzen jedoch einerseits nicht das formale Fundament im Bereich der Workflows wie Workflownetze, und ihnen fehlt andererseits eine Analysemöglichkeit der Nebenläufigkeit.

Es gibt eine Reihe weiterführender und angrenzender Themen, die im Rahmen dieser Arbeit nur angeschnitten, nicht aber vertieft werden. Die Nutzung von Farben in Workflownetzen wird beispielsweise auch von van der Aalst u. a. (2005), Russell und van der Aalst (2009), Jørgensen u. a. (2008) und Mans u. a. (2008,

2009) untersucht, wobei hier der Schwerpunkt auf der praktischen Nutzung von Workflownetzen mit Farben liegt. Eine formale Analyse dieser Netze findet hierbei weniger statt. Eine Übersicht über Netzformalismen, die mit dem Formalismus der Dienstbeschreibungsnetze verwandt sind, wird am Anfang von Kapitel 4 gegeben.

Des Weiteren existiert der umfangreiche Bereich der Ontologien (vgl. Staab und Studer, 2004), die in dieser Arbeit lediglich genutzt werden, ohne dass näher darauf eingegangen wird, was eine gute Ontologie ausmacht und wie verschiedene Ontologien zusammengeführt und angeglichen werden können (vgl. de Bruijn u. a., 2004) oder welche Logiken einer Ontologie zugrundeliegen sollten (vgl. de Bruijn und Heymans, 2008). Ontologien spielen neben der Beschreibung der Fachobjekte auch bei der Verarbeitung von Ereignissen durch Regeln und bei der Entwicklung der Konnektoren eine Rolle. Zusammen mit Regeln dienen sie dazu, zu definieren, wie auf bestimmte Ereignisse reagiert werden soll. Dies wird im Rahmen dieser Arbeit jedoch lediglich in Kapitel 7 kurz angerissen, da sowohl der Bereich der Ontologien als auch der Bereich der Ereignisverarbeitung (vgl. Mühl u. a., 2006) und die Verwendung von Regeln (vgl. Antoniou u. a., 2005) eigene Forschungsfelder darstellen.

Ein weiterer zentraler Baustein bei der Erstellung von Unternehmenssoftware ist die graphische Benutzungsoberfläche. Auch hier stellt der Entwurf einer solchen ein eigenes Forschungsfeld dar. Angefangen bei der zu wählenden Technologie über Ergonomie und Design bis hin zu Sicherheitsaspekten im verteilten Betrieb gibt es hier wiederum eine Reihe von Forschungsfragen, die im Rahmen dieser Arbeit nicht behandelt werden (vgl. Shneiderman und Plaisant, 2010). Vielmehr wird die Schnittstelle zum menschlichen Nutzer lediglich als weiterer Konnektor aufgefasst, dessen Implementation nicht näher diskutiert wird.

Weitestgehend verzichtet wird im Rahmen dieser Arbeit auf konkrete technische Umsetzungen eines Konzepts wie etwa Standards im Bereich der Webservices oder im Java-Bereich, da sich diese zu schnell ändern, als dass eine Diskussion in einer Arbeit sinnvoll erscheint, die die Umsetzung konzeptioneller Ansätze vorstellt[1]. Auf sie wird daher nur eingegangen, sofern sie der Erläuterung dienlich sind.

Konzeptionell schließen sich eine Reihe von Arbeiten an diese Arbeit an, die ebenfalls den dienstbasierten Entwurf komplexer Systeme betrachten. Hier ist insbesondere der aufgrund seiner praktischen Einbettung herausragende Fresco-Ansatz von Zirpins (2007) zu nennen. Zirpins (2007) stellt dort detailliert die Motivation für den Einsatz von Diensten dar, die so in dieser Arbeit nicht in entsprechender Detailltiefe erfolgen kann. Im Kontext von Fresco ist beispielsweise auch der Ansatz von Offermann (2003) zur Komposition von Diensten mit Referenznetzen entstanden, der das Vorgehen in dieser Arbeit mit motiviert hat (vgl. Moldt u. a.,

[1] Nachdem vor wenigen Jahren noch CORBA (Neubauer u. a., 2004) als Mittel der Wahl galt, hat sich in den letzten Jahren SOAP (SOAP, 2003) zunehmend zur Kommunikation im Bereich der Dienste durchgesetzt, welches im Moment zum Teil wieder durch einfachere REST-basierte Anwendungen (Richardson und Ruby, 2007) abgelöst wird. Zudem gibt es eine Reihe plattform- oder programmiersprachenabhängiger Ansätze, die sich ebenfalls regelmäßig ändern.

2004b, 2005). Ebenso verwandt ist der Einsatz von höheren Petrinetzen in verschiedenen angrenzenden Bereichen, wie bei der Steuerung von Workflowsystemen (Reese, 2009), bei der Darstellung einer Plugin-Architektur (Duvigneau, 2009) oder bei der Modellierung von Multiagentensystemen (vgl. Rölke, 2004; Cabac, 2010).

1.5 Aufbau und Gliederung

Im Anschluss an diese Einleitung werden in **Kapitel 2** zunächst die grundlegenden Konzepte von Geschäftsprozessen und dienstorientierten Architekturen dargestellt. In diesem Kapitel werden die drei wesentlichen Bestandteile der prozessorientierten und dienstbasierten Modellierung erläutert. Die drei Bestandteile sind zum einen die Prozesse in einer Organisation, es sind zum anderen die Dienste zur Bearbeitung von Aufgaben, und es ist als dritter Bestandteil die Repräsentation der Daten oder Fachobjekte im System und deren Veränderung. Im hier diskutierten Ansatz können Fachobjekte dabei nicht nur den Wert ihrer Attribute ändern, sondern auch zusätzliche Attribute annehmen, also ihren Typ zur Laufzeit ändern. Dies erleichtert die Modellierung in einer Vielzahl von prozessorientierten Anwendungen erheblich, da etwa Kunde und Person dasselbe Objekt sein können, es aber je nach Kontext unterschiedliche Sichten hierauf geben kann.

Die formalen Grundlagen der Arbeit werden in **Kapitel 3** dargestellt. Zunächst werden hier Algebren vorgestellt. Sie dienen der Beschreibung der Datentypen in Abläufen. Ihre Verwendung ist dabei ähnlich wie bei der Definition abstrakter Datentypen durch Algebren. Im Anschluss hieran folgt die Beschreibung von Petrinetzen, die zur Darstellung der Abläufe verwendet werden. Algebraische Petrinetze verbinden dabei die Darstellung von Datentypen durch Algebren mit der Darstellung von Abläufen durch Petrinetze. Die im Rahmen dieser Arbeit definierten Dienstbeschreibungsnetze sind spezielle algebraische Netze, die zudem Eigenschaften von Workflownetzen übernehmen. Diese werden ebenfalls an dieser Stelle vorgestellt. Abgeschlossen wird das Kapitel durch bekannte Eigenschaften von Petrinetzen, die im weiteren Verlauf der Arbeit zur Analyse der Dienstbeschreibungsnetze herangezogen werden.

Aufbauend auf den in Kapitel 3 dargestellten bestehenden Ergebnissen zu Petrinetzen und Algebren werden in **Kapitel 4** Dienstbeschreibungsnetze als grundlegendes Modellierungswerkzeug dieser Arbeit beschrieben. An dieser Stelle werden die Anweisungen der Dienstbeschreibungsnetze und ihr Schaltverhalten präzisiert. Dabei wird zunächst erläutert, wie die Zustandsveränderung anhand von Updates und Sorten-Updates erfolgt, und es wird im Anschluss hieran geschildert, wie diese Anweisungen in Netzen verwendet werden. Des Weiteren werden verschiedene Kriterien für sinnvoll modellierte Dienste für diese Netzklasse angegeben, die sich zum Teil an den Korrektheitskriterien von Workflownetzen orientieren. Zu diesen zentralen Eigenschaften von Dienstbeschreibungsnetzen gehört neben verschiedenen

Arten der Korrektheit auch die funktional nebenläufige Aktiviertheit von Transitionen. Letztere gibt an, dass bei zwei nebenläufig aktivierten Transitionen die Schaltreihenfolge für das Ergebnis des Feuerns beider Transitionen unerheblich ist.

Die allgemeine Klasse der Dienstbeschreibungsnetze ist jedoch zu ausdrucksstark, um wesentliche Eigenschaften der Netze wie die schwache Korrektheit und die funktional nebenläufige Aktiviertheit von Transition generell überprüfen zu können. Aus diesem Grund wird die Ausdrucksmächtigkeit von Dienstbeschreibungsnetzen durch elementare Dienstbeschreibungsnetze in **Kapitel 5** eingeschränkt. Sie reduzieren die Ausdrucksmächtigkeit von Dienstbeschreibungsnetzen, erlauben im Gegenzug jedoch die direkte Überprüfung der schwachen Korrektheit. Elementarität ist hierbei dadurch charakterisiert, dass sich ein Vielzahl von Eigenschaften anhand einer Analyse des dem Dienstbeschreibungsnetz zugrundeliegenden Netzes und anhand einer Betrachtung der Transitionsanschriften entscheiden lassen. Für Dienstbeschreibungsnetze mit der Free-Choice-Eigenschaft ist dann in Polynomialzeit entscheidbar, ob ein gegebenes Dienstbeschreibungsnetz elementar ist. Um dies analysieren zu können, werden Eigenschaften von Free-Choice-Netzen benötigt, die aufbauend auf den Ergebnissen von Kapitel 3 in Kapitel 5 ermittelt werden. Dies ermöglicht die abschließend in Kapitel 5 dargestellte Überprüfung der Elementarität eines gegebenen Free-Choice-Dienstbeschreibungsnetzes.

Aufbauend auf den Ergebnissen aus Kapitel 5 wird in **Kapitel 6** gezeigt, wie sich elementare Dienstbeschreibungsnetze sequentialisieren lassen. Dies bedeutet, dass das Verhalten des Dienstbeschreibungsnetzes auf eine funktional äquivalente Zustandsmaschine abgebildet wird, so dass sich eine Analyse der Funktionalität auf dieser Zustandsmaschine durchführen lässt. Hieran schließt sich eine weitere Möglichkeit der Überprüfung, ob ein Netz das gewünschte Verhalten umsetzt, an. Objekte erhalten hierzu ebenfalls Zustände, die dann mit dem modellierten Ablauf eines Dienstbeschreibungsnetzes verschmolzen werden können. Das resultierende Netz kann dann daraufhin untersucht werden, ob der modellierte Ablauf die modellierten Zustandsübergänge der Objekte verletzt. Ist dies der Fall, so ist eines der beiden Modelle nicht korrekt.

Kapitel 7 schildert den praktischen Einsatz von Dienstbeschreibungsnetzen. Es werden dazu zunächst mögliche Einsatzszenarien und Vorgangsmodelle diskutiert. Hieran schließt sich die Beschreibung einer petrinetzbasierten Dienstarchitektur an, die dann anhand eines Beispiels erläutert wird. Diese Dienstarchitektur orientiert sich an einer bestehenden Architektur zur Modellierung agentenbasierter Systeme mit Petrinetzen. Abschließend werden die Eigenschaften von Dienstbeschreibungsnetzen für den praktischen Einsatz diskutiert, und es wird dargestellt, wie sich die Notation der Dienstbeschreibungsnetze in der praktischen Umsetzung vereinfachen lässt, so dass die Modelle leichter lesbar werden.

In **Kapitel 8** werden die wesentlichen Teile der Arbeit zusammengefasst, und es wird ein Ausblick auf die sich aus den Ergebnissen dieser Arbeit ergebenden Folgefragen gegeben.

2 Grundlagen dienstbasierter Geschäftsprozessmodellierung

Dieses Kapitel führt die grundlegenden Begriffe der dienstbasierten Geschäftsprozessmodellierung ein. Neben der Klärung der Begriffe des Geschäftsprozesses und des Workflows gehört hierzu die Definition des Dienstbegriffs, und es wird das Konzept der Ontologie erläutert, welches es erlaubt, Daten bei dienstbasierten Systemen einheitlich zu modellieren.

2.1 Aufbau dienstbasierter Geschäftsprozesse

Prozessorientierte Unternehmensorganisation ist angetreten, die nach funktionalen Einheiten gegliederte Struktur herkömmlicher Unternehmensorganisation aufzubrechen und stärker den Geschäftsprozess in den Vordergrund zu stellen. Verschiedene funktionale Einheiten arbeiten zusammen daran, Geschäftsprozesse erfolgreich umzusetzen.

Durch diese Art der Unternehmensorganisation wird somit stärker der Ablauf zur Erreichung eines bestimmten Zwecks in den Vordergrund gestellt. Zentrale Motivation dabei ist auch, die Zusammenarbeit funktionaler Einheiten zu stärken und ein Verständnis für die den Tätigkeiten übergeordneten Prozesse zu schaffen.

Auf technischer Ebene wird dies durch das Konzept der Dienste reflektiert. Dienste stellen den Abläufen im Unternehmen die Funktionalität verschiedener funktionaler Einheiten zur Verfügung, die von diesen genutzt werden können. Wählt man wiederum das in der Einleitung erwähnte Beispiel eines Geschäftsprozesses, der eine Kundenbestellung abwickelt, so sind an diesem Prozess verschiedene funktionale Einheiten beteiligt, wie etwa das Lager, das Rechnungswesen und der Versand. Jede dieser funktionalen Einheiten stellt Dienste bereit, die dann zur Erfüllung des Geschäftsprozesses entsprechend orchestriert werden. Eine automatisierte Koordination der Dienste wird dabei als Workflow bezeichnet. Zwar ist eine Automatisierung im Allgemeinen nicht notwendig, bringt jedoch in vielen Fällen Vorteile bei der Geschwindigkeit und der Qualität der Umsetzung der Geschäftsprozesse.

Nachdem in Abschnitt 2.2 zunächst Geschäftsprozesse und Workflows näher erläutert und voneinander abgegrenzt werden, widmet sich Abschnitt 2.3 der Beschreibung von Diensten und von dienstorientierten Architekturen. Dienstorientierte Architekturen nehmen dabei eine recht technische Sicht auf Dienste ein, die

sich von einem allgemeineren Dienstleistungsbegriff, wie ihn etwa Merz (1999) verwendet, unterscheidet. Dieser Dienstbegriff wird dem späteren technischen Begriff in den Dienstbeschreibungsnetzen entsprechen.

Um die Funktionalität von Diensten beschreiben zu können, ist es notwendig, auch die Veränderungen ausdrücken zu können, die Dienste an ihrer Umwelt vornehmen. Aus diesem Grund widmet sich Abschnitt 2.4 den Darstellungsmöglichkeiten von Daten und ihren Veränderungen. Speziell wird hier auf die Darstellung von Daten anhand von Konzepten einer Ontologie eingegangen, und es wird die Einführung dynamischer Konzepte sowie die Beschreibung der Veränderung von Daten mit Hilfe von Algebren erläutert. Letztere wird hier nur anhand der Evolving Algebras und der abstrakten Zustandsmaschinen aufgezeigt, während eine genauere Definition von Algebren Gegenstand des nächsten Kapitels ist.

2.2 Geschäftsprozesse

Bei der prozessorientierten Modellierung von Organisationen werden diese als Systeme betrachtet, deren Elemente die Mitarbeiter, die Produkte, die Produktionsanlagen, die Infrastruktur und die anderen Elemente einer Organisation sind. Diese verschiedenen Elemente besitzen vielschichtige Abhängigkeitsbeziehungen untereinander und zu ihrer Umwelt, und es ist das Ziel der prozessorientierten Modellierung, einzelne Einheiten zu identifizieren und zu kapseln, und die sie verbindenden Prozesse zu erkennen und zu modellieren, so dass technische Systeme die Steuerung dieser Prozesse unterstützen können. Die automatisierte Steuerung von Abläufen geschieht durch Workflow-Management-Systeme, die hierzu eine operationalisierte Darstellung der Abläufe benötigen, die von Maschinen korrekt interpretiert werden kann.

Dieser Abschnitt liefert zunächst eine Übersicht über die wichtigsten Begriffe, wie Geschäftsprozess und Workflow, in Unterabschnitt 2.2.1. Die verschiedenen Elemente einer solchen Modellierung und die Sichten auf sie werden in Unterabschnitt 2.2.2 betrachtet. Gegenstand von Unterabschnitt 2.2.3 sind dann verschiedene Möglichkeiten der Modellierung von Workflows und Geschäftsprozessen.

2.2.1 Geschäftsprozesse und Workflow-Management

Technische Systeme sind in der Lage, eine Vielzahl von Arbeitsabläufen in Unternehmen zu unterstützen. Solche Arbeitsabläufe, deren Steuerung und Abarbeitung von technischen Systemen unterstützt werden kann, werden als Workflows[1]

[1] Im Deutschen wird das Konzept auch als *Vorgangssteuerung* bezeichnet. Es werden jedoch im Rahmen dieser Arbeit zum Teil englische Begriffe den deutschen vorgezogen, wenn sich diese im deutschen Sprachraum als gebräuchlicher erwiesen haben. Eine Übersetzung der wichtigsten englischen Begriffe im Bereich der Workflows findet sich in Kreplin (2006).

bezeichnet und stellen das zentrale Element des Workflow-Management dar. Geschäftsprozesse hingegen stellen eine stärker geschäfts- und kundenorientierte Sicht auf Abläufe dar, die nicht notwendigerweise in jedem Fall automatisierbar sind. Diese Begriffe werden jedoch häufig synonym zueinander verwendet. Eine Übersicht über die wichtigsten Begriffe im Rahmen von Prozessen in Organisationen liefert dieser Unterabschnitt.

Workflows und Geschäftsprozesse

Aus betriebswirtschaftlicher Sicht ist es der Zweck eines Unternehmens, Leistungen zu erzeugen, die die Bedürfnisse der Kunden befriedigen und deren Vermarktung den wirtschaftlichen Erfolg des Unternehmens sichern. Diese Leistungen werden in Prozessen erstellt, die sich wiederum aus einer Reihe von Aktivitäten zusammensetzen. Diese Aktivitäten erzeugen aus einer Eingabe – einem *Input* – ein definiertes Arbeitsergebnis – einen *Output*. Der Input eines Prozesses sind beispielsweise Arbeitsleistungen, Werkstoffe, Energie oder Informationen. Der Output sind Produkte oder Dienstleistungen (vgl. Schmelzer und Sesselmann, 2004, S.45ff).

Eine *Aktivität* ist die kleinste logische Einheit innerhalb eines Prozesses. Sie ist häufig auf Daten oder Ressourcen angewiesen. Synonym zum Begriff der Aktivität werden häufig auch *Task* oder *Funktion* benutzt[2]. *Ressourcen* stellen menschliche oder maschinelle Arbeitsmittel dar, die benötigt werden, um eine Aktivität ausführen zu können (vgl. Workflow Management Coalition, 1999).

Ein *Prozess* ist nach betriebswirtschaftlichem Verständnis eine „inhaltlich abgeschlossene, zeitliche und sachlogische Folge von Aktivitäten, die zur Bearbeitung eines betriebswirtschaftlich relevanten Objektes notwendig sind“ (Becker u. a., 2005, S. 6). Ein *Geschäftsprozess* ist ein spezialisierter Prozess bestehend aus einer „funktionsüberschreitenden Verkettung wertschöpfender Aktivitäten, die von Kunden erwartete Leistungen erzeugen und deren Ergebnisse strategische Bedeutung für das Unternehmen haben. Sie können sich über Unternehmensgrenzen hinweg erstrecken und Aktivitäten von Kunden, Lieferanten und Partnern einbinden“ (vgl. Schmelzer und Sesselmann, 2004, S.46). Im Gegensatz zu einem Prozess im Allgemeinen betont der Geschäftsprozess also erheblich stärker die Kundensicht. Hierbei ist die Kundenbeziehung jedoch zum Teil recht allgemein interpretiert, so dass auch verschiedene Abteilungen eines Unternehmens in dieser Beziehung zueinander stehen können.

Abgegrenzt zum Prozess wird häufig der Begriff des Workflows. Ein *Workflow* ist ein Ablauf, der aus einzelnen Aktivitäten aufgebaut ist, der im Gegensatz zum Prozess jedoch stärker die operationale Ebene betont. Ein Workflow „besitzt einen

[2]Hier ist die Begriffsbildung nicht immer eindeutig. Häufig wird eine Aktivität auch als eine Menge von Tasks dargestellt, wobei ein Task eine atomare Aktivität ist, also ein nicht weiter teilbarer Ausführungsschritt, während eine Aktivität im Allgemeinen in mehrere Tasks aufgeteilt werden kann. Statt Task wird zum Teil auch der Begriff Aktion verwendet.

definierten Anfang, einen organisierten Ablauf und ein definiertes Ende“ (vgl. Jablonski u. a., 1997, S. 490). Die Spezifikation der einzelnen Aktivitäten ist im Idealfall so exakt, dass sich anhand der Ausgabe einer Aktivität bestimmen lässt, welche Nachfolgeaktivität zu folgen hat. Dies erlaubt es, Workflows durch *Workflow-Management-Systeme* automatisiert zu steuern (vgl. Jablonski u. a., 1997, S. 490). Vielfach wird gerade die technische Unterstützbarkeit als wichtiges Kriterium eines Workflows angesehen wie etwa bei (Gierhake, 1998, S. 54):

> „Ein ‘Workflow’ stellt einen technisch umfassend unterstützten Arbeitsablauf dar, der ausgehend von einem auslösenden Ereignis entlang einer definierten Kette von Teilschritten bis zu einem definierten Arbeitsergebnis führt, wobei der Grad der Vervollständigung des Arbeitsergebnisses mit jedem einzelnen Arbeitsschritt zunimmt.“

Anders als bei Geschäftsprozessen besteht bei einem Workflow nicht notwendigerweise ein Bezug zu einem Geschäftszweck. Workflows beziehen sich vielmehr allgemein auf organisatorische Vorgänge und koordinieren die darin anfallenden Tätigkeiten der Aufgabenträger und technischen Systeme (vgl. Gierhake, 1998, S. 54f). Sie steuern somit die Reihenfolge der Aktivitäten. Diese Charakterisierung soll im Rahmen dieser Arbeit verwendet werden. Sie ist etwas allgemeiner als die eher technisch motivierte Sicht der Workflow Management Coalition in (Hollingsworth, 1995, S.6), die einen Workflow folgendermaßen definiert: „The computerised facilitation or automation of a business process, in whole or part.“ Die Begriffe *Teilworkflow* oder *Subworkflow* bezeichnen Fragmente eines Workflows.

Zur technischen Unterstützung von Prozessen in Unternehmen dienen *Workflow-Management-Systeme* (WfMS). Dies sind Softwaresysteme, die die Vorgänge in Workflows umsetzen, koordinieren, steuern und ausführen. Sie stoßen Aktivitäten an, stellen die notwendigen Daten und Ressourcen für sie bereit, prüfen die Ergebnisse und reichen diese an andere Teilprozesse weiter. Zudem überwachen sie häufig Zeit- und Zugriffsbeschränkungen.

Workflow-Management-Systeme zeichnen sich vor allem durch eine gesamtheitliche Sicht und durch ihre Prozessorientierung aus. Ziel eines Workflow-Management-Systems ist die Steuerung von Aktivitäten. Dies bedarf einer präzisen Beschreibung der Ausführungsreihenfolgen und der den Aktivitäten zugeordneten Ressourcen und Daten. Ein *Workflow-Schema* stellt eine solche Beschreibung dar. Es beschreibt den prinzipiellen Ablauf eines Workflows in einer für das Workflow-Management-System interpretierbaren Sprache.

Eine *Workflow-Instanz* stellt eine spezielle Ausprägung eines Workflow-Schemas dar, die vom Workflow-Management-System durch Instanziierung eines Workflow-Schemas erzeugt wurde. Sie beschreibt einen konkreten Ablauf, dem konkrete Daten und Ressourcen zugeordnet worden sind.

Im Gegensatz zum Workflow beinhaltet ein Prozess auch die Möglichkeit, unstrukturiert oder gar ergebnisoffen zu sein, wie dies etwa bei kreativen Prozessen

der Fall ist. Hier lassen sich zwar ebenfalls strukturierte Teilprozesse identifizieren, die durch Workflows automatisiert werden können, der Prozess als solcher lässt sich im Allgemeinen jedoch nicht automatisieren. Für solche Prozesse sind entsprechend eher kollaborative Group-Ware-Lösungen (vgl. Gross und Koch, 2007) im Einsatz und weniger Workflow-Management-Systeme.

Übersicht der wesentlichen Begriffe

Die wesentlichen Begriffe im Kontext von Workflows werden in diesem Abschnitt noch einmal dargestellt. Die Darstellung orientiert sich an den Vorgaben der Workflow Management Coalition (1999). Jessen und Valk (1987) definieren einige Begriffe aus einer eher technischen Motivation abweichend und auch van der Aalst und van Hee (2002) definieren einige Begriffe leicht unterschiedlich. Ein *Work Item* bezeichnet bei van der Aalst und van Hee (2002) das, was hier als Aktivität beschrieben wird. Eine Aktivität hingegen bezeichnet das was im Folgenden als Aktivitätsinstanz dargestellt wird. Abgesehen von diesen kleinen Unterschieden sind die wesentlichen Begriffe jedoch gleich. Diese Arbeit orientiert sich am Standard des WfMC der Workflow Management Coalition (1999). Abbildung 2.1 gibt eine Übersicht über die wesentlichen Konzepte und ihre Beziehungen.

Ein *Workflow-Schema* ist eine operationalisierte Darstellung eines Workflows, die von einem Workflow-Management-System verarbeitet werden kann. Eine *Workflow-Instanz* ist eine Ausprägung eines Workflow-Schemas und besitzt einen eigenen Kontrollfluss. Ein *Subprozess* ist ein Teil eines Prozesses. Ebenso ist ein *Subworkflow* ein Teil eines Workflows.

Eine *Aktivität* stellt einen elementaren logischen Schritt in einem Prozess dar. Sie kann entweder von einem Menschen oder automatisiert abgearbeitet werden. Eine *strukturierte Aktivität* (auch Block-Aktivität) ist eine Aktivität eines Workflows, die durch einen Subworkflow realisiert wird. Eine *Aktivitätsinstanz* stellt die konkrete Ausprägung einer Aktivität in einer konkreten Workflow-Instanz dar. Ein *Work Item* oder eine Einzelaufgabe ist eine Aktivitätsinstanz, die von einem menschlichen Akteur abgearbeitet wird. Eine *Ressourcenklasse* stellt eine Menge von Ressourcen dar, die ähnliche Eigenschaften aufweisen. *Organisationseinheiten* sind Ressourcenklassen, in denen das die Klassen unterscheidende Element durch die Organisationsstruktur gegeben wird. Dies sind etwa Abteilungen oder Teams. *Rollen* sind Ressourcenklassen, die mögliche Funktionen, Kompetenzen oder Fähigkeiten der Ressourcen ausdrücken. Hierzu zählen Rollen von Mitarbeitern wie *Sachbearbeiter* oder Fähigkeiten von Maschinen wie *Stanzpresse*.

Eine *Workflow-Engine* führt eine Workflow-Instanz aus und überwacht ihre Ausführung. Ein *Workflow-Enactment-Service* ist ein Dienst der aus einer oder mehreren Workflow-Engines besteht und der Workflow-Instanzen verwaltet und durch Instanziierung eines Workflow-Schemas erzeugt. Diese Instanzen werden an die Workflow-Engines zur Ausführung weitergegeben.

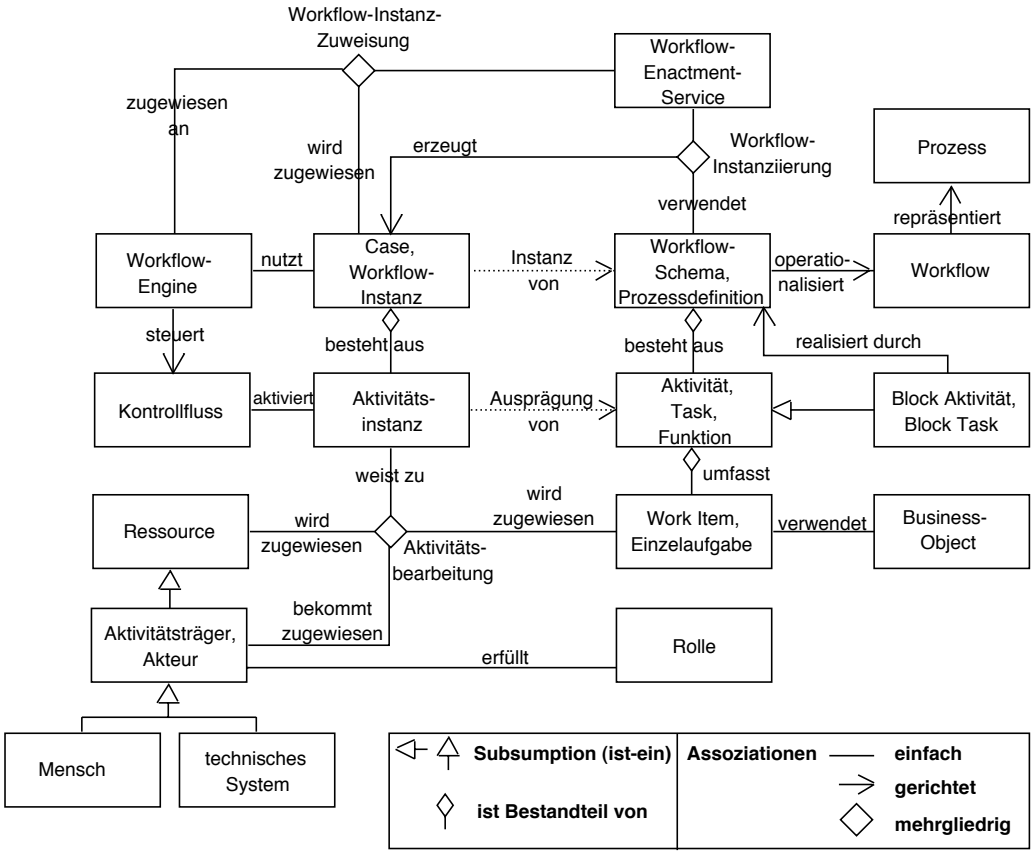


Abbildung 2.1: Begriffe des Workflow-Managements und ihre Beziehungen

Das Workflow-Referenzmodell

Das *Workflow-Referenzmodell* (Hollingsworth, 1995) der Workflow Management Coalition benennt fünf Komponenten eines Workflow-Management-Systems, die über Schnittstellen miteinander und mit ihrer Umgebung interagieren. Diese werden in Abbildung 2.2 dargestellt.

Die wesentlichen Elemente des Modells sind:

Ein Workflow Enactment Service (WFES) besteht aus einer oder auch mehreren Workflow Engines und kann auf diese Weise Workflow-Instanzen in den Engines erzeugen, verwalten und ausführen. Er stellt den *Workflow Engines* die Schnittstelle zu anderen Komponenten und entfernten WFES zur Verfügung.

Eine Workflow Engine (WFE) ist ist für die Ausführung und Laufzeitkontrolle von Workflow-Instanzen verantwortlich. Sie befindet sich innerhalb des *Work-*

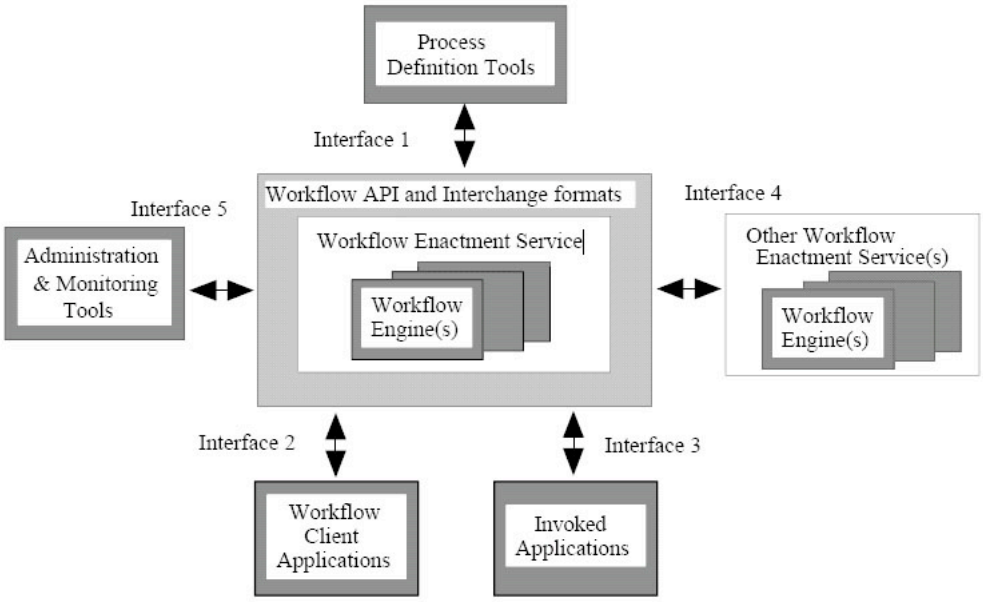


Abbildung 2.2: Das Referenzmodell der Workflow Management Coalition
(Aus (Hollingsworth, 1995))

flow Enactment Service.

Eine Workflow Client Application (WFClA) stellt die Schnittstelle eines Benutzers zum WfMS in Form einer Benutzungsoberfläche bereit. In dieser kann der Benutzer Tasks annehmen und bearbeiten, nachdem sie ihm angeboten wurden. Außerdem kann er damit Workflows instanziieren.

Die *Workflow Client Application* teilt sich in *Worklist Handler* und *User Interface* auf. Der *Worklist Handler* interagiert mit der Worklist des Benutzers, die im *Workflow Enactment Service* gepflegt wird. Die Benutzungsoberfläche präsentiert dem Benutzer seine Aufgaben und bietet ihm Werkzeuge zur Bearbeitung an.

Ein Process Definition Tool (PDT) ist ein Werkzeug zur Erstellung von Workflows für das WfMS. Mit ihm lassen sich Workflow Schemata erzeugen, die durch den *Workflow Enactment Service* instanziiert werden können.

Die Invoked Applications stellen externe Anwendungen dar, die vom WfMS aufgerufen werden.

Die Administration and Monitoring Services dienen der Überwachung und der Administration des Workflows.

Anhand des Modells lassen sich relativ gut die Aufgaben und Schnittstellen eines Workflow-Management-Systems erkennen. Ein vorrangiges Ziel der *Workflow Management Coalition* ist die Schaffung eindeutiger Standards für die Schnittstellen zwischen einem Workflow-Management-System und den Anwendungen, mit denen es interagiert.

2.2.2 Sichten auf Geschäftsprozesse

Neben dem Kontrollfluss modelliert ein Workflow noch weitere Aspekte. Diese *Perspektiven der Workflow-Modellierung* sollen hier kurz dargestellt werden.

Die Modellierung von Workflows ist nicht auf die Modellierung möglicher Abfolgen beschränkt, sondern verlangt auch die Berücksichtigung von Daten- und Informationen auf der einen Seite und Ressourcen auf der anderen Seite. Aus diesem Grund haben sich in den meisten Werkzeugen zu Modellierung von Workflows verschiedene Sichten auf einen Workflow etabliert. So benennen Jablonski u. a. (1997)(S. 98ff) wie auch van der Aalst (1999b); Weske u. a. (2005) und das Werkzeug ARIS (vgl. Scheer, 2002, S.35ff) fünf verschiedene Sichten. Jablonski u. a. (1997) betonen jedoch, dass diese im konkreten Anwendungsfall gegebenenfalls durch weitere Sichten ergänzt werden müssen. Abbildung 2.3 ist aus (Scheer, 2002, S. 57) übernommen und stellt dort das ARIS-Haus dar, welches als Modell für die Modellierung mit dem Werkzeug ARIS gilt und welches die verschiedenen Sichten darstellt. Dort sind die Sichten jedoch leicht anders als hier dargestellt.

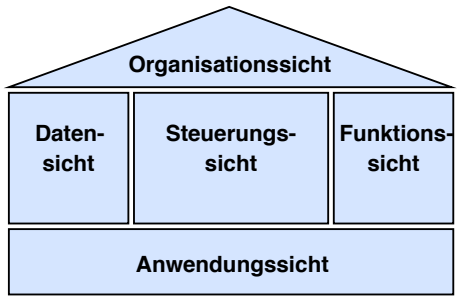


Abbildung 2.3: Die fünf Workflow-Perspektiven
(Nach (Scheer, 2002, S. 57))

Die *Aktivitäts-, Task- oder Funktionssicht* charakterisiert die Aktivitäten und Teilworkflows, die während der Ausführung eines Workflows abgearbeitet werden müssen. Hierbei wird die Transformation des Inputs in den Output beschrieben, die eine Aktivität oder ein Workflow vornimmt. Zudem beschreibt diese Sicht, wie sich diese Teilworkflows in kleinere funktionale Einheiten unterteilen. Dies wird häufig durch eine hierarchische Struktur anhand eines Baumes

wiedergegeben, dessen Blätter die Aktivitäten als kleinste logische Einheit darstellen und deren Wurzelknoten die Funktionalität des gesamten Workflows darstellt. Die inneren Knoten stellen dann die Funktionalität zusammengesetzter Teilworkflows dar. In diesem Zusammenhang werden zusammengesetzte Workflows häufig als komplexe Workflows bezeichnet, während Aktivitäten auch als atomare Workflows bezeichnet werden. Ein Teilworkflow oder eine Aktivität kann hierbei durchaus Bestandteil mehrerer übergeordneter Workflows sein.

Die funktionale Perspektive gibt lediglich das an, was getan werden muss. Es werden jedoch weder die Reihenfolge oder die Bedingungen zum Ausführen der Aktivitäten noch deren beteiligte Ressourcen oder Informationen benannt. Dies geschieht innerhalb der anderen Sichten.

Die *Kontrollfluss- oder Steuerungssicht* spezifiziert die Ausführungsreihenfolge der Aktivitäten und gibt an, wann Teilworkflows alternativ oder parallel abgearbeitet werden müssen. Sie bildet gleichzeitig den integrativen Rahmen der verschiedenen Sichten und zeigt ihr Zusammenspiel auf. Zur Steuerung müssen häufig Ressourcen oder Informationen Beachtung finden, die aus der Organisations- oder Datensicht benötigt werden.

Die Kontrollflusssicht eignet sich in besonderem Maße zur Modellierung durch Petrinetze, da hier der prozesshafte Ablauf im Vordergrund steht. Zudem erlauben Petrinetze, die meisten Kontrollflussmuster von Workflows darzustellen (vgl. van der Aalst, 2000).

Die *Ressourcen- oder Organisationssicht* gibt einen Überblick über die organisationale Struktur und die Struktur der Ressourcen, die einem Workflow zur Verfügung stehen. Die Ressourcen werden zum Teil in Organisationseiheiten gegliedert, die jeweils eine Menge von Aufgabenträgern darstellen. Diese Sicht stellt dar, welche Abhängigkeiten zwischen den Ressourcen bestehen und welche wann und wie verfügbar sind. Insbesondere werden in dieser Perspektive auch die möglichen Rollen modelliert, die ein Aufgabenträger annehmen kann und welche Aktivitäten der Inhaber einer Rolle ausführen kann.

Die *Daten- oder Informationssicht*, beschreibt die zur Ausführung des Workflow verwendeten Daten. Diese unterscheiden sich in Kontrolldaten und Produktionsdaten. Kontrolldaten werden lediglich zur Ablaufsteuerung benutzt, während Produktionsdaten durch Informationsobjekte oder Fachobjekte (etwa Dokumenten oder Formularen) repräsentiert werden, deren Existenz unabhängig vom Workflow ist. Diese Sicht beschreibt die Ein- und Ausgabeparameter einer Aktivität und dient somit auch der Modellierung von Abhängigkeiten bezüglich der Daten zwischen Aktivitäten. Dies wird häufig auch als Datenfluss bezeichnet.

Die *Operations- oder Anwendungssicht* beschreibt die elementaren Operationen, die von Aufgabenträgern ausgeführt werden. Sie beschreibt die möglichen Ausführenden und die benötigten Ressourcen einer Operation. Operationen können sowohl von technischen Systemen als auch von menschlichen Aufgabenträgern oder von einer Mischung aus beidem ausgeführt werden. Sie können auf Daten zugreifen oder auch maschinelle Prozesse anstoßen. Typischerweise besteht der Zugriff auf Daten aus dem Lesen, Schreiben oder Verändern von Kontrollfluss- und Produktionsdaten. Reale Objekte der Anwendungsdomäne werden durch Fachobjekte repräsentiert. Diese können beispielsweise sowohl Vertragspartner als auch Auftragseingänge sein.

Die funktionale Beschreibung von Diensten umfasst in erster Linie die Funktionssicht, jedoch müssen die anderen Sichten zum Teil mit berücksichtigt werden. So ist die Reihenfolge der ausgeführten Aktivitäten ebenso relevant wie die Art der ausgetauschten Daten. Lediglich die Ressourcensicht und die Art, wie eine Aktivität abgearbeitet wird, kann vernachlässigt werden, da ein Aufrufer hierauf keine Kontrolle ausüben kann.

2.2.3 Modellierung von Geschäftsprozessen

Zur Beschreibung von Workflows oder Geschäftsprozessen werden meist ähnliche Notationen herangezogen auch wenn Geschäftsprozesse nicht notwendigerweise in dem Maße automatisierbar sind wie dies für Workflows der Fall ist. Zu den prominentesten Modellierungstechniken dürften die von Keller u. a. (1992) eingeführten ereignisgesteuerten Prozessketten (EPKs), die Business Process Modelling Notation (vgl. BPMN, 2008) sowie die Diagramme der UML, speziell die Aktivitätsdiagramme (UML-AD) (vgl. UML 2.0, 2005) zählen. Ein weiterer eher im akademischen Bereich verbreiteter Ansatz ist durch Workflownetze (vgl. van der Aalst, 2000) gegeben. Workflownetze basieren auf Petrinetzen und erlauben somit die formale Verifikation von Eigenschaften durch Ergebnisse aus diesem Bereich. Neben diesen Modellierungstechniken existieren noch eine Reihe herstellerspezifischer Modellierungstechniken, auf die hier jedoch nicht weiter eingegangen werden soll, da sie mit den vorgestellten Techniken häufig verwandt sind. Eine Übersicht hierzu findet sich bei van der Aalst u. a. (2003) und aus einer mehr betriebswirtschaftlichen Sicht bei Freund und Götzer (2008).

Workflownetze

Workflownetze (van der Aalst, 2000) basieren auf Petrinetzen und geben diesen eine spezielle Struktur. Petrinetze bieten eine formale Semantik und erlauben die Darstellung aller für Workflows relevanten Patterns wie in van der Aalst u. a. (2003) dargestellt. Dort findet sich auch ein Vergleich von Petrinetzen zu anderen in gängigen Werkzeugen verwendeten Darstellungen.

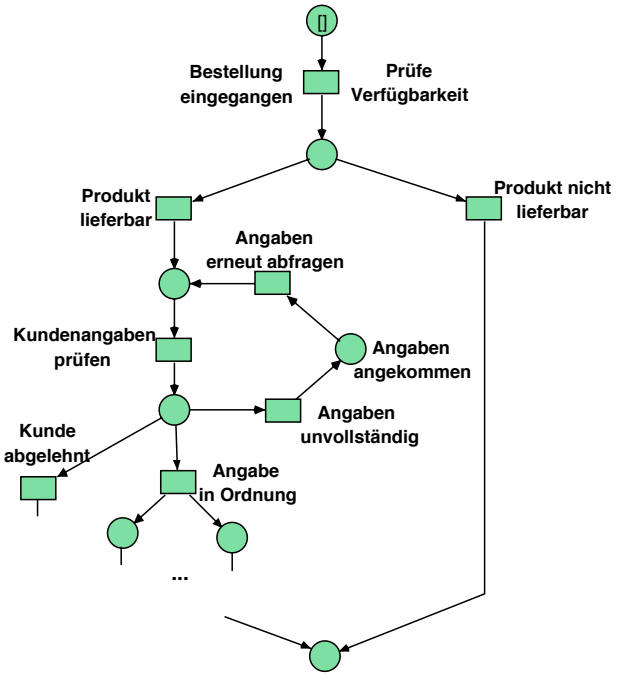


Abbildung 2.4: Ein Workflownetz

Workflownetze modellieren die atomaren Aktivitäten oder Tasks als Transition im Netz und die Ressourcen als Stellen im Netz. Zudem repräsentieren die Stellen auch der Steuerung des Kontrollflusses, da sie die Bedingungen für das Schalten einer Transition angeben. Durch die spezielle Struktur der Workflownetze besitzt ein Workflow stets einen klar definierten Anfang und ein klar definiertes Ende. Ein Workflownetz ist exemplarisch in Abbildung 2.4 dargestellt.

Workflownetze werden häufig auch zur Steuerung von Workflows vorgeschlagen (vgl. Reese, 2009). In diesem Fall steuern sie das Verhalten eines Workflow-Management-Systems beziehungsweise einer Workflow-Engine. Eshuis und Wieringa (2003) schlagen hierzu eine andere Art der Darstellung von atomaren Aktivitäten vor, die aus zwei Transitionen und einer Stelle in der Mitte besteht, wie in Abbildung 2.5 dargestellt. Es wird davon ausgegangen, dass eine Workflow-Engine eine Aufgabe delegiert und dann darauf wartet, dass diese Aufgabe erledigt wird oder ein Timeout auftritt, um dann weitere Aufgaben zu delegieren. Konzeptionell ist dies eine Verfeinerung einer Transition, die eine Task darstellt (Task-Transition). Jacob (2002) und Jacob u. a. (2001) verwenden Referenznetze zur Implementation einer Workflow-Engine, die dann über Java-Anschriften externe Anwendungen steuern kann. Dieses Konzept wird von Reese (2009) ausgebaut und zur Fragmentation und verteilten Ausführung von Workflows eingesetzt.

Im Gegensatz zu vielen anderen Modellierungsansätzen erlauben Workflownetze

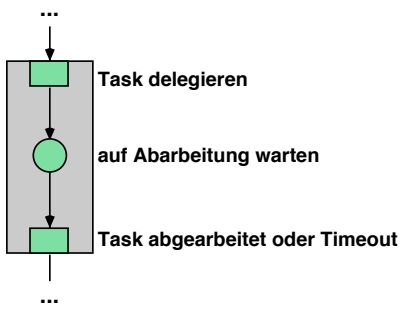


Abbildung 2.5: Task-Transition eines Workflownetzes zur Steuerung

eine explizite Darstellung des aktuellen Zustandes. Zudem ist die Verwendung von Datentypen in Petrinetzen bereits vielfach erforscht. Auf die formalen Eigenschaften von Workflownetzen wird im nächsten Kapitel noch näher eingegangen werden.

Ereignisgesteuerte Prozessketten (EPKs)

Ereignisgesteuerte Prozessketten (EPKs) (engl. event-driven process chain (EPC)) stellen einen weit verbreiteten Ansatz zur Modellierung von Workflows dar. Sie sind besonders im Zusammenhang mit der Modellierung von Prozessen im Rahmen der Werkzeuge von SAP und ARIS weit verbreitet. Entwickelt wurden ereignisgesteuerte Prozessketten von Keller u. a. (1992). Sie unterscheiden vier Arten von Objekten:

- Ereignisse repräsentieren einen eingetretenen Zustand oder eine eingetretene Bedingung. Sie können Funktionen auslösen. Ein Ereignis kann auf Informationsobjekte des Datenmodells verweisen.
- Funktionen stellen die Aktivitäten oder Tasks dar.
- Verknüpfungen erlauben die Verzweigung des Ablaufs.
- Ressourcen (Daten, Organisation, System)

Zentral ist hierbei der Begriff des Ereignisses. Der Kontrollfluss zwischen Funktionen wird über dazwischenliegende Ereignisse gesteuert. Ereignisse haben im Gegensatz zu Funktionen keine Entscheidungskompetenz und besitzen einen passiven Charakter. Näheres zu EPKs und ihrer Verwendung findet sich bei Davis (2001).

EPKs stellen die Ablauflogik, die angibt, welche Ereignisse welche Funktionen auslösen und welche Ereignisse durch welche Funktionen erzeugt werden, als Graph dar. Durch Verknüfungsoperatoren zwischen Ereignissen oder Funktionen kann der Ablauf verzweigt oder zusammengeführt werden. Mögliche Verknüpfungsoperatoren sind Disjunktion (OR), Konjunktion (AND) und Antivalenz (XOR). Bei der Verzweigung (nach einer Funktion stehend) führt die Disjunktion dazu, dass ein oder mehrere Pfade ausgewählt werden, die Konjunktion dazu, dass alle Pfade aus-

gewählt werden, und die Antivalenz dazu, dass genau einer der Pfade ausgewählt wird. Bei der Zusammenführung (vor einer Funktion) kann die Disjunktion durch irgendein Ereignis angestoßen werden, die Konjunktion nur durch alle Ereignisse und die Antivalenz nur durch genau ein Ereignis.

Ein Problem im Zusammenhang mit EPKs stellt die ursprünglich nicht-formale Definition der Semantik dar. Verknüpfungen werden nach dieser Semantik nicht lokal ausgewertet sondern anhand globaler Bedingungen. Dies bedeutet, dass das Verhalten einer Verknüpfung von entfernten Ereignissen abhängen kann. So blockiert etwa eine OR-Verknüpfung solange, bis sichergestellt ist, dass es im System keine Möglichkeit mehr gibt, dass eine noch nicht belegte Eingangskante der Verknüpfung noch belegt wird. Erst wenn das gewährleistet ist, wertet sie die Bedingung aus und schaltet. Es wird von van der Aalst u. a. (2002) gezeigt, dass es EPKs gibt, für die keine formale Semantik angegeben werden kann, da für jede Formalisierung der Semantik ein Beispiel gefunden werden kann, dass der nicht-formalen Semantik widerspricht. Somit kann es keine formale Semantik für EPKs geben, die alle Aspekte ohne Einschränkungen abdeckt.

Trotz ihrer weiten Verbreitung zur Modellierung von Prozessen in Unternehmen werden im Rahmen dieser Arbeit Petrinetze verwendet, da EPKs einerseits weniger formal definiert sind und andererseits die meisten Formalisierungen von EPKs mittels Petrinetzen erfolgen (vgl. Langner u. a., 1998; van der Aalst, 1999a; van Dongen u. a., 2005). Somit ist eine spätere Übertragung der Ergebnisse auf eine Subklasse von EPKs prinzipiell möglich.

Business Process Modelling Notation (BPMN)

Die *Business Process Modelling Notation (BPMN)* (BPMN, 2008) stellt einen relativ neuen Ansatz zur Modellierung von Prozessen dar, der durch die Object Management Group standardisiert wurde. Die derzeit aktuelle Version ist die Version 1.2 (BPMN 1.2, 2008), die jedoch derzeit im Begriff ist, durch die Version 2.0 abgelöst zu werden[3]. Ähnlich den EPKs bietet die BPMN die Möglichkeit, Prozesse dadurch zu modellieren, dass Aktivitäten (Rechtecke mit abgerundeten Kanten) durch Pfeile verbunden werden. Zur Steuerung des Prozessflusses gibt es verschiedene Gateways (OR, XOR, AND) und entsprechende Merger. Eine detailliertere Einführung findet sich bei Allweyer (2005), bei Fettke (2008) oder bei Bridgeland und Zahavi (2008).

Durch Swimlanes wird in der BPMN angezeigt, wer für das Ausführen einer Aktivität zuständig ist und von wo nach wo Nachrichten versandt werden. Eine Reihe von Ereignissen können in der BPMN explzit dargestellt werden. Auch erlaubt die BPMN die Verfeinerung von Tasks durch Subworkflows.

In der Version 2.0 der BPMN erhält auch die Möglichkeit der Modellierung verschiedener Sichten auf einen Prozess Einzug. Zudem werden Möglichkeiten zur Mo-

[3]Die Version 2.0 der BPMN ist für Mitte 2010 angekündigt.

dellierung interorganisationaler Prozesse unterstützt. Insgesamt orientiert sich die BPMN recht stark an Petrinetzen. Sie unterscheidet jedoch deutlich störker zwischen verschiedenen Elementen des Modells und besitzt eine weniger explizite Darstellung des aktuellen Zustands.

Die Überführung der BPMN in Petrinetze wird beispielsweise von Dijkman u. a. (2007, 2008) und Koniewski u. a. (2006) diskutiert.

2.3 Dienste

Eine zentrale Motivation zur Operationalisierung von Prozessen durch Workflows ist die Möglichkeit, die Abarbeitung dieser Prozesse technisch zu unterstützen. Moderne Architekturen basieren hierbei häufig auf Diensten, um eine verteilte Ausführung eines Workflows zu unterstützen. Dienstorientierte Architekturen spiegeln die Kapselung der Aktivitäten auf technischer Ebene wider und bieten eine einheitliche Schnittstelle nach außen.

Dieses Dienstverständnis ist stark durch eine Verteiltheit unabhängiger Systemeinheiten geprägt. Es reflektiert jedoch nicht unbedingt das Dienstverständnis anderer Disziplinen. Zudem weist dieses Begriffsverständnis eine Verwandtschaft zum Agentenkonzept auf und die Abgrenzung beider Begriffe ist häufig nicht eindeutig. Je nach Autor werden Diensten und Agenten oft ähnliche Eigenschaften zugeschrieben. Aus diesem Grund wird der Dienstbegriff in Unterabschnitt 2.3.1 präzisiert. Die Verwendung von Diensten geht in der Regel mit einer speziellen dienstorientierten Architektur der Anwendung einher, deren allgemeine Konzepte im Unterabschnitt 2.3.2 detaillierter betrachtet werden. Unterabschnitt 2.3.3 betrachtet die Interaktion von Diensten in einer dienstorientierten Architektur.

2.3.1 Der Dienstbegriff in verteilten Systemen

Der Begriff des *Dienstes* (engl. service) ist über die Grenzen verschiedener Disziplinen hinweg verbreitet. Im Allgemeinen ist der Begriff verwandt mit dem der Dienstleistung (vgl. Wetzel und Klischewski, 2004) und bezeichnet somit die Erbringung einer Leistung durch einen *Dienstanbieter*. Hierbei zeichnet sich bereits ein wesentliches Element von Diensten ab: Dienste werden von einem *Dienstanbieter* oder *Diensterbringer* (engl. service provider) angeboten und von einem *Dienstnutzer* (engl. service consumer oder client) in Anspruch genommen.

Im Rahmen verteilter Systeme findet der Dienstbegriff häufig im Zusammenhang mit Client/Server-Modellen Verwendung, wobei der Server den Dienstanbieter darstellt und der Client den Dienstnutzer. Popien u. a. (1996) charakterisieren einen Dienst als „eine Funktion, die durch ein Objekt an einer Rechnerschnittstelle zur Verfügung gestellt wird“ (S. 34). Hierbei tritt der deutliche Bezug zur objektorientierten Programmierung hervor, der zwar häufig vorhanden, jedoch nicht notwendig

ist, da ein Dienst durchaus auch von einem beliebigen, nicht objektorientierten Programm realisiert werden kann.

MacKenzie u. a. (2006) (Z. 335ff.) bieten eine Definition, die einem allgemeinen Dienstbegriff in verteilten Systemen näher kommt:

> „A service is a mechanism to enable access to a set of one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description. A service is provided by one entity – the service provider – for use by others, but the eventual consumers of the service may not be known to the service provider and may demonstrate uses of the service beyond the scope originally conceived by the provider“.

Dienste stellen also eine Schnittstelle und eine Dienstbeschreibung zur Verfügung, über die Dienstnutzer einen Dienst aufrufen können. Die Dienstbeschreibung enthält dabei Bedingungen und Richtlinien, die ein Dienstnutzer für einen erfolgreichen Dienstaufruf einhalten muss. Auf diese Weise erlauben Dienste die lose Kopplung zwischen Softwarekomponenten, indem sie eine Schnittstelle bereitstellen über die Dienstnutzer einen Dienst zur Laufzeit auswählen und ansprechen können. Dies wird auch in folgender Definition von Merz (1999) (S. 84) deutlich:

> „Ein Dienst bietet als abgrenzbare Einheit eine Funktionalität, die über eine operationale Schnittstelle anderen Systemkomponenten zur Nutzung angeboten wird. Der Dienst kann auf andere zur Erfüllung des eigenen zurückgreifen“.

Für den Dienstnutzer ist die Implementierung des Dienstes transparent, was bedeutet, dass auf Seiten des Diensterbringers die Implementierung des Dienstes verändert werden kann, solange die Spezifikation des Dienstes erfüllt bleibt. Ebenso ist es denkbar, dass die Dienstanfrage an einen weiteren Diensterbringer delegiert wird, welcher letztlich die konkrete Funktionalität implementiert. Bei einer solchen Vorgehensweise nimmt der Diensterbringer gleichzeitig die Rollen des Dienstanbieters und des Dienstnutzers ein.

Im Allgemeinen dient der Dienstgedanke dazu, Systeme offen und flexibel erweiterbar zu halten. Durch die Spezifikation der Dienstschnittstelle wird von konkreten Implementationen abstrahiert. Gerade im Bereich heterogener verteilter Systeme ist die Spezifikation einer Dienstschnittstelle, die beliebig implementiert werden kann eine wichtige Voraussetzung für den Austausch von Informationen.

Eine derzeit prominente Schnittstellenbeschreibung für Dienste im Bereich der verteilten Systeme wird durch die Webservice-Technologie (Booth u. a., 2004) bereitgestellt. Diese Technologie wird zum Teil sogar als eine Referenzimplementation für verteilte Dienste gesehen, wird andererseits jedoch auch als zu ressourcenhungrig bei der Verarbeitung kritisiert. Auch hier spielt vor allem die lose Kopplung eine wichtige Rolle. So beschreiben etwa Gartner (2001) Webservices folgendermaßen:

> „Web services are loosely coupled software components delivered over the Internet via standards-based technologies like XML and Simple Object Access Protocol (SOAP)“.

Die Begriffe der losen Kopplung und der abgrenzbaren Einheit lassen eine Nähe zum Konzept der Agenten erkennen. So bezeichnen Booth u. a. (2004) Webservices als abstraktes Konzept, welches von einem konkreten *Agenten* implementiert werden muss. Eine weitere wichtige Gemeinsamkeit ist, dass Dienste anders als etwa Objekte nur genau einmal existieren (vgl. Vogels, 2003). Alle Anfragen werden also von derselben „Instanz“ entgegengenommen, deren Lebenszyklus vollkommen unabhängig vom Aufrufer ist und die in der Regel als nicht terminierend angenommen wird.

Dieser Arbeit liegt ein Dienstverständnis zugrunde, welches dem der Webservices und der von Merz (1999) gegebenen Definition sehr nahe kommt. Dieses Dienstverständnis spiegelt sich auch im Rahmen der dienstorientierten Architekturen (vgl. Abschnitt 2.3.2) wieder. Dienste stellen hiernach eine Funktionalität dar, die von einer Systemeinheit gemäß einer vordefinierten Schnittstelle durch Kommunikation über Nachrichten erbracht wird. Ein Dienst ist über eine feste Adresse aufrufbar und ist auf konzeptioneller Ebene ständig ansprechbar. (Dies bedeutet nicht, dass Dienste praktisch immer verfügbar sein müssen.) Um zwischen Diensten als Schnittstellen- und Funktionsbeschreibung und ihrer Implementation unterscheiden zu können, wird der Begriff des *Diensttyps* und der des *Dienstanbieter* bzw. des *Dienstnutzers* eingeführt.

Definition 2.1 (Diensttyp, Dienst)
Ein *Diensttyp* beschreibt eine statische Schnittstelle, eine funktionale Schnittstelle und eine Verhaltensschnittstelle, die von beliebigen Dienstanbietern implementiert werden können. Dabei kann derselbe Diensttyp von verschiedenen Dienstanbietern unterschiedlich implementiert werden. Die Implementation eines Diensttyps bleibt nach außen verborgen. Der *Typbezeichner* eines Diensttyps identifiziert diesen eindeutig.
Ein *Dienst* stellt als abgrenzbare Einheit eine zuvor definierte Funktionalität bereit, die durch eine klar definierte Schnittstellenbeschreibung – den *Diensttyp* – beschrieben wird. Ein Dienst stellt somit eine Art dar, auf die eine Diensttyp konkret realisiert ist. Die konkrete Realisierung bezieht sich dabei nicht notwendigerweise auf die Implementierung, sondern vielmehr auf die verwendeten Ressourcen und gegebenenfalls genutzte weitere Dienste[4].
Ein durch einen Dienstanbieter bereitgestellten *Dienst* kann über das durch den

[4]Ein Dienst abstrahiert von einer konkreten Programmiersprache. Er beschreibt vielmehr die Funktionalität, die erbracht wird. Ein Dienstanbieter kann dann konkret entscheiden, ob ein Dienst beipielsweise in Java oder .Net implementiert ist.

Diensttyp definierte Verhalten hinaus nicht von einem Dienstnutzer verändert werden. Ein *Dienst* ist somit aus Sicht des Dienstnutzers

- ein von einem Anbieter konkret bereitgestelltes reaktives autonomes Teilsystem, dessen Lebenszyklus von dem des Aufrufer unabhängig ist und das vom Anbieter über einen längeren Zeitraum zur Verfügung gestellt wird,
- durch den Austausch von Nachrichten, die dem Diensttyp genügen, aufzurufen und zu steuern,
- über eine eindeutig definierte Schnittstelle – den Diensttyp – nach außen sichtbar,
- mit einem klar definierten und im Rahmen technischer Möglichkeiten vorhersehbares Verhalten ausgestattet und
- in der Lage, weitere Dienste zur Erfüllung seiner Aufgabe zu nutzen.

Zum Teil werden dem Diensttyp noch qualitative Attribute, wie etwa die Dienstgüte, zugeordnet (Quality of Service), die in obiger Definition jedoch nicht aufgeführt wurden. So ist für (Merz, 1999, S. 152) der Typ eines Dienstes durch seinen *Typbezeichner*, durch seine *Schnittstelle* und durch *Dienstattribute*, die Qualitätsmerkmale angeben, gegeben.

Ein Dienstanbieter quittiert normalerweise den Erhalt einer Nachricht mit einer eigenen Nachricht nach Abarbeitung des Dienstes. Dies ist für Dienste, die Ergebnisse zurückliefern, klarerweise der Fall. Es bedienen sich jedoch aus technischer Sicht alle gängigen Standards für Dienste einer Kommunikationsinfrastruktur, die den Erhalt einer Nachricht zumindest quittiert[5]. Ansonsten könnte der Aufrufer nicht feststellen, ob der Aufruf eines Dienstes überhaupt erfolgt ist oder vielleicht ein Ausfall der Infrastruktur dies verhindert hat. Aus diesem Grund ist es gerechtfertigt, das Versenden von Daten an einen Dienstes stets durch eine ausgehende Nachricht und durch eine eingehende Nachricht zu modellieren, wobei letztere entweder wiederum Daten enthält oder nur angibt, dass der Dienst aufgerufen wurde.

Die Daten in einer Nachricht können beliebig komplex sein[6]. Auch können mehrere Nachrichten an einen Dienst geschickt werden, die dann jedoch einzeln quittiert werden. Es wird daher später jede Kommunikation eines Dienstes durch eine eingehende und eine ausgehende Nachricht dargestellt werden, die nicht notwendigerweise Daten enthalten.

[5] Die meisten Systeme lösen eine Ausnahme aus, wenn ein System nicht erreichbar ist, oder wenn keine Quittung innerhalb einer gewissen Zeitspanne auf eine ausgehende Nachricht zurückkommt.

[6] Es ist bei Web Services durchaus möglich, beliebige Anhänge, wie etwa ganze Archive oder Videos, als Parameter an Dienste weiterzugeben.

2.3.2 Service-orientierte Architekturen (SOA)

Softwarearchitekturen beschreiben die Komponenten eines Softwaresystems und ihr Zusammenspiel auf einer verhältnismäßig abstrakten Ebene. In *service-orientierten* oder *dienstorientierten Architekturen* (SOAs)[7] werden diese Komponenten durch Dienste dargestellt. Was genau die Bestandteile einer dienstorientierten Architektur sind wird von unterschiedlichen Autoren verschieden bewertet. Dostal u. a. (2005) (S. 11) etwa geben eine Definition, die in besonderem Maße die Kapselung heterogener Systeme durch Dienste betont:

> „Unter einer SOA versteht man eine Systemarchitektur, die vielfältige, verschiedene und eventuell inkompatible Methoden oder Applikationen als wiederverwendbare und offen zugreifbare Dienste repräsentiert und dadurch eine plattform- und sprachunabhängige Nutzung und Wiederverwendung ermöglicht."

Generell sind die lose Kopplung, die Integration heterogener Systeme durch Kapselung anhand der Dienste und die Verwendung von Standards die wichtigsten Aspekte dienstorientierter Architekturen. Innerhalb solcher Architekturen werden Dienste meist durch Ereignisse gesteuert. Newcomer und Lomow (2005) verwenden hier das Bild eines Fließbandes, bei dem die Dienste die verschiedenen Stationen darstellen (S. xix). Die verwendeten *Fachobjekte* sind dabei die Artefakte, die durch die Dienste bearbeitet werden. Dieses Bild visualisiert die zugrundeliegenden Konzepte recht gut, wenn auch die Flexibilität und Wiederverwendbarkeit der Elemente dienstorientierter Architekturen in diesem Bild zu kurz kommt.

Dienstorientierte Architekturen zeichnen sich durch verschiedene konzeptuelle Unterscheidungen aus, die im folgenden näher erläutert werden. Die Darstellung folgt hierbei verschiedenen Autoren wie (Dostal u. a., 2005; Krafzig u. a., 2005; Erl, 2005; Newcomer und Lomow, 2005).

Elemente einer SOA

Die wesentlichen in einer dienstorientierten Architektur unterschiedenen Elemente sind einerseits die verschiedenen Rollen, die die Beteiligten in einer SOA annehmen können und andererseits die Aktionen, die zwischen diesen Rollen ausgeführt werden können. Mögliche Rollen in einer SOA sind die bereits im Zusammenhang mit Diensten erwähnten Rollen des *Dienstanbieters* und die des *Dienstnutzers* sowie die Rolle des *Verzeichnisdienstes* (engl. *(Service) Registry*). Der Verzeichnisdienst ist für die Vermittlung zwischen Dienstanbieter und Dienstnutzer verantwortlich. Abbildung 2.6 gibt eine schematische Übersicht der Dienstvermittlung in einer dienstorientierten Architektur.

[7] Im weiteren Verlauf wird der Term „dienstorientierte Architektur" verwendet, der jedoch ebenfalls mit SOA abgekürzt wird.

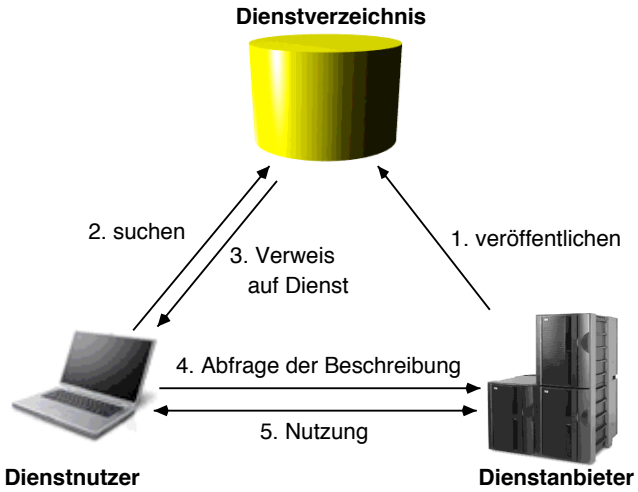


Abbildung 2.6: Dienstvermittlung anhand des Dienstverzeichnisses einer SOA

Eine *Dienstbeschreibung* ist die notwendige Grundlage dafür, dass ein Dienst überhaupt gefunden werden kann. Was genau der Inhalt einer Dienstbeschreibung ist, ist hierbei nicht näher spezifiziert. Generell wird zumindest die Schnittstelle eines Dienstangebotes auf syntaktischer Ebene anhand der Art der Operationen und ihrer Parameter festgelegt.

Die Dienstvermittlung unterscheidet die folgenden drei Rollen.

Ein Dienstanbieter stellt eine Implementation eines Dienstes bereit. Damit Nutzer auf diesen Dienst zugreifen können, muss der Dienstanbieter die Dienstbeschreibung bei einem Verzeichnisdienst veröffentlichen und auf dem neuesten Stand halten. Die Realisierung des Dienstes kann hierbei beliebig geschehen. So kann ein Dienstanbieter seinen Dienst unter Verwendung anderer Dienste realisieren und einem anderen Dienst gegenüber wiederum als Dienstnutzer auftreten. Dies zeigt, dass eine Rolle immer vom Betrachtungskontext abhängig ist. Der Dienstanbieter kann zudem noch weitere Informationen wie etwa die Qualität eines Dienstes oder die Verfügbarkeit veröffentlichen. Die Einhaltung obliegt dabei dem jeweiligen Dienstanbieter.

Ein Dienstnutzer benötigt eine bestimmte Funktionalität, die der Spezifikation eines Dienstes entspricht. Nachdem ein Dienstnutzer über das Dienstverzeichnis einen Dienstanbieter gefunden hat, kann er den Dienst an diesem Anbieter direkt aufrufen. Hierzu ruft er beim Dienstanbieter die von der Dienstspezifikation vorgegebene Sequenz von Operationen auf. Durch die Verwendung des Dienstverzeichnisses weiß der Dienstnutzer im Allgemeinen im Vorfeld nicht, welchen Dienst er aufrufen wird. Durch die lose Bindung kann somit erst zur

Laufzeit entschieden werden, welcher Dienst aufgerufen wird.

Ein Dienstverzeichnis stellt die Vermittlungsinstanz zwischen dem Dienstnutzer und dem Dienstanbieter dar. Dienstanbieter können hier ihre Dienstbeschreibung hinterlegen. Dienstnutzer hingegen können anfragen an das Verzeichnis stellen und nach Anbietern von Diensten suchen. Da es sich hierbei wiederum um einen Dienst handelt, wird auch vom *Verzeichnisdienst* gesprochen. Die Möglichkeiten der Suche und ihre Komplexität hängen vom Anbieter des Verzeichnisdienstes und von der Beschreibung der Dienste ab.

Die in Abbildung 2.6 dargestellte Übersicht ergibt somit folgenden Ablauf. Zunächst (1) teilt ein Dienstanbieter dem Dienstverzeichnis mit, dass er einen oder mehrere Dienste anbieten möchte, indem er die entsprechenden Dienstbeschreibungen veröffentlicht. Hiernach (2) kann der Dienst jetzt von einem Dienstnutzer durch eine Suchanfrage gefunden werden. Handelt es sich bei einem Dienst um den gewünschten Dienst, bekommt der Dienstnutzer vom Dienstverzeichnis einen Verweis (3) auf den Dienstanbieter und kann dort nun die (4) Dienstbeschreibung abrufen. Hierauf kann der Dienst vom Dienstnutzer verwendet werden (5).

Der Enterprise Service Bus (ESB)

Da Dienste nicht notwendigerweise dieselben Technologien verwenden müssen, andererseits jedoch eine Interoperabilität zwischen den verschiedenen Ansätzen gewährleistet bleiben soll, bieten sich prinzipiell zwei Möglichkeiten: (1) Jeder Dienstnutzer muss die Technologie des Dienstanbieter beherrschen, dessen Dienste er nutzen möchte oder (2) durch eine zusätzliche Schicht wird von Implementationsdetails abstrahiert. Letzteres erlaubt es einem Dienstnutzer, eine zusätzliche Instanz zur Vermittlung der unterschiedlichen Technologien heranzuziehen. Dies wird häufig auch als Bus bezeichnet, weswegen das logische Konzept hierzu als *Enterprise Service Bus* (*ESB*) bezeichnet wird.

Der Enterprise Service Bus ist ein offener Nachrichtenbus, der verschiedene Standards unterstützt. In Abbildung 2.7 wird das Modell des ESB als zentraler Bestandteil der Nachrichtenkommunikation dargestellt. Aufgrund des ESB können Dienste, die JMS oder J2EE verwenden mit solchen kommunizieren, die WSDL, .NET oder CORBA. Um dies leisten zu können, muss der ESB

- die transparente Transformation von Daten in verschiedene Darstellungen unterstützen,
- die transparente Transformation von einem Protokoll in ein anderes ermöglichen und
- die Nachrichten intelligent weiterleiten, um nicht zum Flaschenhals zu werden.

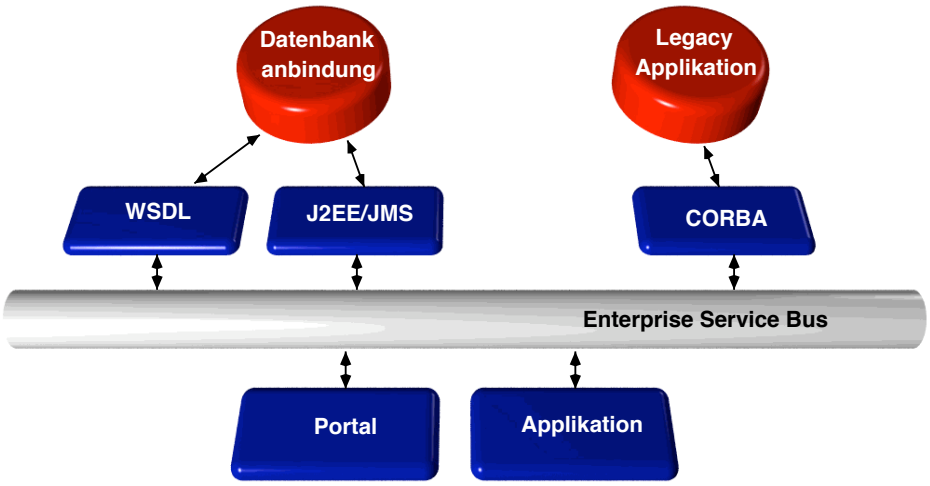


Abbildung 2.7: Der Enterprise Service Bus
In Anlehnung an (Krafzig u. a., 2005) und (Papazoglou u. a., 2006)

Der ESB besitzt somit eine Funktion, die der Funktion nachrichtenorientierter Middleware ähnelt. Ziel ist es, virtuelle Kanäle zwischen einem Dienstanbieter und seinem Nutzer zu schaffen. Diese Kanäle werden auch von Message Brokern bereitgestellt, wie etwa im Rahmen von JMS (Hapner u. a., 2002). Ein ESB ist zudem häufig auch in der Lage, verschiedene Nachrichtenformate ineinander zu übersetzen, Nachrichten in Warteschlangen vorzuhalten oder Nachrichten zu persistieren, um im Fall eines Systemabsturzes Datenverlust (bzw. Nachrichtenverlust) zu vermeiden. Die Verwendung einer zentralen Instanz zur Nachrichtenweiterleitung und die sich daraus ergebende Sterntopologie reduziert die Anzahl der zwischen den Diensten bestehenden Kommunikationsverbindungen gegenüber einer Netzwerktopologie und erleichtert die Übersetzung der Datenformate. Der potentiell durch eine zentrale Instanz entstehende Flaschenhals wird häufig durch eine physikalische Verteilung des logischen Busses umgangen. Dies unterscheidet einen ESB von den Hub-and-Spoke-Architekturen früherer Systeme im Bereich der Enterprise Application Integration (EAI).

Weitere Anforderungen an einen ESB sind häufig noch die Überprüfung von Policies und Sicherheitsbestimmungen (vgl. Bajaj u. a., 2006), die Fehlertoleranz und die Unterstützung verteilter Transaktionen. Hinzu kommen meist noch Loggingfunktionalitäten und die Möglichkeiten der Skalierbarkeit des Systems (vgl. Krafzig u. a., 2005).

Die Überprüfung von Policies dient der Wartbarkeit und der Sicherheit von Systemen und erlaubt es, die Dienstnutzer eines Dienstes einzuschränken. Häufig ist es nicht gewollt, dass jeder Dienst jeden anderen Dienst aufrufen kann, da dies

die Struktur der Software unübersichtlich macht und zu Sicherheitsrisiken führen kann. Aus dem gleichen Grund gibt es auch im Rahmen der objektorientierten Programmierung die Möglichkeit, die Sichtbarkeit von Attributen und Methoden eines Objekts einzuschränken. Im Bereich der ESBs haben sich Policies etabliert, die den Zugriff auf einen Dienst einschränken (vgl. Bajaj u. a., 2006) und so sicherstellen, dass nur autorisierte Dienstnutzer einen Dienst verwenden.

Dienstarten

Gerade bei der Dienstkomposition werden häufig verschiedene Dienstarten unterschieden. So unterscheidet OWL-S (Martin u. a., 2004a,b) etwa atomare (atomic) und zusammengesetzte (composite) Dienste. Letztere bedienen sich anderer Dienste, um ihre Aufgaben zu erledigen, während erstere dies nicht tun. Eine etwas differenziertere Einteilung bieten (Krafzig u. a., 2005, S. 69ff), die hier um Informationsdienste weiter ergänzt worden ist:

Informationsdienste sind einfache zustandslose Dienste, die lediglich der Abfrage eines Wertes oder eines Zustandes dienen.

Basisdienste sind einfache datenzentrierte oder logikzentrierte Dienste. Sie sind ausschließlich Dienstanbieter und sind entweder fachlicher Art (domänenspezifische Dienste) oder technischer Art (Sicherheit, Logging oder Persistenz). Sie sind zudem zustandslos und kapseln häufig Backend-Funktionalität, die sich in verschiedenen Anwendungsszenarien wiederverwenden lässt.

Vermittelnde Dienste (Konnektoren) kapseln andere Dienste und sind sowohl Dienstanbieter als auch Dienstnutzer. Sie bilden eine technische oder konzeptionelle Brücke zwischen Diensten und stellen Adapter, bestimmte Sichten oder Gateways dar. Sie dienen vor allem dazu, Technologie- oder Designunterschiede auszugleichen. Sie sind in der Regel zustandslos, nicht jedoch notwendigerweise der Dienst, den sie kapseln.

Prozesszentrierte Dienste steuern die Kernlogik eines Workflows und modellieren anwendungsspezifische Abläufe. Sie besitzen Prozesslogik und fungieren somit sowohl als Dienstanbieter als auch als Dienstnutzer. Durch eine dynamische Auswahl der aufgerufenen Dienste können sie Ausfall oder Überlastung eines Dienstes kompensieren. Diese Dienste sind meist anwendungsabhängig und besitzen häufig einen internen Zustand[8].

Öffentliche Unternehmensdienste sind Dienste, die von außerhalb des Unternehmens aufgerufen werden können. Sie lösen entweder Ereignisse aus oder rufen ihrerseits wiederum interne Dienste auf.

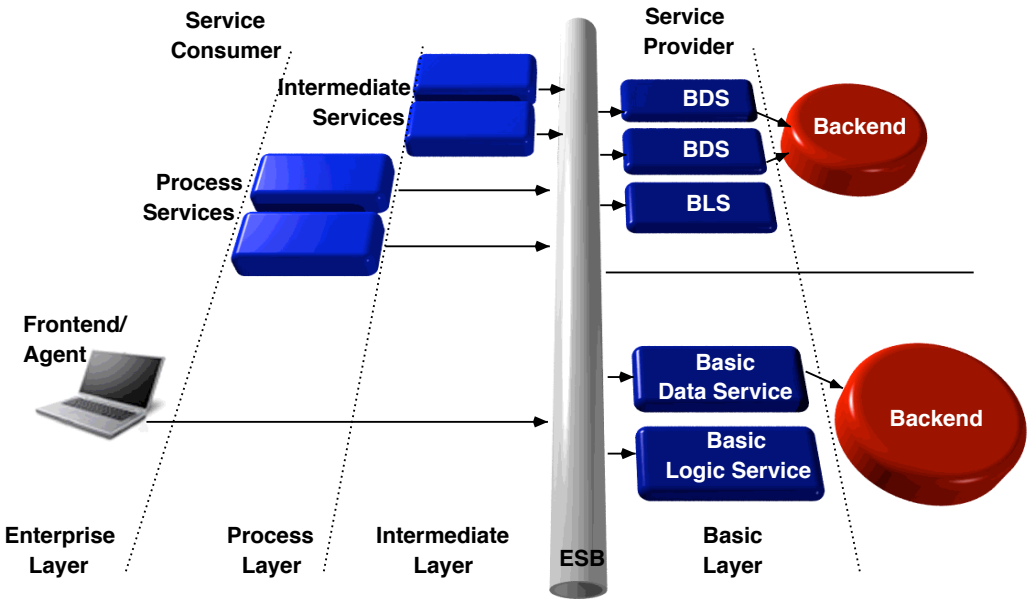


Abbildung 2.8: Die Dienstarten einer dienstorientierten Architektur
Angelehnt an Krafzig u. a. (2005)

Prozessorientierte Dienste werden auch als *zusammengesetzte Dienste* oder *komplexe Dienste* bezeichnet. Basisdienste werden hingegen auch als *atomare Dienste* bezeichnet. Atomare Dienste bestehen hierbei aus einem Aufruf, einem eventuellen Zustandsübergang der Umwelt und einem Zurücksenden einer Nachricht. Wird keine Zustandsänderung der Umwelt vorgenommen, so ist der Dienst ein *Informationsdienst*, andernfalls ist er ein *zustandsverändernder Dienst*. Der Zustand, der hierbei verändert wird, ist der Zustand der Umwelt respektive der Zustand des Systems. Zusammengesetzte Dienste bedienen sich mehrerer atomarer oder auch zusammengesetzter Dienste, um ihre Aufgaben zu erfüllen. Komplexe oder zusammengesetzte Dienste entsprechen somit den strukturierten Aktivitäten. Sie bilden eine komplexe Struktur von Diensten, die über ein Workflowschema miteinander koordiniert werden. Eine Übersicht über die verschiedenen Dienstarten liefert Abbildung 2.8.

[8] Dies bedeutet, dass während verschiedener Aufrufe von Diensten ein interner Zustand verwaltet wird. Die Kommunikation zwischen Diensten ist jedoch weiterhin zustandslos.

2.3.3 Interaktion in dienstorientierten Architekturen

Nachdem der letzte Unterabschnitt dargestellt hat, wie eine SOA im Allgemeinen aufgebaut ist, soll in diesem Unterabschnitt noch einmal die Interaktion der verschiedenen Dienste untersucht werden. Dies umfasst einerseits die Kommunikation zwischen den Diensten und ihren Aufrufern und andererseits die Ereignisse und die Regeln, über die Dienste initiiert werden. Neben diesen beiden Elementen sind auch die beschränkenden Elemente wichtig. So sorgen Policies dafür, dass Unternehmensrichtlinien eingehalten werden.

Konversationen

Im Gegensatz zu Objekten werden Dienste nicht aus dem System heraus erzeugt. Sie werden vielmehr einem System bereitgestellt und bieten verschiedenen Dienstnutzern eine gewisse Funktionalität an. Wie in Abschnitt 2.3.1 bereits angesprochen besitzen Dienste zwar einen allgemeinen Zustand, meistens jedoch keinen aufruferspezifischen Zustand oder gar eine Instanz wie ein Objekt. Da es häufig notwendig ist, mehrere Aufrufe bei einem oder mehreren Diensten als Einheit zu identifizieren, werden bei jedem Aufruf eindeutige Token mit übergeben, so dass ein Dienst einen Aufrufer identifizieren kann.

Anhand dieses Tokens kann ein Dienst erkennen, dass die aktuelle Nachricht zu vorherigen Nachrichten in Beziehung steht. Dies ist auch für mehrere verschiedene Dienste möglich, so dass auf diese Weise bei einem Dienstaufruf spezifiziert werden kann, welche Dienste der Aufrufer bereits wie aufgerufen hat. Hiervon abhängig kann nun der Dienst ausgeführt werden. Der Aufruf von Diensten besitzt wie der Abruf einer Webseite von einem Webserver einen sehr starken Request/Reply-Charakter (vgl. Dunkel u. a. (2008)). Der Nachrichtenaustausch geschieht dabei über eine Message oriented Middleware, die dafür sorgt, dass Dienste korrekt angesprochen werden und dass Folgeaufrufe korrekt identifiziert werden können. Diese Middleware ist häufig ebenfalls Teil des ESB.

Ereignisse und Regeln

Die Aktivierung von Diensten erfolgt in dienstorientierten Architekturen in der Regel durch Ereignisse. So werden ereignisgesteuerte Architekturen (Event Driven Architectures (EDA)) auch häufig als Erweiterung dienstorientierter Architekturen gesehen (vgl. Luckham (2008) und Taylor u. a. (2009)). Andererseits ist die Ereignissteuerung nicht notwendigerweise an Dienste gekoppelt (vgl. Mühl u. a. (2006)). Viele moderne SOA-Ansätze legen jedoch einen starken Fokus auf Ereignisse und auch in vielen Modellierungsansätzen für Geschäftsprozesse nehmen Ereignisse einen hohen Stellenwert ein (vgl. EPKs und BPMN in Abschnitt 2.2.3). Auch der in dieser Arbeit verfolgte Ansatz verwendet Ereignisse, um die Bearbeitung eines Vorgangs durch einen Dienst initial zu aktivieren.

Ereignisse werden in EDAs als Initiator für einen Ablauf betrachtet. Beispiele für Ereignisse sind der Eingang einer Warenlieferung, das Eintreffen einer Bestellung oder auch das Eintreten eines kritischen Zustands wie die Unterschreitung des minimalen Warenbestandes. Jedes dieser Ereignisse wird anhand von Regeln (Business Rules) entsprechenden Diensten zugeordnet, die dann die Steuerung eines komplexen Ablaufs übernehmen.

Zur Verarbeitung komplexer Ereignisse (Complex Event Processing (CEP)) dienen Regeln wie von Luckham (2008) dargestellt wird. Diese erlauben es, Ereignisse zu klassifizieren und durch den Aufruf entsprechender (komplexer) Dienste zu bearbeiten. Um das proaktive Verarbeiten von Ereignissen zu unterstreichen führen McGovern u. a. (2006) in diesem Zusammenhang die Unterscheidung zwischen Agenten und Diensten ein. Agenten verarbeiten Ereignisse anhand von Regeln und weisen somit proaktives Verhalten (und unter Umständen auch Adaptivität) auf. Zur Erfüllung der durch Ereignisse angesteuerten Abläufe nutzen die Agenten dann wiederum Dienste, die das System ihnen bereitstellt.

Die Verarbeitung komplexer Ereignisse erfordert in der Regel zusätzliches Domänenwissen. Komplexe Ereignisse können dabei etwa das Erreichen von bestimmten Kursen am Devisenmarkt bei einer gleichzeitigen Bewegung des Anleihemarktes sein. Solche Ereignisse triggern dann spezielle Prozesse (gekapselt durch Dienste), die auf die neue Situation angemessen reagieren. Um Regeln komplexer Ereignisse verarbeiten zu können, nennen McGovern u. a. (2006) zudem die Notwendigkeit einheitlicher Konzepte, die durch eine Ontologie systemweit vorgegeben werden können. Eine detaillierte Diskussion der Nutzung von Ontologien ist Gegenstand des nächsten Abschnitts.

Policies

Das Management und die Umsetzung externer und interner Unternehmensrichtlinien wird meist als Governance bezeichnet. Governance umfasst dabei sowohl die Umsetzung gesetzlicher Vorgaben als auch die Überwachung unternehmenskritischer Vorgänge. Policies dienen dazu, die durch das Governance-Modell vorgegebenen Richtlinien umzusetzen und automatisiert zu überwachen, sofern dies möglich ist. Eine detaillierte Schilderung von Governance in dientbasierten Systemen findet sich in Biske (2008).

Durch die zentrale Aufgabe, Sicherheitrichtlinien und allgemeine Bedingungen gewährleisten zu müssen, werden Policies häufig als wesentlicher Bestandteil einer SOA angesehen. Sie überwachen das Verhalten von Diensten und sorgen dafür, dass nicht-erwünschte Aktionen nicht ausführbar sind. Eine detailliertere Erläuterung hierzu findet sich bei Hondo u. a. (2008) oder bei Biske (2008).

2.4 Repräsentation von Konzepten und Funktionalität

Für den im Rahmen dieser Arbeit verfolgten Ansatz sind von den in Abschnitt 2.2.2 betrachteten Sichten auf Geschäftsprozesse neben der Kontrollflusssicht vor allem die Daten- und die Funktionssicht relevant. Zur Beschreibung der Datensicht werden häufig *Fachobjekte* (auch Business-Objekte) herangezogen, die die wesentlichen Konzepte einer Organisation darstellen. Fachobjekte werden durch *Fachklassen* angegeben, die darstellen welche Attribute ein bestimmtes Fachobjekt besitzt.

Zur formalen Darstellung von Fachobjekten soll im Rahmen dieser Arbeit eine explizite Konzeptionalisierung dieser Daten in Form einer Ontologie verwendet werden. Im Ablauf werden diese Objekte dann durch eine Algebra dargestellt. Mögliche Operatoren auf den Konzepten der Ontologie werden durch die Operatoren dieser Algebra dargestellt. Aus diesem Grund widmet sich dieser Abschnitt zunächst der Darstellung von Konzepten durch Ontologien, um im Anschluss die Repräsentation von Funktionalität durch Algebren und den darauf aufbauenden abstrakten Zustandsmaschinen zu untersuchen.

2.4.1 Ontologien

Beim Austausch von Nachrichten zwischen Dienstanbieter und Dienstnutzer muss sichergestellt sein, dass der Inhalt einer Nachricht auf beiden Seiten gleich interpretiert wird. Dies lässt sich durch ein gemeinsames Vokabular beziehungsweise durch Übersetzungsregeln von einem Vokabular in ein anderes erreichen. Die Explikation der Bedeutung der auszutauschenden Daten geschieht anhand einer Ontologie.

Der Begriff der Ontologie stammt aus der Philosophie und beschreibt dort ursprünglich die Lehre vom Sein (vgl. Zúñiga, 2001). Der Begriff ist zunächst im Bereich der künstlichen Intelligenz und der Sprachverarbeitung aufgegriffen worden und wird mittlerweile ebenfalls im Rahmen verteilter Systeme und im Rahmen von Datenbanksystemen verwendet. In diesen Bereichen hat der Begriff jedoch viel von seiner ursprünglichen Bedeutung eingebüßt und wird nun vielmehr als Bezeichnung für eine „explizite Spezifizierung einer Konzeptualisierung“ (Gruber, 1993) verwendet. Eine Ontologie dient dazu, Konzepte explizit und auf einer formalen Grundlage aufbauend zu spezifizieren. Gruber (1993) rechtfertigt den Begriff Ontologie damit, dass das, was für ein Computersystem (an Umwelt) existiert, das ist, was repräsentiert werden kann.

Grundlegende Bestandteile sämtlicher Definitionen einer Ontologie sind Konzepte, Objekte (Instanzen von Konzepten) und Beziehungen zwischen diesen beiden (vgl. Chandrasekaran u. a., 1999; Guarino, 1998; Sowa, 2000; Staab und Studer, 2004). Zusammenfassend lässt sich sagen, dass eine Ontologie die Objekte einer Domäne anhand von Eigenschaften beschreibt, wobei sich diese Objekte verschiedenen

Kategorien zuordnen lassen. Konzepte sind durch eine partielle Ordnung (Subtypbeziehung) geordnet und spezifizieren die Eigenschaften ihrer Objekte und die Art der Beziehung zu anderen Objekten. Die Spezifikation einer Ontologie geschieht im Allgemeinen anhand von logischen Ausdrücken.

Der Fokus der Ontologiebetrachtung im Rahmen dieser Arbeit liegt auf dem Kommunikationsaspekt, was bedeutet, dass klar entscheidbar sein muss, welche Beziehungen ein Objekt erfüllen muss, um einem Konzept anzugehören. In Anlehnung an Beschreibungslogiken (vgl. Baader u. a., 2002) wird Motik u. a. (2002) und Hotho u. a. (2003) folgend zwischen der *terminologischen Struktur* und der *assertorischen Struktur* einer Ontologie unterschieden. Während erstere die Konzepte und ihre Beziehungen beschreibt, gibt letztere die Zugehörigkeit von Objekten zu Konzepten und die konkreten Beziehungen zwischen einzelnen Objekten an. Im Kontext von Abläufen ist diese Unterscheidung deshalb wichtig, weil die terminologische Struktur als zeitinvariant angesehen wird, während die assertorische Struktur sich ändern kann[9].

Definition 2.2 (Terminologische Struktur)
Eine *terminologische Struktur* ist ein Tupel $\mathfrak{T} \stackrel{\text{def}}{=} (\mathfrak{C}, \mathfrak{R}, \preceq_{\mathfrak{C}}, \mathfrak{ar}, \mathfrak{A}_{\mathfrak{X}}, \top)$, wobei

- $\mathfrak{C}$ eine Menge von Konzeptsymbolen ist,
- $\mathfrak{R}$ eine Menge von Relationssymbolen ist,
- $\preceq_{\mathfrak{C}} \subseteq \mathfrak{C} \times \mathfrak{C}$ eine partielle Ordnung – die *Konzepthierarchie* – auf $\mathfrak{C}$ ist,
- $\mathfrak{ar} : \mathfrak{R} \to \mathfrak{C} \times \mathfrak{C}$ die Arität (Sortigkeit) der Relationen angibt,
- $\mathfrak{A}_{\mathfrak{X}}$ eine Menge von logischen Axiomen in einer beliebigen Logik ist und
- $\top$ das Wurzelkonzeptsymbol der Ontologie ist, so dass $\forall \mathfrak{c} \in \mathfrak{C} : \mathfrak{c} \preceq^*_{\mathfrak{C}} \top$.

♦

Eine terminologische Struktur weist große Ähnlichkeit zu einer logischen Struktur auf. Dies resultiert daraus, dass alle existenten Formalisierungen einer Ontologie auch auf die Prädikatenlogik (gegebenenfalls unter Verwendung von Konstrukten zweiter Ordnung) abbildbar sind. In obiger Definition wäre es durchaus möglich gewesen, die partielle Ordnungen bezüglich der Konzepte ebenfalls anhand der Axiome auszudrücken, jedoch betont obige Definition die entscheidende Bedeutung der Konzepthierarchie[10].

Im Gegensatz zu einer reinen Taxonomie erlaubt es eine terminologische Struk-

[9] Dies muss nicht so sein, wird jedoch im Rahmen dieser Arbeit so angenommen. Dem Autor sind keine Ansätze bekannt, bei denen Dienste die terminologische Struktur einer Ontologie ändern.

[10] Die obige Definition weist eine relativ starke Ähnlichkeit zur Beschreibungslogik (Brachman und Schmolze, 1985; Baader u. a., 2002) und zur F-Logik (Kifer u. a., 1995) wie auch zur ordnungssortierten Prädikatenlogik auf. Dies begründet sich daraus, dass diese drei Formalismen die gängigsten Repräsentationen von Ontologien sind.

tur, zusätzliche Kriterien anhand der Axiome anzugeben, die für die Zugehörigkeit zu einem Konzept gelten müssen. Diese Kriterien können etwa besagen, dass ein Konzept invers zu einem anderen ist und ein Objekt genau dann in einem Konzept enthalten ist, wenn es in einem anderen nicht enthalten ist. Eine Ontologie wird folglich anhand eines Axiomensystems repräsentiert, dass die in einer Begriffswelt geltenden Regeln formalisiert.

Anders als in logischen Formeln im Allgemeinen machen die Axiome keine Aussagen über konkrete Dinge der Domäne, sondern beschreiben vielmehr allgemeingültige Zusammenhänge. Axiome stellen somit eine Menge von allquantifizierten oder existenzquantifizierten Aussagen über die Domäne dar. Dabei gibt es auch Formalismen, die die Definition konkreter Domänen erlauben und sogar die Verwendung von Operationen hierauf, die eigentlich nicht Teil der terminologischen Struktur sind. Konkrete Domänen sind häufig Zahlen und Zeichenketten, mit Hilfe derer etwa ausgedrückt werden kann, dass beispielsweise alle Erwachsenen älter als 18 sind.

Definition 2.3 (Assertorische Struktur, Gültigkeit)
Eine *assertorische Struktur* zu einer terminologischen Struktur $\mathfrak{T}$ ist ein Tupel $\mathfrak{A}_\mathfrak{T} \stackrel{\text{def}}{=} (\mathfrak{Ci}, \cdot^\mathfrak{I})$, wobei

- $\mathfrak{Ci}$ eine Menge von Konzeptinstanzsymbolen ist und
- $\cdot^\mathfrak{I}$ ein Paar von Abbildungen $\cdot^\mathfrak{I} \stackrel{\text{def}}{=} (\cdot^{\mathfrak{I}_\mathfrak{c}}, \cdot^{\mathfrak{I}_\mathfrak{R}})$ ist, so dass
  - $\cdot^{\mathfrak{I}_\mathfrak{c}} : \mathfrak{C} \to 2^{\mathfrak{Ci}}$ jedem Konzeptsymbol eine Menge von Instanzsymbolen zuweist mit $\forall \mathfrak{c}_1, \mathfrak{c}_2 \in \mathfrak{C} : \mathfrak{c}_1 \preceq_\mathfrak{c} \mathfrak{c}_2 \implies {\mathfrak{c}_1}^{\mathfrak{I}_\mathfrak{c}} \subseteq {\mathfrak{c}_2}^{\mathfrak{I}_\mathfrak{c}}$,
  - $\cdot^{\mathfrak{I}_\mathfrak{R}} : \mathfrak{R} \to 2^{\mathfrak{Ci} \times \mathfrak{Ci}}$ jedem Relationssymbol eine Menge von Instanzsymboltupeln zuweist mit $\forall \mathfrak{r} \in \mathfrak{R} : (\mathfrak{ci}_1, \mathfrak{ci}_2) \in \mathfrak{r}^{\mathfrak{I}_\mathfrak{R}} \implies (\mathfrak{ci}_1, \mathfrak{ci}_2) \in \mathfrak{ar}(r)^{\mathfrak{I}_\mathfrak{c}}$[11],
  - $\top^{\mathfrak{I}_\mathfrak{c}} = \mathfrak{Ci}$ ist.

Eine assertorische Struktur $\mathfrak{A}_\mathfrak{T}$ *genügt* einer terminologischen Struktur $\mathfrak{T}$, wenn sie der Menge der Axiome $\mathfrak{Ax}$ und der Konzepthierarchie von $\mathfrak{T}$ genügt. In diesem Fall ist $\mathfrak{A}_\mathfrak{T}$ *gültig*. ♦

Es wird im Folgenden die Unterscheidung zwischen $\cdot^{\mathfrak{I}_\mathfrak{c}}$ und $\cdot^{\mathfrak{I}_\mathfrak{R}}$ nur dann getroffen, wenn diese nicht aus dem Kontext hervorgeht. Stattdessen wird stets $\cdot^\mathfrak{I}$ notiert.

Führt man die Analogie mit Logiken fort, so entspricht die assertorische Struktur der Interpretation einer logischen Struktur. Im Rahmen von Ontologien existieren jedoch zunächst keine Terme. Stattdessen werden die Konstanten als Instanzen bezeichnet. Die Existenz von Termen wird jedoch notwendig sobald Operatoren auf die Instanzen angewendet werden sollen. Dies wird in Kapitel 4 dargestellt.

Die Relationen einer Ontologie werden auch häufig als *Rollen* bezeichnet. Dies ist jedoch missverständlich, da der Begriff der Rolle auch zur Bezeichnung dyna-

[11] Hierbei wurde $\cdot^{\mathfrak{I}_\mathfrak{c}}$ auf Tupel erweitert, indem die Anwendung komponentenweise erfolgt.

mischer Konzepte dient. Eine Unterscheidung zwischen unären Rollen und binären Rollen würde zu verwirrend werden. Aus diesem Grund wird die Bezeichnung Rolle im Rahmen dieser Arbeit ausschließlich für dynamische Konzepte, wie sie in Abschnitt 2.4.2 erläutert werden, reserviert sein. Da der Ansatz in dieser Arbeit lediglich funktionale Rollenbeziehungen zulässt, wird stattdessen die Bezeichnung Attribut gewählt.

Es gibt Formalismen zur Darstellung von Ontologien, die eine hierarchische Ordnung auf den Relationen zulassen, so dass ein Attribut eine Subbeziehung zu einem anderen Attribut sein kann. Dies erhöht jedoch die Komplexität der verwendeten Formalismen, ohne einen wesentlichen Vorteil für die Modellierung zu bringen. Hiervon soll daher im Rahmen dieser Arbeit abgesehen werden. Ebenso sind im Rahmen dieser Arbeit sämtliche Konzepte und Relationen klar abgegrenzt, so dass es keine Fuzzy-Ontologien (Calegari und Ciucci, 2007) gibt.

In Anlehnung an Hotho u. a. (2003) besteht eine Ontologie aus einer terminologische Struktur $\mathfrak{T}$ und einer dazu gültigen assertorischen Struktur $\mathfrak{A}_{\mathfrak{T}}$ sowie einem Lexikon, dass die Konzepte der Ontologie mit realweltlichen Konzepten und Begriffen in Verbindung bringt, die logische Axiomatisierung also zur Realität in Beziehung setzt.

2.4.2 Beschreibung dynamischer Konzepte

Rollen sind im Zusammenhang mit Abläufen und Workflows zwar weit verbreitet, bezeichnen dabei jedoch im Allgemeinen eine oder mehrere Eigenschaften, die eine Ressource besitzen muss, um eine Aktivität verrichten zu können. Rollen als dynamisches Konzept sind jedoch allgemeiner und bezeichnen Konzepteigenschaften, die einem Objekt für einen gewissen Zeitraum zugeschrieben werden. Diese Eigenschaften können auch rein auf Konventionen beruhen und müssen nicht anhand von Attributen erkennbar sein[12]. Da der Begriff der Rolle in vielen verschiedenen Kontexten unterschiedlich interpretiert wird, wird in dieser Arbeit meist von *dynamischen Konzepten* in Abgrenzung zu *statischen Konzepten* gesprochen. Statische Konzepte werden häufig auch als (natürlicher) *Typ* bezeichnet.

Die Verwendung dynamischer Konzepte ist bei der Modellierung von Diensten wenig verbreitet. Gerade in heterogenen Systemen stellt jedoch die Möglichkeit, ein Objekt aufgrund dynamischer Konzeptzugehörigkeit zu verschiedenen Zeitpunkten in verschiedenen Kontexten einzusetzen, ein wichtiges Merkmal dar und sollte entsprechend im Modell abgebildet werden können. Zudem gibt es eine Vielzahl von Aktionen in einem Workflow, die die Rollen eines Konzeptes verändern. So kann etwa eine Aktion *Immatrikulieren* ein Objekt vom Konzept *Person* mit einer zusätzlichen Rolle *Student* versehen. Rollen sind also ein wichtiges Mittel zur

[12] Die Eigenschaft, dass bestimmte Münzen als Währung gelten, stellt auch eine temporäre Eigenschaft und somit eine Rolle der Münzen dar.

Beschreibung von Objekten und ihrem Verhalten in Abläufen.

Im Bereich der Ontologien haben sich Guarino und Welty (2000) der Problematik der dynamischen Konzepte angenommen, deren Ziel dabei jedoch eher die Konsistenzprüfung einer gegebenen Ontologie ist. Dabei wird nicht nur zwischen statischen und dynamischen Konzepten unterschieden, sondern es werden auch Meta-Eigenschaften von Kategorien und verschiedene Arten von Rollen differenziert. Diese sollen hier jedoch nicht näher behandelt werden.

Mit dem Problem der Rollen bei der Modellierung beschäftigt sich Steimann (2000). Er hebt ein Problem hervor, das entsteht, wenn man natürliche Typen (statische Konzepte) und Rollen (dynamische Konzepte) auf derselben Ebene betrachtet. Wenn dynamische und statische Konzepte auf derselben Ebene angesiedelt sind, stellt sich die Frage, wie diese zueinander in Beziehung stehen. Es scheint zunächst, dass ein Rollentyp spezieller ist, als ein natürlicher Typ, da die Angehörigen der Rolle ja lediglich eine Teilmenge darstellen. Sei etwa eine *Person* ein statisches Konzept, so wären die Rollen *Käufer* und *Verkäufer* Subtypen, wie in Abbildung 2.9a dargestellt. Ein Objekt muss somit die Möglichkeit haben, einen Subtyp abzulegen oder wiederzuerlangen. – eine Problematik, die auch im Rahmen objektorientierter Datenbanken entsteht (vgl. Mendelzon u. a., 1994; Su, 1997; Coulondre und Libourel, 2002).

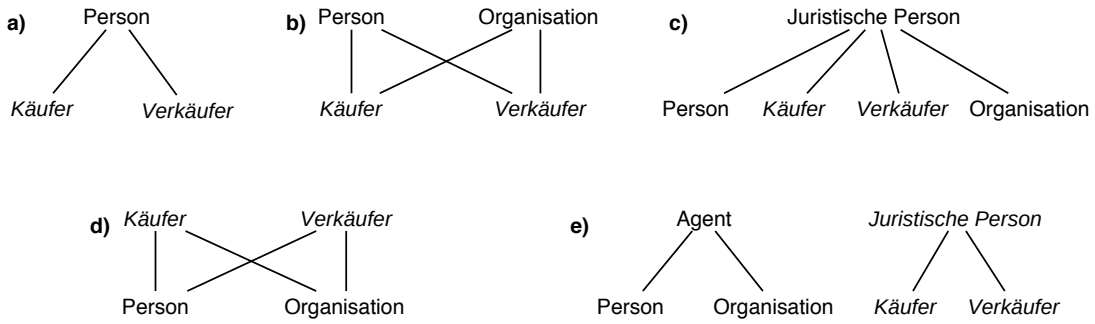


Abbildung 2.9: Modellierung von Rollen
Nach Steimann (2000)

Die Darstellung der Rollen als Subtypen des natürlichen Typs stellt jedoch in erster Linie ein Problem auf konzeptioneller Ebene dar. So können nicht nur Personen sondern auch *Organisation*en *Käufer* oder *Verkäufer* werden. Die Rollen zum Subtyp beider statischen Konzepte zu machen, wie in Abbildung 2.9b dargestellt, ist nicht sinnvoll, da die Schnittmenge der beiden Typen im Allgemeinen leer ist. Sinnvoller ist die Einführung eines gemeinsamen Supertyps von *Organisation* und *Person* wie etwa *juristische Person*. *Käufer* und *Verkäufer* sind, wie in Abbildung 2.9c zu sehen, nun beide *juristische Personen*. Dies wirft wiederum die Frage auf, welche Subtypen einer *juristischen Person* nun Rollen sind und welche nicht

und ob *juristische Person* selbst eine Rolle darstellt. Zudem stellen *Organisation* und *Person* eine statische Partitionierung des Konzeptes *juristische Person* dar, während *Käufer* und *Verkäufer* eine dynamische Aufteilung darstellen, die jedoch weder disjunkt noch vollständig sein muss.

Auf der anderen Seite stellen Rollen auch keine Supertypen dar. Eine *Person* muss nicht die Attribute eines *Verkäufer*s besitzen und ist trotzdem eine *Person* (vgl. Abbildung 2.9d). Vielmehr scheint das Rollenkonzept orthogonal zu dem des statischen Konzeptes wie in Abbildung 2.9e dargestellt.

Ein anderer Ansatz stellt Rollen als Adjunkte dar, die eine eigene Instanz besitzen und in einer `erfüllt_von`-Beziehung mit einer Instanz eines natürlichen Typen stehen. Dies ist der Mechanismus, den die meisten modernen objektorientierten Sprachen gewählt haben, wie etwa Java (Coad und Mayfield, 1997)[13]. Die dynamische Typänderung ist insbesondere im Zusammenhang mit Methoden von Objekten schwierig, da ein Objekt hierdurch nicht nur andere Attribute besitzt, sondern auch neue Methoden, die unter Umständen auf andere Attribute zugreifen. Im Bereich der objektorientierten Datenbanken besitzen Objekte im Allgemeinen keine Methoden. Hier existieren verschiedene Ansätze zur Repräsentation von Rollen, die die dynamische Typänderung eines Datenbankobjektes erlauben (vgl. Jodlowski u. a., 2003; Wong u. a., 1997; Coulondre und Libourel, 2002). Nachteil des Ansatzes, Rollen als Adjunkte darzustellen ist, dass auf diese Weise ein Objekt und seine Rollen mehrere verschiedene Identitäten besitzen. Zudem müssen Anfragen an eine Rolle häufig an das Objekt weiter dirigiert werden.

Da Rollen nicht Teil der Hierarchie statischer Konzepte sind, verfolgt Steimann (2000) den Ansatz, Rollen und natürliche Typen als orthogonal zueinander zu betrachten. Dies führt dazu, dass die Rollenhierarchie unabhängig von der Hierarchie der natürlichen Typen ist. Zudem wird eine Relation eingeführt, die natürliche Typen und Rollen miteinander in Beziehung setzt.

Um Angeben zu können, wie sich Rollen ändern können, schlagen Wieringa u. a. (1995) einen Ansatz vor, in dem der Lebenszyklus eines Objekts auch mögliche Rollen beinhaltet. Migrationsdiagramme, denen ein Automatenmodell zugrundeliegt, geben wieder, welche Rolle Voraussetzung für die Annahme einer anderen Rolle ist. Diese Überlegung wird später noch einmal im Zusammenhang mit der Analyse von Objektzuständen in Kapitel 6 aufgegriffen.

[13] Die dynamische Typisierung ist in vielen modernen objektorientierten Programmiersprachen wie Java oder C++ nicht mehr möglich, da dies häufig zu Konflikten führt, wenn auch Code vererbt wird. Sprachen wie Smalltalk, ACTOR oder CLOS erlauben es jedoch, den Typ eines Objektes zur Laufzeit zu ändern. Das Konzept der dynamischen Typisierung wird in Nierstrasz (1989); Stein u. a. (1989) näher erläutert.

2.4.3 Abstrakte Zustandsmaschinen (ASMs)

Im Rahmen dieser Arbeit werden wie im letzten Unterabschnitt dargestellten Ontologien dazu verwendet, die Konzepte in einer Organisation zu Beschreiben. Diese Konzepte werden Fachobjekte beschrieben.

Neben der Darstellung der Fachobjekte in Form einer Ontologie ist auch die Darstellung des aktuellen Zustands eines Fachobjektes notwendig. Zudem ist die Veränderung, die ein Fachobjekt beim Durchlauf durch einen Ablauf erfährt, relevant, um Entscheiden zu können, ob ein Ablauf sinnvoll ist oder nicht. Die Darstellung des aktuellen Zustands eines Fachobjektes erfolgt durch eine Algebra, die eine Signatur (die benannten Attribute des Fachobjektes) interpretiert. Dies ist verwandt mit dem Konzept der abstrakten Zustandsmaschinen, die in diesem Unterabschnitt kurz skizziert werden sollen.

Evolving Algebras (EAlgebras) sind von Gurevich (1988, 1991) vorgeschlagen worden, um Abläufe und ihre Zuständen zu modellieren, insbesondere im Rahmen von Computerprogrammen. EAlgebras sind endliche dynamische Algebren, die Zustandsübergänge repräsentieren und auf diese Weise dynamische diskrete Systeme beschreiben.

Das Konzept der Evolving Algebras ist heute meist unter dem Begriff der *abstrakten Zustandsmaschine* (*Abstract State Machine*) (vgl. Börger und Bolognesi, 2003; Börger und Stärk, 2003) zu finden. Abstrakte Zustandsmaschinen beschreiben ein Transitionssystem, dessen Zustände Algebren sind. Eine abstrakte Zustandsmaschine besitzt eine (boolesche) Signatur als *Vokabular*, wodurch auf syntaktischer Ebene definiert wird, welche Sorten und welche Operationen zwischen diesen Sorten der Maschine bekannt sind. Jeder Term wird in einem Zustand durch eine Algebra interpretiert. Transitionsregeln bestimmen den Übergang zwischen den abstrakten Zuständen. Auch wenn es eine Vielzahl von Erweiterungen (vgl. Börger und Stärk (2003)) gibt, hat die grundlegende Regel die Form

if *Guard* **then** *Updates*

Hierbei stellt *Guard* die Bedingung dar, unter der eine Regel angewandt wird. *Guard* ist eine beliebige prädikatenlogische Formel ohne freie Variablen, die zu wahr oder falsch auswertet. *Updates* ist eine endliche Menge von Zuweisungen, die die Form $f(\tau_1, \ldots, \tau_n) \stackrel{\text{def}}{=} t$ haben und deren Ausführung das parallele Verändern (bzw. Definieren) der Werte der aufgeführten Funktionen bewirkt. In einem gegebenen Zustand werden zunächst alle Parameter τ_i, τ ihren Werten entsprechend ausgewertet. Seien dies etwa $(v_1, \ldots, v_n)$ und v, so wird anschließend der Wert von $f(v_1, \ldots, v_n)$ auf v gesetzt, was den Wert im Folgezustand repräsentiert.

Das Paar von Funktionssymbol f, gegeben durch eine Signatur, und dem Tupel von optionalen Argumenten $(v_1, \ldots, v_n)$, die durch die dynamischen Parameterwerte gegeben sind, wird *Lokation loc* genannt. Lokationen entsprechen Speicherreferenzen, an die die Werte des Datentyps geschrieben werden. Lokations-Wert Paare (loc, v) werden *Updates* genannt und repräsentieren die grundlegenden Einheiten

von Zustandsänderungen. Die Operatorsymbole einer Signatur werden also je nach Lokation verschiedenen Werten einer Grundmenge zugeordnet. Betrachtet man dies als heterogene Algebra, so stellt die Grundmenge die Vereinigung der Trägermengen der Algebra dar und die Zuweisung von Werten ist durch die Interpretation der Operatorsymbole gegeben. Zustände stellen hierbei meist partielle Algebren dar, welche jedoch durch die Einführung eines speziellen Symbols $\perp\!\!\!\perp$, welches in allen Sorten vorkommt, zu totalen Algebren erweitert werden können. Eine Übersicht über die Entwicklung des Formalismus der abstrakten Zustandsmaschinen wird von Börger und Huggins (1998) gegeben.

2.5 Zusammenfassung

Dieses Kapitel hat die im Rahmen dieser Arbeit zentralen Konzepte der Geschäftsprozesse, der Dienste und der Repräsentation von Daten und Konzepten dargestellt. Zur Operationalisierung von Geschäftsprozessen werden Workflows genutzt, die sich ihrerseits wiederum durch dienstbasierte Systeme umsetzen lassen. Die Modellierung der bei der Interaktion verschiedener Dienste notwendigen Daten erfolgt anhand einer Ontologie, die auf diese Weise auch die statische Schnittstelle eines Dienstes anhand von Konzepten der Ontologie bestimmt. Neben der statischen Schnittstelle verfügt ein Dienst über eine funktionale und über eine Verhaltensschnittstelle, die anhand einer veränderlichen Algebra und anhand eines Petrinetzes dargestellt werden können. Beide werden im nächsten Kapitel daher näher erläutert.

Die verschiedenen Sichten auf einen Geschäftsprozess werden später zur Beschreibung der Notation zur Modellierung dienstbasierter Systeme wieder herangezogen. Gleichzeitig werden die existierenden Notationen herangezogen, um bestehende Vorgehensweisen übernehmen zu können.

Durch die Verwendung einer dienstbasierten Infrastruktur wird von der konkreten Kommunikation zwischen den verschiedenen an einem Geschäftsprozess beteiligten Systemen abstrahiert. Diese kann auf einem höheren Abstraktionsniveau vielmehr als Kommunikation mittels der Konzepte einer Ontologie verstanden werden, die durch einen Enterprise Service Bus entsprechend den Anforderungen der Kommunikationsteilnehmer angepasst wird.

Die Verwendung dynamischer Konzepte erlaubt die intuitivere Modellierung von Verhaltens- und Rollenänderungen bei den Fachobjekten, die modelliert werden sollen. Dies erleichtert später die Darstellung von Geschäftsprozessen. Die praktische Verwendung wird in Kapitel 7 noch einmal zusammen mit der Modellierung aufgegriffen.

3 Formalismen: Betrachtung des aktuellen Forschungsstandes

Nachdem im letzten Kapitel die grundlegenden Konzepte zur Modellierung dienstbasierter Geschäftsprozesse vorgestellt wurden, ist der Gegenstand dieses Kapitels die Betrachtung des aktuellen Forschungsstandes im Bereich der in dieser Arbeit verwendeten Formalismen und seine Einbettung in den Rahmen dieser Arbeit. Die Betrachtung konzentriert sich dabei auf die im Rahmen dieser Arbeit verwendeten Konzepte der Workflownetze und der algebraischen Datenrepräsentation.

3.1 Übersicht verwendeter Formalismen

Die Repräsentation von Daten und Datentypen in Workflownetzen soll anhand ordnungssortierter Algebren und anhand von Workflownetzen formalisiert werden. Hierzu werden in diesem Kapitel die Notationen für heterogene und ordnungssortierte Algebren wie auch für Petrinetze und Workflownetze festgelegt. Die Festlegung der Notationen erfolgt ohne detaillierte Erläuterungen, da bereits eine Reihe hervorragender Einführungen in diesem Bereich existieren. Genannt seien hier für den Bereich der heterogenen und ordnungssortierten Algebren das Buch von Ehrig u. a. (2001) und der Artikel von Goguen und Meseguer (1992). Für den Bereich der Petrinetze sei auf die Bücher von Desel und Esparza (1995), Jessen und Valk (1987) und Reisig und Rozenberg (1998), für den Bereich der Workflownetze auf die Artikel von van der Aalst (2000) sowie für den Bereich der algebraischen Netze auf den Artikel von Reisig (1991b) verwiesen.

Dieses Kapitel gibt zunächst einen Überblick über heterogene und ordnungssortierte Algebren, bevor dann Petrinetze, Workflownetze und algebraische Petrinetze eingeführt werden. Abschließend werden noch bekannte Ergebnisse aus dem Bereich der Analyse von Petrinetzen wiederholt, die dann in den Kapiteln 5 und 6 verwendet werden, um Dienstbeschreibungsnetze analysieren zu können.

Algebren

Algebren dienen der Repräsentation von Daten und von Datentypen. Dies ist ähnlich der Verwendung von Algebren im Bereich der Abstract Data Types. Neben der Verwendung traditioneller heterogener Algebren werden in dieser Arbeit vor allem

ordnungssortierte Algebren verwendet, die im Unterschied zu heterogenen Algebren im Allgemeinen eine Ordnung auf den Sorten besitzen. Dies wird später genutzt, um die Ordnung auf den Konzepten einer Ontologie widerzuspiegeln.

Die zentralen Elemente der Beschreibung von Datentypen durch Algebren sind die algebraische Signatur, die angibt, welche Symbole es gibt. Dies sind die Konstanten, die Operatoren und die Sorten. Spezifikationen besitzen neben diesen drei Mengen von Symbolen noch eine Menge von Gleichungen, die angibt, wann Terme äquivalent sind. Terme sind dabei die aus den Operatoren und Konstanten entstehenden komplexeren Strukturen. Ein Term ist etwa $+(x, 3)$, wobei $+$ ein Operator, x eine Variable und 3 eine Konstante ist.

Durch die Verwendung einer Ordnung auf den Sorten kann nun definiert werden, dass ein Operator nur für eine Submenge einer Menge von Konstanten verwendet werden darf. Die Ergebnisse aus diesem Kapitel werden dann in Kapitel 4 und in Kapitel 5 zur Definition von Dienstbeschreibungsnetzen und zur Analyse elementarer Dienstbeschreibungsnetze benötigt. Dienstbeschreibungsnetze stellen die Daten und die Veränderung auf den Daten in einem Netz dar.

Die zentralen Konzepte dieses Abschnitts sind Sorten und Operatoren, die der Bildung von Termen über einer Menge von Variablen dienen. Die Auswertung der Terme geschieht dann durch eine Variablenbelegung. Jeder Term wird dabei durch ein Element der Trägermenge der Algebra interpretiert. Im Rahmen der algebraischen Netze sind zudem die Gleichungen wichtig, da sie es erlauben, eine Spezifikation anzugeben.

Petrinetze

Die Beschreibung der Abläufe erfolgt durch den auf Petri (1962) zurückgehenden Formalismus der Petrinetze. Hierbei werden an dieser Stelle Definitionen gegeben, die dann in Kapitel 5 zur Analyse der einem Dienstbeschreibungsnetz zugrundeliegenden Workflownetze benötigt werden. Da eine Analyse von Workflownetzen im Allgemeinen nicht in Polynomialzeit möglich ist, wird diese Klasse hier eingeschränkt auf Free-Choice-Netze. Da die Analyse der Netze auch durch die Betrachtung der Inzidenzmatrix erfolgt, werden zudem die wesentlichen linear algebraischen Eigenschaften eines Petrinetzes bei der Betrachtung als lineares Gleichungssystem wiedergegeben.

Algebraische Petrinetze ermöglichen es, strukturierte Marken in Form algebraischer Konstanten zu verwenden. Ein algebraisches Netz nutzt dazu eine Spezifikation, deren Terme als Marken auf den Stellen verwendet werden können. Die entsprechenden Terme können zudem gemäß den Gleichungen der Spezifikation reduziert werden. Durch das Schalten einer Transition entstehen so aus den Termen auf den Eingangsstellen der Transition neue Terme auf den Ausgangsstellen der Transition. Algebraische Netze stellen eine der Grundlagen der in Kapitel 4 beschriebenen Dienstbeschreibungsnetze dar und werden daher in Unterabschnitt 3.3.5 genauer

dargestellt.

Zentrale Begriffe dieses Abschnitts sind die grundlegenden Begriffe der Petrinetze, wie die des Schaltverhaltens und der Markierung. Zudem werden Invarianten und S- und T-Komponenten im weiteren Verlauf der Arbeit bei der Analyse notwendig sein. Zum Nachweis der Korrektheit von Workflownetzen sind die Eigenschaften der Sicherheit und der Lebendigkeit von Netzen zentral. Für Workflownetze sind die Korrektheit sowie ihre Ermittlung anhand des Abschlusses wichtig. Gleichzeitig wird später auch von der Transitionsverfeinerung für Workflownetze Gebrauch gemacht. Algebraische Netze dienen als Ausgangsbasis für Dienstbeschreibungsnetze. Als zugrundeliegende Netzklasse elementarer Dienstbeschreibungsnetze dienen Free-Choice-Netze. Hierbei ist vor allem das Rang-Theorem im weiteren Verlauf der Arbeit von Bedeutung.

Allgemeine Definitionen

Im weiteren Verlauf dieses Kapitels und in den formalen Teilen späterer Kapitel werden eine Reihe von Notationen verwendet, die hier kurz dargestellt werden sollen. Hierbei handelt es sich um gebräuchliche Darstellungen mathematischer Zusammenhänge.

Die *reflexive transitive Hülle* einer Relation R wird durch R^* notiert. Ebenso ist für eine Menge M die *Menge aller endlichen Folgen* über Elemente aus M notiert durch M^*. Die *leere Folge* wird als ϵ notiert. Zu einer Folge $w \in M^*$ bezeichnet $w[j]$ das j-te Element der Folge und N_w die Anzahl der Elemente in der Folge. Hierbei sind Werte für j zwischen 1 und N_w möglich.

Zu einer Menge X wird die *Kardinalität* von X notiert als $|X|$. Für eine Teilmenge $Y \subseteq X$ von X wird die *charakteristische Funktion* von Y als $\chi[Y] : X \to \{0, 1\}$ notiert und ist definiert als $\chi[Y](x) = 1 \Leftrightarrow x \in Y$. Geht die Menge X nicht aus dem Kontext hervor, so wird sie als $\chi[Y]_X$ notiert.

Wie üblich werden die Mengen der rationalen, der ganzen und der natürlichen Zahlen durch $\mathbb{Q}$, $\mathbb{Z}$ und $\mathbb{N}$ repräsentiert. In der Regel wird jedoch zwischen $\mathbb{N}_0$ und $\mathbb{N}_+$ unterschieden, wobei erstere Menge die Null einschließt, während letztere dies nicht tut. Wird lediglich $\mathbb{N}$ verwendet, so ist $\mathbb{N} = \mathbb{N}_0$. Es werden Abbildungen auf $\mathbb{Q}$, $\mathbb{Z}$ oder $\mathbb{N}$ zum Teil als Vektoren interpretiert. Hierbei wird von zwei verschiedenen Notationen Gebrauch gemacht. Es wird die Notation $f : A \to \mathbb{Q}$ verwendet, wenn der Abbildungscharakter im Vordergrund steht, und es wird die Notation $f \in \mathbb{Q}^{|A|}$ verwendet, wenn der Charakter als Vektor im Vordergrund steht. Letzteres findet insbesondere im Zusammenhang mit linearen Abbildungen und Matrizen Verwendung. In diesem Zusammenhand wird auch die Funktion $\chi[\cdot]$ als Vektor interpretiert, so dass $\chi[Y] \in \{0, 1\}^{|X|}$ ein Vektor ist, der genau an den Stellen, die ein Element aus Y repräsentieren, den Wert 1 besitzt.

Die Einschränkung oder *Restriktion* einer Abbildung $f : A \to B$ auf eine Teilmenge ihres Definitionsbereichs $C \subseteq A$ wird als $f|_C : C \to B$ notiert, wobei

$f|_C(c) = f(c)$ für alle $c \in C$ gilt.

Für Vektoren über $\{0, 1\}$ wird eine Notation in eckigen Klammern verwendet, die nur die Werte angibt, die verschieden von 0 sind. Dies geschieht in der Form $[a, b]$, so dass $[a, b](a) = [a, b](b) = 1$ und $[a, b](c) = 0$ für $c \notin \{a, b\}$ gilt.

3.2 Heterogene und ordnungssortierte Algebren

Dieser Abschnitt dient der Einführung grundlegender Begriffe heterogener Algebren, die im weiteren Verlauf der Arbeit verwendet werden, um Datenstrukturen zu definieren, zu interpretieren und zu verändern. Algebren erlauben eine Trennung von Syntax und Semantik. Hierzu wird der Zeichenvorrat durch eine Signatur definiert und die Interpretation dieser Zeichen durch eine Algebra gegeben. Abstrakte Datentypen (ADTs) verwenden dieses Konzept ebenfalls, um generelle Eigenschaften von Datentypen zu definieren, die von einer konkreten Programmiersprache abstrahieren (vgl. Loeckx u. a. (1997)).

In diesem Abschnitt sollen die wichtigsten Begriffe im Zusammenhang mit Algebren definiert und erläutert werden. Als weiterführende Literatur sei auf Ehrig und Mahr (1985) und Ehrig u. a. (2001) verwiesen sowie auf Goguen und Meseguer (1992) im Bereich der ordnungssortierten Algebren.

3.2.1 Signaturen und Terme

Um einen Datentyp zu definieren, müssen die Objekte, die diesem Datentyp zugeordnet sind, und die Operationen, die auf diesen Objekten möglich sind, definiert werden. Auf syntaktischer Ebene entspricht dies dem mathematischen Konstrukt der Signatur, welche aus einer Menge von Operatorsymbolen besteht und zu jedem Operatorsymbol seine Sortigkeit angibt. Aus den Operatorsymbolen werden unter Berücksichtigung der Sorten Terme gebildet. Diese Terme können Variablen enthalten, denen ein Wert zugewiesen werden muss, um den Term zu interpretieren.

Eine Spezifikation erweitert eine Signatur um Gleichungen. Dies erlaubt eine Termersetzung auf syntaktischer Ebene. Eine Algebra interpretiert eine Signatur oder Spezifikation, indem sie den Sorten Mengen und den Operatorsymbolen Operationen auf diesen Mengen zuweist. Eine Algebra zu einer Spezifikation muss zudem den Gleichungen der Spezifikation genügen.

Die Notation von indizierten oder sortierten Mengen erleichtert den Umgang mit heterogenen Signaturen. Gegeben eine Menge von Sorten S, so ist eine S-indizierte Menge A eine Familie A_s für jede Sorte $s \in S$. Seien S-indizierte Mengen A und B gegeben, so stellt die Abbildung $f : A \to B$ eine S-indizierte Familie von Abbildungen $f = \{f_s : A_s \to B_s \mid s \in S\}$ dar.

Definition 3.1 (Signatur, Sortensymbol, Operatorsymbol)
Eine (heterogene) *Signatur* $\Sigma = (S, \Omega)$ besteht aus einer endlichen Menge S von *Sortensymbolen* und einer endlichen Familie $\Omega = (\Omega_{w,s})_{(w,s)\in S^*\times S}$ von *Operatorsymbolen*[1]. Weiterhin gelte, dass jeder Term eine eindeutige Sorte hat, d.h. für alle $w \in S^*$ und alle $s, s' \in S, s \neq s'$ gilt $\Omega_{w,s} \cap \Omega_{w,s'} = \emptyset$.
Ein *Operatorsymbol* $\omega \in \Omega_{w,s}$ steht für einen Operator mit der *Domäne* (Definitionsbereich) $w = s_1 \dots s_n$ und der *Codomäne* (Wertebereich) s. $\omega \in \Omega_{\epsilon,s}$ bezeichnet ein *Konstantensymbol* der Sorte $s \in S$, wobei ϵ das leere Wort über den Sorten darstellt. ⧫

In der vorangegangenen Definition wird zwar lediglich die Präfixnotation definiert, es wird jedoch im späteren Verlauf auch von der Infixnotation gebraucht gemacht werden, wo dies der Intuition entspricht (bspw. $(a + b)$ statt $+(a, b)$).

Dadurch, dass die Familie Ω in Definition 3.1 nicht als paarweise disjunkt angenommen wird, ist es möglich, das gleiche Operatorsymbol auf verschiedenen Mengen zu verwenden (operator overloading). So ist es möglich, dass für dasselbe $\omega \in \Omega$ gilt, dass $\omega \in \Omega_{w,s}$ und $\omega \in \Omega_{w',s'}$ für $w, w' \in S^*$ und $s, s' \in S$ und $w \neq w'$.

Weitergehende Definitionen lassen für die Codomäne auch Produkte von Sorten zu, so dass die Familie der Operatoren als $\Omega = (\Omega_{w,v})_{(w,v)\in S^*\times S^*}$ definiert ist. Dies wird jedoch von wichtigen Werkzeugen nicht unterstützt und lässt sich zudem leicht nachbilden, indem die Sorte s ein Produkt mehrerer Sorten darstellt.

Definition 3.2 (Untersignatur)
Für zwei Signaturen $\Sigma_1 = (S_1, \Omega_1)$ und $\Sigma_2 = (S_2, \Omega_2)$ mit den Eigenschaften $S_1 \subseteq S_2$ und $\Omega_1 \subseteq \Omega_2$ heißt Σ_1 *Untersignatur* von Σ_2. ⧫

Definition 3.3 (Variablen und Terme)
Zu einer Signatur $\Sigma = (S, \Omega)$ wird eine Familie $X = (X_s)_{s\in S}$ mit $X \cap \Omega = \emptyset$, wobei die X_s paarweise disjunkt sind, Σ-*Variablen* (oder Variablen) genannt, wobei X_s die Menge der Variablen der Sorte s ist.
Die Menge $\mathbb{T}_{\Sigma,s}(X)$ der Σ-*Terme* (oder Terme) der Sorte s mit Variablen X ist induktiv über die Variablen- und die Operatorsymbole der Signatur Σ definiert:

1. $X_s \subseteq \mathbb{T}_{\Sigma,s}(X)$ (jede Variable ist ein Term),
2. $\Omega_{\epsilon,s} \subseteq \mathbb{T}_{\Sigma,s}(X)$ (jede Konstante ist ein Term),
3. $\omega(\tau_1, \dots, \tau_n) \in \mathbb{T}_{\Sigma,s}(X)$ für $\omega \in \Omega_{s_1\dots s_n,s}$ sofern $\tau_i \in \mathbb{T}_{\Sigma,s_i}(X),\ 1 \leq i \leq n$.
4. Dies sind alle Terme der Sorte s in $\mathbb{T}_{\Sigma,s}(X)$.

[1] Die Menge der Operatoren kann hierbei unendlich sein. Lediglich die Unterteilung soll an dieser Stelle als endlich angenommen werden.

Die Menge aller Terme ist $\mathbb{T}_\Sigma(X) \stackrel{\text{def}}{=} \bigcup_{s \in S} \mathbb{T}_{\Sigma,s}(X)$. Sie enthält die Menge der Grundterme (Terme ohne Variablen) definiert als $\mathbb{T}_\Sigma \stackrel{\text{def}}{=} \mathbb{T}_\Sigma(\emptyset)$.
Für einen Term $\tau \in \mathbb{T}_{\Sigma,s}(X)$ ist die *Menge der Variablen* in τ, notiert als $\textsc{Var}(\tau)$, induktiv definiert über die Termstruktur als $\textsc{Var}(x) \stackrel{\text{def}}{=} \{x\}$ für $x \in X$ und als $\textsc{Var}(\omega(\tau_1, \ldots, \tau_n)) \stackrel{\text{def}}{=} \bigcup_{i=1}^{n} \textsc{Var}(\tau_i)$ sonst. ♦

Definition 3.4 (Subterm)
Die *Subtermrelation* $\preccurlyeq_T$ ist rekursiv definiert als

1. $\tau \preccurlyeq_T \tau$ für alle $\tau \in \mathbb{T}_\Sigma(X)$,
2. $\tau \preccurlyeq_T \omega(\tau_1, \ldots, \tau_n)$, wenn $\tau = \tau_i$ für ein $i \in \{1, \ldots, n\}$
3. $\tau_1 \preccurlyeq_T \tau_2 \wedge \tau_2 \preccurlyeq_T \tau_3 \Rightarrow \tau_1 \preccurlyeq_T \tau_3$ für alle $\tau_1, \tau_2, \tau_3 \in \mathbb{T}_\Sigma(X)$.

Die Menge aller *Subterme* eines Terms $\tau \in \mathbb{T}_\Sigma(X)$ ist definiert als $\textsc{Subterm}(\tau) \stackrel{\text{def}}{=} \{\tau^* \in \mathbb{T}_\Sigma(X) | \tau^* \preccurlyeq_T \tau\}$. ♦

3.2.2 Algebren und Spezifikationen

Während eine Signatur zunächst lediglich beschreibt, welche Sorten es gibt und welche Operatorsymbole mit diesen Sorten identifiziert werden und Terme komplexere Strukturen von Operatorsymbolen und Variablen darstellen, findet auf dieser Ebene noch keine Interpretation der Terme statt. Um Terme und Signaturen zu „interpretieren“ bedarf es einer Algebra. Diese ordnet den Sorten Elemente konkreter Mengen und den Operatorsymbolen konkrete Abbildungen zu.

Definition 3.5 (Σ-Algebra, Trägermenge, partielle Algebra)
Zu einer Signatur $\Sigma = (S, \Omega)$ ist eine (heterogene) *Algebra* mit der Signatur Σ oder kurz Σ-*Algebra* definiert als $\mathcal{A} \stackrel{\text{def}}{=} (S_A, \Omega_A)$, wobei $S_A = (A_s)_{s \in S}$ eine Familie von Mengen ist, die den Sortensymbolen entspricht, und $\Omega_A \stackrel{\text{def}}{=} (\omega_A)_{\omega \in \Omega}$ eine Familie von Funktionen ist, die den Operatorsymbolen entspricht, so dass $\omega_A : A_{s_1} \times \ldots \times A_{s_n} \rightarrow A_s$ für alle $\omega \in \Omega_{s_1 \ldots s_n, s}$. Jede der nicht notwendigerweise disjunkten Mengen A_s wird als *Trägermenge* (der Sorte s) der Algebra bezeichnet. Die Funktionen ω_A heißen *Operationen* (auch Grundoperationen) der Algebra.
Eine Algebra wird *partielle* Σ-*Algebra* genannt, wenn die Funktionen ω_A partielle Funktionen sind. ♦

Bei einer gegebenen Signatur (S, Ω) mit zugehöriger Algebra (S_A, Ω_A) wird auch A_w anstatt $A_{s_1} \times \cdots \times A_{s_n}$ und $\overline{a} \in A_w$ respektive $\overline{a} \in A_{s_1} \times \cdots \times A_{s_n}$ anstatt $(a_1, \ldots, a_n) \in A_{s_1} \times \cdots \times A_{s_n}$ notiert.

Definition 3.6 (Unteralgebra, Redukt)
Sei $\Sigma = (S, \Omega)$ eine Signatur und seien $\mathcal{A}_1$ und $\mathcal{A}_2$ zwei Σ-Algebren für die gilt:

- $\forall s \in S : A_{1,s} \subseteq A_{2,s}$
- $\forall \omega \in \Omega : \omega_{A_1} \subseteq \omega_{A_2}$

Dann heißt $\mathcal{A}_1$ *Unteralgebra* von $\mathcal{A}_2$.
Sei $\Sigma' = (S', \Omega')$ eine Untersignatur zu $\Sigma = (S, \Omega)$. Das Σ'-*Redukt* zu einer Σ-Algebra $\mathcal{A}$, notiert als $\mathcal{A}_{/\Sigma'}$, ist die Σ'-Algebra, die wie folgt definiert ist:

- $(A_{/\Sigma'})_s \stackrel{\text{def}}{=} A_s$ für alle $s \in S'$,
- $\omega_{A_{/\Sigma'}} \stackrel{\text{def}}{=} \omega_A$ für alle $\omega \in \Omega'$.

⧫

Das Redukt überträgt die Veränderung der Struktur, wie sie durch Bildung der Unteralgebra geschieht, auf eine Algebra, wobei die übernommenen Bestandteile der Ausgangsalgebra unverändert bleiben. Um Terme mit Variablen interpretieren zu können, müssen die Variablen mit Werten aus der ihnen zugeordneten Trägermenge versehen werden. Diese Variablenbelegung erlaubt die Auswertung eines Terms.

Definition 3.7 (Belegung und Auswertung)
Zu einer Signatur $\Sigma = (S, \Omega)$, Σ-Variablen $X = (X_s)_{s \in S}$ und einer Σ-Algebra $\mathcal{A} = (S_A, \Omega_A)$ ist eine *Belegung* (oder Variablenbelegung) definiert als eine Abbildung $\alpha_{\mathcal{A}} : X \to S_A$ wobei $\alpha_{\mathcal{A}}(x) \in A_s$ für jedes $s \in S$ und $x \in X_s$ gilt. Eine Variablenbelegung weist jeder Variable einen Wert der korrespondierenden Trägermenge zu. Durch eine gegebene Variablenbelegung kann ein Term zu einem Wert ausgewertet werden.
Eine *Auswertung* ist eine Abbildung $[\![\cdot]\!]^{\mathcal{A}}_{\alpha} : \mathbb{T}_{\Sigma}(X) \to S_A$ von Σ Termen auf die Trägermenge einer Σ-Algebra S_A, die induktiv definiert ist durch

1. $[\![x]\!]^{\mathcal{A}}_{\alpha} \stackrel{\text{def}}{=} \alpha_{\mathcal{A}}(x)$ für $x \in X$,
2. $[\![\omega(\tau_1, \ldots, \tau_n)]\!]^{\mathcal{A}}_{\alpha} \stackrel{\text{def}}{=} \omega_A([\![\tau_1]\!]^{\mathcal{A}}_{\alpha}, \ldots, [\![\tau_n]\!]^{\mathcal{A}}_{\alpha})$ für alle $\omega(\tau_1, \ldots, \tau_n) \in \mathbb{T}_{\Sigma}(X)$.

Die Menge aller Belegungen einer Variablenmenge durch eine Algebra $\mathcal{A}$ wird als $\mathbf{Ass}_{\mathcal{A}}(X)$ notieren. Die Angabe der Algebra wird dabei auch weggelassen, wenn diese aus dem Kontext hervorgeht. ⧫

Gleichungen definieren die Gleichheit von Termen auf syntaktischer Ebene. Hierdurch ist es möglich, in Spezifikationen Gleichungen anzugeben, die von allen Algebren erfüllt werden müssen.

Definition 3.8 (Gleichung, Grundgleichung)
Gegeben eine Signatur $\Sigma = (S, \Omega)$ und eine S-indizierte Familie X von Σ-Variablen. Eine *(Σ-)Gleichung* e der Sorte $s \in S$ ist ein Paar von Termen $\tau_l, \tau_r \in \mathbb{T}_{\Sigma,s}(X)$

zusammen mit den Variablen X notiert als $e = (X : \tau_l = \tau_r)$. Die Menge aller Gleichungen über X wird notiert als $Eqns_\Sigma(X) \stackrel{\text{def}}{=} \{(X : \tau_l = \tau_r) \mid \exists s \in S : \tau_l, \tau_r \in \mathbb{T}_{\Sigma,s}(X)\}$. *Grundgleichungen* sind Gleichungen $e = (X : \tau_l = \tau_r)$ mit $X = \emptyset$. In diesem Fall sind τ_l und τ_r Grundterme. Wenn die Menge der Variablen X aus dem Kontext hervorgeht, wird diese nicht explizit genannt und Gleichungen auch als $\tau_l = \tau_r$ spezifiziert. ♦

Eine Algebra genügt einer Gleichung, wenn diese für alle möglichen Variablenbelegungen stets zu wahr auswertet. Eine Algebra genügt einer Menge von Gleichungen entsprechend, wenn sie allen Gleichungen dieser Menge genügt. Dies erlaubt es, bereits auf syntaktischer Ebene zu überprüfen, ob eine Algebra gewissen Kriterien genügt. Im Rahmen der Spezifikation wird dies dazu genutzt, die Zahl der möglichen interpretierenden Algebren einzuschränken.

Definition 3.9 (Lösung, Gültigkeit von Gleichungen)
Gegeben eine Signatur $\Sigma = (S, \Omega)$, eine Σ-Algebra $\mathcal{A} = (S_A, \Omega_A)$ und eine S-indizierten Familie X von Σ-Variablen.

Eine Belegung $\alpha : X \to S_A$ ist *Lösung* einer Σ-Gleichung $e = (\tau_l = \tau_r)$ in $\mathcal{A}$, in Zeichen $\mathcal{A}, \alpha \vDash (\tau_l = \tau_r)$ gdw. $[\![\tau_l]\!]^{\mathcal{A}}_\alpha = [\![\tau_r]\!]^{\mathcal{A}}_\alpha$ gilt. Die Menge aller Lösungen wird als $Sol_\Sigma((\tau_l = \tau_r), \mathcal{A}) \stackrel{\text{def}}{=} \{\alpha : X \to S_A \mid \tau_l, \tau_r \in dom([\![\cdot]\!]^{\mathcal{A}}_\alpha) \wedge [\![\tau_l]\!]^{\mathcal{A}}_\alpha = [\![\tau_r]\!]^{\mathcal{A}}_\alpha\}$ notiert.

Die Gleichung $e = (X : \tau_l = \tau_r)$ wird *gültig* in $\mathcal{A}$ genannt, wenn für alle Variablenbelegungen $\alpha : X \to S_A$ gilt $[\![\tau_l]\!]^{\mathcal{A}}_\alpha = [\![\tau_r]\!]^{\mathcal{A}}_\alpha$.

Wenn e in $\mathcal{A}$ gültig ist, sagt man auch $\mathcal{A}$ *genügt* e und notiert dies als $\mathcal{A} \vDash_\Sigma e$. Die Gültigkeit einer Menge von Gleichungen E wird als $\mathcal{A} \vDash_\Sigma E$. Wenn die Menge der Variablen X aus dem Kontext hervorgeht, wird diese nicht explizit genannt und Gleichungen auch als $\tau_l = \tau_r$ spezifiziert. Die Menge aller Gleichungen über einer Signatur Σ wird als $Eqns_\Sigma$ notiert. ♦

Eine Spezifikation nutzt eine Menge von Gleichungen, um die möglichen interpretierenden Algebren einzuschränken. Auch lassen sich durch die Gleichungen bereits auf syntaktischer Ebene gleiche Terme durcheinander ersetzen. Eine Gleichung wie $(1 + 1) = 2$ erlaubt es auf syntaktischer Ebene den Term $(1 + 1)$ durch den Term 2 zu ersetzen.

Definition 3.10 (Spezifikation und Γ-Algebra)
Eine *Spezifikation* $\Gamma = (S, \Omega, E)$ besteht aus einer Signatur $\Sigma = (S, \Omega)$ und aus einer Menge von Σ-Gleichungen E.

Eine *Algebra* $\mathcal{A}$ der Spezifikation Γ (Γ-*Algebra*) ist eine Algebra der Signatur Σ, die allen Gleichungen aus E genügt. ♦

Eine mögliche Interpretation einer Signatur ist stets dadurch gegeben, dass die Terme durch sich selbst interpretiert werden. Diese Interpretation wird als Termalgebra bezeichnet. Die Trägermengen sind dabei sämtliche möglichen Terme der Signatur.

Definition 3.11 (Termalgebra, Grundtermalgebra)
Gegeben eine Signatur Σ und eine Menge von Variablen X. Die Algebra (S_T, Ω_T) mit

- $S_T = (\mathbb{T}_{\Sigma,s}(X))_{s \in S}$ als der Familie von Trägermengen
- $\omega_T : \mathbb{T}_{\Sigma,s_1}(X) \times \ldots \times \mathbb{T}_{\Sigma,s_n}(X) \to \mathbb{T}_{\Sigma,s}(X)$ definiert durch $\omega_T(\tau_1, \ldots, \tau_n) \stackrel{\text{def}}{=} \omega(\tau_1, \ldots, \tau_n)$ für $\omega \in \Omega_{s_1 \ldots s_n, s}$ und $\tau_i \in \mathbb{T}_{\Sigma,s_i}(X), 1 \leq i \leq n$ als Operationen

wird *Termalgebra* bzgl. Σ und X genannt. Diese Algebra soll im Folgenden ebenfalls als $\mathbb{T}_\Sigma(X)$ notiert werden, genau wie die Menge der Terme über X. Des Weiteren wird auch $\mathbb{T}_{\Sigma,s}$ statt $\mathbb{T}_{\Sigma,s}(\emptyset)$ verwendet und es wird die *Grundtermalgebra* – die Termalgebra ohne Variablen – mit $\mathbb{T}_\Sigma$ statt $\mathbb{T}_\Sigma(\emptyset)$ bezeichnet. ♦

Eine Spezifikation besitzt Gleichungen, die ebenfalls in die Interpretation der Spezifikation einbezogen werden sollten. Dies geschieht, indem zunächst eine Kongruenz auf den Termen definiert wird, die als Trägermengen dienen. Hierdurch kann dann die Quotiententermalgebra definiert werden, die der Termalgebra für Spezifikationen entspricht.

Definition 3.12 (Kongruenz, erzeugte Kongruenz)
Sei $\Sigma = (S, \Omega)$ eine Signatur, $\mathcal{A}$ eine Algebra und $K = (K_s \subseteq A_s \times A_s)_{s \in S}$ eine Familie von Äquivalenzrelationen auf den Trägermengen von $\mathcal{A}$. K heißt *Kongruenzrelation* auf $\mathcal{A}$, wenn für jedes Operatorsymbol $\omega \in \Omega$ und alle $a_i, a_i' \in A_{s_i} (1 \leq i \leq n)$ gilt, dass $(a_1, a_1') \in K_{s_1} \wedge \ldots \wedge (a_n, a_n') \in K_{s_n} \implies (\omega_A(a_1, \ldots, a_n), \omega_A(a_1', \ldots, a_n')) \in K_s$.
Sei E eine Menge von Σ-Gleichungen. Die durch E *Kongruenzrelation* $\sim_s^E \subseteq \mathbb{T}_{\Sigma,s} \times \mathbb{T}_{\Sigma,s}$ auf den Grundtermen der Signatur ist induktiv gegeben durch:

- $[\![\tau_l]\!]_\alpha^{\mathbb{T}_\Sigma} \sim_s^E [\![\tau_r]\!]_\alpha^{\mathbb{T}_\Sigma}$ für jede Gleichung $(\tau_l = \tau_r) \in E$ und alle Variablenbelegungen $\alpha : X \to \mathbb{T}_\Sigma$.
- $\omega(\tau_1, \ldots, \tau_n) \sim_s^E \omega(\tau_1', \ldots, \tau_n')$ für alle $\omega \in \Omega_{s_1 \ldots s_n, s}$ und alle $\tau_i, \tau_i', 1 \leq i \leq n$ mit $\tau_i \sim_s^E \tau_i'$.
- $\tau_1 \sim_s^E \tau_1$, $\tau_1 \sim_s^E \tau_2 \implies \tau_2 \sim_s^E \tau_1$ und $\tau_1 \sim_s^E \tau_2 \wedge \tau_2 \sim_s^E \tau_3 \implies \tau_1 \sim_s^E \tau_3$ für $\tau_1, \tau_2, \tau_3 \in \mathbb{T}_{\Sigma,s}$.

- Dies sind alle Elemente von $\sim_s^E$.

⧫

Definition 3.13 (Äquivalenzklasse, Quotient)
Seien $R \subseteq A \times A$ eine beliebige *Äquivalenzrelation* und $a \in A$ ein beliebiges Element ihrer Grundmenge. Die *Äquivalenzklasse* von a, geschrieben $[a]_R$ ist definiert als $[a]_R \stackrel{\text{def}}{=} \{x \in A \mid (a, x) \in R\}$.
Sei $R \subseteq A \times A$ eine Äquivalenzrelation. Die Menge aller Äquivalenzklassen von R heißt *Quotient* von A bzgl. R und wird notiert als $A/_R \stackrel{\text{def}}{=} \{[a]_R \mid a \in A\}$ ⧫

Die Quotiententermalgebra nutzt die durch die Gleichungen erzeugte Kongruenzrelation um Terme induktiv gleich zu setzen. Dies ermöglicht es, auf syntaktischer Ebene bereits Vereinfachungen vorzunehmen.

Definition 3.14 (Quotiententermalgebra)
Sei $\Gamma = (S, \Omega, E)$ eine Spezifikation und sei $\sim^E \stackrel{\text{def}}{=} (\sim_s^E)_{s \in S}$ die durch E erzeugte Kongruenzrelation. Dann heißt $\mathbb{T}_\Gamma \stackrel{\text{def}}{=} (S_T, \Omega_T)$ *Quotiententermalgebra*, wobei

- $S_T \stackrel{\text{def}}{=} (T_s)_{s \in S}$ mit $(T_s)_{s \in S} \stackrel{\text{def}}{=} (\mathbb{T}_{\Sigma,s}/_{\sim_s^E})_{s \in S}$,
- Für alle $\omega \in \Omega_{s_1 \ldots s_n, s}$ und $[\tau_i] \in T_{s_i}, (1 \leq i \leq n)$ gilt:

$$\omega_{\mathbb{T}_\Gamma}([\tau_1], \ldots, [\tau_n]) \stackrel{\text{def}}{=} [\omega_{\mathbb{T}_\Sigma}(\tau_1, \ldots, \tau_n)].$$

Hierbei bezeichnet $[x]$ die Äquivalenzklasse von x und $M/_R$ den Quotienten. ⧫

An dieser Stelle soll nicht weiter auf Quotiententermalgebren eingegangen werden. Es sei stattdessen auf Ehrig u. a. (2001) verwiesen. Quotiententermalgebren besitzen eine wesentliche Eigenschaft, die hier noch erwähnt werden soll. Jede Quotiententermalgebra stellt eine initiale Algebra einer Spezifikation dar. Dies bedeutet, dass zu jeder Algebra einer Spezifikation genau ein Homomorphismus, der die Quotiententermalgebra auf diese Algebra abbildet, existiert. Hiervon wird im Rahmen algebraischer Netze Gebrauch gemacht. Ein Homomorphismus ist dabei wie folgt definiert.

Definition 3.15 (Homomorphismus)
Gegeben zwei Σ-Algebren $\mathcal{A}$ und $\mathcal{B}$ zur Signatur $\Sigma = (S, \Omega)$. Ein (Σ-)*Homomorphismus* $h : \mathcal{A} \to \mathcal{B}$ ist eine eindeutige Familie von Abbildungen $h = (h_s : A_s \to B_s)_{s \in S}$ zwischen den Trägermengen der Algebren, so dass für alle $\omega \in \Omega_{s_1 \ldots s_n, s}$ und für alle Elemente $a_i \in A_{s_i}, i \in \{1, \ldots, n\}$ gilt:

$$h_s(\omega_A(a_1, \ldots, a_n)) = \omega_B(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$$

♦

Die Gesamtheit aller Algebren zu einer gegebenen Signatur Σ bzw. zu einer gegebenen Spezifikation Γ zusammen mit den Homomorphismen zwischen den Algebren bildet eine Kategorie (vgl. Ehrig u. a., 2001). Die Kategorie der Σ-Algebren wird als $\mathbf{Alg}(\Sigma)$ notiert. Eine Kategorie ist dabei wie folgt definiert.

Definition 3.16 (Kategorie)
Eine *Kategorie* $\mathbf{C} = (Obj_{\mathbf{C}}, Mor_{\mathbf{C}}, \circ, Id)$ besteht

- aus einer Klasse von Objekten $Obj_{\mathbf{C}}$,
- aus einer Menge von *Morphismen* $Mor_{\mathbf{C}}(A, B)$ zu jedem Paar von Objekten $A, B \in Obj_{\mathbf{C}}$, notiert als $f : A \to B$ für $f \in Mor_{\mathbf{C}}(A, B)$,
- aus $\circ_{(A,B,C)} : Mor_{\mathbf{C}}(B, C) \times Mor_{\mathbf{C}}(A, B) \to Mor_{\mathbf{C}}(A, C)$ als einer Menge von Kompositionsoperationen für je drei Objekte $A, B, C \in Obj_{\mathbf{C}}$ und
- aus einem *Identitätsmorphismus* $id_A \in Mor_{\mathbf{C}}(A, A)$ für jedes $A \in Mor_{\mathbf{C}}$,

die folgende Bedingungen erfüllen: **Assoziativiät**, d.h. für alle $A, B, C, D \in Obj_{\mathbf{C}}$ und alle $f \in Mor_{\mathbf{C}}(A, B)$, $g \in Mor_{\mathbf{C}}(B, C)$ und $h \in Mor_{\mathbf{C}}(C, D)$ gilt$(h \circ_{(B,C,D)} g) \circ_{(A,B,D)} f = h \circ_{(A,C,D)} (g \circ_{(A,B,C)} f)$ und **Neutralität**, d.h. für alle $A, B \in Obj_{\mathbf{C}}$ und alle $f \in Mor_{\mathbf{C}}(A, B)$ gilt $f \circ_{(A,A,B)} id_A = f$ und $id_B \circ_{(A,B,B)} f = f$.

♦

Für einen Morphismus $f \in Mor_{\mathbf{C}}(A, B)$ heißt das Objekt A Quelle (domain, source) und das Objekt B Ziel (codomain, target) von f, was für f auch als $A \xrightarrow{f} B \in Mor_{\mathbf{C}}$ notiert wird.

3.2.3 Ordnungssortierte Algebren

Algebren trennen zwischen der Syntax, die als Signatur gegeben wird, und der Interpretation als Algebra. Eine heterogene Signatur $\Sigma = (S, \Omega)$ besteht aus einer endlichen Menge S von *Sortensymbolen* (auch Sorten) und einer endlichen Familie $\Omega = (\Omega_{w,s})_{(w,s)\in S^* \times S}$ von *Operatorsymbolen*, wie in Abschnitt 3.2.1 dargestellt. Das Konzept der ordnungssortierten Algebra (OSA) ergänzt die Typisierung heterogener Algebren dadurch, dass Vererbung und Subsorten-Polymorphismus eingeführt werden. Dies erlaubt die Darstellung von Mehrfachvererbung und das Überladen von Operatoren. Die folgenden Definitionen und Theoreme sind an die Ausführungen von Goguen und Meseguer (1992) angelehnt.

Das zentrale Element von ordnungssortierten Algebren ist die Einführung einer partiellen Ordnung $\preceq$ auf der Menge der Sorten S. Diese *Subsorten*-Relation führt eine Einschränkung dergestalt ein, dass für $s \preceq s'$ in S auch $A_s \subseteq A_{s'}$ gilt. Im weiteren Verlauf wird $\preceq$ auf Zeichenketten $w \in S^*$ gleicher Länge erweitert, dergestalt,

dass $s_1 \dots s_n \preceq s'_1 \dots s'_n$ gdw. $s_i \preceq s'_i$ für alle $1 \leq i \leq n$. Des Weiteren sei $w_i \in S^*$ im Folgenden eine Zeichenkette über S.

Definition 3.17 (Ordnungssortierte Signatur (OS-Signatur))
Eine *ordnungssortierte Signatur* ist ein Tripel $\Sigma = (S, \preceq, \Omega)$ wobei

- (S, Ω) eine heterogene Signatur,
- $\preceq$ eine partielle Ordnung auf S ist und
- $\omega \in \Omega_{w_1,s_1} \cap \Omega_{w_2,s_2}$ und $w_1 \preceq w_2$ impliziert $s_1 \preceq s_2$ gilt.

⧫

Die letzte Bedingung ist eine Monotonieeigenschaft, die besagt, dass ein Operatorsymbol den Typ seiner Codomäne respektiert. Diese Eigenschaft impliziert, dass Konstanten nicht überladen werden können, da $\epsilon = w_1 = w_2$ impliziert, dass $s_1 = s_2$.

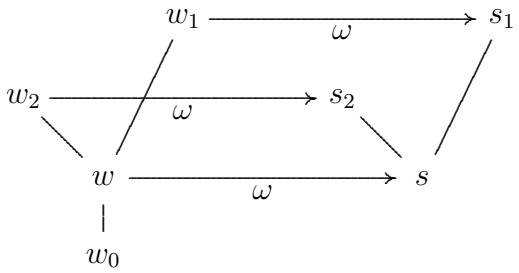
Eine wichtige Eigenschaft ordnungssortierter Signaturen ist die Eigenschaft der Regularität. Diese besagt, dass jeder Term in einer OS-Signatur eine kleinste Sorte besitzt. Anders formuliert besagt Regulärität, dass *überladene Operatoren* gleiches Verhalten aufweisen, wenn sie auf die gleiche Subsorte eingeschränkt werden. Wenn eine OS-Signatur regulär ist, können die Eigenschaften einer heterogenen Signatur direkt auf diese OS-Signatur übertragen werden.

Definition 3.18 (Reguläre Signaturen)
Eine ordnungssortierte Signatur Σ ist *regulär* gdw. es für jedes $\omega \in \Omega_{w_1,s_1}$ und jedes w_0 mit $w_0 \preceq w_1$ für $w_0, w_1 \in S^*$ ein (bzgl. $\preceq$) kleinstes $(w, s) \in (S^* \times S)$ gibt, so dass $w_0 \preceq w$ und $\omega \in \Omega_{w,s}$. ⧫

Das kleinste w kann natürlich mit w_0 zusammenfallen, wird dies jedoch im Allgemeinen nicht tun. Regularität besagt, dass jeder Operator eine kleinste Domäne besitzen muss. Sie impliziert ein kleinstes w und somit aufgrund der Monotonieeigenschaft aus der Definition auch ein kleinstes s bzgl. $\preceq$ für ein Operatorsymbol. Wenn Σ regulär ist, dann gibt es für ein $\omega \in \Omega_{w_1,s_1}$ auch ein kleinstes $s \in S$, so dass $\omega \in \Omega_{w,s}$ für ein $w \in S^*$. Dieses s ist das gleiche s wie das im kleinsten Paar $(w, s) \in (S^* \times S)$ in Definition 3.18.

Dies erlaubt es, folgendes Lemma zu formulieren, welches in Abbildung 3.1 visualisiert wird. Die Sorteninklusion bezeichnet dabei die Teilmengenbeziehung bzgl. der Sortenhierarchie $\preceq$.

Lemma 3.19 (Regularität)
Eine OS-Signatur Σ über einer partiell geordneten Menge von Sorten $(S, \preceq)$ ist



Diagonale und vertikale Linien bezeichnen Sorteninklusion, während horizontale Pfeile für Instanzen des Operationssymbols ω stehen. (Das Diagramm wurde von Goguen und Meseguer (1992) übernommen.)

Abbildung 3.1: Visualisierung von Lemma 3.19

regulär gdw. für alle $\omega \in \Omega_{w_1,s_1} \cap \Omega_{w_2,s_2}$ die Existenz eines $w_0 \leq w_1, w_2$ impliziert, dass ein $w \leq w_1, w_2$ existiert, so dass $\omega \in \Omega_{w,s}$ und $w_0 \leq w$.

Der Beweis findet sich in Goguen und Meseguer (1992), Beweis zu Lemma 2.6.

Eine OS-Signatur wird durch eine ordnungssortierte Algebra interpretiert, indem wie bei heterogenen Signaturen jedem Element der Menge der Sortensymbole eine Trägermenge zugewiesen wird und jedem Operatorsymbol eine Abbildung. Hinzu kommen noch Bedingungen bezüglich der Relation $\preceq$.

Definition 3.20 (Ordnungssortierte Algebra)
Zu einer Signatur $\Sigma = (S, \preceq, \Omega)$ ist eine (ordnungssortierte) Σ-*Algebra* $A = (S_A, \Omega_A)$ eine Algebra (vgl. Def. 3.5) zu (S, Ω), so dass

- $s_1, s_2 \in S$, $s_1 \preceq s_2$ impliziert, dass $A_{s_1} \subseteq A_{s_2}$ (Subtypisierung impliziert Teilmengenbeziehung der Trägermengen),
- $\omega \in \Omega_{w_1,s_1} \cap \Omega_{w_2,s_2}$ und $w_1 \preceq w_2$ implizieren, dass $\omega_A : A_{w_1} \to A_{s_1}$ gleich $\omega_A : A_{w_2} \to A_{s_2}$ auf A_{w_1} [2] (Gleiche Operatoren werten auf Subsorten gleich aus).

♦

Definition 3.21 (Homomorphismen und die Kategorie OSAlg(Σ))
Sei $\Sigma = (S, \preceq \Omega)$ eine OS-Signatur und seien $\mathcal{A}$ und $\mathcal{B}$ zwei Σ-Algebren. Ein Σ-*Homomorphismus* $h : \mathcal{A} \to \mathcal{B}$ ist ein (S, Ω)-Homomorphismus für den zusätzlich

[2] Dies bedeutet, dass sich ein Operator auf gleicher Eingabe gleich verhalten muss. So muss etwa der Wert von $op(1, 2)$ gleich dem von $op(1.0, 2.0)$ sein, wobei $1, 2 \in nat$ und $1.0, 2.0 \in rat$ mit $1 = 1.0$ und $2 = 2.0$ gilt.

gilt $s \leq s'$ und $a \in A_s$ impliziert $h_s(a) = h_{s'}(a)$.
Die Klasse aller Σ-Algebren zusammen mit den Σ-Homomorphismen bilden die Kategorie **OSAlg**(Σ). ⧫

Wie im Falle der heterogenen Signatur sind Terme induktiv definiert. Sie sind wiederum durch Variablen und Operatorsymbole einer Signatur aufgebaut. Außerdem ist die Menge der Terme einer Subsorte $s' \preceq s$ in der Menge der Terme von s. Somit stellt $\mathbb{T}_\Sigma$ wiederum eine Termalgebra dar.

Definition 3.22 (Variablen, Terme)
Zu einer OS-Signatur $\Sigma = (S, \preceq, \Omega)$ wird eine paarweise disjunkte Familie $X = (X_s)_{s \in S}$ mit $X \cap \Omega = \emptyset$ Σ-*Variablen* genannt.
Die Menge $\mathbb{T}_{\Sigma,s}(X)$ von Σ-Termen der Sorte s mit Variablen aus X ist induktiv definiert als:

- $X_s \subseteq \mathbb{T}_{\Sigma,s}(X)$ (jede Variable ist ein Term),
- $\Omega_{\epsilon,s} \subseteq \mathbb{T}_{\Sigma,s}(X)$ (jede Konstante ist ein Term),
- $\mathbb{T}_{\Sigma,s}(X) \subseteq \mathbb{T}_{\Sigma,s'}(X)$ für jedes $s \preceq s'$ (jeder Subsortenterm ist wiederum ein Term),
- $\omega(\tau_1, \ldots, \tau_n) \in \mathbb{T}_{\Sigma,s}(X)$ für $\omega \in \Omega_{s_1 \ldots s_n, s}$ wenn $\tau_i \in \mathbb{T}_{\Sigma,s_i}(X)$ für $i = 1, \ldots n$.

⧫

Lemma 3.23 (Kleinste Sorten regulärer OS-Signaturen)
Gegeben eine reguläre OS-Signatur Σ. Für jedes $\tau \in \mathbb{T}_\Sigma$ gibt es eine *kleinste Sorte* $s \in S$ von τ notiert als $LS(\tau)$, so dass $\tau \in \mathbb{T}_{\Sigma,s}$.

Die Termalgebra $\mathbb{T}_\Sigma$ ist für reguläre Signaturen wiederum initial.Der Beweis findet sich in Goguen und Meseguer (1992), Beweis zu Theorem 2.12.

Ein Term τ hat die *Nominalsorte* s, wenn er die Form $\tau = \omega(\tau_1, \ldots, \tau_n)$ mit $\omega \in \Omega_{w,s}$ hat oder τ eine Variable ist und $\tau \in X_s$ gilt. Belegung und Auswertung sind definiert wie im Fall der heterogenen Algebren (vgl. Def. 3.7). Ebenso sind Spezifikationen dem heterogenen Fall sehr ähnlich, allerdings gibt es einen Unterschied bei den Gleichungen, da die rechte und die linke Seite einer Gleichung verschiedene Sorten besitzen können. Aus diesem Grund verlangt die folgende Definition für ordnungssortierte Gleichungen, dass die kleinsten Sorten (Lemma 3.23) der rechten und der linken Seite einer Gleichung in derselben Zusammenhangskomponente der partiell geordneten Menge $(S, \preceq)$ liegen. Die Zusammenhangskomponenten von $(S, \preceq)$ sind die Äquivalenzklassen der Äquivalenzrelation, die durch die transitive und symmetrische Hülle von $\preceq$ entsteht: $\equiv_\preceq \stackrel{\text{def}}{=} \preceq \cup \preceq^{-1}$.

Definition 3.24 (Gleichung, Spezifikation)
Zu einer regulären OS-Signatur $\Sigma = (S, \preceq, \Omega)$ ist eine Σ-*Gleichung* ein Tripel (X, l, r), notiert als $e = (X : l = r)$, wobei X eine Menge von Variablen ist und $l, r \in \mathbb{T}_\Sigma(X)$ sind, so dass $LS(\tau)$ und $LS(\tau')$ in derselben Zusammenhangskomponente von $(S, \preceq)$ sind. Die Gleichung $e = (X : l = r)$ wird *gültig* in einer OS-Algebra $\mathcal{A}$ genannt, wenn für alle Variablenbelegungen $\alpha : X \to A$ gilt $[\![l]\!]_\alpha^{\mathcal{A}, LS(l)} = [\![r]\!]_\alpha^{\mathcal{A}, LS(r)}$, wobei $[\![\tau]\!]_\alpha^{\mathcal{A}, LS(\tau)}$ die Auswertung des Terms $\tau \in \mathbb{T}_{\Sigma, LS(\tau)}$ ist. Dies wird wiederum als $\mathcal{A} \models_\Sigma e$ notiert. Die Gültigkeit einer Menge von Gleichungen E wird als $\mathcal{A} \models_\Sigma E$ notiert (vgl. auch Def. 3.9).
Analog zum heterogenen Fall ist eine ordnungssortierte *Spezifikation* (OS-Spezifikation) $\Gamma = (S, \preceq, \Omega, E)$ eine Signatur $(S, \preceq, \Omega)$ zusammen mit einer Menge von Gleichungen E. ⧫

Die Angabe der Signatur bei der Gültigkeit wird im Allgemeinen weggelassen, wenn diese aus dem Kontext hervorgeht.

Die Begrifflichkeiten sind in diesem Zusammenhang nicht immer eindeutig. So findet sich in der Literatur statt Spezifikation auch die Bezeichnung *Theorie*, wobei die einzelnen Gleichungen in diesem Fall als *Sätze* (engl. *Sentences*) bezeichnet werden (vgl. Goguen, 1991). Dies schlägt sich beispielsweise auch in der Sprache OBJ nieder. Eine einer Spezifikation genügende Algebra wird in diesem Fall auch als Modell der Theorie bezeichnet. Des Weiteren wird eine Spezifikation auch als Präsentation bezeichnet (vgl. Goguen, 1991; Große-Rhode, 2004).

Definition 3.25 ((Lokal) gefilterte, kohärente Signatur)
Eine partiell geordnete Menge $(S, \preceq)$ ist (aufwärts) *gefiltert* gdw. es für je zwei Elemente $s, s' \in S$ ein Element $s'' \in S$ gibt, so dass $s, s' \preceq s''$. Eine partiell geordnete Menge S ist *lokal gefiltert* gdw. jede ihrer Zusammenhangskomponenten gefiltet ist. Eine OS-Signatur $\Sigma = (S, \preceq, \Omega)$ ist *lokal gefiltert* gdw. $(S, \preceq)$ lokal gefiltert ist und die Signatur Σ ist *kohärent* gdw. sie lokal gefiltert und regulär ist. ⧫

In der Praxis kann Kohärenz leicht dadurch erreicht werden, dass ein größtes Element bzgl. $\preceq$ für jede nicht gefilterte Zusammenhangskomponente hinzugefügt wird. Dies wird etwa in Java durch die `Object` Klasse oder in OWL durch das Konzept `Thing` erreicht. Terme, die in Gleichungen kohärenter Signaturen erscheinen, haben eine gemeinsame Supersorte zu der sie gehören. In Goguen und Meseguer (1992) wird im Beweis zu Proposition 2.20 gezeigt, dass jede OS-Signatur, die eine obere Schranke (bzgl. $\preceq$) aufweist auch kohärent ist, sofern sie regulär ist. In unserem Fall ist dieses kleinste Element stets durch das $\bot$ Element der Ontologie gegeben, die Regularität folgt aufgrund von $\top$. Dies soll hier nur als Lemma festgehalten werden. Für einen Beweis sei auf Goguen und Meseguer (1992) verwiesen.

Lemma 3.26 (Kohärente OS-Signatur)
Eine OS-Signatur, deren Ordnung $\preceq$ ein kleinstes Element $\bot$ und ein größtes Element $\top$ besitzt, ist kohärent.

Definition 3.27 (Unteralgebra)
Zu einer OS-Signatur Σ und einer Σ-Algebra $\mathcal{A}$ ist eine ordnungssortierte Σ-*Unteralgebra* $\mathcal{B}$ von $\mathcal{A}$ eine heterogene Σ-Unteralgebra (vgl. Def. 3.6), so dass $B_s \subseteq B_{s'}$ gilt, wenn $s \leq s'$. ⧫

In Goguen und Meseguer (1992) wird dargestellt, dass zu jeder kohärenten OS-Signatur Σ eine heterogene Signatur $\Sigma^{\#}$ existiert, so dass die Kategorie der Algebren zu Σ und die derjenigen zu $\Sigma^{\#}$ äquivalent sind. Die entsprechende Argumentation findet sich in Goguen und Meseguer (1992) auf Seite 27ff. im Zusammenhang mit Theorem 4.2. Dort wird auch dargestellt, dass sich kohärente OS-Signaturen und ihre OS-Algebren folglich genauso einsetzen lassen wie heterogene Algebren.

Da jede kohärente OS-Algebra auch eine heterogene Algebra ist, besitzt die Kategorie $\mathbf{OSAlg}(\Sigma)$ einen Vergissfunktor in die Kategeorie der Algebren $\mathbf{Alg}((S, \Omega))$ zur Signatur (S, Ω), der die Ordnungsrelation vergisst. Außerdem ist jede heterogene Algebra eine OS-Algebra mit der trivialen Ordnungsrleation $\preceq = \{(s, s') \in S \times S \mid s = s'\}$. In diesem Fall ist der Vergissfunktor die Identität.

3.3 Petrinetze und Workflownetze

Dieser Abschnitt beschreibt die Begriffe und Notationen, die in den folgenden Abschnitten verwendet werden, und präsentiert bereits existierende Ergebnisse aus anderen Quellen. Diese wurden an dieser Stelle noch einmal zusammengefasst, so dass später leichter nachvollziehbar ist, auf welche Ergebnisse in den Beweisen Bezug genommen wird.

Zunächst werden Grundbegriffe von Petrinetzen und Graphen allgemein in Abschnitt 3.3.1 definiert. Die Darstellung orientiert sich an den üblichen Definitionen in diesem Bereich, wie sie etwa bei Jessen und Valk (1987), bei Girault und Valk (2003) oder bei Desel und Esparza (1995) zu finden sind. Anschließend folgt eine kurze Zusammenfassung von Eigenschaften von Petrinetzen und Workflownetzen. In erster Linie werden hierbei Ergebnisse aus Desel und Esparza (1995) wiederholt.

3.3.1 Grundlegende Definitionen von Petrinetzen

Dieser Unterabschnitt beschreibt die grundlegenden Eigenschaften von Petrinetzen, auf denen dann die folgenden Unterabschnitte aufbauen. Hierbei werden zunächst die statischen Eigenschaften der Petrinetze dargestellt und im Anschluss daran wird auf die dynamischen Eigenschaften der Netze eingegangen.

Definition der Grapheneigenschaften

Ein Petrinetz ist ein gerichteter bipartiter Graph. Hiervon ausgehend werden hier zunächst die Grapheneigenschaften der Netze beschrieben. Hierzu gehört zunächst die Beschreibung der Petrinetze selber und die Einführung von Subnetzen als Teilgraph eines Netzes. Durch Invertierung der Kanten eines Petrinetzes entsteht das inverse Netz.

Definition 3.28 (Petrinetz)
Ein *Petrinetz* (Netz) $\mathcal{N}$ ist ein Tripel $\mathcal{N} = (P, T, F)$, wobei

- P eine endliche Menge *Stellen* (auch Plätzen),
- T eine endliche Menge von *Transitionen* mit $P \cap T = \emptyset$ und
- $F \subseteq (P \times T) \cup (T \times P)$ eine *Flussrelation* darstellt.

Jedes $x \in P \cup T$ wird auch *Knoten* genannt. ⧫

Definition 3.29 (Subnetz, Vereinigung zweier Netze)
Ein *Subnetz* $\mathcal{N}' = (P', T', F')$ eines Petrinetzes $\mathcal{N}$ ist ein Petrinetz, so dass $P' \subseteq P$, $T' \subseteq T$ und $F' = F \cap ((P' \times T') \cup (T' \times P'))$ ist.
Die *Vereinigung* zweier Netze $\mathcal{N}_1 = (P_1, T_1, F_1)$ und $\mathcal{N}_2 = (P_2, T_2, F_2)$ ist definiert als das Netz $\mathcal{N}_1 \oplus \mathcal{N}_2 = (P_1 \cup P_2, T_1 \cup T_2, F_1 \cup F_2)$. ⧫

Zu einem Knoten $x \in P \cup T$ bezeichnet ${}^\bullet_{(\mathcal{N})}x \stackrel{\text{def}}{=} \{y \mid (y, x) \in F\}$ die Menge der Knoten im *Vorbereich* und $x^\bullet_{(\mathcal{N})} \stackrel{\text{def}}{=} \{y \mid (x, y) \in F\}$ die Menge der Knoten im *Nachbereich* von x. Die Definition wird auf Mengen erweitert, wobei ${}^\bullet_{(\mathcal{N})}U \stackrel{\text{def}}{=} \bigcup_{x \in U} {}^\bullet_{(\mathcal{N})}x$ und $U^\bullet_{(\mathcal{N})} \stackrel{\text{def}}{=} \bigcup_{x \in U} x^\bullet_{(\mathcal{N})}$ gilt. Das Netz $\mathcal{N}$, auf das sich die Operationen ${}^\bullet\cdot$ und $\cdot^\bullet$ beziehen, wird im Allgemeinen nicht mit angegeben, so dass im weiteren Verlauf von der Notation ${}^\bullet x$ Gebrauch gemacht wird, sofern das entsprechende Netz aus dem Kontext hervorgeht.

Eine nicht-leere Folge von Knoten $\pi = (x_0 \dots x_k)$ mit $k \in \mathbb{N}_0, x_i \in P \cup T$ eines Netzes $\mathcal{N}$ wird (gerichteter) *Pfad* von x_0 nach x_k genannt gdw. $\forall i \in \{0, \dots, k-1\} : (x_i, x_{i+1}) \in F$ gilt. Ein Pfad $(x_0 \dots x_k)$ wird als *elementar* bezeichnet gdw. $\forall i, j \in \{0, \dots, k\} : x_i = x_j \Rightarrow i = j$. Ein Pfad $(x_0 \dots x_k)$ wird als *konfliktfrei* bezeichnet gdw. $\forall x_i \in P \forall x_j \in T : i \neq j - 1 \Rightarrow x_i \notin {}^\bullet x_j$. Die Menge aller gerichteten Pfade von x_0 nach x_k in $\mathcal{N}$ wird als $X_d^{\mathcal{N}}(x_0, x_k)$ notiert. Die Menge aller gerichteten elementaren Pfade von x_0 nach x_k wird als $E_d^{\mathcal{N}}(x_0, x_k)$ notiert. Für eine Folge $\pi = (x_0 x_1 \dots x_k)$ sei $\textsc{Alph}(\pi) \stackrel{\text{def}}{=} \{x_0, x_1, \dots, x_k\}$ die Menge aller Elemente von π.

Ein elementarer Pfad von x_0 nach x_k wird als *Zyklus* γ bezeichnet gdw. $(x_k, x_0) \in F$. Ein Netz ist *zyklenfrei* gdw. es keine Zyklen enthält. In zyklenfreien Petrinetzen ist jeder Pfad elementar, so dass dieser Zusatz weggelassen wird. Ebenso wird die Spezifikation des Netzes weggelassen, wenn diese aus dem Kontext hervorgeht, und $E_d^{\mathcal{N}}$ stattdessen als E_d notiert.

Ein Petrinetz $\mathcal{N} = (P, T, F)$ wird als *(schwach) zusammenhängend* bezeichnet gdw. $(x, y) \in (F \cup F^{-1})^*$ für je zwei Knoten $x, y \in (P \cup T)$ gilt. Ein Netz wird als *stark zusammenhängend* bezeichnet gdw. $(x, y) \in F^*$ für je zwei Knoten $x, y \in (P \cup T)$ gilt, d.h. es gibt für je zwei Knoten x und y einen Pfad von x nach y.

Definition 3.30 (Inverses Netz, Duales Netz)
Zu einem Petrinetz $\mathcal{N} = (P, T, F)$ ist das *inverse Netz* $\mathcal{N}^- = (P^-, T^-, F^-)$ definiert durch $P^- \stackrel{\text{def}}{=} P$, $T^- \stackrel{\text{def}}{=} T$ und $F^- \stackrel{\text{def}}{=} F^{-1}$, wobei F^{-1} die inverse Relation von F ist. Zu einem Petrinetz $\mathcal{N} = (P, T, F)$ ist das *duale Netz* $\mathcal{N}^d = (P^d, T^d, F^d)$ definiert durch $P^d \stackrel{\text{def}}{=} T$, $T^d \stackrel{\text{def}}{=} P$ und $F^d \stackrel{\text{def}}{=} F^{-1}$. ⧫

Definition dynamischer Eigenschaften

Die dynamischen Eigenschaften eines Petrinetzes ergeben sich durch die Markierung des Netzes und die sich hieraus ergebende Möglichkeit, von einer Netzmarkierung in eine andere Netzmarkierung zu schalten. Schaltfolgen stellen eine Folge solcher Markierungsübergänge dar. Eine Markierung ist von einer Markierung aus erreichbar, wenn es eine Schaltfolge gibt, die zu dieser Markierung führt. Wesentliche Eigenschaften von Petrinetzen zusammen mit einer Initialmarkierung sind die Lebendigkeit und die Beschränktheit des Netzes unter dieser Markierung.

Definition 3.31 (Markierung, Schalten, Folgemarkierung, Netzsystem)
Eine *Markierung* $\mathbf{m}$ eines Petrinetzes $\mathcal{N}$ ist eine Abbildung $\mathbf{m} : P \to \mathbb{N}_0$, die jeder Stelle eine Anzahl von Marken zuordnet. Eine Stelle ist markiert gdw. $\mathbf{m}(p) > 0$. Es wird die Notation $\mathbf{m}_\emptyset$ für die *Nullmarkierung*, die keiner Stelle eine Marke zuweist, und $[q]$ für die Markierung, die nur genau der Stelle q eine Marke zuweist, d.h. $\forall p \in (P \setminus \{q\}) : [q](p) = 0$ und $[q](q) = 1$, verwendet.
Eine Transition $t \in T$ ist *aktiviert* in einer Markierung $\mathbf{m}$ gdw. $\forall p \in {}^\bullet t : \mathbf{m}(p) \geq 1$ (notiert als Relation $\mathbf{m} \xrightarrow[\mathcal{N}]{t}$). Eine aktivierte Markierung $\mathbf{m}$ kann in ihre *Folgemarkierung* $\mathbf{m}'$ *schalten*, was als $\mathbf{m} \xrightarrow[\mathcal{N}]{t} \mathbf{m}'$ notiert wird. In diesem Fall ist die Folgemarkierung durch $\mathbf{m}'(p) \stackrel{\text{def}}{=} \mathbf{m}(p) - |F \cap \{(p,t)\}| + |F \cap \{(t,p)\}|$ definiert. Eine Markierung wird *tot* genannt, wenn sie keine Transition aktiviert. Statt $\mathbf{m} \xrightarrow[\mathcal{N}]{t}$ wird auch verkürzend $\mathbf{m} \xrightarrow{t}$ verwendet.

Ein *Netzsystem* $\mathcal{NS} = (\mathcal{N}, \mathbf{m}_0)$ ist ein zusammenhängendes Petrinetz $\mathcal{N}$ mit mindestens zwei Knoten zusammen mit einer initialen Markierung $\mathbf{m}_0$. ♦

Definition 3.32 (Schaltfolgen, erreichbare Markierungen, Aktiviertheit)
Sei $\mathbf{m}$ eine Markierung des Netzes $\mathcal{N}$ und sei $\mathbf{m} \xrightarrow{t_1} \mathbf{m}_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} \mathbf{m}_n$ das aufeinanderfolgende Schalten der Transitionen t_1 bis t_n, so wird $\sigma = t_1 t_2 \ldots t_n \in T^*$ eine *Schaltfolge* von $\mathbf{m}$ nach $\mathbf{m}_n$ genannt und als $\mathbf{m} \xrightarrow{\sigma} \mathbf{m}_n$ notiert. Hierbei kann σ auch die *leere Schaltfolge* sein. Durch $\mathbf{m} \xrightarrow{*} \mathbf{m}$ wird notiert, dass die Markierung $\mathbf{m}'$ von der Markierung $\mathbf{m}$ aus durch eine Schaltfolge $\sigma \in T^*$ erreichbar ist. $[\mathbf{m}\rangle_{\mathcal{N}} \stackrel{\text{def}}{=} \{\mathbf{m}' \mid \exists \sigma \in T^* : \mathbf{m} \xrightarrow[\mathcal{N}]{\sigma} \mathbf{m}'\}$ definiert die Menge aller von $\mathbf{m}$ aus in $\mathcal{N}$ *erreichbaren Markierungen*. (Im Folgenden wird $\mathcal{N}$ in der Regel weggelassen, wenn es aus dem Kontext hervorgeht.)
Wenn $\mathbf{m} \xrightarrow{t_1} \mathbf{m}_1 \xrightarrow{t_2} \xrightarrow{t_3} \ldots$ für eine unendliche Folge von Transitionen $\sigma = t_1 t_2 t_3 \ldots$ gilt, so ist σ eine *unendliche Schaltfolge*, notiert als $\mathbf{m} \xrightarrow{\sigma}$. Eine Schaltfolge σ ist *aktiviert* in einer Markierung $\mathbf{m}$ gdw. $\mathbf{m} \xrightarrow{\sigma} \mathbf{m}'$ für eine Markierung $\mathbf{m}'$ ist.
Zwei Transitionen t_1 und t_2 sind *nebenläufig* aktiviert in $\mathbf{m}$ gdw. ausreichend Marken im Vorbereich beider Transitionen vorhanden sind, dass sowohl t_1 als auch t_2 gleichzeitig in der Markierung $\mathbf{m}$ schalten können, also wenn $\mathbf{m} \geq (\chi[{}^\bullet t_1] + \chi[{}^\bullet t_2])$ ist. Dies wird als $\mathbf{m} \xrightarrow{\{t_1,t_2\}}$ notiert. Zwei Transitionen t_1 und t_2 sind in einem Netzsystem $(\mathcal{N}, \mathbf{m}_0)$ *nebenläufig aktivierbar* gdw. es ein $\mathbf{m} \in [\mathbf{m}_0\rangle$ mit $\mathbf{m} \xrightarrow{\{t_1,t_2\}}$ gibt. ♦

Es sei darauf hingewiesen, dass die Notation $\mathbf{m} \xrightarrow{\{t_1,t_2\}}$ nicht zu verwechseln mit der Notation $\mathbf{m} \xrightarrow{t_1 t_2}$ ist. Erstere bezeichnet die nebenläufige Aktiviertheit von t_1 und t_2, während letztere die Aktiviertheit der Schaltfolge $t_1 t_2$ beschreibt.

Definition 3.33 (Lebendigkeit, Beschränktheit, Deadlock-Freiheit)
Sei $\mathcal{NS} = (\mathcal{N}, \mathbf{m}_0)$ ein Netzsystem. Eine Transition t ist

- *tot* in der Markierung $\mathbf{m}$ gdw. $\neg\exists \mathbf{m}' \in [\mathbf{m}\rangle : \mathbf{m}' \xrightarrow{t}$,
- *lebendig* gdw. $\forall \mathbf{m} \in [\mathbf{m}_0\rangle \, \exists \mathbf{m}' : \mathbf{m} \xrightarrow{*} \mathbf{m}' \xrightarrow{t}$, d.h. für jede erreichbare Markierung $\mathbf{m}$ gibt es eine von $\mathbf{m}$ erreichbare Markierung $\mathbf{m}'$, die t aktiviert.

Ein Netzsystem $\mathcal{NS} = (\mathcal{N}, \mathbf{m}_0)$ ist *lebendig* gdw. alle Transitionen $t \in T$ lebendig sind. Ein Netzsystem $\mathcal{NS} = (\mathcal{N}, \mathbf{m}_0)$ ist *beschränkt* gdw. $\forall p \in P \, \exists k \in \mathbb{N}_+ \, \forall \mathbf{m} \in [\mathbf{m}_0\rangle : \mathbf{m}(p) < k$, d.h. wenn es für jede Stelle p eine natürliche Zahl k gibt, so dass für jede erreichbare Markierung $\mathbf{m}$ die Anzahl der Marken auf p in $\mathbf{m}$ kleiner k ist. Ein Netzsystem ist *sicher* gdw. $\forall p \in P \, \forall \mathbf{m} \in [\mathbf{m}_0\rangle : \mathbf{m}(p) \leq 1$. Ein Netzsystem ist *deadlock-frei* gdw. $\forall \mathbf{m} \in [\mathbf{m}_0\rangle \exists t \in T : \mathbf{m} \xrightarrow{t}$. ♦

Die Begriffe der Lebendigkeit und der Beschränktheit werden häufig unter dem Begriff der Wohlgeformtheit subsumiert.

Definition 3.34 (Wohlgeformtheit)
Ein Petrinetz $\mathcal{N}$ ist *wohlgeformt* gdw. es eine Markierung $\mathbf{m}_0$ gibt, so dass $(\mathcal{N}, \mathbf{m}_0)$ beschränkt und lebendig ist. ⧫

3.3.2 Betrachtung von Netzen mittels linearer Algebra

Interpretiert man Petrinetze als Graphen, so lassen sich Techniken der linearen Algebra zur Analyse von Petrinetzen verwenden. Die Markierung wird hierzu als Vektor interpretiert, der jeder Stelle des Netzes eine Anzahl von Marken zuweist. Die Inzidenzmatrix repräsentiert in diesem Fall das Netz. Sofern es im Netz keine Schleifen gibt, ist eine solche Repräsentation möglich.

Definition 3.35 (Vektor, Inzidenzmatrix, Träger, Invariante)
Gegeben ein Petrinetz $\mathcal{N} = (P, T, F)$.

- Ein *T-Vektor* (*S-Vektor*) ist ein T-indizierter (S-indizierter) Spaltenvektor $\vec{\iota} : T \to \mathbb{Q}$ ($\vec{\kappa} : P \to \mathbb{Q}$).
- Der *Nullvektor* $\vec{\mathbf{0}}$ ist derjenige Vektor, der an jeder Position eine Null aufweist. (Es wird im Weiteren nicht zwischen T- und S-indiziertem Nullvektor unterschieden, wenn dies aus dem Kontext hervorgeht. Auch wird der Zeilenvektor $\vec{\mathbf{0}}^t$ im Allgemeinen als $\vec{\mathbf{0}}$ notiert.)
- Die *Inzidenzmatrix* $[\mathcal{N}]$ von $\mathcal{N}$ ist die durch P und T indizierte Matrix $[\mathcal{N}] : P \times T \to \mathbb{Z}$, so dass

$$[\mathcal{N}](p,t) = \begin{cases} -1 & \text{wenn } p \in {}^\bullet t \setminus t^\bullet \\ 1 & \text{wenn } p \in t^\bullet \setminus {}^\bullet t \\ 0 & \text{sonst} \end{cases}$$

- Ein T-Vektor $\vec{\iota}$ von $\mathcal{N}$ ist eine *T-Invariante* gdw. $[\mathcal{N}] \cdot \vec{\iota} = \vec{\mathbf{0}}$.
- Ein S-Vektor $\vec{\kappa}$ von $\mathcal{N}$ ist eine *S-Invariante* gdw. $\vec{\kappa}^t \cdot [\mathcal{N}] = \vec{\mathbf{0}}^t$.
- Der *Träger* (die *Trägermenge*) $\|\vec{\iota}\|$ eines T-Vektors $\vec{\iota}$ ist definiert als $\|\vec{\iota}\| \stackrel{\text{def}}{=} \{t \in T \mid \vec{\iota}(t) \neq 0\}$.
- Der *Träger* $\|\vec{\kappa}\|$ eines S-Vektors $\vec{\kappa}$ ist als $\|\vec{\kappa}\| \stackrel{\text{def}}{=} \{p \in P \mid \vec{\kappa}(p) \neq 0\}$ definiert.
- Eine T-Invariante $\vec{\iota}$ (S-Invariante $\vec{\kappa}$) ist

- *semi-positiv*(*nicht-negativ*) gdw. $\vec{\iota} \neq \vec{\mathbf{0}}$ und $\forall t \in T: \vec{\iota}(t) \geq 0$ ($\vec{\kappa} \neq \vec{\mathbf{0}}$ und $\forall p \in P: \vec{\kappa}(p) \geq 0$),
- *positiv* gdw. $\forall t \in T: \vec{\iota}(t) > 0$ ($\forall p \in P: \vec{\kappa}(p) > 0$) und
- *minimal* gdw.
  - $\vec{\iota}$ ($\vec{\kappa}$) semi-positiv ist,
  - es keine T-Invariante $\vec{\iota}'$ mit $\vec{\iota}' \neq \vec{\iota}$ (S-Invariante $\vec{\kappa}' \neq \vec{\kappa}$) gibt, so dass $\|\vec{\iota}'\| \subseteq \|\vec{\iota}\|$ ($\|\vec{\kappa}'\| \subseteq \|\vec{\kappa}\|$), und
  - $\vec{\iota}: T \to \mathbb{N}_0$ ($\vec{\kappa}: P \to \mathbb{N}_0$) gilt und der größte gemeinsame Teiler aller Einträge von $\vec{\iota}$ ($\vec{\kappa}$) 1 ist.

♦

In der Literatur wird für Invarianten häufig die Einschränkung vorgenommen, dass sie Vektoren über $\mathbb{Z}$ darstellen. Für T-Invarianten wird zum Teil sogar gefordert, dass sie Vektoren über $\mathbb{N}$ darstellen. Diese Einschränkung wird Desel und Esparza (1995) (Def. 2.26 und Def. 2.35) folgend hier nicht gemacht, da sich hierdurch der Vorteil ergibt, dass die Invarianten einen Vektorraum bilden. Dies erlaubt es, Ergebnisse aus der linearen Algebra auf Netze zu übertragen.

Gleichzeitig wird hier Lautenbach (2002) folgend gefordert, dass minimale Invarianten Abbildungen auf die natürlichen Zahlen (inklusive der 0) darstellen und dass der größte gemeinsame Teiler aller ihrer Einträge 1 ist. Dies erfolgt abweichend von Desel und Esparza (1995), wo die letzte Bedingung minimaler Invarianten nicht existiert, so dass eine T-Invariante $\vec{\iota}$ (S-Invariante $\vec{\kappa}$ minimal ist gdw. $\vec{\iota}$ ($\vec{\kappa}$) semi-positiv ist, und es keine T-Invariante $\vec{\iota}'$ mit $\vec{\iota}' \neq \vec{\iota}$ (S-Invariante $\vec{\kappa}' \neq \vec{\kappa}$) gibt, so dass $\|\vec{\iota}'\| \subseteq \|\vec{\iota}\|$ ($\|\vec{\kappa}'\| \subseteq \|\vec{\kappa}\|$) (*). Um die Ergebnisse aus Desel und Esparza (1995) auf die hier verwendete Definition der minimalen Invarianten übertragen zu können, muss gezeigt werden, dass eine minimale Invariante, die eine Abbildungen auf die natürlichen Zahlen darstellt existiert gdw. eine minimale Invariante gemäß der Definition (*) existiert. Dass jede minimale Invariante aus Definition 3.35 den Bedingungen aus (*) genügt, ist trivialerweise richtig. Gibt es eine minimale T-Invariante nach (*), so kann diese für jeden von 0 verschiedenen Eintrag der T-Invariante, der kleiner als 1 ist, mit dem Kehrwert dieses Eintrags multipliziert werden. Sei c das Produkt dieser Kehrwerte, so ist $c \cdot \vec{\iota} \in \mathbb{N}^{|T|}$ ($c \cdot \vec{\kappa} \in \mathbb{N}^{|P|}$). Ist der größte gemeinsame Teiler der Einträge dann c_2, so stellt $\frac{c}{c_2} \cdot \vec{\iota} \in \mathbb{N}^{|T|}$ ($\frac{c}{c_2} \cdot \vec{\kappa} \in \mathbb{N}^{|P|}$) die gesuchte minimale Invariante dar nach obiger Definition dar.

Die *Anzahl der Elemente* eines semi-positiven T-Vektors wird im Folgenden durch $|\vec{\iota}| \stackrel{\text{def}}{=} \sum_{t \in T} \vec{\iota}(t)$ notiert. Entsprechend notiert $|\vec{\kappa}| \stackrel{\text{def}}{=} \sum_{p \in P} \vec{\kappa}(p)$ die Anzahl der Elemente eines semi-positiven S-Vektors, wobei wie für Invarianten ein Vektor semi-positiv ist, wenn $\forall t \in T: \vec{\iota}(t) \geq 0$ ($\forall p \in P: \vec{\kappa}(p) \geq 0$) gilt.

Zu einer Matrix Ma bezeichnet $\text{rg}(Ma)$ den *Rang der Matrix* und zu einem Vektorraum V bezeichnet $\dim(V)$ die *Dimension dieses Vektorraums*.

Häufig verwendet wird der Begriff des *Parikh-Vektor*s, welcher für ein Petrinetz $\mathcal{N}$ und eine Schaltfolge σ definiert ist als Vektor $\vec{\sigma} : T \to \mathbb{N}_0$ von σ, der jede Transition $t \in T$ auf die Anzahl der Vorkommen dieser Transition in σ abbildet. Für die Abbildung $\textsc{Alph}(\sigma)$ gilt folglich $\textsc{Alph}(\sigma) = \|\vec{\sigma}\|$ für eine Schaltfolge σ, wobei Alph oben definierte Abbildung auf die Menge der Elemente der Folge σ ist.

Definition 3.36 (Realisierung, Vektorensubnetz, Überdeckung)
Gegeben ein Petrinetz $\mathcal{N} = (P, T, F)$.

- Ein T-Vektor $\vec{\iota}$ ist *realisierbar* unter der Markierung $\mathbf{m}$ gdw. es eine Schaltfolge σ gibt, so dass $\mathbf{m} \xrightarrow{\sigma}$ und $\vec{\sigma} = \vec{\iota}$. Eine *Realisierung* von $\vec{\iota}$ in $\mathbf{m}$ ist eine Schaltfolge σ mit $\mathbf{m} \xrightarrow{\sigma}$ und $\vec{\sigma} = \vec{\iota}$.

- Die *Netzrepräsentation*(*Vektorensubnetz*) $\mathcal{N}_{\vec{\iota}}$ eines T-Vektors $\vec{\iota}$ ist gegeben durch das Netz $\mathcal{N}_{\vec{\iota}} = (P_{\vec{\iota}}, T_{\vec{\iota}}, F_{\vec{\iota}})$ mit

$$\begin{aligned} T_{\vec{\iota}} &\stackrel{\text{def}}{=} \|\vec{\iota}\| \\ P_{\vec{\iota}} &\stackrel{\text{def}}{=} {}^\bullet T_{\vec{\iota}} \cup T_{\vec{\iota}}^\bullet \\ F_{\vec{\iota}} &\stackrel{\text{def}}{=} F \cap ((P_{\vec{\iota}} \times T_{\vec{\iota}}) \cup (T_{\vec{\iota}} \times P_{\vec{\iota}})) \end{aligned}$$

- Die *Netzrepräsentation* $\mathcal{N}_{\vec{\kappa}}$ eines S-Vektors $\vec{\kappa}$ ist gegeben durch das Netz $\mathcal{N}_{\vec{\kappa}} = (P_{\vec{\kappa}}, T_{\vec{\kappa}}, F_{\vec{\kappa}})$ mit

$$\begin{aligned} P_{\vec{\kappa}} &\stackrel{\text{def}}{=} \|\vec{\kappa}\| \\ T_{\vec{\kappa}} &\stackrel{\text{def}}{=} {}^\bullet P_{\vec{\kappa}} \cup P_{\vec{\kappa}}^\bullet \\ F_{\vec{\kappa}} &\stackrel{\text{def}}{=} F \cap ((P_{\vec{\kappa}} \times T_{\vec{\kappa}}) \cup (T_{\vec{\kappa}} \times P_{\vec{\kappa}})) \end{aligned}$$

- $\mathcal{N}$ ist von einer T-Invariante $\vec{\iota}$ (S-Invariante $\vec{\kappa}$) *überdeckt*, wenn $\forall t \in T : \vec{\iota}(t) \neq 0$ ($\forall p \in P : \vec{\kappa}(p) \neq 0$) ist.

⧫

Ist ein T- oder S-Vektor eine T- oder S-Invariante, so wird im Folgenden auch vom *Invariantensubnetz* gesprochen, welches die Netzrepräsentation des der Invariante entsprechenden Vektors ist.

3.3.3 Definition struktureller Netzeigenschaften

Zu den Petrinetzen im Allgemeinen haben sich eine Reihe von Subklassen etabliert, die sich in der Regel leichter analysieren lassen. Eine der bekanntesten Subklassen dürfte die der Free-Choice-Netze sein (vgl. Desel, 1992; Desel und Esparza, 1995; Desel, 1996). Da die betrachteten Free-Choice-Netze S- und T-überdeckbar sind,

sind auch die Eigenschaften von Zustandsmaschinen und von markierten Graphen im weiteren Verlauf der Arbeit relevant.

Definition 3.37 (Free-Choice-Eigenschaft)
Ein Petrinetz $\mathcal{N} = (P, T, F)$ ist *free-choice* gdw. für alle $p \in P$ und alle $t \in T$ gilt, dass $(p, t) \in F$ impliziert, dass ${}^\bullet t \times p^\bullet \subseteq F$. ⧫

Definition 3.38 (Zustandsmaschine, S-Komponente, S-überdeckbar)
Gegeben ein Petrinetz $\mathcal{N} = (P, T, F)$. $\mathcal{N}$ ist eine *Zustandsmaschine*, wenn für alle $t \in T$ gilt $|{}^\bullet t| = |t^\bullet| = 1$.
Sei $\mathcal{N}_S = (P_S, T_S, F_S)$ ein Subnetz eines Petrinetzes $\mathcal{N}$, so dass $(P_S \cup T_S) \neq \emptyset$. $\mathcal{N}_S$ ist eine *S-Komponente* von $\mathcal{N}$, wenn $\mathcal{N}_S$ eine stark zusammenhängende Zustandsmaschine ist und wenn für alle $p \in P_S$ und alle $t \in T$ gilt $(p, t) \in F \implies (p, t) \in F_S$ und $(t, p) \in F \implies (t, p) \in F_S$.
Ein Netz $\mathcal{N}$ ist *S-überdeckbar*, wenn es eine Menge von S-Komponenten $\mathcal{N}_{S_j}, j \in \mathbb{N}_+$ gibt, so dass jede Stelle $p \in P$ Element einer S-Komponente ist. ⧫

Definition 3.39 (Markierter Graph, T-Komponente, T-überdeckbar)
Gegeben ein Petrinetz $\mathcal{N} = (P, T, F)$. $\mathcal{N}$ ist ein *Markierter Graph*, wenn für alle $p \in P$ gilt $|{}^\bullet p| = |p^\bullet| = 1$.
Sei $\mathcal{N}_T = (P_T, T_T, F_T)$ ein Subnetz eines Petrinetzes $\mathcal{N}$, so dass $(P_T \cup T_T) \neq \emptyset$. $\mathcal{N}_T$ ist eine *T-Komponente*, wenn $\mathcal{N}_T$ stark zusammenhängend ist, $\mathcal{N}_T$ ein markierter Graph ist und wenn für alle $p \in P$ und alle $t \in T_T$ gilt $(p, t) \in F \implies (p, t) \in F_T$ und $(t, p) \in F \implies (t, p) \in F_T$.
Ein Netz $\mathcal{N}$ ist *T-überdeckbar*, wenn es eine Menge von T-Komponenten $\mathcal{N}_{T_j}, j \in \mathbb{N}_+$ gibt, so dass jede Transition $t \in T$ Element einer T-Komponente ist. ⧫

3.3.4 Workflownetze

Workflownetze stellen einen weit verbreiteten Formalismus zur Darstellung von Prozessen in Organisationen dar. Während der folgende Unterabschnitt zunächst die grundlegenden Definitionen für Workflownetze wiedergibt, werden Ergebnisse zu Workflownetzen im nächsten Abschnitt dargestellt. Die Definitionen stammen aus van der Aalst (2000), Dehnert und Rittgen (2001) und Verbeek (2004).

Definition 3.40 (Workflownetz)
Ein Netz $\mathcal{N} = (P, T, F)$ ist ein *Workflownetz* (WF-Netz) gdw.

1. es eine *Anfangsstelle* (engl. source place) $i \in P$ gibt, so dass ${}^\bullet i = \emptyset$,

2. es eine *Senke* (engl. sink) $o \in P$ gibt, so dass $o^\bullet = \emptyset$ und

3. jeder Knoten $x \in P \cup T$ auf einem Pfad von i nach o liegt.

⧫

Ein wichtiges Kriterium dafür, ob ein Ablauf durch ein Workflownetz sinnvoll modelliert worden ist, ist das Korrektheitskriterium. Dieses besagt, dass alle Transitionen in einer von $[i]$ erreichbaren Markierung schalten können müssen und dass stets die finale Markierung erreichbar sein muss. Da die initiale Markierung von Workflownetzen stets $[i]$ ist, wird diese zum Teil nicht explizit genannt. So wird ein Workflownetz beispielsweise beschränkt genannt, wenn es in der Markierung $[i]$ beschränkt ist.

Definition 3.41 (Korrektheit von Workflownetzen)
Ein Workflownetz (WF-Netz) $\mathcal{N} = (P, T, F)$ heißt *korrekt* gdw.

1. $\forall \mathbf{m} \in [[i]\rangle : \mathbf{m} \xrightarrow{*} [o]$,

2. $\forall\, t \in T\, \exists\, \mathbf{m}, \mathbf{m}' : [i] \xrightarrow{*} \mathbf{m} \xrightarrow{t} \mathbf{m}'$.

⧫

Die Bedingung $\forall \mathbf{m} \in [[i]\rangle : \mathbf{m} \geq [o] \Rightarrow (\mathbf{m} = [o])$, die häufig ebenfalls als Korrektheitskriterium angegeben wird, folgt aus den in Definition 3.41 gegebenen Bedingungen.

Zum Teil wird das Kriterium der Korrektheit als zu stark empfunden, da es sich in der Regel nur recht aufwendig überprüfen lässt. Aus diesem Grund hat sich zudem das Kriterium der relaxten Korrektheit etabliert.

Definition 3.42 (Relaxte Korrektheit von Workflownetzen)
Ein Workflownetz (WF-Netz) $\mathcal{N}$ heißt *relaxt korrekt* gdw.

$$\forall\, t \in T\, \exists\, \mathbf{m}, \mathbf{m}' : [i] \xrightarrow{*} \mathbf{m} \xrightarrow{t} \mathbf{m}' \xrightarrow{*} [o].$$

⧫

Das Konzept der relaxten Korrektheit wurde von Dehnert und Rittgen (2001) eingeführt. Offensichtlich implizieren die Bedingungen der Korrektheit auch die relaxte Korrektheit.

3.3.5 Algebraische Petrinetze

Algebraische Petrinetze wurden in Reisig (1991a) vorgestellt. Sie sind eine Ausprägung der gefärbten Petrinetzen von Jensen (1983) und definieren die möglichen Farbmengen als Sorten einer heterogenen Signatur. An dieser Stelle sollen lediglich algebraische Netze vorgestellt werden, eine Einführung in gefärbte Petrinetze findet sich bei Jensen (1992) und bei Jensen und Kristensen (2009).

Algebraische Petrinetze verwenden eine algebraische Spezifikation, die die Struktur der im Netz verwendeten Marken vorgibt. Mehrere Marken auf einer Stelle werden durch Multimengen mit entsprechender Multimengenspezifikation dargestellt. Nähere Erläuterungen zu Multimengen finden sich ebenfalls bei Jensen und Kristensen (2009). Um Multimengen in einem algebraischen Netz nutzen zu können, wird an dieser Stelle zunächst eine Multimengenspezifikation eingeführt, bevor dann algebraische Netze definiert werden.

Multimengen

Multimengen erlauben gegenüber Mengen das Mehrfachvorhandensein eines Objektes in einer Menge. Sie eignen sich daher zur Repräsentation von Markierungen, da jede Marke im Allgemeinen mehrfach auf einer Stelle vorhanden sein kann.

Definition 3.43 (Multimengen)
Eine *Multimenge* über der Menge M ist eine Abbildung $\mathbf{a} : M \to \mathbb{N}$. Die *leere Multimenge* $\mathbf{0}$ ist definiert als $\mathbf{0}(x) \stackrel{\text{def}}{=} 0$ für alle $x \in M$. Die *Kardinalität* einer Multimenge ist definiert als $|\mathbf{a}| \stackrel{\text{def}}{=} \sum_{x \in M} \mathbf{a}(x)$. Eine Multimenge $\mathbf{a}$ wird *endlich* genannt, wenn ihre Kardinalität verschieden von unendlich ist. Die Menge aller endlichen Multimengen über eine Menge M wird als $MS(M)$ notiert. Eine Multimenge $\mathbf{a}$ kann als formale Summe $\mathbf{a} = \sum_{x \in M} \mathbf{a}(x) \cdot x$ dargestellt werden. Eine Multimenge mit genau einem Element a wird auch als $[a]$ dargestellt.
Die Summe zweier Multimengen $\mathbf{a} + \mathbf{b}$ ist definiert als $(\mathbf{a} + \mathbf{b})(x) \stackrel{\text{def}}{=} \mathbf{a}(x) + \mathbf{b}(x)$. Die *Differenz* zweier Multimengen $\mathbf{a} - \mathbf{b}$ ist definiert als $(\mathbf{a} - \mathbf{b})(x) \stackrel{\text{def}}{=} \max(\mathbf{a}(x) - \mathbf{b}(x), 0)$. Die Gleichheit $\mathbf{a} = \mathbf{b}$ ist elementweise definiert als $\forall x \in M : \mathbf{a}(x) = \mathbf{b}(x)$. Eine Abbildung $f : M \to M'$ kann in eine entsprechende Abbildung von Multimengen $f : MS(M) \to MS(M')$ überführt werden, wobei gilt

$$f\left(\sum_{i=1}^{n} a_i\right) = \sum_{i=1}^{n} f\left(a_i\right)$$

♦

Multimengen stellen eine Generalisierung von Mengen dar, da jede Menge M einer Multimenge $\mathbf{a}$ entspricht, für die gilt $\mathbf{a}(x) \leq 1$ für alle $x \in M$. Multimengen besitzen eine partielle Ordnung $\mathbf{a} \leq \mathbf{b}$ gdw. $\forall x \in M : \mathbf{a}(x) \leq \mathbf{b}(x)$.

Die Definition der Multimengenspezifikationen baut auf Defintion 3.43 auf und stellt auch Terme über Multimengen bereit. Die Sorte der Multimengen über einer Sorte s wird ebenfalls als $MS(s)$ bezeichnet, wie dies auch für die Menge aller Multimengen über einer Menge der Fall war.

Definition 3.44 (Multimengenspezififkation)
Zu einer gegebenen Spezifikation $\Gamma = (S, \Omega, E)$ ist die *Multimengenspezifikation* gegeben durch ein Tupel $\Gamma^{MS} = (S^{MS}, \Omega^{MS}, E^{MS})$, definiert durch

- $S^{MS} \stackrel{\text{def}}{=} S \cup \{MS(s) | s \in S\}$,
- $\Omega^{MS} \stackrel{\text{def}}{=} \Omega \cup \{0_s, make_s, +_s, -_s \mid s \in S\}$ und
- $E^{MS} \stackrel{\text{def}}{=} E \cup \bigcup_{s \in S} E_s$,

wobei $0_s \in \Omega^{MS}_{MS(s)}$, $+_s \in \Omega^{MS}_{MS(s)MS(s),MS(s)}$, $-_s \in \Omega^{MS}_{MS(s),MS(s)}$ und $make_s \in \Omega^{MS}_{s,MS(s)}$ die Einbettung eines Terms der Sorte s in die entsprechende Multimenge ist. Die Menge der Gleichungen E_s ist definiert als

$$\begin{aligned} E_s \stackrel{\text{def}}{=} \; & \{+_s(a,\, 0_s) = a, \\ & +_s(a,\, b) = +_s(b,\, a), \\ & +_s(a,\; +_s(b,\, c)) = +_s(+_s(a,\, b),\, c), \\ & +_s((a,\; -_s(a))) = 0_s\} \end{aligned}$$

wobei $a, b, c \in X_{MS(s)}$ Variablen der Sorte $MS(s)$ sind. ⧫

Wie üblich wird die Infix-Notation verwendet und es werden der *make*-Operator und überflüssige Klammern wie auch die Angabe der Sorte bei den Operatoren auf Multimengen weggelassen, so dass für zwei Konstanten a, b einer beliebigen Sorte s die Multimengenaddition der Negation von b als $a - b$ oder als $a + (-b)$ notiert wird, anstatt umständlich $+_s(make_s(a), -_s(make_s(b)))$ schreiben zu müssen. Wie auf Multimengen ist auch auf Termen die Beziehung $\geq$ definiert, so dass für zwei Multimengenterme a und b gilt $a \geq b$ wenn $a - b$ nicht negativ ist, d.h. der entsprechende Multimengenterm kann ohne Verwendung von $-_s, s \in S$ definiert werden.

Algebraische Netzdefinition

Mittels der Multimengenspezifikation kann nun beschrieben werden, was ein algebraisches Netz ausmacht. Dieses ist wie folgt definiert, wobei eine boolesche Signatur eine Signatur ist, die die Sorte *bool* besitzt.

Definition 3.45 (Algebraisches Petrinetz)
Ein algebraisches Petrinetz ist ein Tupel $\mathcal{APN} \stackrel{\text{def}}{=} (\Gamma, X, \mathcal{N}, d, W, G)$ wobei

- $\Gamma = (S, \Omega, E)$ eine Spezifikation ist mit (S, Ω) als einer booleschen Signatur und mit der Multimengenspezifikation Γ^{MS},
- X eine S-indizierte Familie von Variablen ist,
- $\mathcal{N} = (P, T, F)$ ein Petrinetz ist,
- $d : P \to S$ eine *Stellentypisierung* ist,
- $W : F \to \mathbb{T}_{\Gamma^{MS}}(X)$ eine *Kantenbeschriftung* ist mit $W(f) \in \mathbb{T}_{\Gamma^{MS}, MS(d(p))}(X)$ für $f = (p, t) \in F$ oder $f = (t, p) \in F$ und
- $G : T \to \mathbb{T}_{\Gamma, bool}(X)$ eine *Aktivierungsbedingung* (oder *Guard*) ist.

Die *Initialmarkierung* $\mathbf{m}_0$ eines algebraischen Netzes ist gegeben durch eine Abbildung der Form $\mathbf{m}_0 : P \to \mathbb{T}_{\Gamma^{MS}}$ mit $\mathbf{m}_0(p) \in \mathbb{T}_{\Gamma^{MS}, MS(d(p))}(\emptyset)$. ♦

Um die Ausführbarkeit algebraischer Netze zu gewährleisten wird häufig zusätzlich gefordert, dass, wenn eine Variable an einer Ausgangskante einer Transition verwendet wird, diese auch an einer Eingangskante verwendet werden muss. Dies erleichtert die Bindungssuche für diese Variablen. Formal seien $\text{VAR}^-(t) \stackrel{\text{def}}{=} \bigcup_{p \in \bullet t} \text{VAR}(W(p, t))$ und $\text{VAR}^+(t) \stackrel{\text{def}}{=} \bigcup_{p \in t^\bullet} \text{VAR}(W(t, p))$ die Mengen der Variablen and Ein- und Ausgangskanten der Transition t. Die Einschränkung ist dann als $W(t, p) \in \mathbb{T}_{\Gamma^{MS}, MS(d(p))}(\text{VAR}^-(t))$ definiert.

Definition 3.46 (Markierung und Schaltregel algebraischer Netze)

Gegeben ein algebraisches Petrinetz $\mathcal{APN} = (\Gamma, X, \mathcal{N}, d, W, G)$. Eine *Markierung* $\mathbf{m}$ von $\mathcal{APN}$ ist eine Abbildung $\mathbf{m} : P \to \mathbb{T}_{\Gamma^{MS}}$ von den Stellen auf die Grundterme der Signatur Σ, so dass jedem Platz Werte über die Multimengen-Sorte $MS(d(p))$ zugewiesen werden, d.h. $\mathbf{m}(p) \in \mathbb{T}_{\Gamma^{MS}, MS(d(p))}(\emptyset)$.

Eine Transition t wird *aktiviert* unter der Variablenbelegung $\alpha : X \to \mathbb{T}_\Gamma$ genannt, gdw. die Aktivierungsbedingung $[\![G(t)]\!]_\alpha = \mathit{true}$ erfüllt ist und für alle $p \in P$ auch $\mathbf{m}(p) \geq [\![W(p, t)]\!]_\alpha$ gilt. Das *Schalten* einer Transition unter einer Variablenbelegung α wird als $\mathbf{m} \xrightarrow{t,\alpha} \mathbf{m}'$ notiert. Die *Folgemarkierung* des Schaltens $\mathbf{m}'$ ist für jeden Platz $p \in P$ definiert als $\mathbf{m}'(p) = \mathbf{m}(p) - [\![W(p, t)]\!]_\alpha + [\![W(t, p)]\!]_\alpha$. ♦

Algebraische Netze verwenden meist allein die Syntax zur Definition des Schaltverhaltens eines Netzes. Variablen werden durch die Belegung Elemente (also Termklassen) der Quotiententermalgebra zugewiesen, die die Terme interpretiert. Dies bedeutet, dass eine nicht-initiale Algebra (zu Γ) unter Umständen ein anderes Schaltverhalten aufweist, da Terme anders interpretiert werden.

3.4 Bekannte Eigenschaften von Petrinetzen und Workflownetzen

Nachdem der letzte Abschnitt die wichtigsten im Rahmen dieser Arbeit verwendeten Konzepte definiert hat, widmet sich dieser Abschnitt der Zusammenfassung bekannter Ergebnisse aus verschiedenen Quellen. Der Fokus liegt auf den Eigenschaften, die zur Bestimmung der Erreichbarkeit einer Markierung und zur Bestimmung der Lebendigkeit und Sicherheit eines Netzes notwendig sind. Die Ergebnisse aus diesem Abschnitt werden in erster Linie in Kapitel 5 und in Kapitel 6 verwendet.

3.4.1 Erreichbarkeit von Markierungen

Die Erreichbarkeit einer Markierung stellt eine der zentralen Eigenschaften bei der Analyse von Netzen im weiteren Verlauf dieser Arbeit dar. In diesem Bereich kann bereits auf verschiedene Ergebnisse für einfachere Netze zurückgegriffen werden, die an dieser Stelle zusammengefasst werden sollen.

Eine Markierung $\mathbf{m}$ wird im Folgenden auch als Spaltenvektor interpretiert. Es wird daher im Folgenden $[q]$ auch den Spaltenvektor bezeichnen, der genau an der der Stelle q entsprechenden Position eine 1 besitzt, während alle anderen Positionen 0 sind. Dies wird im weiteren Verlauf auch für T-indizierte Vektoren genutzt. Es ergibt sich dann folgender Satz.

Satz 3.47 (Markierungsgleichung)
Gegeben ein Petrinetz $\mathcal{N}$. Für jede endliche Schaltfolge σ mit $\mathbf{m} \xrightarrow{\sigma} \mathbf{m}'$ von $\mathcal{N}$ gilt die folgende Markierungsgleichung:

$$\mathbf{m}' = \mathbf{m} + [\mathcal{N}] \cdot \vec{\sigma}.$$

Der Beweis findet sich in Desel und Esparza (1995) als Beweis zu Proposition 2.12.

Definition 3.48 (Auf S-Invarianten übereinstimmende Markierungen)
Sei $\mathcal{N}$ ein Netz. Zwei Markierungen $\mathbf{m}_1$ und $\mathbf{m}_2$ von $\mathcal{N}$ stimmen auf allen *S-Invarianten überein* gdw. $\vec{\kappa}^t \cdot \mathbf{m}_1 = \vec{\kappa}^t \cdot \mathbf{m}_2$ für alle S-Invarianten $\vec{\kappa}$ von $\mathcal{N}$. ⧫

Satz 3.49 (Notwendige Bedingung für Erreichbarkeit)
Sei $\mathcal{N}$ ein Netz mit Initialmarkierung $\mathbf{m}_0$. Jede erreichbare Markierung $\mathbf{m} \in [\mathbf{m}_0\rangle$ stimmt mit $\mathbf{m}_0$ auf allen S-Invarianten überein.

Der Beweis findet sich in Desel und Esparza (1995) als Beweis zu Theorem 2.33.

Satz 3.50 (Hinreichende Bedingung für Erreichbarkeit)
Sei $\mathcal{N}$ ein zyklenfreies Petrinetz mit Initialmarkierung $\mathbf{m}_0$. Es ist $\mathbf{m} \in [\mathbf{m}_0\rangle$ gdw. es einen semi-positiven T-Vektor $\vec{\iota_*}$ gibt, so dass

$$\mathbf{m}_0 + [\mathcal{N}] \cdot \vec{\iota_*} = \mathbf{m}.$$

Der Beweis findet sich in Murata (1989) als Beweis zu Theorem 16.

Satz 3.51 (Bedingung für Übereinstimmung auf allen S-Invarianten)
Sei $\mathcal{N}$ ein Petrinetz. Zwei Markierungen $\mathbf{m}_1$ und $\mathbf{m}_2$ von $\mathcal{N}$ stimmen auf allen S-Invarianten überein gdw. $\mathbf{m}_1 + [\mathcal{N}] \cdot X = \mathbf{m}_2$ für einen T-indizierten Spaltenvektor $X \in \mathbb{Q}^{|T|}$.

Der Beweis findet sich in Desel und Esparza (1995) als Beweis zu Theorem 2.34.

3.4.2 Eigenschaften von Invarianten

Verwendet man linear-algebraische Methoden zur Analyse der Netze, so bietet sich die Analyse über Invarianten an. In diesem Zusammenhang existieren bereits verschiedene Ergebnisse zu den Eigenschaften von Invarianten und zu ihrer Verwendung bei der Analyse von Netzeigenschaften.

Satz 3.52 (Eigenschaften von Invarianten)
Sei $\mathcal{N}$ ein Petrinetz.

1. Eine Abbildung $\vec{\iota} : T \to \mathbb{Q}$ ist eine T-Invariante gdw. für alle $p \in P$ gilt
$$\sum_{t \in {}^\bullet p} (\vec{\iota}(t)) = \sum_{t \in p^\bullet} (\vec{\iota}(t)).$$
2. Eine Abbildung $\vec{\kappa} : P \to \mathbb{Q}$ ist eine S-Invariante gdw. für alle $t \in T$ gilt
$$\sum_{p \in {}^\bullet t} (\vec{\kappa}(p)) = \sum_{p \in t^\bullet} (\vec{\kappa}(p)).$$
3. Es ist für jede semi-positive S-Invariante ${}^\bullet\|\vec{\kappa}\| = \|\vec{\kappa}\|^\bullet$ und für jede semi-positive T-Invariante ${}^\bullet\|\vec{\iota}\| = \|\vec{\iota}\|^\bullet$.
4. Jedes zusammenhängende Netz $\mathcal{N}$ mit einer positiven T-Invariante und einer positiven S-Invariante ist stark zusammenhängend.

Der Beweis findet sich in Desel und Esparza (1995) als Beweis zu Proposition 2.36 (1), Proposition 2.28 (2), Proposition 2.41 (3) und Theorem 2.40 (4).

Wenn T-Invarianten realisiert werden können, so reproduzieren sie eine Markierung. Dies ist für lebendige und beschränkte Netze interessant, da eine positive T-Invariante eine Voraussetzung für die Lebendigkeit und die Beschränktheit ist.

Lemma 3.53 (Schaltfolgen und T-Invarianten)
Sei $\mathcal{N}$ ein Petrinetz, $\mathbf{m}$ eine Markierung von $\mathcal{N}$ und sei σ eine endliche nicht-leere in $\mathbf{m}$ aktivierte Schaltfolge. Dann ist der Parikh-Vektor $\vec{\sigma}$ eine T-Invariante gdw. $\mathbf{m} \xrightarrow{\sigma} \mathbf{m}$ (also wenn die Schaltfolge σ $\mathbf{m}$ reproduziert).

Der Beweis findet sich in Desel und Esparza (1995) als Beweis zu Proposition 2.37.

Satz 3.54 (Wohlgeformtheit und T-Invarianten)
Jedes wohlgeformte Petrinetz $\mathcal{N}$ besitzt eine positive T-Invariante.

Der Beweis findet sich in Desel und Esparza (1995) als Beweis zu Theorem 2.38.

Minimale Invarianten werden im Verlauf dieser Arbeit häufiger verwendet, da sich aus ihnen eine Basis des Vektorraums der Invarianten erstellen lässt, wenn ein Netz eine positive Invariante besitzt. Dies wird in Kapitel 5 detaillierter dargestellt.

Satz 3.55 (Eigenschaften minimaler Invarianten)
Gegeben ein Petrinetz $\mathcal{N} = (P, T, F)$.

- Jede semi-positive S-Invariante (T-Invariante) ist die Linearkombination minimaler S-Invarianten (T-Invarianten).
- Wenn ein Petrinetz eine positive S-Invariante (T-Invariante) besitzt, dann ist jede S-Invariante (T-Invariante) eine Linearkombination minimaler S-Invarianten (T-Invarianten).

Der Beweis findet sich in Desel und Esparza (1995) als Beweis zu Theorem 2.43. Da hier eine andere Definition minimaler Invarianten verwendet wird, wurde im Anschluss an Definition 3.35 gezeigt, dass sich jede minimale Invariante nach Desel und Esparza (1995) durch Multiplikation einer minimalen Invariante nach Definition 3.35 mit einer Konstanten erzeugen lässt.

Zur Analyse der Lebendigkeit eines Netzes wird häufig die Analyse über Siphons herangezogen. Ein Netz, in dem ein Siphon nicht markiert ist, kann nicht lebendig sein. Daher sind folgende Begriffe von Interesse, die im Bereich der Free-Choice-Netze noch einmal aufgegriffen werden.

Definition 3.56 (Siphon, Falle)
Eine Menge von Stellen $S \subseteq P$ eines Petrinetzes $\mathcal{N} = (P, T, F)$ ist

- ein *Siphon* gdw. ${}^\bullet S \subseteq S^\bullet$ und $S \neq \emptyset$ ist. Ein Siphon S ist *minimal*, wenn kein anderes Siphon S' mit $S' \subset S$ existiert,
- eine *Falle* gdw. $S^\bullet \subseteq {}^\bullet S$ und $S \neq \emptyset$ ist. Eine Falle S ist *minimal*, wenn keine andere Falle S' mit $S' \subset S$ existiert.

⧫

Ein Siphon bleibt unmarkiert, wenn es einmal unmarkiert ist, eine Falle bleibt markiert, wenn sie einmal markiert ist. Das nicht-leere Siphon und die nicht-leere Falle werden häufig auch als *ordentlich* (engl. proper) bezeichnet, wobei der Zusatz $S \neq \emptyset$ dann für allgemeine Siphons und Fallen fehlt. Im Folgenden werden jedoch lediglich ordentliche Siphons betrachtet.

3.4.3 Eigenschaften von Free-Choice-Netzen

Free-Choice-Netze stellen eine besonders gut erforschte Subklasse von Petrinetzen dar. Da diese Netzklasse einerseits bereits eine Vielzahl von Ergebnissen bereitstellt und sie andererseits für die meisten Workflows zur Modellierung ausreichend ist, liegt der Fokus dieser Arbeit auch auf der Verwendung dieser Netzklasse. Von besonderem Interesse ist das Rang-Theorem zur Entscheidung, ob ein Free-Choice-Netz sicher und lebendig ist. Eine umfangreiche Sammlung der Ergebnisse zu Free-Choice-Netzen bietet Desel und Esparza (1995), wobei auf Ergebnissen aus Desel (1992) aufgebaut wird.

Definition 3.57 (Cluster)
Gegeben ein Netz $\mathcal{N}$. Sei $n \in (P \cup T)$. Das *Cluster* von n wird als $[n]_C$ notiert und ist definiert als die minimale Menge von Knoten für die gilt:

- $n \in [n]_C$.
- Wenn $p \in [n]_C \cap P$, so ist $p^\bullet \subseteq [n]_C$.
- Wenn $t \in [n]_C \cap T$, so ist ${}^\bullet t \subseteq [n]_C$.

Die *Menge der Cluster* von $\mathcal{N}$ wird als $C_\mathcal{N}$ notiert. ⧫

Mit Hilfe von Clustern lässt sich einer der zentralen Sätze zur Charakterisierung der Wohlgeformtheit von Free-Choice-Netzen formulieren. Das von Campos u. a. (1991) beschriebene Rang-Theorem, das hier nach Desel und Esparza (1995) zitiert ist.

Satz 3.58 (Rang-Theorem)
Sei $\mathcal{N}$ ein Netz mit der Free-Choice-Eigenschaft und sei $C_{\mathcal{N}}$ die Menge der Cluster von $\mathcal{N}$. $\mathcal{N}$ ist wohlgeformt gdw.

1. $\mathcal{N}$ zusammenhängend ist und mindestens eine Stelle und eine Transition besitzt,
2. $\mathcal{N}$ eine positive T-Invariante besitzt,
3. $\mathcal{N}$ eine positive S-Invariante besitzt und
4. $\text{rg}([\mathcal{N}]) = |C_{\mathcal{N}}| - 1$ ist.

Der Beweis findet sich in Desel und Esparza (1995) als Beweis zu Theorem 6.14.

Satz 3.59 (Lebendige und beschränkte Free-Choice-Netze)
Sei $\mathcal{N}$ ein Netz mit der Free-Choice-Eigenschaft und sei $\mathbf{m}_0$ eine Markierung von $\mathcal{N}$. Das Netzsystem $(\mathcal{N}, \mathbf{m}_0)$ ist lebendig und beschränkt gdw.

1. $\mathcal{N}$ wohlgeformt ist und
2. $\mathbf{m}_0$ alle S-Komponenten markiert.

Der Beweis findet sich in Desel und Esparza (1995) als Beweis zu Theorem 5.8.

Satz 3.60 (Erreichbarkeit in wohlgeformten Free-Choice-Netzen)
Sei $(\mathcal{N}, \mathbf{m}_0)$ ein lebendiges und beschränktes Netzsystem und besitze $\mathcal{N}$ die Free-Choice-Eigenschaft. Weiterhin sei $\mathbf{m}_0$ von jeder Markierung in $[\mathbf{m}_0\rangle$ erreichbar. Dann ist eine Markierung $\mathbf{m} \in [\mathbf{m}_0\rangle$ gdw. $\mathbf{m}$ und $\mathbf{m}_0$ auf allen minimalen S-Invarianten übereinstimmen und $\mathbf{m}$ jede ordentliche Falle von $\mathcal{N}$ markiert.

Der Beweis findet sich in Desel und Esparza (1995) als Beweis zu Theorem 9.6.

Satz 3.61 (Minimale Siphons in wohlgeformten Free-Choice-Netzen)
Sei S ein minimales Siphon eines wohlgeformten Free-Choice-Netzes $\mathcal{N}$. Dann ist

- S eine Falle und
- das Subnetz aus S und ${}^\bullet S$ eine S-Komponente.

Der Beweis findet sich in Desel und Esparza (1995) als Beweis zu Proposition 5.4.

Interessant für die weitere Analyse ist vor allem auch die Tatsache, dass das duale Netz eines wohlgeformten Free-Choice-Workflownetzes ebenfalls ein wohlgeformtes Free-Choice-Workflownetz ist.

Satz 3.62 (Dualitäts-Theorem)
Ein Netz $\mathcal{N}$ ist ein wohlgeformtes Free-Choice-Netz gdw. $\mathcal{N}^d$ ein wohlgeformtes Free-Choice-Workflownetz ist.

Der Beweis findet sich in Desel und Esparza (1995) als Beweis zu Theorem 6.21.

3.4.4 Eigenschaften von Zustandsmaschinen und markierten Graphen

Zustandsmaschinen und markierte Graphen stellen eine besonders einfache Netzklasse dar, die aus diesem Grund besonders gut untersucht ist. Vielfach lassen sich Ergebnisse von dieser Netzklasse auf Netze übertragen, die lediglich aus einer Kombination von Zustandsmaschinen bestehen, weil sie S-überdeckbar oder T-überdeckbar sind. Aus diesem Grund werden wesentliche Ergebnisse für diese Netzklassen hier wiedergegeben.

Satz 3.63 (Zustandsmaschinen, S-Komponenten und S-Invarianten)
Für eine zusammenhängende Zustandsmaschine $\mathcal{N}_1$ ist ein S-Vektor $\vec{\kappa}$ eine S-Invariante gdw. $\vec{\kappa} = c \cdot (1 \dots 1)$ mit $c \in \mathbb{Q}$.
Ist $\mathcal{N}_S$ eine S-Komponente eines Netzes $\mathcal{N}$, so ist $\vec{\kappa} = \chi[P_S]$ eine minimale S-Invariante von $\mathcal{N}$.

Der Beweis findet sich in Desel und Esparza (1995) als Beweis zu Proposition 3.7 und Proposition 5.7.

Satz 3.64 (Eigenschaften von T-Komponenten)
Sei $\mathcal{N}_T = (P_T, T_T, F_T)$ eine T-Komponente von $\mathcal{N}$. Dann gilt

- für alle Stellen $p \in P_T$, dass $|{}^\bullet_{(\mathcal{N})}p \cap T_T| = 1 = |p^\bullet_{(\mathcal{N})} \cap T_T|$,
- für jede Markierung $\mathbf{m}$ von $\mathcal{N}$ und jede Schaltfolge σ mit $\textsc{Alph}(\sigma) \subseteq T_T$, dass $\mathbf{m} \xrightarrow{\sigma} \mathbf{m}'$ in $\mathcal{N}$ gdw. $\mathbf{m}|_{P_T} \xrightarrow{\sigma} \mathbf{m}'|_{P_T}$, und
- für den Vektor $\vec{\iota} = \chi[T_T]$, dass er eine minimale T-Invariante von $\mathcal{N}$ ist.

Der Beweis findet sich in Desel und Esparza (1995) als Argumentation zu Proposition 5.12 und Proposition 5.14.

Satz 3.65 (Lebendigkeit und Beschränktheit von Zustandsmaschinen)
Sei $\mathcal{N}$ eine stark zusammenhängende Zustandsmaschine und $\mathbf{m}$ eine Markierung von $\mathcal{N}$.

- $(\mathcal{N}, \mathbf{m})$ ist lebendig gdw. $\mathbf{m}(p) > 0$ für ein $p \in P$.

- $(\mathcal{N}, \mathbf{m})$ ist sicher gdw. $\mathbf{m}$ genau eine Stelle aus P mit genau einer Marke markiert.

Der Beweis findet sich in Desel und Esparza (1995) als Beweis zu Theorem 3.3 und als Beweis zu Theorem 3.5.

Satz 3.66 (Lebendigkeit und Beschränktheit von markierten Graphen)
Sei $\mathcal{N}$ ein markierter Graph und $\mathbf{m}$ eine Markierung von $\mathcal{N}$.

- $\mathcal{N}$ ist unter der Markierung $\mathbf{m}$ lebendig gdw. jeder Zyklus in $\mathcal{N}$ initial markiert ist.

- $\mathcal{N}$ ist unter der Initialmarkierung $\mathbf{m}$ sicher gdw. es zu jeder Stelle $p_* \in P$ einen Zyklus γ in $\mathcal{N}$ mit $p_* \in \textsc{Alph}(\gamma)$ gibt, so dass $\mathbf{m}(\gamma) \leq 1$ ist, wobei $\mathbf{m}(\gamma) = \sum_{p \in \textsc{Alph}(\gamma)} \mathbf{m}(p)$ ist.

Der Beweis findet sich in Desel und Esparza (1995) als Beweis zu Theorem 3.15 und als Beweis zu Theorem 3.18.

Lemma 3.67 (Zyklen in Zustandsmaschinen sind T-Invarianten)
Sei $\mathcal{N}$ eine Zustandsmaschine, γ ein Zyklus von $\mathcal{N}$ und $T_\gamma = \textsc{Alph}(\gamma) \cap T$ die Menge der Transitionen in γ. Dann ist der charakteristische Vektor von T_γ $\chi[T_\gamma]$ bezüglich T eine T-Invariante von $\mathcal{N}$.

Der Beweis findet sich in Desel und Esparza (1995) als Beweis zu Lemma 3.9.

Satz 3.68 (Erreichbarkeit in markierten Graphen)
Sei $(\mathcal{N}, \mathbf{m}_0)$ ein lebendiges Netzsystem und $\mathcal{N}$ ein markierter Graph. Dann ist $\mathbf{m} \in [\mathbf{m}_0\rangle_{\mathcal{N}}$ gdw. $\mathbf{m}$ und $\mathbf{m}_0$ auf allen S-Invarianten übereinstimmen.

Der Beweis findet sich in Desel und Esparza (1995) als Beweis zu Theorem 3.21.

3.4.5 Bekannte Ergebnisse zu Workflownetzen

Im Rahmen der Workflownetze gibt es eine Reihe etablierter Ergebnisse. Ein wesentliches Ergebnis ist hierbei die Bestimmung der Korrektheit eines Workflownetzes über seinen Abschluss. Weiterhin ist die Verfeinerung einer Transition eines Workflownetzes durch ein weiteres Netz interessant. Beides wird hier kurz vorgestellt. Eine detailliertere Diskussion der Konzepte findet sich jeweils an der zu den Sätzen angegebenen Stelle.

Definition 3.69 (Abschluss eines Workflownetzes)
Der *Abschluss* eines WF-Netzes $\mathcal{N}$ ist definiert als Netz $\overline{\mathcal{N}} \stackrel{\text{def}}{=} (\overline{P}, \overline{T}, \overline{F})$, welches durch eine zusätzliche Transition t^* entsteht, die o mit i verbindet, d.h. $\overline{\mathcal{N}} = (P, T \cup \{t^*\}, F \cup \{(o, t^*), (t^*, i)\})$. ♦

Satz 3.70 (Starker Zusammenhang des Abschlusses)
Der Abschluss $\overline{\mathcal{N}}$ eines Workflows $\mathcal{N}$ ist stark zusammenhängend.

Der Beweis findet sich in van der Aalst (1997).

Satz 3.71 (Korrektheit, Lebendigkeit und Beschränktheit)
Ein Workflownetz $\mathcal{N}$ ist korrekt gdw. $(\overline{\mathcal{N}}, [i])$ lebendig und beschränkt ist.

Der Beweis findet sich in van der Aalst (1997).

Wiederum wird hierbei zum Teil auf die explizite Nennung der Initialmarkierung $[i]$ verzichtet, da diese für Workflownetze aus dem Kontext hervorgeht. In diesem Fall wird lediglich davon gesprochen, dass $\mathcal{N}$ korrekt ist gdw. $\overline{\mathcal{N}}$ lebendig und beschränkt ist.

Eine relativ einfache Klasse von Workflownetzen stellen die wohlstrukturierten Workflownetze dar. Zu ihrer Definition wird zunächst die Definition des Handles benötigt.

Definition 3.72 (Handle, well-handled)
In einem Petrinetz $\mathcal{N} = (P, T, F)$ wird ein Knoten-Paar $(n_1, n_2) \in (P \cup T) \times (P \cup T)$ *Handle* genannt gdw. es zwei elementare Pfade von n_1 nach n_2 gibt, die lediglich die beiden Knoten n_1 und n_2 gemeinsam haben, d.h. $\exists \pi_0, \pi_1 \in E_d(n_1, n_2) : \pi_0 \neq \pi_1 \wedge \textsc{Alph}(\pi_0) \cap \textsc{Alph}(\pi_1) = \{n_1, n_2\}$. Für ein *PP-Handle* gilt $(n_1, n_2) \in P \times P$, für ein *PT-Handle* gilt $(n_1, n_2) \in P \times T$. Analog sind *TT-Handle* und *TP-Handle* definiert.
Ein Petrinetz ist *well-handled* gdw. es keine PT-handle und keine TP-handle besitzt. ♦

Definition 3.73 (Wohlstrukturierte Workflownetze)
Ein WF-Netz $\mathcal{N}$ ist *wohlstrukturiert* gdw. sein Abschluss $\overline{\mathcal{N}}$ well-handled ist. ♦

Eine Konsequenz aus Satz 3.49 ist, dass S-überdeckbare Workflownetze in der Markierung $[i]$ sicher sind.

Lemma 3.74 (S-Überdeckbarkeit impliziert Sicherheit)
Sei $\mathcal{N}$ ein WF-Netz, so dass der Abschluss $\overline{\mathcal{N}}$ S-überdeckbar ist. Dann ist $\overline{\mathcal{N}}$ unter der Initialmarkierung $[i]$ sicher.

Der Beweis findet sich in Verbeek (2004), Beweis zu Theorem 4.3.

Die Möglichkeit, Transitionen zu verfeinern, erlaubt es Workflows mit Hilfe von Workflownetzen sukzessive zu modellieren und aufzubauen.

Definition 3.75 (Transitionsverfeinerung)
Seien $\mathcal{N}_i = (P_i, T_i, F_i),\ i \in \{1, 2\}$ zwei WF-Netze, so dass $T_1 \cap T_2 = \emptyset$, $P_1 \cap P_2 = \emptyset$ und $t^+ \in T_1$ und seien i_i und o_i die Anfangsstelle bzw. die Senke des Netzes $\mathcal{N}_i$. $\mathcal{N}_3 = (P_3, T_3, F_3)$ ist das WF-Netz, das man erhält, wenn man Transition t^+ in $\mathcal{N}_1$ durch $\mathcal{N}_2$ ersetzt, d.h. $P_3 = (P_1 \cup P_2) \setminus \{i_2, o_2\}$, $T_3 = (T_1 \setminus \{t^+\}) \cup T_2$ und

$$\begin{aligned} F_3 \quad = \quad & \{(x,y) \in F_1 | x \neq t^+ \wedge y \neq t^+\} \cup \{(x,y) \in F_2 | \{x,y\} \cap \{i_2, o_2\} = \emptyset\} \cup \\ & \{(x,y) \in P_1 \times T_2 | (x, t^+) \in F_1 \wedge (i_2, y) \in F_2\} \cup \\ & \{(x,y) \in T_2 \times P_1 | (t^+, y) \in F_1 \wedge (x, o_2) \in F_2\} \end{aligned}$$

$i_3 = i_1$ and $o_3 = o_1$. $\mathcal{N}_3$ ist das zusammengesetzte WF-Netz von $\mathcal{N}_1$ und $\mathcal{N}_2$, welches durch *Verfeinerung* der Task t^+ entsteht. ⧫

Bei der Transitionsverfeinerung bleiben wesentliche Eigenschaften der Workflownetze erhalten, wie folgender Satz zeigt.

Satz 3.76 (Eigenschaften der Transitionsverfeinerung)
Für die Netze $\mathcal{N}_i = (P_i, T_i, F_i),\ i \in \{1, 2, 3\}$, wobei $\mathcal{N}_3$ das durch die Verfeinerung von $t^+ \in T_1$ durch $\mathcal{N}_2$ entstandene WF-Netz ist und i_i die Anfangsstelle von Netz $\mathcal{N}_i$ ist, gelten folgende Aussagen:

1. Wenn $\mathcal{N}_3$ wohlstrukturiert ist, dann sind auch $\mathcal{N}_1$ und $\mathcal{N}_2$ wohlstrukturiert.
2. Wenn $(\mathcal{N}_1, [i_1])$ sicher ist und $\mathcal{N}_1$ und $\mathcal{N}_2$ korrekt sind, dann ist $\mathcal{N}_3$ korrekt.
3. $(\mathcal{N}_1, [i_1])$ und $(\mathcal{N}_2, [i_2])$ sind sicher und korrekt gdw. $(\mathcal{N}_3, [i_3])$ sicher und korrekt ist.

Der Beweis findet sich in (van der Aalst, 2000, Beweis zu Theorem 3).

3.5 Zusammenfassung

Dieses Kapitel hat Algebren und Petrinetze eingeführt und erste Ergebnisse aus beiden Bereichen geliefert. Durch die Verwendung einer ordnungssortierten Algebra lässt sich bereits erkennen, wie Daten in einem Netz modelliert werden können.

In algebraischen Netzen findet sich dann eine konkrete Verwendung von Signaturen und Spezifikationen in Netzen. Dies legt ein ähnliches Vorgehen für Dienstbeschreibungsnetze nahe. Die Nutzung von Workflownetzen als zugrundeliegender Netzstruktur erlaubt die Verwendung des Korrektheitsbegriffs für Workflownetze, so dass die Korrektheit des zugrundeliegenden Workflownetzes als notwendige Bedingung für die Korrektheit eines Dienstbeschreibungsnetzes genutzt werden kann.

Auch lassen sich die bekannten Ergebnisse zu Petrinetzen nutzen, um die Analysemöglichkeiten für Dienstbeschreibungsnetze aufzuzeigen. Die hierzu notwendigen zusätzlichen Ergebnisse werden in Kapitel 5 ausgehend von den Ergebnissen aus Abschnitt 3.4 hergeleitet. Bislang existiert noch kein Verfahren, um festzustellen, wann zwei Transitionen nebenläufig aktivierbar sind. Im Zusammenhang mit der Verwendung von Daten wird diese Frage wichtig, da entschieden werden muss, ob es zu Race-Conditions kommen kann, weil zwei nebenläufige Transitionen auf die gleichen Daten zugreifen. Des Weiteren ist es interessant, ob ein Netz dadurch vereinfacht werden kann, dass die Transitionen zu einer Zustandsmaschine sequentialisiert werden. Dies erlaubt im Zusammenhang mit Daten ebenfalls eine bessere Analysemöglichkeit. Beiden Fragen wird in Kapitel 5 nachgegangen.

4 Dienstbeschreibungsnetze

Dieses Kapitel beschreibt den in dieser Arbeit zentralen Formalismus der Dienstbeschreibungsnetze zur Modellierung von dienstbasierten Systemen, dessen Einführung auf Köhler und Ortmann (2005) zurückgeht. Dienstbeschreibungsnetze besitzen einen Workflownetzcharakter und erlauben es, über Anschriften die Veränderungen an den Daten im Netz darzustellen. In diesem Kapitel werden zunächst in Abschnitt 4.1 der Formalismus allgemein erläutert und die zu klärenden Fragen erörtert. Hieran schließt sich die Beschreibung der Anschriften von Dienstbeschreibungsnetzen in Abschnitt 4.2 an, bevor dann Dienstbeschreibungsnetze in Abschnitt 4.3 definiert werden. In Abschnitt 4.4 werden hieran anschließend die zentralen Eigenschaften dieser Netzklasse beleuchtet.

4.1 Eine informelle Einführung in Dienstbeschreibungsnetze

Die Modellierung von Workflows soll in diesem Abschnitt zunächst an einem Beispiel entwickelt werden, bevor dann im weiteren Verlauf des Abschnitts dargestellt wird, welche Eigenschaften sich analysieren lassen und wie dies erfolgen kann.

Modellierung von Workflows

Dienstbeschreibungsnetze dienen der Modellierung dienstbasierter Workflows in Organisationen. Klassische Beispiele für solche Workflows sind die Auftragsabwicklung etwa bei Eingang einer Bestellung in einem Unternehmen oder das Beschwerdemanangement. Der Eingang einer Bestellung soll hier als Beispiel dienen, um die Konzepte hinter den Dienstbeschreibungsnetzen besser erläutern zu können. Ein Kunde A bestellt bei einem Unternehmen B eine Reihe von Artikeln. Dies führt dazu, dass bei Unternehmen B eine Bestellung eingeht, in der die Daten von Kunde A zusammen mit den bestellten Artikeln aufgeführt sind. Unter Umständen werden weitere Daten übergeben, wie eine Lieferadresse oder nähere Angaben zu den Bezahlmodalitäten, von denen hier jedoch abstrahiert werden soll, so dass der Kunde nur über Kreditkarte zahlen kann.

Der Eingang der Bestellung startet einen Prozess, nach dessen Abarbeitung der Kunde A die Artikel erhalten hat und das liefernde Unternehmen B einen entsprechenden Betrag für die Artikel erhalten hat, der zuvor Kunde A in Rechnung gestellt

wurde. Hierzu müssen im Wesentlichen zwei Schritte erfolgen. Nach Eingang der Bestellung muss die Zahlung veranlasst werden, und es muss die Ware ausgeliefert werden. Dies lässt sich als Workflownetz mit drei Stellen und zwei Transitionen darstellen, wie dies in Abbildung 4.1 zu sehen ist.

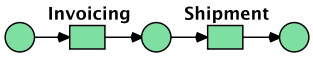


Abbildung 4.1: Einfache Darstellung eines Bestelleingangs

Diese Darstellung reicht zur Dokumentation der Prozesse aus, sie erlaubt jedoch keine weitergehenden Betrachtungen bezüglich der Daten, der Schnittstellen oder bezüglich der Funktionalität der Prozesse. So wäre es beispielsweise möglich, dass verschiedene Implementationen des Workflows unter der Auslieferung völlig verschiedene Dinge verstehen. Auch wäre es denkbar, dass die eintreffende Bestellung vollkommen unterschiedlich interpretiert wird. So mag in einer Implementation die Bestellung eine Liste der bestellten Ware besitzen, während in einer anderen Implementation in jeder Bestellung nur ein Artikel bestellt werden kann, so dass eine Bestellung auf mehrere Bestellungen aufgeteilt werden muss. Um zu gewährleisten, dass die eingehenden Aufträge das gleiche Format haben, werden Konzeptklassen oder Fachklassen (vgl. Abschnitt 2.4) gebildet, die angeben, wie ein Objekt aufgebaut ist. Für eine Bestellung könnte etwa die Fachklasse *PurchaseOrder* existieren, die die Attribute *item*, *customer* und *payment* besitzt.

Durch eine Angabe des Typs der eingehenden Daten und der ausgehenden Daten ist zumindest sichergestellt, dass alle relevanten Informationen auch tatsächlich für die übergebenen Daten existieren. So ist es nicht möglich, dass ein *PurchaseOrder*-Objekt kein *customer*-Attribut besitzt und somit unklar ist, wer eigentlich der Kunde ist. Die Typsicherheit wird erreicht, indem die Stellen des Workflownetzes mit den Fachklassen typisiert werden, so dass sichergestellt ist, dass die übergebenen Daten den richtigen Typ besitzen. Dies führt zu einem Workflownetz der Art, wie es in Abbildung 4.2 dargestellt ist. In diesem Netz sind die Datentypen in blauer Schrift dargestellt, um sie von den Namen der Knoten unterscheiden zu können.

Abbildung 4.2: Darstellung eines Bestelleingangs mit Datentypen

Führt man obiges Beispiel fort, so schickt Kunde A eine Bestellung, die ein Fachobjekt der Fachklasse *PurchaseOrder* darstellt. Diese besitzt als *customer*-Attribut Kunde A und als *payment*-Attribut eine Bezahlmethode wie beispielsweise die

Zahlung per Kreditkarte. Das *payment*-Attribut verweist folglich auf ein weiteres Fachobjekt, das dann die Kreditkarteninformationen enthält. Ebenso verweist das *item*-Attribut auf ein Fachobjekt vom Typ *Item*. Dies ist graphisch in Abbildung 4.3 dargestellt. Die Klassen *Payment* und *CreditcardPayment* stehen in einer Subklassen-Beziehung, so dass statt einer Bezahlung per Kreditkarte auch andere Bezahlungsmodi denkbar wären, was hier jedoch nicht dargestellt ist.

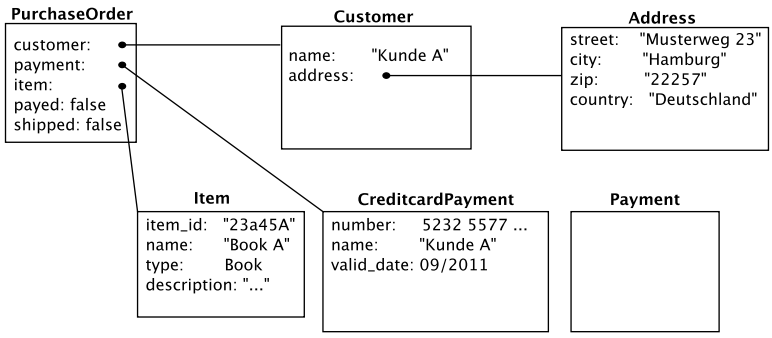


Abbildung 4.3: Die Fachobjekte mit konkreten Werten für einen Bestelleingang

Die Ausführung der Dienste hat wiederum einen Einfluss auf die übergebenen Objekte. So ändert sich das Attribut *payed* nach Zahlungseingang zu *true*. Dies sollte ebenfalls darstellbar sein, so dass sich drei Ebenen ergeben. Einmal die Ebene der Prozesse dargestellt in Abbildung 4.1, in der lediglich der Ablauf dargestellt ist, einmal die Ebene des Datenflusses in Abbildung 4.2, in der der Datenfluss dargestellt ist, und einmal die Ebene der Funktionalität, die darstellt, wie sich ein Fachobjekt ändert, wenn es an einem Dienstaufruf beteiligt ist. Betrachtet man lediglich das Auftragsobjekt, so mag eine solche Veränderung wie in Abbildung 4.4 dargestellt vor sich gehen. Für das *PurchaseOrder*-Objekt wird zunächst das Attribut *payed* auf *true* gesetzt und dann das Attribut *shipped* auf *true* gesetzt.

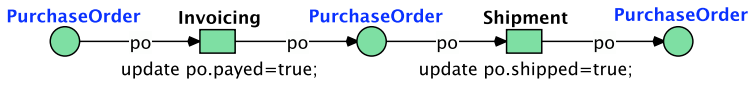


Abbildung 4.4: Ein Bestelleingang mit Daten und deren Veränderungen

Dabei enthält die Eingangsstelle des Netzes zunächst eine Referenz auf ein Objekt vom Typ *PurchaseOrder*, das die in Abbildung 4.3 dargestellten Verknüpfungen zu anderen Objekten besitzt. Eine Änderung eines Attributwertes erfolgt anhand von *Update*-Anweisungen, die die Interpretation der Attribute ändern. Sie werden mit dem Schlüsselwort *update* eingeführt. Die Auswirkungen einer solchen Update-Anweisung sind global sichtbar, da eine entsprechende Änderung für das gesamte

System gilt, was den realen Auswirkungen in einer Organisation entspricht. Die in Abbildung 4.4 dargestellten Update-Anweisung ändern zunächst den Wert des Attributes *payed* und dann den Wert des Attributes *shipped* jeweils zu *true*.

Neben der Änderung von Attribut-Werten ist auch die Erzeugung neuer Objekte häufig notwendig. In obigem Beispiel ist es beispielsweise notwendig, ein Objekt vom Typ *Invoice* zu erzeugen, damit später nachvollzogen werden kann, welche Rechnung an einen Kunden geschickt wurde. In der Regel soll diese Rechnung dann dem Bestellungsauftrag zugeordnet werden. Es kann nun entweder ein zusätzliches Attribut *invoice* im Fachobjekt *PurchaseOrder* erstellt werden, oder es kann ein dynamisches Konzept geben, wie sie in Abschnitt 2.4.2 eingeführt wurden, das das *PurchaseOrder*-Objekt erweitert und zusätzliche Attribute bereitstellt.

Der Ansatz der dynamischen Konzepte bietet mehr Flexibilität und erlaubt es, auch bestehende Systeme zu erweitern, so dass bei der initialen Modellierung von Objekten nicht auf alle potentiellen Änderungen Rücksicht genommen werden muss. Zudem sind häufig dynamische Konzepte nur in bestimmten Situationen von Interesse, so dass eine generelle Modellierung nicht notwendig ist. Besitzt beispielsweise ein Mitarbeiter im Unternehmen die Rolle Vertriebsleiter, so ist dies eine dynamisches Konzept, da die entsprechenden Eigenschaften dynamisch hinzukommen können oder auch wieder entfernt werden können. Gleichzeitig wäre es jedoch nicht sinnvoll bei jedem Mitarbeiter die Attribute des Vertriebsleiters vorzusehen, da diese Attribute bei allen anderen Mitarbeitern nicht sinnvoll sind, so dass die beste Lösung zur Modellierung ein dynamisches Konzept ist.

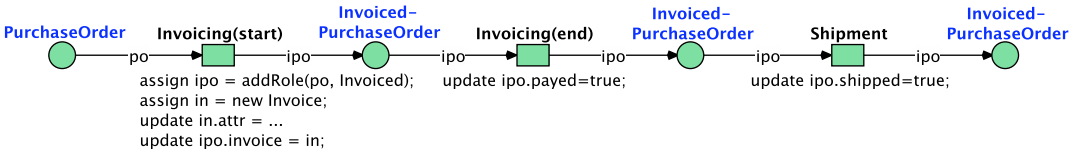


Abbildung 4.5: Ein Bestelleingang mit dynamischen Konzepten

Abbildung 4.5 stellt den Ablauf aus Abbildung 4.4 dar, wobei drei neue Fachklassen hinzugekommen sind. Einmal gibt es die statische Fachklasse *Invoice*, die eine Rechnung repräsentiert, und einmal gibt es die dynamischen Fachklassen *Invoiced* und *InvoicedPurchaseOrder*. Die Klasse *Invoiced* stellt eine dynamische Klasse dar, die verschiedene Klassen erweitern kann und unter anderem auch die Klasse *PurchaseOrder* erweitert. Sie stellt das zusätzliche Attribut *invoice* bereit. Alle Objekte, die sowohl der Klasse *Invoiced* als auch der Klasse *PurchaseOrder* angehören, gehören dann auch der Klasse *InvoicedPurchaseOrder* an, so dass im System Mehrfachvererbung möglich ist. Die möglichen Attribute des *Invoice*-Objektes wurden hier nur schematisch anhand von Punkten dargestellt.

Neben dem Schlüsselwort *update* gibt es auch Zuweisungsanweisungen, die über

das Schlüsselwort *assign* eingeführt werden. Sie weisen einer Variablen einen Wert zu, der dann später im Netz genutzt werden kann. Um beispielsweise das neue Objekt der Klasse *Invoice* nutzen zu können, wird dieses einer Variablen *in* zugewiesen. Der an diese Variable gebundene Wert (in diesem Fall die Objektreferenz auf das neu erzeugte Objekt) kann dann in einer Update-Anweisung genutzt werden. Ebenso wird das um einen dynamischen Typ erweiterte *PurchaseOrder*-Objekt an die Variable *ipo* gebunden, damit diese dann die richtige Typisierung besitzt.

Gilt vor der Ausführung des Ablaufs die Objektbeschreibung aus Abbildung 4.3, so hat die Ausführung des Ablaufs die Objekte und die Attribute erweitert. Dies ist in Abbildung 4.6 dargestellt. Das ursprüngliche Objekt gehört nun der dynamischen Klasse *InvoicedPurchaseOrder* mit den Subklassen *PurchaseOrder* und *Invoiced* an, deren Attribute *payed*, *shipped* und *invoice* sich geändert haben. Zudem haben sich die Attribute des Objektes der Klasse *Invoice* geändert, die hier wiederum durch Punkte angedeutet sind.

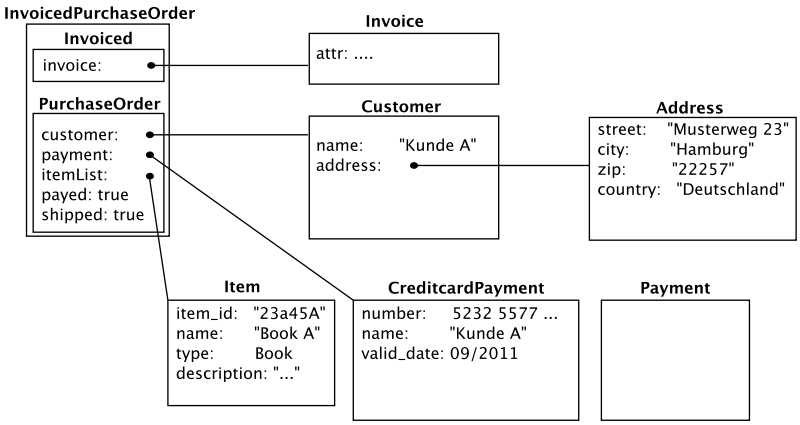


Abbildung 4.6: Die Fachobjekte nach Ausführung des Ablaufs in Abbildung 4.5

Diese Möglichkeiten der Modellierung sollen in einen Formalismus überführt werden. Dabei wurden Petrinetze als zugrundeliegender Formalismus für die Darstellung der Abläufe gewählt, da Petrinetze in diesem Bereich eine gut erforschte Grundlage bieten. Wie in dem Beispiel zu sehen ist, ist zudem eine Notation zur Darstellung der Daten und eine Notation zur Darstellung der Veränderung der Daten notwendig. Zur Darstellung der Daten dienen ordnungssortierte Algebren, die in Abschnitt 3.2.3 eingeführt wurden. Das Verändern der Algebren orientiert sich an der Vorgehensweise abstrakter Zustandsmaschinen, die in Abschnitt 2.4.3 kurz erläutert worden sind.

Detailliertere Beschreibung von Dienstbeschreibungsnetzen

Dienstbeschreibungsnetze sind Workflownetze, die gefärbte Marken besitzen und die einen Systemzustand durch Anweisungen an den Transitionen ändern. Um die im letzten Unterabschnitt am Beispiel diskutierten Modellierungsmöglichkeiten zu bieten, müssen neben den Petrinetzen zur Darstellung des Ablaufs auch Formalismen zur Darstellung von Daten verwendet werden. Hierzu werden die aus den algebraischen Netzen (vgl. Abschnitt 3.3.5) bekannten Algebren verwendet, wobei in diesem Fall die Besonderheit besteht, dass die Sorten der Algebra eine Ordnung besitzen. Diese Ordnung erlaubt es Subtyp-Beziehungen darzustellen. Im Beispiel in Abbildung 4.3 besitzt die Klasse *Payment* die Subklasse *CreditcardPayment*. Auf diese Weise können verschiedene Zahlungswege im Modell angegeben werden. Diese Möglichkeit wird durch eine ordnungssortierte Algebra formalisiert und auf Netze übertragen. Die Marken des Netzes sind dabei Elemente einer ordnungssortierten Algebra, die Objektreferenzen oder primitive Datentypen repräsentieren.

Die wesentlichen Elemente algebraischer Netze bleiben bei Dienstbeschreibungsnetzen erhalten, allerdings wird die Beschreibung der Zustandsänderungen von den Kanten an die Transitionen verschoben, wie dies auch bei den Java-Referenznetzen von Kummer (2002) der Fall ist. Dies besitzt den Vorteil, dass bei einer Transitionsverfeinerung durch einen Dienst klar definiert ist, welche Anweisungen dieser Dienst erfüllen muss. Zudem besitzt dies für die Analyse den Vorteil, dass es eine starke Korrespondenz zwischen dem Verhalten des Dienstbeschreibungsnetzes und dem Verhalten des zugrundeliegenden Workflownetzes gibt, da gefordert wird, dass an den Kanten nur Variablen (keine Terme) als Anschrift stehen. Zudem darf jede Variable an jeder Eingangskante einer Transition nur maximal einmal vorkommen, so dass keine Unifikation von Eingangsvariablen stattfindet. Wenn zwei Eingangsvariablen gleich sein sollen, muss dies vielmehr explizit durch eine Aktivierungsbedingung ausgedrückt werden. Soll eine Variable einen bestimmten Wert haben, muss dies ebenfalls über Aktivierungsbedingungen zugesichert werden.

Eine Möglichkeit der Modellierung, die in algebraischen Netzen nicht vorhanden ist, bieten Dienstbeschreibungsnetze durch die Möglichkeit, die Sortenzugehörigkeit von Elementen der Algebra zu ändern. Dies reflektiert die dynamischen Fachklassen, die im Beispiel im letzten Unterabschnitt eingeführt wurden. Das Netz in Abbildung 4.5 nutzt das dynamische Konzept *Invoiced*, das anzeigt, dass ein Objekt abgerechnet wurde. Ein bestehendes Objekt erhält diesen Typ zusätzlich und besitzt somit weitere Attribute. Dies erlaubt eine direktere Abbildung des tatsächlichen Verhaltens in organisationalen Abläufen auf das Modell.

Neben der Darstellung des Ablaufs durch Petrinetze und der Verwendung von gefärbten Marken ist ein wesentlicher Bestandteil des vorgestellten Modells die Darstellung des aktuellen Zustands eines Systems sowie die Darstellung seiner Zustandsveränderung. Die Darstellung eines Zustands als Algebra ist durch den Formalismus der Evolving Algebra (vgl. Gurevich, 1988, 1991, 1995) und der hieraus

hervorgegangenen abstrakten Zustandsmaschinen (Börger und Stärk, 2003) relativ weit verbreitet. Überlegungen, den Zustand einer verteilten abstrakten Zustandsmaschine durch algebraische Petrinetze zu repräsentieren, finden sich für den Dienstkontext bei Köhler und Ortmann (2005). Eine Diskussion der Verwandtheit von Petrinetzen und abstrakten Zustandsmaschinen erfolgt bei Glausch und Reisig (2006). Eine andere verwandte Herangehensweise findet sich bei Große-Rhode (2004), wo ein System ebenfalls durch eine Signatur beschrieben wird, deren Interpretation durch Gleichungen – also anhand sich ändernder Spezifikationen – geschieht.

Die möglichen Anweisungen zur Veränderung der Objekte im Netz sind durch die folgende Tabelle 4.1 gegeben. Wie im Beispiel im letzten Unterabschnitt gezeigt, können Objekte mit einem neuen dynamischen Typ versehen werden, oder es können dynamische Typen entfernt werden. Gleichzeitig können Objekte neu erzeugt werden, und es können die Attribute bestehender Objekte verändert werden. Das Symbol τ bezeichnet dabei jeweils einen Term, dessen Auswertung unter einer Variablenbelegung dann ein Objekt referenziert bzw. im Fall der Zuweisung auch einen primitiven Datentyp darstellt. Jeder Zustand stellt eine Algebra dar. Die Auswertung des Terms τ im Zustand Q unter der Variablenbelegung α wird dabei wie in Abschnitt 3.2.1 definiert als $[\![\tau]\!]^Q_\alpha$ notiert.

Anweisung	Umgangssprachliche Interpretation
$addRole(\tau, \mathfrak{c})$	Fügt dem Objekt, das der Auswertung des Terms τ entspricht, das dynamische Konzept $\mathfrak{c}$ hinzu.
$removeRole(\tau, \mathfrak{c})$	Entfernt das dynamische Konzept $\mathfrak{c}$ von dem Objekt, das der Auswertung des Terms τ entspricht.
$new\ (v, \mathfrak{c})$	Erzeugt ein neues Objekt, das dem statischen Konzept $\mathfrak{c}$ angehört, und weist es v zu.
$update\ \tau.\omega = \tau'$	Ändert den Wert des Attributs ω des Objektes, das der Auswertung des Terms τ entspricht.
$assign\ v = \tau$	Weist der Variable v die Auswertung des Terms τ zu.

Tabelle 4.1: Anweisungen und ihre umgangssprachliche Bedeutung

Die diesem Vorgehen zugrundeliegende Sicht auf den Zustand eines Netzes ist in Abbildung 4.7 schematisch dargestellt. Hierbei wird die Möglichkeit des parallelen Zugriffs auf den Zustand vernachlässigt, da diese für die Modellierung nicht relevant ist. Wie später diskutiert wird, lässt sich ein gleichzeitiges Schalten zweier Transitionen in der praktischen Ausführung dadurch erreichen, dass nur Teile eines Zustands durch eine Transition reserviert werden. Das Netz in Abbildung 4.7 stellt

kein Dienstbeschreibungsnetz dar. Es soll lediglich illustrieren, wie diese Netzklasse durch gefärbte oder algebraische Petrinetze als bekanntem Formalismus repräsentiert werden kann. Der zentrale Zustand stellt dabei ein Objekt dar, auf dem die entsprechenden Operationen und die zugehörigen Abbildungen aus Tabelle 4.1 definiert sind.

Sämtliche Anschriften zur Zustandsänderung befinden sich in einem Dienstbeschreibungsnetz an den Transitionen. Ebenso werden Zuweisungen über Transitionsanschriften vorgenommen, so dass die Kanten lediglich durch Variablennamen beschriftet sind. Auch dies grenzt Dienstbeschreibungsnetze von algebraischen Netzen ab, bei denen Operatoren an den Kanten zugelassen sind. Der Vorteil hierbei ist, dass Dienstbeschreibungsnetze direkt die Kommunikation zwischen Diensten über Kanäle, die als Stellen modelliert sind, darstellen können. Da ein Kanal keine Modifikation vornimmt, sind hier lediglich Variablen erlaubt.

Das Schalten einer Transition führt sämtliche Zustandsänderungsanweisungen auf dem aktuellen Zustand aus und ersetzt diesen durch einen Folgezustand. In Abbildung 4.7 stellt Q den aktuellen Zustand dar, der sich aufgrund der Eingangsvariablen (p) und aufgrund der Transitionsanschriften, die in **Cmd** zusammengefasst wurden, ändert. Die Darstellung orientiert sich am Formalismus der algebraischen Netze. Hierbei wird $[\![\mathbf{Cmd}]\!]^q_\alpha$ als Operator verstanden, der die Liste von Anweisungen **Cmd** im Zustand q unter der Variablenbelegung α auswertet und q entsprechend auf den Folgezustand abbildet.

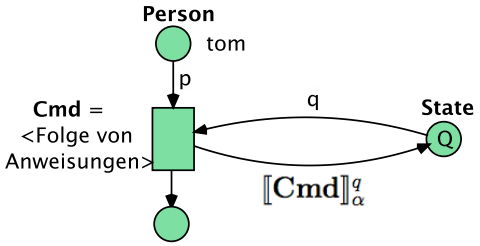


Abbildung 4.7: Netz mit zentralem Zustand

Durch den Systemzustand wird das Lokalitätsprinzip der Netze verletzt, da Änderungen, die durch eine Transition durchgeführt werden, Auswirkungen auf eine Transition haben können, ohne dass diese Transitionen gemeinsame Stellen im Vor- oder Nachbereich besitzen. Dies ist explizit gewollt, da es gerade der Vorteil der Dienstbeschreibungsnetze ist, auch solche Szenarien untersuchen zu können, in denen Transitionen sich über den globalen Zustand synchronisieren müssen.

Eine weitere Konsequenz des zentralen Systemzustands und seiner Reservierung während des Schaltens einer Transition, ist, dass, die Anschriften an Transitionen immer atomar ausgeführt werden. Wie dies dann praktisch implementiert wird,

ist nicht Gegenstand dieser Arbeit. Eine mögliche stärkere Parallelisierung der Zugriffe wird in Unterabschnitt 4.4.4 kurz skizziert. Nicht diskutiert wird an dieser Stelle auch die Verwendung von transaktionalem Verhalten, das garantiert, dass eine Variable sich über eine Ausführungsfolge hinweg nicht ändert.

Eigenschaften von Dienstbeschreibungsnetzen

Zur Charakterisierung von Dienstbeschreibungsnetzen dienen eine Reihe von Eigenschaften, die sich zum Teil an den entsprechenden Eigenschaften von Workflownetzen orientieren. Die wichtigsten Eigenschaften sind in diesem Zusammenhang die funktionale Nebenläufigkeit, die Typisierungskonformität und die Korrektheit, die hier kurz skizziert werden sollen und in Abschnitt 4.4 detaillierter behandelt werden.

Bei der nebenläufigen Ausführung von Teilen eines Dienstbeschreibungsnetzes kann es dazu kommen, dass Race-Conditions auftreten, die dazu führen, dass das funktionale Verhalten eines Dienstbeschreibungsnetz von der zufälligen Schaltreihenfolge der Transitionen abhängt. Dies soll unterbunden werden, so dass Dienstbeschreibungsnetze bei einer Eingabe stets ein vorhersehbares Verhalten aufweisen. Zwei Transitionen, die nebenläufig schalten können, bei denen der Ergebniszustand jedoch unabhängig von der Schaltreihenfolge ist, werden als funktional nebenläufig aktiviert bezeichnet. Bei nebenläufig aktivierten Transitionen führt folglich eine beliebige Permutation der Schaltsequenz zum gleichen Systemzustand.

Die Typisierungskonformität eines Dienstbeschreibungsnetzes beschreibt hingegen die Eigenschaft, dass Netze nicht durch das Entfernen einer Marke in einer Markierung schalten können, wenn sie dies nicht auch vorher konnten. Dies kann geschehen, wenn Dienstbeschreibungsnetze *removeRole*-Anweisungen besitzen. Typisierungskonformität und funktional nebenläufige Aktiviertheit sorgen also dafür, dass ein Netz sich so verhält, wie man es von ihm erwartet, also dass bei ausreichender Zahl von Marken im Vorbereich einer Transition und bei Auswertung der Aktivierungsbedingung dieser Transition zu wahr auch ein Schalten der Transition möglich ist.

An die Korrektheit von Workflownetzen angelehnt ist der Korrektheitsbegriff von Dienstbeschreibungsnetzen. Wie für Workflownetze wird die eindeutige Termination und die Fortsetzbarkeit einer Schaltfolge gefordert. Die Aktivierbarkeit aller Transitionen wird hingegen nur für einen starken Korrektheitsbegriff gefordert, da gerade bei der Nutzung von Diensten, die für verschiedene Aufgaben konzipiert wurden, nicht immer alle Transitionen aktivierbar sein müssen. Neben diesen Forderungen darf es für korrekte Dienstbeschreibungsnetze keine Race-Conditions bei der Ausführung von Transitionen geben. Sie müssen also funktional nebenläufig aktivierbar sein.

Da nicht jede Anfangsmarkierung eines Netzes gültig sein muss, können die Markierungen von Dienstbeschreibungsnetzen durch initialbeschriftete Dienstbeschrei-

bungsnetze eingeschränkt werden. Initialbeschriftete Dienstbeschreibungsnetze besitzen eine initiale Netzmarkierung, die einem Prädikat genügen muss. Dieses Prädikat schränkt die möglichen initialen Netzzustände über die Typisierung der Anfangsstelle hinaus ein, was es erlaubt, das Verhalten eines Netzes für bestimmte Klassen initialer Netzzustände zu analysieren. Initialbeschriftete Dienstbeschreibungsnetze umfassen dabei die Klasse der Dienstbeschreibungsnetze, da für letztere das Prädikat stets als wahr angenommen werden kann. Die Korrektheit eines Dienstbeschreibungsnetzes wird für initialbeschriftete Dienstbeschreibungsnetze formalisiert, da hierdurch bei Bedarf die Menge möglicher initialer Netzzustände eingeschränkt werden kann.

Aufbau des Kapitels

Dieses Kapitel führt Dienstbeschreibungsnetze ein. Ein wesentlicher Bestandteil der Dienstbeschreibungsnetze sind die Anweisungen zur Änderung eines Systemzustandes. Daher wird in Abschnitt 4.2 zunächst das Konzept des Zustandes und des Systemzustandes erläutert, bevor dann auf die Anweisungen eines Dienstbeschreibungsnetzes eingegangen wird. Die Interpretation der Transitionsanschriften, die hier zunächst informell eingeführt wurde, wird dazu in Unterabschnitt 4.2.2 präzisiert. Die Anweisungen zur Veränderung eines Zustandes orientieren sich an den Veränderungsanweisungen abstrakter Zustandsmaschinen von Börger und Stärk (2003). Hierbei geben Lokationen den Operator und das Element des Definitionsbereichs an, an dem die Interpretation eines Operators verändert werden soll.

In Abschnitt 4.3 werden Dienstbeschreibungsnetze eingeführt. Dies erfolgt über eine Beschreibung der Syntax, die die wesentlichen Bestandteile eines Dienstbeschreibungsnetzes anhand ihres syntaktischen Aufbaus einführt. Das Schaltverhalten von Dienstbeschreibungsnetzen ist Gegenstand von Unterabschnitt 4.3.2, wobei zwischen Dienstbeschreibungsnetzen mit *removeRole*-Anweisungen und solchen ohne diese Anweisungen unterschieden wird, da erstere Anweisungen dazu führen können, dass Stellen beziehungsweise die Marken auf ihnen das Schalten einer Transition verhindern, was mit dem Verhalten von Inhibitor-Kanten (vgl. Christensen und Hansen, 1993) verwandt ist.

Unterabschnitt 4.3.3 bettet den Formalismus der Dienstbeschreibungsnetze in bestehende Formalismen ein. Hierbei wird einmal die Beziehung zu Workflownetzen und einmal die Beziehung zu algebraischen Netzen dargestellt. Ein Ergebnis dieses Vergleichs ist, dass die Fragen der Erreichbarkeit, der Beschränktheit und der Lebendigkeit des Abschlusses für Dienstbeschreibungsnetze im Allgemeinen unentscheidbar sind.

Abschnitt 4.4 widmet sich der Darstellung der wichtigsten Eigenschaften von Dienstbeschreibungsnetzen, die dann im nächsten Kapitel für elementare Dienstbeschreibungsnetze untersucht werden. Dies sind einmal die Schwierigkeiten bei der nebenläufigen Ausführung von Transitionen, die die gleiche Lokation verändern.

Da hierdurch Race-Conditions auftreten können, ist dieses Verhalten grundsätzlich nicht wünschenswert. Dienstbeschreibungsnetze, die ein solches Verhalten nicht aufweisen, werden in Unterabschnitt 4.4.1 als funktional (nebenläufig) ausführbar eingeführt.

Die Einführung von Korrektheitskriterien für Dienstbeschreibungsnetze in Unterabschnitt 4.4.3 orientiert sich an den Korrektheitskriterien für Workflownetze. Diese Kriterien werden erweitert, und es wird zwischen schwacher Korrektheit, bei der jeder mögliche initiale Netzzustand zu einem finalen Netzzustand führen muss, und starker Korrektheit, bei der zusätzlich jede Transition in einem von einem initialen Netzzustand erreichbaren Netzzustand aktivierbar sein muss, unterschieden. Die Aktivierbarkeit jeder Transition ist in praktischen Szenarien nicht immer notwendig und ist zudem weit aufwendiger zu zeigen, so dass diese Unterscheidung sinnvoll ist.

In Unterabschnitt 4.4.4 erfolgt abschließend eine kurze Diskussion, wie Dienstbeschreibungsnetze praktisch umgesetzt werden können. Hierbei steht die Möglichkeit im Zentrum, eine höhere Parallelisierung zu ermöglichen, als dies durch eine vollständige Reservierung des Systemzustandes möglich ist. Eine weitergehende Diskussion der Sequentialisierung von Zugriffen, wie sie im Bereich der Datenbanken interessant ist, erfolgt an dieser Stelle nicht. Hier sei etwa auf Jessen und Valk (1987) (Abschnitt 2.3) verwiesen.

Verwandte Netzformalismen

Es existieren zahlreiche Ansätze, die neben anonymen Marken auch strukturierte Marken zulassen. Diese Marken repräsentieren zum Teil Daten und zum Teil Funktionalität, wie etwa bei Netzen in Netzen. Die vor dem Hintergrund der Fragestellung dieser Arbeit verwandten Ansätze sollen hier kurz vorgestellt werden.

Gefärbte und algebraische Netze

Gefärbte Petrinetze (Coloured Petri Nets) sind von Jensen (1983) vorgestellt worden und erlauben die Verwendung von gefärbten Marken, wobei eine Farbe einen beliebigen Datentyp darstellen kann. So ist es etwa möglich, Zahlen, Zeichenketten oder jede andere Menge von Symbolen innerhalb der Netze als Marken zu verwenden. Zudem erlauben gefärbte Petrinetze die bedingte Ausführung von Transitionen durch Aktivierungsbedingungen. Die Kanten von gefärbten Petrinetzen werden mit Multimengenausdrücken über Variablen beschriftet, wobei die Variablen durch die Menge der Farben getypt sind. Die Stellen sind ebenfalls durch Farbmengen typisiert. Eine detaillierte Einführung bietet Jensen und Kristensen (2009).

Eng verwandt mit gefärbten Petrinetzen sind die bereits in Abschnitt 3.3.5 erwähnten algebraischen Netze von Reisig (1991b). Statt einer beliebigen Farbmenge werden bei algebraischen Netzen die Sorten einer Signatur verwendet um Stellen

und Kantenanschriften zu typisieren. Kantenanschriften stellen Terme über dieser Signatur dar und Aktivierungsbedingungen werden als Terme dargestellt, die die Sorte der booleschen Symbole *bool* besitzen. Eine formale Definition von algebraischen Netzen bietet Definition 3.45 auf Seite 72.

Petrinetze zur Modellierung objektorientierter Systeme

Es existieren verschiedene Formalismen, die algebraische Ansätze und Petrinetze zur Beschreibung objektorientierter Systeme vereinen. CO-OPN/2 (Biberstein u. a., 2001) verwendet algebraische Spezifikationen in Form von abstrakten Datentypen, um das interne Verhalten eines Objektes zu spezifizieren. Es gibt verschiedene Schnittstellen eines Objektes. Bei Ankunft einer Nachricht an einer dieser Schnittstellen schaltet ein Petrinetz, was zu einer Ausgabe führt.

Auch CLOWN (Battiston u. a., 2001) nutzt algebraische Spezifikationen um Objekte und ihr Verhalten zu spezifizieren. Hierbei dienen wiederum abstrakte Datentypen als Basis, deren Verhalten dann mit Hilfe von Petrinetzen spezifiziert wird. Als Spezifikationssprache dient hierbei das auf ordnungssortierten Algebren basierende OBJ3. LOOPN++ (Lakos, 1996) ist ebenfalls eine Spezifikationssprache für objektorientierte Petrinetze. Sie ähnelt in weiten Teilen CO-OPN/2 und CLOWN.

Einen weiteren Ansatz zur Darstellung von Objekten anhand von Netzen bietet Moldt (1996). Dieser Ansatz verwendet gefärbte Petrinetze zur Repräsentation der Datentypen. Moldt (1996) geht dabei sehr detailliert auf die Möglichkeiten der objektorientierten Programmierung ein und modelliert diese durch gefärbte Petrinetze. Ähnlich dem hier verwendeten Ansatz werden die Attribute der Objekte ebenfalls durch Paare bestehend aus einem Identifikationsschlüssel des Objektes und einem Attributwert auf Stellen repräsentiert. Im Gegensatz zu Moldt (1996) liegt der Fokus hier jedoch weniger auf Klassen und Vererbung, sondern stärker auf der Analyse des Verhaltens. Dies wird durch die Verwendung algebraischer Netze begünstigt, da sich neben den Farben auch Terme im Netz zur Analyse nutzen lassen.

Prädikats-/Transitionsnetze

Prädikat/Transitions-Netze (vgl. Genrich und Lautenbach, 1981; Genrich, 1986) stellen einen weiteren Formalismus dar. Die Marken dieser Netze basieren auf einer logischen Struktur. Jeder Stelle ist ein Prädikat zugewiesen und jeder Transition eine offene Formel über dieser logischen Struktur. Die Kanten tragen Summen von Tupeln von Variablen als Anschriften. Damit eine Transition schalten kann, muss es eine Belegung geben, die alle Variablen an den Eingangskanten und die freien Variablen der Formel einer Transition belegt, und die die Formel zu wahr auswertet. Gläs„ser (1997) stellt die Nähe von Prädikats-/Transitionsnetzen zu abstrakten Zustandsmaschinen dar. Auch die zur Modellierung von Workflows entwickelten FUNSOFT-

Netze (vgl. Emmerich und Gruhn, 1991) basieren auf Prädikat/Transitions-Netzen.

Feature-Structure-Netze und Referenznetze

Einen weiteren Formalismus zur Modellierung von Informationen und Daten in Netzen stellen Feature-Structure-Netze (FS-Netze) von Wienberg (2001) dar. Diese nutzen Feature-Structures als Marken, die unifiziert werden können. Auf diese Weise können Feature-Structures manipuliert werden, und es kann dargestellt werden, wie verschiedene Transitionen diese verändern.

Kummer (2002) führt den Formalismus der Referenznetze ein. Dieser Formalismus bietet die auf Valk (1986, 1991, 1998) zurückgehende Möglichkeit, Netze in Netzen darzustellen. Bei Referenznetzen kann eine Marke somit eine Referenz auf ein anderes Netz darstellen. Dieser Formalismus ist im Werkzeug Renew (REference NEt Workshop) implementiert (Renew, 2005). Er erlaubt es, dass die Marken eines Netzes eine Referenz auf ein anderes Netz sind. Über synchrone Kanäle kann zwischen den Netzen kommuniziert werden. Neben Referenzen auf Netze erlauben die in Renew implementierten Netze auch gefärbte Marken, so dass jedes Netz ein gefärbtes Netz darstellt, welches zudem Referenzen auf andere Netze enthalten kann. Die Verwendung des Netze-in-Netzen-Formalismus zur Darstellung und Modellierung von Workflows ist in van der Aalst u. a. (1999) dargestellt.

4.2 Zustände und Anweisungen

Wie bereits in Abschnitt 4.1 erläutert, besitzen Dienstbeschreibungsnetze einen globalen Zustand, der durch Anweisungen verändert werden kann. Die Dynamik von Dienstbeschreibungsnetzen wird also durch die Anweisungen an den Transitionen und die damit verbundenen Zustandsänderungen einerseits und durch die Marken im Netz andererseits bestimmt. Um die Schaltregel vollständig definieren zu können, muss zunächst das Verhalten der Transitionsanschriften präzisiert werden. Dies erfolgt in diesem Unterabschnitt. Hieran schließen sich die Definitionen der Dienstbeschreibungsnetze und ihres Schaltverhaltens im nächsten Unterabschnitt an.

Die Anweisungen werden dabei, wie oben umgangssprachlich in Tabelle 4.1 beschrieben, interpretiert, was hier formalisiert wird. Da jeder Zustand eine Algebra ist, stellen die Anweisungen Abbildungen von der Klasse der Zustände in die Klasse der Zustände dar, wobei jede Anweisung als eine Abbildung von einem Zustand $\mathfrak{Q}$ auf einen Folgezustand $\mathfrak{Q}'$ verstanden wird, so dass sich aus den Anweisungen und den Zuständen die Kategorie der Zustände ergibt, wobei die Anweisungen die Morphismen darstellen. Die Ausführung einer Folge von Anweisungen ist entsprechend die Komposition dieser Abbildungen und stellt ebenfalls eine Abbildung von der Klasse der Zustände in die Klasse der Zustände dar. Sie entspricht folglich der

Morphismenkomposition der Kategorie.

In diesem Abschnitt werden Syntax und Semantik der Anweisungen angegeben. Zunächst wird dazu der Begriff des Zustandes eingeführt, woran sich die Angabe der Syntax von Anweisungen anschließt, die dann um deren Semantik ergänzt wird.

4.2.1 Zustandssignatur und Zustand

Zur Darstellung der Objekte im Netz muss es möglich sein, die Objekte und deren Attribute zu referenzieren. Hierzu werden die Symbole einer ordnungssortierten Signatur (vgl. Def. 3.17) genutzt, die jedem Attribut einen Operator zuordnet und jedem Konzept eine Sorte. Die Ordnung auf den Sorten spiegelt dann die Subtypbeziehung wider. Die wesentlichen Begriffe der Signatur und der Algebra werden hier unter Rückgriff auf die Definitionen aus Abschnitt 3.2 auf die Repräsentation von Zuständen übertragen.

Da lediglich ein Teil der Operatoren veränderbar sein soll, muss die Gesamtsignatur in zwei Signaturen aufgeteilt werden, so dass nicht etwa Operatoren wie die Multiplikation von Zahlen oder das Zusammenfügen von Zeichenketten veränderbar sind, die stets gleich interpretiert werden sollen. Diese Unterscheidung wird durch Einführung der statischen Signatur mit den unveränderlichen Operatoren und der dynamischen Signatur mit den veränderlichen Operatoren, die beispielsweise Attribute repräsentieren, erreicht. Die veränderbaren Operatoren werden durch die Systemsignatur repräsentiert. Sie beschreibt die veränderlichen Konzepte und Operatoren. Eine Zustandssignatur erweitert diese Signatur um nicht veränderliche Operatoren, wie etwa die Operatoren auf Zahlen oder Zeichenketten. Dieses Vorgehen ist vergleichbar mit dem Vorgehen bei Hidden Algebras (vgl. Goguen, 1999; Goguen und Malcolm, 2000; Goguen und Rosu, 2004), wo es ebenfalls veränderliche Signaturen und statische Signaturen gibt. Attribute von Objekten werden hierbei als Operatoren verstanden, nicht als Prädikate, wie dies in Darstellungsweisen der Fall ist, die stärker durch Logiken geprägt sind.

Eine Systemsignatur repräsentiert den Systemzustand und besitzt für jedes Attribut einen Operator, der dieses Attribut repräsentiert. Über zwei spezielle Sorten ($\top^{\Omega}$ und $\bot^{\Omega}$) ist sichergestellt, dass eine Systemsignatur kohärent ist. Zudem besitzt eine Systemsignatur die spezielle Sorte $\top_{\overline{use}}$, die dem Typ der nicht verwendeten Objekte entspricht.

Definition 4.1 (Systemsignatur)
Eine *Systemsignatur* $\Sigma^{sys} = (S, \preceq, \Omega^{sys})$ ist eine reguläre OS-Signatur derart, dass

- zwei ausgezeichnete Sorten $\top^{\Omega}$ und $\bot^{\Omega}$ existieren und für alle $s \in S$ gilt $\bot^{\Omega} \preceq s \preceq \top^{\Omega}$ und

- eine spezielle Sorte $\top_{\overline{use}}$ existiert, die die nicht genutzten Elemente der Algebra repräsentiert.

⧫

Es sei darauf hingewiesen, dass Objekte der Systemsignatur, die ihren Typ ändern sollen, keine Konstanten darstellen können, da diese stets vom gleichen Typ sind. Aus diesem Grund kann es sinnvoll sein, zu fordern, dass die Systemsignatur keinerlei Konstanten besitzt. Diese Einschränkung soll an dieser Stelle jedoch nicht erfolgen, da der Einsatz von Konstanten durchaus sinnvoll sein kann, um gewisse realweltliche Dinge zu fixieren, wie beispielsweise gewisse Ressourcen.

In einem Dienstbeschreibungsnetz stellen sämtliche Objekte der Marken später Elemente der Algebra und nicht Konstanten der Signatur dar. Dies ist ein Unterschied gegenüber algebraischen Netzen. Der Vorteil bei diesem Vorgehen liegt darin, dass die Elemente der Algebra auf diese Weise ihre Sorte wechseln können. Würde man Konstanten wählen, die ihre Interpretation ändern, so müsste man bei unterschiedlicher Typisierung eine Konstante im Netz durch eine Konstante eines anderen Typs austauschen. Dies müsste insbesondere auch dann erfolgen, wenn eine Konstante auf zwei Stellen vorhanden ist und sich der Typ dieser Konstante ändert. Dies würde dann dazu führen, dass Marken auf einer Stelle durch eine Transition ausgetauscht würden, die weder im Vor- noch im Nachbereich der Transition wäre, was sehr unintuitiv wäre und den Formalismus unnötig kompliziert werden ließe. Statt der initialen Quotiententermalgebra zur Interpretation einer Signatur wird also eine nicht-initiale Algebra verwendet (diese kann ebenfalls nur Terme und Symbole enthalten, wie etwa Einträge einer Datenbank), der dann durch einen Isomorphismus realweltliche Objekte zugeordnet werden.

Es ist sinnvoll anzunehmen, dass jede Systemsignatur die Sorte der booleschen Werte *bool*, die Sorte der rationalen Zahlen *rat* sowie die Sorte der Zeichenketten *string* enthält, so dass es Attribute geben kann, die auf Werte dieser Sorten abbilden. Hierbei enthält die Systemsignatur nur die Sorten, nicht aber Operatoren auf diesen primitiven Sorten oder Konstanten der Sorten. Diese Sorten dienen nur der Typisierung möglicher Terme. Da sich die Interpretation der Operatoren primitiver Sorten nicht ändern soll, werden diese später durch eine statische Signatur ergänzt, die stets gleich interpretiert wird.

Aus Lemma 3.26 und den zugehörigen Erläuterungen folgt durch die Sorten $\top^{\Omega}$ und $\bot^{\Omega}$ sofort, dass jede Systemsignatur Σ^{sys} eine kohärente OS-Signatur ist.

Übertragen auf das Beispiel in Abschnitt 4.1 besitzt die Systemsignatur neben den Sorten *bool*, *rat* und *string* und $\top_{\overline{use}}$ die Sorten *PurchaseOrder*, *Customer*, *Address*, *Item*, *CreditcardPayment* und *Payment* sowie eine Sorte *ItemList*, die eine Liste von Objekten der Klasse *Item* darstellt. Des Weiteren sind die Operatoren durch die Attribute der Fachklassen gegeben. Dies sind, um ein paar Operatoren zu nennen, beispielsweise *customer*, *name*, *address*, *street* und *city*. Da es sich bei den Operatoren um Attribute handelt, besitzt jeder dieser Operatoren eine Quell- und

eine Zielsorte. Dies wird wie folgt notiert, wobei die ausgelassenen Operatoren durch Punkte dargestellt sind und die Sorten $\top^{\Omega}$ und $\bot^{\Omega}$, die jeder Signatur angehören, weggelassen wurden.

```
sorts PurchaseOrder Customer Address Item .
sorts CreditcardPayment Payment .
subsorts CreditcardPayment < Payment .

op customer: PurchaseOrder -> Customer .
op name: Customer -> string .
op address: Customer -> Address .
op street: Address -> string .
op city: Address -> string .
...
```

Ein Systemzustand interpretiert diese Signatur, indem er eine entsprechende Algebra zu ihr darstellt.

Definition 4.2 (Systemzustand)
Ein *Systemzustand* $\mathfrak{Q}_{\Sigma^{sys}} = (S_Q, \Omega_Q)$ zu einer gegebenen Systemsignatur Σ^{sys} ist eine ordnungssortierte Algebra (vgl. Def. 3.20), die Σ^{sys} interpretiert. ♦

Betrachtet man wiederum das Beispiel in Abschnitt 4.1 so stellt ein Zustand beispielsweise die in Abbildung 4.3 dargestellten tatsächlichen Werte eines Objektes dar. Es gibt in diesem Fall ein Objekt vom Typ *PurchaseOrder*, das ein Attribut *customer* besitzt, welches wiederum auf ein Objekt vom Typ *Customer* verweist. Letzteres Objekt besitzt ein Attribut *name*, das den konkreten Wert `"Kunde A"` besitzt, und ein Attribut *address*, das wiederum auf ein Objekt vom Typ *Address* verweist.

In der Systemsignatur sind diese Attribute die Operatoren. So wird das *PurchaseOrder*-Objekt durch ein Element der Trägermenge S_Q der Algebra repräsentiert, das der Sorte *PurchaseOrder* angehört. Dieses Element ist beliebig und soll als q bezeichnet werden, so dass $q \in Q_{PurchaseOrder}$ gilt. Es sei darauf hingewiesen, dass q Teil der Algebra ist und keine Konstante der Signatur ist. Da q von der Sorte *PurchaseOrder* ist, sind die entsprechenden Operatoren auf q definiert. In diesem Fall verweist also der Operator *payed* auf ein $q_b \in Q_{bool}$, das einen booleschen Wert repräsentiert, und der Operator *customer* auf ein $q_c \in Q_{Customer}$. Anders formuliert ist $customer(q) = q_c$ und $payed(q) = q_b$. Während auf q_b keine veränderlichen Operatoren definiert sind, sind auf q_c die Operatoren der Sorte *Customer* definiert. Einer dieser Operatoren ist *name*, der auf die Sorte *string* abbildet. Für diesen Operator ist im aktuellen Beispiel dann $name(q_c) =$`"Kunde A"`. Dies gilt entsprechend für die anderen Operatoren.

Im Beispiel zeigte sich bereits, dass es ungünstig ist, dass die Elemente der Sorte *bool* nicht statisch sind. Es gibt in diesem Fall keine Beschränkung der booleschen Werte, so dass Q_{bool} beliebig viele Elemente besitzen könnte. Gleichzeitig sind die elementaren Operatoren auf dieser Sorte nicht definiert, so dass es keine logischen Verknüpfungen geben kann. Diese Problematik wird durch die Verwendung der statischen Sorten gelöst, deren Interpretation stets gleich ist. Die statische Signatur zusammen mit der Systemsignatur stellen dann die Zustandssignatur dar.

Definition 4.3 (Zustandssignatur, Statische Signatur, Statische Algebra) Eine *Zustandssignatur* $\Sigma^{\mathbf{Q}} = (S, \preceq, \Omega)$ ist eine reguläre OS-Signatur bestehend aus

- einer Systemsignatur $\Sigma^{sys} = (S, \preceq, \Omega^{sys})$ als Untersignatur von $\Sigma^{\mathbf{Q}} = (S, \preceq, \Omega)$ und
- einer *statischen Signatur* $\Sigma^{stat} = (S^{stat}, \preceq^{stat}, \Omega^{stat})$ als Untersignatur von $\Sigma^{\mathbf{Q}} = (S, \preceq, \Omega)$, die die spezielle Sorte $s_{\mathbf{token}} \in S^{stat}$ mit der Konstanten $\bullet \in \Omega^{stat}_{\epsilon, s_{\mathbf{token}}}$ als dem einzigen Operatorsymbol dieser Sorte besitzt.

Es ist dabei $S^{stat} \subseteq S$, $s_1 \preceq^{stat} s_2$ gdw. $s_1 \preceq s_2$ für $s_1, s_2 \in S^{stat}$ und $\Omega^{stat} \cap \Omega^{sys} = \emptyset$. Es sei die *statische Algebra* $\mathcal{A}^{stat} = (S_A, \Omega_A)$ eine fest definierte Algebra der statischen Signatur Σ^{stat}, die deren Interpretation festlegt. ♦

Die statische Signatur und die Systemsignatur sind jeweils Untersignaturen der Zustandssignatur, wobei die Ordnung auf die Untersignatur entsprechend übertragen wurde (vgl. Def. 3.2). In einer statischen Signatur befinden sich beispielsweise die statischen Sorten *bool* und *nat*, die die üblichen Konstanten $true, false \in \Omega^{stat}_{\epsilon,bool}$ und $1, 2, \ldots \in \Omega^{stat}_{\epsilon,nat}$ besitzen. In der statischen Algebra $\mathcal{A}^{stat} = (S_A, \Omega_A)$ wäre dann $S_A = (A_s)_{s \in S^{stat}}$ mit $A_{bool} = \mathbb{B}$ und $A_{nat} = \mathbb{N}$. Für die Operatoren $\wedge, \vee \in \Omega^{stat}_{boolbool,bool}$ und $\neg \in \Omega^{stat}_{bool,bool}$ würde entsprechend in $\mathcal{A}^{stat}$ die übliche Interpretation boolescher Operatoren gelten, also beispielsweise $\wedge_A(true_A, true_A) = true_A$ und $\neg_A(true_A) = false_A$. Entsprechendes gilt für die Operatoren aus $\Omega^{stat}_{natnat,nat}$ wie $+$ (Addition) und $\cdot$ (Multiplikation). Im weiteren Verlauf wird für die Interpretation der Symbole der statischen Signatur das Symbol der Signatur genutzt, so dass beispielsweise statt $true_A \in \mathbb{B}$ dann *true* und statt $1_A \in \mathbb{N}$ entsprechend 1 notiert wird. Für die Definition der Dienstbeschreibungsnetze wird die Existenz der Sorte *bool* später vorausgesetzt, so dass diese stets Teil der statischen Signatur ist. Die übrigen Sorten sind dann vom Modellierungskontext abhängig.

Die Konstante $\bullet$ beschreibt die *anonyme Marke* im Netz. Die Sorte $s_{\mathbf{token}} \in S$ ist entsprechend die Sorte dieser Konstanten. Die Konstante wird der Einfachheit halber in der statischen Algebra durch sich selbst interpretiert, so dass jeweils dasselbe Symbol genutzt wird. Die statische Signatur und die Systemsignatur einer Zustandssignatur besitzen die gleiche Menge von Sorten, da so jede Sorte auch die

Zielsorte eines Attributs sein kann. Beide Signaturen, die statische Signatur und die Systemsignatur, sind Untersignaturen (vgl. Def. 3.2) der Zustandssignatur.

Die Systemsignatur legt alle veränderlichen Operatoren fest. Die Interpretation der in dieser Signatur angegebenen Operatorsymbole kann sich während des Schaltens eines Dienstbeschreibungsnetzes ändern.

Für beide Signaturen wird die Existenz von *Cast-Operatoren* angenommen, die ein Element einer Sorte auf eine andere Sorte abbilden. Hierbei umfasst die statische Signatur alle Cast-Operatoren, die auf eine Sorte aus S^{stat} abbilden, während die Systemsignatur alle Cast-Operatoren umfasst, die auf eine Sorte aus $S \setminus S^{stat}$ abbilden. Diese Cast-Operatoren sind nur definiert, wenn das Objekt einer Sorte angehört und liefern andernfalls ein Element $\perp\!\!\!\perp \in \perp^{\Omega}$ zurück. Die Konstante $\perp\!\!\!\perp$ stellt dabei einen undefinierten Wert dar. Das Objekt bleibt dabei gleich, es ändert sich lediglich die Sorte eines Terms. Für zwei Sorten $s, s' \in S$ ist ein Cast-Operator $(s)\cdot \in \Omega_{s',s}$ definiert gdw. $s \preceq s'$ ist. Dieser ist somit gegeben durch

$$(s)(q) = \begin{cases} q, & \text{wenn } q \in Q_s \\ \perp\!\!\!\perp, & \text{sonst} \end{cases}.$$

Ein Zustand ist eine Algebra, die die Zustandssignatur interpretiert. Dabei bleibt die Interpretation der statischen Signatur stets gleich. Um dies zu definieren wird vom Redukt einer Algebra Gebrauch gemacht, welches in Definition 3.6 definiert wurde. Das Σ^{stat}-Redukt eines Zustandes ist dabei stets gleich der Algebra $\mathcal{A}^{stat}$. Sämtliche Zustände zu einer Algebra stellen die Klasse der Zustände dar.

Definition 4.4 (Zustand, Klasse der Zustände)
Ein *Zustand* $\mathfrak{Q}_{\Sigma^{\mathbf{Q}}} = (S_Q, \Omega_Q)$ zu einer gegebenen Zustandssignatur $\Sigma^{\mathbf{Q}} = (S, \preceq, \Omega)$ ist eine ordnungssortierte Algebra, die $\Sigma^{\mathbf{Q}}$ interpretiert, so dass $\mathfrak{Q}_{\Sigma^{\mathbf{Q}}}{\restriction_{\Sigma^{stat}}} = \mathcal{A}^{stat}$ ist.
Die *Klasse der Zustände* zu einer Zustandssignatur $\Sigma^{\mathbf{Q}}$ wird im Folgenden als $\mathbf{States}(\Sigma^{\mathbf{Q}})$ notiert. ⧫

Wenn die Zustandssignatur aus dem Kontext hervorgeht, wird statt $\mathfrak{Q}_{\Sigma^{\mathbf{Q}}}$ auch $\mathfrak{Q}$ notiert. Die Klasse $\mathbf{States}(\Sigma^{\mathbf{Q}})$ stellt die möglichen Algebren dar, die die Signatur $\Sigma^{\mathbf{Q}}$ interpretieren. Übergänge zwischen den Zuständen sind Abbildungen von der Klasse der Zustände in die Klasse der Zustände. Die Sorten *bool*, *nat* und *string* und deren Operatoren werden durch die statische Algebra auf die übliche Art interpretiert. Anhand der statischen Algebra lässt sich somit sicherstellen, dass es nur zwei boolesche Werte gibt, die sich als wahr und falsch interpretieren lassen. Zudem sind die üblichen Operatoren auf den Sorten *bool*, *rat* und *string* Teil von Σ^{stat}.

Übertragen auf das Beispiel aus Abschnitt 4.1 stellt die Zustandssignatur nun auch die Möglichkeit bereit, Zeichenketten im Netz zu nutzen, wie dies etwa mit

`"Kunde A"` geschieht. Die Zeichenkette `"Kunde A"` ist dabei eine Konstante der statischen Signatur, die durch die Algebra entsprechend interpretiert wird. Diese Interpretation ist stets gleich. Ebenso lassen sich nun auch Operatoren auf eine solche Konstante anwenden wie etwa die Konkatenation von Zeichenketten der Art `"Kunde A"+": beliefert"`, was dann zu `"Kunde A: beliefert"` auswertet, sofern die statische Algebra wie gefordert die naheliegende Interpretation der Operatoren liefert.

4.2.2 Syntax der Anweisungen

Der Zustand gibt die aktuelle Interpretation der Operatoren wieder. Hierbei ist noch nicht betrachtet worden, wie sich diese Interpretation ändert. Im Netz in Abbildung 4.5 wurden hierzu Update-Anweisungen und Zuweisungsanweisungen genutzt. Deren Syntax soll hier definiert werden, bevor auf die Definition von Dienstbeschreibungsnetzen eingegangen wird. Der prinzipielle Aufbau der Anweisungen ist bereits in Tabelle 4.1 angegeben. Hier werden die verschiedenen Arten von Anweisungen detaillierter erläutert.

Update-Anweisungen ändern die Interpretation eines Attributes, das durch ein Operatorsymbol der Systemsignatur repräsentiert wird. Veränderbar sind somit lediglich Operatorsymbole aus der Systemsignatur Σ^{sys}. Eine Update-Anweisung hat dabei die in folgender Definition gegebene Form.

Definition 4.5 (Update-Anweisungen)
Sei $\Sigma^{\mathbf{Q}}$ eine Zustandssignatur und X eine S-indizierte Menge von Variablen. Eine *Update-Anweisung* ist von der folgenden Art:

> *update* $\omega(\tau_1, \ldots, \tau_n) = \tau$ für $\tau_j \in \mathbb{T}_{\Sigma^{\mathbf{Q}},s_j}(X)$ mit $1 \leq j \leq n$ und $\tau \in \mathbb{T}_{\Sigma^{\mathbf{Q}},s}(X)$ für $\omega \in \Omega^{sys}_{s_1 \ldots s_n,s}$.

♦

Ist $n = 1$ und gibt es somit nur eine Sorte in der Domäne des Operators, so wird statt $\omega(\tau_1)$ kürzer $\tau_1.\omega$ notiert. Dies ist enger verwandt mit der Notation in objektorientierten Programmiersprachen und ist daher vielfach leichter verständlich.

Die Update-Anweisung wird entsprechend ihrer Verwendung im Beispiel in Abschnitt 4.1 interpretiert. Der Operator ω wird nach Ausführung der Anweisung in einem Zustand neu interpretiert und hat dann den Wert der Auswertung von τ. Dies wird im Netz in Abbildung 4.5 zur Änderung der Attribute genutzt. Beispielsweise erhält das Attribut *payed* durch die Anweisung *update ipo.payed = true* den Wert der Konstante *true*, der dem Wert wahr der Algebra entspricht. Dies ist dann in der Darstellung des Objektzustands in Abbildung 4.6 zu sehen.

Konzeptänderungsanweisungen ändern die Konzeptzugehörigkeit eines Elementes der Trägermenge S_Q. Hierzu ist es hilfreich, die Sorten einer Zustandssignatur in die Menge der statischen Sorten S^{stat} und in die Menge der dynamischen Sorten S^{dyn} zu partitionieren, so dass $S^{stat} \cup S^{dyn} = S$ ist. Die statischen Sorten stellen dabei die unveränderlichen Typen der Systemsignatur dar, während die dynamischen Sorten die veränderlichen Typen repräsentieren. Dies bezieht sich in diesem Fall nicht auf die Operatoren, da alle Operatoren der Systemsignatur veränderlich sind, sondern bezieht sich vielmehr auf die Sortenzugehörigkeit eines Elements der Algebra. Im Netz in Abbildung 4.5 ist *Invoiced* solch eine dynamische Sorte.

Es können einem Objekt dynamische Konzepte hinzugefügt werden oder Objekte mit statischen Konzepten neu erzeugt werden. Da sich eine statische Konzeptzugehörigkeit nicht ändern kann und da dynamische Konzepte immer ein statisches Konzept benötigen, auf das sie sich beziehen, sind dies die einzigen Möglichkeiten, Konzeptzugehörigkeiten zu ändern. Dass sich eine dynamische Konzeptzugehörigkeit immer auf eine statische Konzeptzugehörigkeit bezieht ist notwendig, damit jedes Objekt stets ein Konzept besitzt. Zudem entspricht dies der Modellierungsnotwendigkeit der meisten Formalismen zur Darstellung von Ontologien (vgl. Abschnitt 2.4). Die *new*-Anweisung ist hierbei ebenfalls eine Änderung der Konzeptzugehörigkeit, nur handelt es sich hierbei um statische Konzepte, deren Zugehörigkeit geändert wird. Dies erlaubt es überhaupt erst, Objekte einer statischen Sorte zu erzeugen.

Definition 4.6 (Konzeptänderungsanweisungen)
Sei $\Sigma^{\mathfrak{Q}}$ eine Zustandssignatur und X eine S-indizierte Menge von Variablen, dann ist eine *Konzeptänderungsanweisung* von der folgenden Art:

1. *new* $(v, s_{\mathfrak{c}})$, wobei $s_{\mathfrak{c}} \in S^{stat}$ und $v \in X_{\top^{\mathfrak{Q}}}$,
2. *addRole*$(\tau, s_{\mathfrak{c}})$, wobei $\tau \in \mathbb{T}_{\Sigma^{\mathfrak{Q}},s}(X)$, $s \in S$ und $s_{\mathfrak{c}} \in S^{dyn}$ sowie $s_{\mathfrak{c}} \preceq s$ oder
3. *removeRole*$(\tau, s_{\mathfrak{c}})$, wobei $\tau \in \mathbb{T}_{\Sigma^{\mathfrak{Q}},s}(X)$, $s \in S$ und $s_{\mathfrak{c}} \in S^{dyn}$ sowie $s \preceq s_{\mathfrak{c}}$.

♦

Die Konzeptänderungsanweisungen enthalten keine Anweisung, die ein Objekt wieder dem Pool der ungenutzten Objekte zuführen, so dass dieses wiederverwendet werden kann. Eine solche Anweisung könnte etwa die Form *release*(v) besitzen. Da das Speichermanagement jedoch nicht Gegenstand der Geschäftsprozessmodellierung ist, sondern vielmehr implizit durch die technische Ebene bereitgestellt werden sollte, wird von einer solchen Anweisung abgesehen.

Bei den *new*-Anweisungen wird die Variable v im Allgemeinen weggelassen, da sie eine beliebige unbenutzte Variable darstellt, an die ein Wert gebunden wird. Da diese Variable jedoch vom Typ $\top^{\mathfrak{Q}}$ ist, sind keine Operatoren auf ihr definiert, so dass anschließend ein Cast auf eine andere Sorte stattfinden muss und damit

$v_1 = (s_\mathfrak{c})v$ und $v_1 \in X_{s_\mathfrak{c}}$ ist. Dies lässt sich abkürzen durch die Notation

$$v = new\ s_\mathfrak{c}$$

Die Konzeptänderungsanweisungen haben die zu erwartende Interpretation. Eine genaue Definition des Verhaltens der Konzeptänderungsanweisungen erfolgt in Abschnitt 4.2.3. Die *new*-Anweisung erzeugt ein neues Objekt vom Typ $s_\mathfrak{c}$, indem in der Algebra ein Element der Sorte $\top_{\overline{use}}$ ausgesucht wird und der Sorte $s_\mathfrak{c}$ hinzugefügt wird. Hierdurch ändert sich die Gesamtmenge der Objekte nicht. Vielmehr wird ein Element von $Q_{\top_{\overline{use}}}$ nach $Q_{s_\mathfrak{c}}$ verlagert. Die *addRole*-Anweisung geht ähnlich vor, nur dass diesmal das Element der Algebra einer weiteren Menge $Q_{s_\mathfrak{c}}$ hinzugefügt wird. Bei der *removeRole*-Anweisung wird hingegen das Element der Algebra aus der Menge $Q_{s_\mathfrak{c}}$ entfernt. Ein entsprechendes Beispiel der Verwendung findet sich im Netz in Abbildung 4.5.

Zuweisungen bieten die Möglichkeit, Variablen anzulegen und diese mit einem Wert innerhalb einer Menge von Anweisungen zu belegen. Im Netz können diese Variablen auch als Kantenanschrift dienen und somit weitergereicht werden. Zur Belegung dieser Variablen dienen Zuweisungsanweisungen, die eine Gleichung darstellen, wobei auf der linken Seite eine Variable und auf der rechten Seite ein Term steht.

Definition 4.7 (Zuweisungsanweisungen, gebundene Variable)
Sei $\Sigma^\mathbf{Q}$ eine Zustandssignatur und sei X eine S-indizierte Menge von Variablen. Eine *Zuweisungsanweisung* ist von der folgenden Art:

$$assign\ v = \tau \text{ für } v \in X_s \text{ und } \tau \in \mathbb{T}_{\Sigma^\mathbf{Q},s}(X \setminus \{v\}) \text{ mit } s \in S.$$

Die Variable v wird als durch die Zuweisungsanweisung *gebundene Variable* bezeichnet. Dies wird als $Bound(assign\ v = \tau) = \{v\}$ notiert. ⧫

Die Interpretation der Zuweisungsanweisung ist eine Zuweisung der Auswertung des Terms τ an die Variable v, so dass τ und v gleich auswerten. In Abbildung 4.5 wird die Zuweisung beispielsweise genutzt, um das neu erzeugte Objekt vom Typ *Invoice* an die Variable *in* zu binden.

Ein Anweisungsblock ist eine Folge von Anweisungen. Jede dieser Anweisungen ändert den Zustand oder interpretiert Variablen.

Definition 4.8 (Anweisungsblock, Variablen, Wohlgeformtheit)
Gegeben eine Zustandssignatur $\Sigma^\mathbf{Q}$ und eine S-indizierte Menge von Variablen X über dieser Signatur. Eine *Anweisung cmd* über $\Sigma^\mathbf{Q}$ und X ist entweder eine Konzeptänderungsanweisung, eine Update-Anweisung oder eine Zuweisungsanweisung

über $\Sigma^{\mathbf{Q}}$ und X. Es ist $\textsc{Var}(cmd)$ die Menge der in cmd genutzten Variablen mit $\textsc{Var}(cmd) \stackrel{\text{def}}{=} \{v \in \textsc{Var}(\tau) \mid \tau \text{ ist Term in } cmd\}$.
Ein *Anweisungsblock* $\mathbf{Cmd} = cmd_1 \ldots cmd_n$ ist eine Folge von Anweisungen über $\Sigma^{\mathbf{Q}}$ und X. **Cmd** ist *wohlgeformt* gdw. für zwei Zuweisungsanweisungen cmd_k und cmd_l $(1 \leq k, l \leq n)$ gilt

- $Bound(cmd_k) = Bound(cmd_l)$ impliziert $k = l$ (Jede Variable wird nur einmal gebunden (per Definition ist $|Bound(cmd)| = 1$)),
- $v \in Bound(cmd_k)$ und $v \in \textsc{Var}(cmd_l)$ impliziert $k < l$ (Jede genutzte Variable muss vorher gebunden sein) und

wenn gilt $cmd_k = (removeRole(\tau, s))$ und $x \in \textsc{Var}(cmd_l)$ mit $k < l$ impliziert $x \notin X_{s'}$ für $s' \preceq s$ (Variablen entfernter Sorten werden nach dem Entfernen nicht mehr genutzt). ⧫

Die Wohlgeformtheit eines Anweisungsblocks gibt vor, dass jede Variable eindeutig gebunden ist. Die Variablenmenge $\textsc{Var}(cmd)$ wird auf **Cmd** ausgedehnt, so dass $\textsc{Var}(\mathbf{Cmd})$ die Vereinigung aller $\textsc{Var}(cmd)$ für $cmd \in \mathbf{Cmd}$ bezeichnet. Weiterhin bezeichnet $Bound(\mathbf{Cmd})$ die Menge der *Variablen*, die in einer Zuweisungsanweisung von **Cmd** *gebunden* sind. Es bezeichnet $Free(\mathbf{Cmd}) \stackrel{\text{def}}{=} \textsc{Var}(\mathbf{Cmd}) \setminus Bound(\mathbf{Cmd})$ die Menge der *freien Variablen* von **Cmd**.
Die *Menge aller Anweisungsblöcke* über einer Zustandssignatur $\Sigma^{\mathbf{Q}}$ und einer Variablenmenge X wird als $\mathbf{Cmds}_{\Sigma^{\mathbf{Q}}}(X)$ notiert.

4.2.3 Interpretation der Anweisungen

Jede Update-Anweisung und jede Konzeptänderungsanweisung stellt eine Abbildung von einem Zustand auf einen Folgezustand dar. Dies wird für eine Anweisung cmd als $[\![cmd(\alpha)]\!]^{\mathfrak{Q}} = \mathfrak{Q}'$ notiert, was der Notation für die Auswertung von Termen entspricht. Die genaue Interpretation von $[\![\cdot]\!]$ ist Gegenstand dieses Unterabschnitts. Die Variablenbelegung α ist dabei in einem Dienstbeschreibungsnetz abhängig von den Eingangskanten und von den Zuweisungsanweisungen. Zuweisungsanweisungen erweitern die Variablenbelegung der Eingangsvariablen. In wohlgeformten Anweisungsblöcken ist eine solche erweiterte Variablenbelegung stets eindeutig definiert, da jede Variable eindeutig interpretiert wird.

Update-Anweisungen

Zur Ausführung von Updates werden die zu ändernden Operatorinterpretationen der Algebra als Lokationen bezeichnet, deren Wert sich ändert. Eine Lokation stellt eine Kombination von Elementen der Trägermenge zusammen mit einem Operatorsymbol dar. Da hier lediglich der Systemzustand betrachtet und verändert wird,

werden nur Operatoren der Systemsignatur verwendet. Für ein Tupel von Elementen der Trägermenge wird verkürzt $\overline{q} \in Q_w$ für $w = s_1 \dots s_n$ notiert, anstatt $(q_1, \dots, q_n) \in Q_{s_1} \times \dots \times Q_{s_n}$ zu schreiben, wie dies in Abschnitt 3.2.2 eingeführt wurde.

Definition 4.9 (Lokation, Inhalt einer Lokation, Update)
Sei $\mathfrak{Q} = (S_Q, \Omega_Q)$ ein Zustand einer Zustandssignatur $\Sigma^{\mathbf{Q}}$. Eine *Lokation* von $\mathfrak{Q}$ ist ein Paar $l^{\Omega} \stackrel{\text{def}}{=} (\omega, \overline{q})$ mit $\omega \in \Omega^{sys}_{w,s}$ und $\overline{q} \in Q_w$.
Der *Inhalt einer Lokation* l^{Ω} im Zustand $\mathfrak{Q}$ wird als $\mathfrak{Q}(l^{\Omega})$ notiert und ist definiert als $\mathfrak{Q}(l^{\Omega}) \stackrel{\text{def}}{=} \omega_Q(\overline{q})$ für $l^{\Omega} = (\omega, \overline{q})$.
Ein *Update* ist ein Paar (l^{Ω}, q_v), wobei $l^{\Omega} = (\omega, \overline{q})$ eine Lokation von $\mathfrak{Q}$ ist und $q_v \in Q_s$ für $\omega \in \Omega^{sys}_{w,s}$. ♦

Ein Update ändert den Wert einer Lokation. Greift man das im Zusammenhang mit dem Systemzustand gegebene Beispiel von Seite 100 noch einmal auf, so ergibt sich dort eine Lokation beispielsweise als $(\mathit{customer}, q)$, wobei wiederum $q \in Q_{\mathit{PurchaseOrder}}$ gilt. Der Inhalt der Lokation wäre in diesem Beispiel $q_c \in Q_{\mathit{Customer}}$. Eine andere Lokation wäre (payed, q), deren Inhalt *false* ist, wobei hier $\mathit{false} \in Q_{\mathit{bool}}$ als Element der Algebra angesehen wird. Ein Update wäre $((\mathit{payed}, q), \mathit{true})$ mit $\mathit{true} \in Q_{\mathit{bool}}$, welches dazu führt, dass die Lokation (payed, q) im Anschluss den Inhalt *true* besitzt.

Updates können feuern, was zu einer Neuinterpretation einer Lokation führt. Wie in obigem Beispiel dargestellt führt das Feuern des Updates $((\mathit{payed}, q), \mathit{true})$ dazu, dass im Folgezustand $\mathit{payed}(q) = \mathit{true}$ gilt. Die Menge aller Lokationen eines Zustandes $\mathfrak{Q}$ wird als $\mathbf{Loc}^{\mathbf{\Omega}}(\mathfrak{Q})$ notiert.

Definition 4.10 (Feuern eines Updates, Folgezustand)
Das *Feuern* eines Updates $\mathbf{u}$ in einem Zustand $\mathfrak{Q}$ führt zu einem neuen Zustand $\mathfrak{Q}' = \mathfrak{Q} \oplus_Q \mathbf{u}$ mit der gleichen Familie von Trägermengen wie $\mathfrak{Q}$, in dem für jede Lokation $l^{\Omega} \in \mathbf{Loc}^{\mathbf{\Omega}}(\mathfrak{Q})$ gilt

$$\mathfrak{Q}'(l^{\Omega}) = \begin{cases} v & \text{wenn } \mathbf{u} = (l^{\Omega}, v) \\ \mathfrak{Q}(l^{\Omega}) & \text{sonst} \end{cases}$$

Der Zustand $\mathfrak{Q}' = \mathfrak{Q} \oplus_Q \mathbf{u}$ wird als *Folgezustand* von $\mathfrak{Q}$ bezüglich $\mathbf{u}$ bezeichnet. ♦

Da sowohl $\mathfrak{Q}$ als auch $\mathfrak{Q}'$ Zustände und somit reguläre ordnungssortierte Algebren sind, werden bei einem Update die Operatoren der Subsorten ebenfalls an der entsprechenden Lokation neu interpretiert.

Wie im Netz in Abbildung 4.5 zu sehen ist, sollen die Updates je nach Markierung des Netzes unterschiedlich ausfallen, was durch Variablen an den Eingangskanten

erreicht wird. Durch Terme in den Updates, die dann in einem Zustand unter einer Variablenbelegung ausgewertet werden, ergeben sich die konkreten Updates. Die Auswertung einer Update-Anweisung in einem Zustand $\mathfrak{Q}$ unter einer Variablenbelegung α entspricht somit einem Update. Dies wird für eine Update-Anweisung $\mathbf{cmd}_U = (\mathit{update}\ \omega(\tau_1, \ldots, \tau_n) = \tau)$ als $[\![\mathbf{cmd}_U(\alpha)]\!]^{\mathfrak{Q}}$ notiert. Eine entsprechende Interpretation ist in Tabelle 4.2 dargestellt.

Anweisung $\mathbf{cmd}_U$	Update $[\![\mathbf{cmd}_U(\alpha)]\!]^{\mathfrak{Q}}$
$\mathit{update}\ \omega(\tau_1, \ldots, \tau_n) = \tau$	$\mathbf{u} = \Big(\omega, ([\![\tau_1]\!]^Q_\alpha, \ldots, [\![\tau_n]\!]^Q_\alpha)), [\![\tau]\!]^Q_\alpha\Big)$

Tabelle 4.2: Update-Anweisungen und ihre Interpretation

Die *Ausführung einer Update-Anweisung* $\mathbf{cmd}_U$ in einem Zustand $\mathfrak{Q}$ unter einer Variablenbelegung α entspricht dem Feuern des Updates $[\![\mathbf{cmd}_U(\alpha)]\!]^{\mathfrak{Q}}$. Aus diesem Grund wird $[\![\mathbf{cmd}_U(\alpha)]\!]^{\mathfrak{Q}}$ als Abbildung $[\![\mathbf{cmd}_U(\alpha)]\!] : \mathbf{States}(\Sigma^{\mathfrak{Q}}) \to \mathbf{States}(\Sigma^{\mathfrak{Q}})$ interpretiert, die den Übergang zwischen zwei Zuständen darstellt, wobei α jede Variable in $\mathbf{cmd}_U$ mit einem Wert aus der Trägermenge des Startzustands belegt.

Das Ergebnis der Ausführung des Updates ist wiederum ein Zustand. Dies soll an einem einfachen Beispiel illustriert werden.

Beispiel 4.1: Sei die Signatur dieses Beispiels durch folgende Sorten und Operatoren gegeben.

```
sorts Person .
op name: Person -> string .
op age: Person -> nat .
var v1, v2: Person .
```

Sei $\mathfrak{Q}$ ein Zustand, so dass $Q_{Person} = \{q_1, q_2\}$ ist und sei die Operatorinterpretation der Algebra gegeben durch $name_Q(q_1) =$`"bob"`, $name_Q(q_2) =$`"tom"`, $age_Q(q_1) = 21$ und $age_Q(q_2) = 23$.

Eine Update-Anweisung ist nun gegeben durch $v_1.name =$`"paul"`. Eine Variablenbelegung weist dann der Variable v_1 beispielsweise den Wert q_1 zu, so dass die Auswertung zu einer Änderung des Zustands führt. Im Folgezustand $\mathfrak{Q}'$ wäre dann $name(q_1) =$`"paul"` und $age(q_1) = 21$. Anders formuliert wäre für $\alpha : v_1 \mapsto q_1$ und $\mathbf{cmd}_U = (v_1.name =$`"paul"`$)$ dann $[\![\mathbf{cmd}_U(\alpha)]\!] = \mathfrak{Q}'$ mit $name_{Q'}(q_1) =$`"paul"`, $name_{Q'}(q_2) =$`"tom"`, $age_{Q'}(q_1) = 21$ und $age_{Q'}(q_2) = 23$. ◇

Konzeptänderungsanweisungen

Update-Anweisungen ändern die Interpretation eines Operators, was durch das Feuern eines Updates umgesetzt wird. Ein solches Update ändert die durch eine

Lokation gegebene Interpretation eines Operatorsymbols. Neben der Änderung der Operatorinterpretation lässt sich auch die Sortenzugehörigkeit eines Objektes durch Konzeptänderungsanweisungen ändern. Diese Konzeptänderungsanweisungen werden analog durch Sorten-Updates interpretiert, die einem Element der Algebra die hinzuzufügenden und die zu entfernenden Sorten zuweisen.

Definition 4.11 (Sorten-Update)
Ein *Sorten-Update* eines Zustandes $\mathfrak{Q}$ ist ein Tripel

$$(q, S_{add}, S_{rem}) \in S_Q \times 2^{S^{dyn}} \times 2^{S^{dyn}} \cup S_Q \times 2^{S^{stat}} \times \{\{\top_{\overline{use}}\}\},$$

wobei q ein Element der Trägermenge von $\mathfrak{Q}$ ist und S_{add} und S_{rem} Mengen von Sorten sind, für die gilt $S_{add} \cap S_{rem} = \emptyset$ und die die Sorten angeben, deren Trägermengen um q ergänzt beziehungsweise verringert werden sollen. ⧫

Ein Sorten-Update ändert entweder die dynamischen Sorten eines Objektes der Algebra, oder es fügt statische Sorten hinzu, um ein Objekt zu initialisieren. Das Entfernen der Objekte aus dem System ist nicht Teil des Modells und geschieht wie beispielsweise bei Java durch ein Entfernen sämtlicher Referenzen auf ein Objekt, so dass ein Entfernen statischer Konzepte nicht benötigt wird. Die Wohlgeformtheit von Sorten-Updates garantiert, dass das Resultat des Updates der Signatur gehorcht und dass es eindeutig definiert ist.

Definition 4.12 (Wohlgeformtheit von Sorten-Updates)
Gegeben ein Zustand $\mathfrak{Q}$. Zwei Mengen S_{add} und S_{rem} sind *Sortenänderungsmengen* zu $s_* \in S$ gdw. für S_{add} und S_{rem} gilt

- $s \in S_{add} \cup S_{rem}$ impliziert, dass $s \preceq s_* \vee s_* \preceq s$,
- $s \in S_{add}$ impliziert, dass $\forall s' \in S : s \preceq s' \implies s_* \preceq s' \vee s' \in S_{add}$ und
- $s \in S_{rem}$ impliziert, dass $\forall s' \in S : s' \preceq s \implies s' \in S_{rem}$.

Ein Sorten-Update (q, S_{add}, S_{rem}) wird als *wohlgeformt* bezeichnet gdw. es ein $s_* \in S$ mit $q \in Q_{s_*}$ gibt und S_{add} und S_{rem} Sortenänderungsmengen zu s_* sind. ⧫

Wohlgeformtheit sorgt dafür, dass die veränderten Sorten nach wie vor die Sortenhierarchie respektieren und dass ein Objekt entweder genutzt oder ungenutzt ist. Wenn ein q einer bestimmten Sorte s_* angehört, so gehört q auch allen Supersorten von s_* an. Das Feuern eines Sorten-Updates bewirkt, dass q den Sorten aus S_{add} hinzugefügt wird und aus den Sorten aus S_{rem} entfernt wird. Dabei können wie in der ersten Bedingung gefordert nur Sorten hinzugefügt oder entfernt werden, die

bzgl. $\preceq$ in einer Hierarchie mit s_* stehen. Zudem müssen alle Supersorten ebenfalls hinzugefügt beziehungsweise alle Subsorten ebenfalls entfernt werden, damit die Sortenhierarchie nicht unterbrochen ist. Dies entspricht der zweiten und dritten Forderung aus Definition 4.12.

Bei Sorten-Updates muss berücksichtigt werden, dass Operatoren unter Umständen nicht mehr wie vor dem Sorten-Update interpretiert werden können. Wenn zum Beispiel ein Operator $\omega \in \Omega_{s_1,s_2}$ existiert und $\omega_Q(a) = b$ ist, b aber durch das Sorten-Update das Konzept s_2 verliert, muss $\omega_Q(a)$ undefiniert sein, was durch $\omega_Q(a) = \perp\!\!\!\perp$ ausgedrückt wird[1]. Das Symbol $\perp\!\!\!\perp$ ist hierbei eine spezielle Konstante, die angibt, dass ein Wert nicht definiert ist und die bereits beim Cast-Operator Verwendung fand. Hierzu ist $\perp\!\!\!\perp \in \Omega_{\epsilon,\perp\!\!\!\perp}$, so dass jeder Operator undefiniert sein kann[2].

Definition 4.13 (Feuern wohlgeformter Sorten-Updates, Folgezustand)
Das *Feuern eines wohlgeformten Sorten-Updates* $\mathbf{u}_{sort} = (q, S_{add}, S_{rem})$ führt von einem Zustand $\mathfrak{Q} = ((Q_s)_{s\in S}, (\omega_A)_{\omega\in\Omega})$ einer Systemsignatur Σ^{sys} zu einem *Folgezustand* $\mathfrak{Q}' = (S'_Q, \Omega'_Q)$, so dass

- $S_{Q'} = (Q'_s)_{s\in S}$ mit $Q'_s = \begin{cases} Q_s \cup \{q\} & \text{wenn } s \in S_{add} \\ Q_s \setminus \{q\} & \text{wenn } s \in S_{rem} \\ Q_s & \text{sonst} \end{cases}$
- $\Omega_{Q'} = (\omega'_Q)_{\omega\in\Omega}$, wobei für $\omega \in \Omega_{w,s}$ und $(q_1, \ldots, q_n) \in Q'_w$ gilt

$$\omega_{Q'}(q_1, \ldots, q_n) = \begin{cases} \perp\!\!\!\perp & \text{wenn } q_j \in (Q'_{s_j} \setminus Q_{s_j}) \text{ für } 1 \le j \le n \\ & \text{oder } \omega'_Q(q_1, \ldots, q_n) = q \text{ und } q \in (Q_s \setminus Q'_s) \\ \omega_Q(q) & \text{sonst} \end{cases}$$

Das Feuern wird notiert als $\mathfrak{Q}' = \mathfrak{Q} \oplus_Q \mathbf{u}_{sort}$. ♦

Zu zeigen bleibt, dass das Ergebnis des Feuerns auch wirklich ein Zustand ist, also dass für $s_1, s_2 \in S$ mit $s_1 \preceq s_2$ gilt, dass $Q_{s_1} \subseteq Q_{s_2}$ ist. Dies folgt daraus, dass für alle $s \in S_{add}$ gilt, dass $\forall s' \in S : s \preceq s' \rightarrow q \in Q_{s'} \vee s' \in S_{add}$ und für alle $s \in S_{rem}$ gilt, dass $\forall s' \in S : s' \preceq s \rightarrow q \notin Q_{s'} \vee s' \in S_{rem}$. Wenn also ein $q \in Q'_{s_1} \setminus Q_{s_1}$

[1] Nimmt man etwa das Konzept *Studentenausweis* mit einem Attribut *belegtStudentenstatusVon*, so gilt beispielsweise $belegtStudentenstatusVon(a) = tom$ für einen Studenten *tom* und einen Ausweis a. Ist *tom* nun kein Student mehr, sollte der Ausweis auch nicht mehr seinen Studentenstatus belegen, und es ist $belegtStudentenstatusVon(a) = \perp\!\!\!\perp$

[2] Das Problem der undefinierten Werte durch Sortenänderungen ist verwandt mit Fremdschlüsseln in Datenbanken (vgl. Kemper und Eickler, 1999) oder dem Dangling-Pointer-Problem bei objektorientierten Programmiersprachen ohne Garbage Collection (vgl. Dhurjati und Adve, 2006).

und $s_1 \preceq s_2$ gilt, so gilt auch $q \in Q'_{s_2} \setminus Q_{s_2}$ (Wenn q zu Q_{s_1} hinzugefügt wird, dann auch zu Q_{s_2}.). Ebenso gilt, dass wenn $q \in Q_{s_2} \setminus Q'_{s_2}$ und $s_1 \preceq s_2$ gilt, so ist auch $q \in Q_{s_1} \setminus Q'_{s_1}$ (Wenn q aus Q_{s_2} entfernt wird, dann auch aus Q_{s_1}.). Somit genügen die Trägermengen der Zustandsdefinition.

Die Operatoren der entfernten Sorten sind nach einem Sorten-Update nicht mehr referenzierbar. In einer konkreten Implementation könnte der Speicher der Attributwerte freigegeben werden, sofern die Werte nicht anderweitig referenziert werden. Wenn ein Objekt eine Sorte ablegt, kann es sein, dass Informationen verloren gehen. Dies ist bei der Modellierung zu berücksichtigen. Meist ist es daher sinnvoll, zu fordern, dass vor dem Entfernen einer Sorte sämtliche möglichen Operatoren auf $\perp\!\!\!\perp$ gesetzt werden.

Ein Sorten-Update feuert auch Updates der Operatoren, bei dem die veränderten Operatorinterpretationen durch den Zustand $\mathfrak{Q}'$ in Definition 4.13 gegeben sind. Diese Neuinterpretation der hinzugekommenen Operatoren ist notwendig, damit ein Zustand eine ordnungssortierte Algebra bleibt.

Neben den Operatoren auf der Sorte muss auch der Cast-Operator neu interpretiert werden. Dies erfolgt durch ein herkömmliches Update, das mit dem Sorten-Update einhergeht. Hierbei ist $\omega = (s)$ mit $\omega \in \Omega_{\top_{\mathfrak{Q}},bool}$ und $q \in Q'_s$ dann

$$\omega_{Q'}(q) = \begin{cases} \perp\!\!\!\perp & \text{wenn } q \in (Q_s \setminus Q'_s) \\ q & \text{wenn } q \in (Q'_s \setminus Q_s) \\ \omega_Q(q) & \text{sonst} \end{cases}$$

Nimmt man wiederum eine Signatur mit den Sorten *Person* und *Student* und einen Zustand $\mathfrak{Q}$ mit einem Element $q \in Q_{Student}$, so lässt sich durch ein Sorten-Update erreichen, dass q die dynamische Sorte *Student* entzogen wird. Dies würde als Sorten-Update $(q, \{\}, \{Student\})$ dargestellt. Das Resultat des Feuern des Sorten-Updates wäre, dass $q \notin Q_{Student}$ gelten würde.

Wie leicht überprüft werden kann, gilt für den Folgezustand $\mathfrak{Q}'$ eines Zustands $\mathfrak{Q}$ bezüglich eines Sorten-Updates (q, S_{add}, S_{rem}), dass $s \in S_{add}$ und $q \notin Q_s$ impliziert $\mathfrak{Q}'(\omega, (q_1, \ldots, q, \ldots, q_n)) = \perp\!\!\!\perp$ für $\omega \in \Omega_{w,s}$ mit $(q_1, \ldots, q, \ldots, q_n) \in Q_w$. Somit sind alle Attribute zunächst undefiniert für eine neu hinzugekommene Sorte.

Mit Hilfe von Sorten-Updates lässt sich auch das Erzeugen von Objekten elegant lösen. Noch nicht erzeugte Objekte besitzen zunächst die spezielle Sorte $\top_{\overline{use}}$ und wechseln diese sobald sie erzeugt werden. Somit muss die Sorte $\top_{\overline{use}}$ lediglich einen ausreichenden Vorrat an Elementen aufweisen, was durch eine unendliche Trägermenge in jedem Fall möglich ist.

Es sei auch darauf hingewiesen, dass durch Updates keine neuen Objekte hinzukommen und auch keine Objekte gelöscht werden, da jedes Objekt mindestens einer Sorte angehören muss. Vielmehr gilt stets $\bigcup_{s \in S} Q_s = \bigcup_{s \in S} Q'_s$ und somit $S_Q = S_{Q'}$ für $\mathfrak{Q}' = \mathfrak{Q} \oplus_Q \mathbf{u}_{sort}$ mit einem beliebigen $\mathbf{u}_{sort}$. Es ändert sich somit lediglich die Zuordnung zu Sorten.

In Dienstbeschreibungsnetze werden Sorten-Updates durch Konzeptänderungsanweisungen umgesetzt. Hierbei werden wiederum zunächst die Terme der Konzpetänderungsanweisung im aktuellen Zustand $\mathfrak{Q}$ unter einer Variablenbelegung α interpretiert. Dies wird für eine gegebene Konzeptänderungsanweisung $\mathbf{cmd}_C$ als $[\![\mathbf{cmd}_C(\alpha)]\!]^{\mathfrak{Q}}$ notiert und ist in Tabelle 4.3 dargestellt.

Anweisung $\mathbf{cmd}_C$	Sorten-Update $[\![\mathbf{cmd}_C(\alpha)]\!]^{\mathfrak{Q}}$
$addRole(\tau, s_{\mathfrak{c}})$	$\mathbf{u}_{sort} = ([\![\tau]\!]^{\mathfrak{Q}}_{\alpha}, S_{add}, \emptyset)$ wobei für $\tau \in \mathbb{T}_{\Sigma^{\mathfrak{Q}},s}(X)$ gilt $S_{add} = \{s' \in S^{dyn} \mid s_{\mathfrak{c}} \preceq s' \preceq s \wedge s \neq s'\}$
$removeRole(\tau, s_{\mathfrak{c}})$	$\mathbf{u}_{sort} = ([\![\tau]\!]^{\mathfrak{Q}}_{\alpha}, \emptyset, S_{rem})$ wobei für $\tau \in \mathbb{T}_{\Sigma^{\mathfrak{Q}},s}(X)$ gilt $S_{rem} = \{s' \in S^{dyn} \mid s' \preceq s_{\mathfrak{c}}\}$
$new\ (v, s_{\mathfrak{c}})$	$\mathbf{u}_{sort} = (v, \{s \in S^{stat} \mid s_{\mathfrak{c}} \preceq s\}, \{\top_{\overline{use}}\})$ mit $v \in X_{\top_{\overline{use}}}$ und $\alpha(v) = q_{\top_{\overline{use}}}$ für ein $q_{\top_{\overline{use}}} \in Q_{\top_{\overline{use}}}$

Tabelle 4.3: Konzeptänderungsanweisungen und ihre Interpretation

Es lässt sich nun zeigen, dass jede dieser Abbildungen ein wohlgeformtes Term-Sorten-Update erzeugt. Da alle Sorten s' mit $s_{\mathfrak{c}} \preceq s' \preceq s \wedge (s \neq s')$ ebenfalls in S_{add} sind beziehungsweise alle Sorten s' mit $s' \preceq s_{\mathfrak{c}}$ ebenfalls in S_{rem} sind, sind die drei Kriterien von Definition 4.12 erfüllt, da die Auswertung eines Terms zwar von einer Subsorte der Sorte des Terms sein kann, es jedoch nicht möglich ist, dass der Term einer spezielleren Sorte angehört als seine Interpretation.

Lemma 4.14 (Auswertung der Konzeptänderungsanweisungen)
Die Interpretation $[\![\mathbf{cmd}_C(\alpha)]\!]^{\mathfrak{Q}}$ einer Konzeptänderungsanweisung $\mathbf{cmd}_C$ in einem Zustand $\mathfrak{Q}$ unter einer Variablenbelegung α erzeugt ein wohlgeformtes Sorten-Update.

Die *Ausführung einer Konzeptänderungsanweisung* $\mathbf{cmd}_C$ in einem Zustand $\mathfrak{Q}$ unter einer Variablenbelegung α entspricht dem Feuern des korrespondierenden Sorten-Updates $[\![\mathbf{cmd}_C(\alpha)]\!]^{\mathfrak{Q}}$, so dass eine Abbildung $[\![\mathbf{cmd}_C(\alpha)]\!] : \mathbf{States}(\Sigma^{\mathfrak{Q}}) \rightarrow \mathbf{States}(\Sigma^{\mathfrak{Q}})$ den Übergang der Zustände darstellt. Dies ist analog zur Ausführung von Update-Anweisungen auf Seite 108. Somit stellen sowohl Update-Anweisungen als auch Konzeptänderungsanweisungen Zustandsübergänge dar, die als Abbildungen auf der Klasse der Zustände interpretiert werden.

Ein Beispiel soll die Nutzung von Konzeptänderungsanweisungen illustrieren. Dabei wird die Signatur aus Beispiel 4.1 wieder aufgegriffen und um eine dynamische Sorte *Student* erweitert.

Beispiel 4.2: Sei die Zustandssignatur wie folgt gegeben.

```
sorts Person, Student .
subsort Student < Person .
op name: Person -> string .
op age: Person -> nat .
op student_id: Student -> string .
var v1, v2: Person .
var vs: Student .
```

Sei Q ein Zustand, so dass $Q_{Person} = \{q_1, q_2\}$ und $Q_{Student} = \emptyset$ ist und sei die Operatorinterpretation der Algebra wie folgt gegeben: $name_Q(q_1) =$"bob", $name_Q(q_2) =$"tom", $age_Q(q_1) = 21$ und $age_Q(q_2) = 23$. Statt die Interpretation der Operatoren zu verändern, kann über eine Konzeptänderungsanweisung der Form $addRole(v_1, Student)$ einem Objekt vom Typ *Person* die Sorte *Student* zugewiesen werden. Durch die Variablenbelegung α mit $\alpha(v_1) = q_1$ erhält das Objekt q_1 die neue Sorte *Student*. Für den Folgezustand $\mathfrak{Q}'$ ist dann $Q'_{Student} = \{q_1\}$ und $student\_id_{Q'}(q_1) = \perp$. Dabei wird die Konzeptänderungsanweisung in ein Sorten-Update $(q_1, \{Student\}, \{\})$ überführt, da $[\![v_1]\!]^{\mathfrak{Q}}_{\alpha} = q_1$ ist. ◇

Auswertung eines Anweisungsblocks

Die Auswertung eines Anweisungsblocks entspricht der Hintereinanderausführung aller Update-Anweisungen und Konzeptänderungsanweisungen in der im Anweisungsblock gegebenen Reihenfolge. Da die Ausführung jeder dieser Anweisungen einem Zustandsübergang, also einer Funktion von einem Zustand in einen Folgezustand, entspricht lassen sich diese Funktionen komponieren, was der Interpretation des Anweisungsblocks entspricht. Nicht behandelt wurden bislang die Zuweisungsanweisungen, deren Interpretation zunächst folgt.

Zuweisungsanweisungen In Beispiel 4.2 wäre es sinnvoll, das Element der Algebra q_1, das um eine weitere Sorte ergänzt wurde, weiter zu verwenden, so dass die neu hinzugewonnenen Attribute initialisiert werden können. Hierzu ist es jedoch notwendig, q_1 der Variable v_s zuzuweisen, da die Variablen v_1 und v_2 nicht der Sorte *Student* angehören und auf ihnen der Operator *student_id* folglich nicht definiert ist. Diese Zuweisung erfolgt durch Zuweisungsanweisungen wie sie bereits in Definition 4.7 definiert worden sind. Die Interpretation einer solchen Zuweisungsanweisung erfolgt durch eine Erweiterung der Variablenbelegung, die der Variablen der linken Seite der Zuweisung einen Wert zuweist, der der Auswertung der rechten Seite der Zuweisung entspricht. Die Variable der linken Seite wurde in Definition 4.7 durch die Menge $Bound(\mathbf{cmd}_A)$ bezeichnet.

Definition 4.15 (Erweiterte Variablenbelegung)
Sei X eine S-indizierte Menge von Variablen, $\alpha : X \to S_Q$ eine Variablenbelegung in einem Zustand $\mathfrak{Q}$ und $\mathbf{cmd}_A = (assign\ v = \tau)$ eine Zuweisungsanweisung.
Zu $\mathbf{cmd}_A$ ist die *erweiterte Variablenbelegung* $\bar{\alpha}^{\mathfrak{Q}}_{\mathbf{cmd}_A} : X \cup Bound(\mathbf{cmd}_A) \to S_Q$ definiert durch

$$\bar{\alpha}^{\mathfrak{Q}}_{\mathbf{cmd}_A}(x) \stackrel{\text{def}}{=} \begin{cases} \alpha(x) & \text{wenn } x \neq v \\ [\![\tau]\!]^{\mathfrak{Q}}_{\alpha} & \text{sonst} \end{cases}$$

♦

Die erweiterte Variablenbelegung $\bar{\alpha}^{\mathfrak{Q}}_{\mathbf{cmd}_A}$ ist hierbei abhängig vom Zustand $\mathfrak{Q}$, von der Variablenbelegung α und von der Zuweisungsanweisung $\mathbf{cmd}_A$. Eine Zuweisungsanweisung ändert den Zustand $\mathfrak{Q}$ nicht und stellt somit eine identische Abbildung eines Zustandes auf sich selbst dar.

Anweisungsblöcke und Belegungskonsistenz Da die Ausführung jeder Anweisung als Abbildung von der Klasse der Zustände $\mathbf{States}(\Sigma^{\mathbf{Q}})$ in die Klasse der Zustände $\mathbf{States}(\Sigma^{\mathbf{Q}})$ interpretiert wird, entspricht die Auswertung eines wohlgeformten Anweisungsblocks der Komposition der einzelnen Anweisungen. Für eine Anweisung $cmd(\alpha)$ ist $[\![cmd(\alpha)]\!] : \mathbf{States}(\Sigma^{\mathbf{Q}}) \to \mathbf{States}(\Sigma^{\mathbf{Q}})$. Für einen Anweisungsblock ohne Zuweisungsanweisung, also für eine Folge von Anweisungen, $\mathbf{Cmd} = cmd_1 \ldots cmd_n$ über $\Sigma^{\mathbf{Q}}$ und X entspricht die Auswertung der Funktionskomposition

$$[\![\mathbf{Cmd}(\alpha)]\!]^{\mathfrak{Q}} = [\![cmd_1(\alpha_1)]\!] \circ \ldots \circ [\![cmd_n(\alpha_n)]\!](\mathfrak{Q})$$

Hierbei wurde die Notation $[\![cmd_j(\alpha_j)]\!](\mathfrak{Q})$ statt der Notation $[\![cmd_j(\alpha_j)]\!]^{\mathfrak{Q}}$ zur Verdeutlichung der Komposition gewählt. Die Variablenbelegung α_j ist abhängig von den ihr vorausgegangenen Zuweisungsanweisungen. So ist $\alpha_1 = \alpha$ und α_j in Abhängigkeit der Zuweisungsanweisungen definiert, wobei die erweiterte Variablenbelegung $\bar{\alpha}^{\mathfrak{Q}}_{\mathbf{Cmd}}$ genutzt wird. Diese ist für wohlgeformte Anweisungsblöcke (vgl. Def 4.8) wohldefiniert, da eine wohlgeformte Zuweisungsanweisung jeder Variablen genau einen Wert zuweist. Die Variablenbelegung $\bar{\alpha}^{\mathfrak{Q}}_{\mathbf{Cmd}}$ weist jeder Variablen den Wert der Auswertung der Terme auf der rechten Seite einer Zuweisungsanweisung zu. Hierbei kann es jedoch sein, dass eine Update-Anweisung den Wert einer Lokation ändert, die später für die Auswertung einer Variablen benötigt wird, so dass sich die Variablenbelegung während der Ausführung des Anweisungsblocks ändert.

Ein Anweisungsblock, bei dem die Zuweisungsanweisungen, die einen Operator aus Ω^{sys} verwenden, in der Folge $\mathbf{Cmd}$ sämtlich vor den Update-Anweisungen angeordnet sind wird als *belegungskonsistent* bezeichnet. Für einen solchen Anweisungsblock ergibt sich die erweiterte Variablenbelegung direkt aus den Anweisungen und

wird durch die weitere Ausführung der Updates nicht mehr verändert. Ein Anweisungsblock, der nicht belegungskonsistent ist, wäre beispielsweise durch folgende Anweisungen gegeben:

```
update x.name = "Tom";
assign v = x.name;
```

Dadurch, dass sich die Lokation $(name, [\![x]\!]_\alpha)$ ändert, wäre die Zuweisung gleichbedeutend mit der Zuweisung *assign* $v =$`"Tom"`. Belegungskonsistente Anweisungen besitzen den Vorteil, dass die Auswertung des Anweisungsblocks dergestalt erfolgen kann, dass zunächst sämtliche Zuweisungsanweisungen ausgewertet werden können, woraus sich eine erweiterte Variablenbelegung ergibt, durch die dann die Anweisungen in **Cmd** interpretiert werden können. In diesem Fall lässt sich der Anweisungsblock, in dem sämtliche Variablen belegt sind, wiederum als $\mathbf{Cmd}(\alpha)$ angeben, und es ist $[\![\mathbf{Cmd}(\alpha)]\!] : \mathbf{States}(\Sigma^{\mathbf{Q}}) \to \mathbf{States}(\Sigma^{\mathbf{Q}})$ eine Abbildung von einem Zustand auf einen Folgezustand, die der *Auswertung des Anweisungsblocks* **Cmd** entspricht. Die Zuweisungsanweisungen stellen dabei identische Abbildungen auf der Klasse der Zustände dar.

Ein Anweisungsblock $\mathbf{Cmd} = cmd_1 \dots cmd_n$ ist also *belegungskonsistent* gdw. $cmd_j = (assign\ v = \tau)$ und $cmd_k = (update\ \omega(\tau_1, \dots, \tau_n) = \tau)$ impliziert, dass $j < k$ ist oder τ keinen Operator aus Ω^{sys} enthält. In einem gegebenen Zustand $\mathfrak{Q}$ unter einer Variablenbelegung α lässt sich jeder wohlgeformte Anweisungsblock in einen belegungskonsistenten Anweisungsblock überführen, indem, wie in obigem Beispiel dargestellt, die Zuweisungen ersetzt werden. Dort konnte die Zuweisungsanweisung nach vorne gezogen werden, indem stattdessen *assign* $v =$`"Tom"` genutzt wird.

Zur Überführung eines Anweisungsblocks in einen belegungskonsistenten Anweisungsblock wird zu jeder Update-Anweisung die (einelementige) *Menge der veränderten Lokationen* berechnet, die für $\mathbf{cmd}_U = (update\ \omega(\tau_1, \dots, \tau_n) = \tau)$ und $\overline{q} = [\![\tau_1]\!]^{\mathfrak{Q}}_\alpha, \dots, [\![\tau_n]\!]^{\mathfrak{Q}}_\alpha$ als $\mathbf{Loc^U}(\mathbf{cmd}_U, \mathfrak{Q}, \alpha) \stackrel{\text{def}}{=} \{(\omega, \overline{q})\}$ definiert ist. Gleichzeitig wird für jede Zuweisungsanweisung $\mathbf{cmd}_A$ die *Menge der involvierten Lokationen* $\mathbf{Loc^I}(\mathbf{cmd}_A, \mathfrak{Q}, \alpha)$ gebildet. Diese ist für einen Term τ durch $\mathbf{Loc^I}(\tau, \mathfrak{Q}, \alpha) \stackrel{\text{def}}{=} \{(\omega, \overline{q}) \mid \omega(\tau_1, \dots, \tau_n) \in \textsc{Subterm}(\tau) \wedge \omega \in \Omega^{sys} \wedge \overline{q} = [\![\tau_1]\!]^{\mathfrak{Q}}_\alpha, \dots, [\![\tau_n]\!]^{\mathfrak{Q}}_\alpha\}$ gegeben. Diese Menge beschreibt alle Lokationen, die veränderlich sind und die zur Auswertung des Terms τ ausgelesen werden müssen. Entsprechend ist $\mathbf{Loc^I}(\mathbf{cmd}_A, \mathfrak{Q}, \alpha) \stackrel{\text{def}}{=} \mathbf{Loc^I}(\tau, \mathfrak{Q}, \alpha)$ für $\mathbf{cmd}_A = (assign\ v = \tau)$[3].

Ist cmd_k eine Zuweisungsanweisung und cmd_j eine Update-Anweisung für $j < k$ und ist $\mathbf{Loc^I}(\mathbf{cmd}_A, \mathfrak{Q}, \alpha) \cap \mathbf{Loc^U}(\mathbf{cmd}_U, \mathfrak{Q}, \alpha) \neq \emptyset$, so wird im Term der Zuweisungsanweisung der Subterm durch den Wert des Updates ersetzt, der der Loka-

[3] Streng genommen müsste hier der Begriff der erweiterten Lokation eingeführt werden, da durch Konzeptänderungsanweisungen die Möglichkeit besteht, dass eine Lokation noch nicht für einen Zustand existiert, sondern vielmehr erst nach dem Hinzufügen einer Sorte zu einem Element der Algebra existiert. Um die Definition nicht übermäßig komplex werden zu lassen, wird hiervon jedoch an dieser Stelle abstrahiert.

tion entspricht. Hierdurch ist gewährleistet, dass sich sämtliche Terme im Zustand $\mathfrak{Q}$ unter einer erweiterten Variablenbelegung $\bar{\alpha}$ auswerten lassen. Ist ein Anweisungsblock wohlgeformt, so lassen sich diese Schritte stets durchführen, da es keine undefinierten Variablen geben kann, da eine Variable einer zuvor entfernten Sorte wie in Definition 4.8 dargestellt, nicht mehr in einer Anweisung verwendet werden darf. Dies bedeutet, dass sich jeder wohlgeformte Anweisungsblock in einen belegungskonsistenten Anweisungsblock überführen lässt. Somit lässt sich $\mathbf{Cmd}(\alpha)$ stets als Zustandsübergang verstehen.

Bei der Auswertung eines Anweisungsblocks ist es meist sinnvoll anzunehmen, dass für spezielle Symbole wie $\perp\!\!\!\perp$ die Update- und Konzeptänderungsanweisungen die Identität auf den Zuständen darstellen, so dass sich die Interpretation von $\perp\!\!\!\perp$ nicht ändert. Denkbar wäre jedoch auch statt $\perp\!\!\!\perp$ spezielle Fehlersymbole zu verwenden, die der Ausnahmebehandlung dienen. Dies soll hier jedoch nicht weiter thematisisert werden. Abschließend lässt sich somit folgende Aussage festhalten.

> *Ein wohlgeformter Anweisungsblock lässt sich stets in eine belegungskonsistente Form überführen und stellt dann eine Abbildung von der Klasse der Zustände in die Klasse der Zustände dar.*

Aus den Zuständen $\mathbf{States}(\Sigma^{\mathbf{Q}})$ als Objekten und aus den Anweisungen als Morphismen ergibt sich die *Kategorie der Zustände*. Die Menge der Morphismen für je zwei Objekte $\mathfrak{Q}, \mathfrak{Q}'$ ist dabei durch die Tripel aus Anweisungen und Zuständen gegeben, die den Zustand $\mathfrak{Q}$ in den Zustand $\mathfrak{Q}'$ überführen (also $(\mathfrak{Q}, \mathbf{cmd}, \mathfrak{Q}')$). Die Morphismenkomposition entspricht dann der Anweisungshintereinanderausführung, so dass sich die Kategorie der Zustände ergibt. Hierbei sind die Morphismen also nicht die Homomorphismen der Algebra, sondern die möglichen Zustandsübergänge durch die Anweisungen.

Kompakte Notation und Anmerkungen zur praktischen Umsetzung

Häufig werden Zuweisungen und Updates in bestimmten Mustern verwendet, für die im Folgenden eine Kurznotation eingeführt wird. Für ein unäres Operatorsymbol $\omega \in \Omega_{s_1,s_2}$ wird die Term-Operatorlokation (ω, τ) durch $\tau.\omega$ notiert. Diese Notation wird als abkürzende Schreibweise für Terme genutzt, so dass die Schreibweise $\tau.\omega_1.\omega_2$ verwendet werden kann. Dies steht für die Hintereinanderausführung der Operatoren $\omega_2(\omega_1(\tau))$. Hierdurch sind auch Ketten der Form $\tau.\omega_1.\ldots.\omega_n$ möglich. Für Updates kann ebenso *update* $\tau.\omega = \tau$ notiert werden.

Bei *new*-Anweisungen wird nicht jedesmal explizit die Variable und ihre Zuweisung angeben, sondern stattdessen kürzer *assign* $v = new\ s_{\mathfrak{c}}$ notiert. Zudem werden die folgenden verkürzenden Schreibweisen verwendet.

$assign\ v = addRole(\tau, s_\mathfrak{c})$	statt	$addRole(\tau, s_\mathfrak{c})$ $assign\ v = (s)\tau$	mit $s_\mathfrak{c} \preceq s$ und $v \in X_s$
$assign\ v = removeRole(\tau, s_\mathfrak{c})$	statt	$assign\ v = \tau$ $removeRole(\tau, s_\mathfrak{c})$	mit $s_\mathfrak{c} \preceq s$ und $v \in X_s$
$assign\ v = new\ s_\mathfrak{c}$	statt	$new\ (v', s_\mathfrak{c})$ $assign\ v = (s)v'$	mit $s_\mathfrak{c} = s$ und $v \in X_s, v' \in X_{\top_{\overline{use}}}$

Gibt es Operatoren, die abhängig von der Sortenzugehörigkeit sind, so werden diese implizit ebenfalls durch Konzeptänderungsanweisungen geändert. Ein Cast-Operator ist etwa davon abhängig, ob ein Objekt eine Sorte besitzt. So ist $(s)\tau$ von der Sorte s, wenn die Auswertung von τ ein Objekt ist, das auch von der Sorte s ist. Ebenso ist die Abfrage, ob eine Term von der Sorte s ist, gegeben durch $(\tau\ \mathbf{inst}\ s) \stackrel{\text{def}}{=} (\perp\!\!\!\perp \neq (s)\tau)$ veränderlich und wird ebenfalls durch ein implizites Update verändert, so dass *removeRole*- und *addRole*-Anweisungen die Interpretation entsprechend verändern.

Beispiel 4.3: Dieses Beispiel zeigt einen kurzen Anweisungsblock, in dem ein durch die Variable x referenziertes Element der Algebra die Rolle Student hinzugefügt bekommt, also eine zusätzliche Sorte erlangt. Das Ergebnis wird der Variable $v1$ zugewiesen. Zudem wird *en* ein Element der Trägermenge zugewiesen, dass vorher nicht benutzt war und nun der Sorte *Enrolment* angehört. Im Rahmen der Update-Anweisungen finden verschiedene Neuinterpretationen der Operatorsymbole statt. Der Operator *enrolments* wird durch *en* neu interpretiert. Zudem werden die Operatoren *id* und *university* für *en* definiert. Dies geschieht durch folgende Anweisungen.

```
/* Konzeptanweisungen */
assign v1 = addRole(x, Student);
assign en = new Enrolment;

/* Update-Anweisungen */
update v1.enrolment = en;
update en.id = "593322333";
update en.university = "University of Hamburg";
```

◇

Die Variable x in Beispiel 4.3 kann nun über eine Eingangskante einer Transition mittels einer Variablenbelegung an ein Objekt der Algebra gebunden werden. Zudem können v_1 und *en* Ausgangsvariablen dieser Transition sein.

Zur praktischen Umsetzung ist es sinnvoll, die Anweisungen um den Fall zu erweitern, dass einer der Terme undefiniert ist. In diesem Fall sollte entweder die

Anweisung oder der gesamte Anweisungsblock zu undefiniert auswerten, so dass allen Ausgangsvariablen der Wert undefiniert zugewiesen wird. Dies erlaubt es beispielsweise eine Ausnahmebehandlung zu implementieren, indem dieses Vorgehen durch verschiedene undefinierte Werte der Sorte $\perp^{\Omega}$ ergänzt wird. So kann dann beispielsweise ein Fehlercode, eine Nachricht oder der Auslöser eines Fehlers übergeben werden, wie dies bei der Ausnahmebehandlung in Programmiersprachen der Fall ist. Dieses Verhalten kann jedoch auch durch spezielle Transitionen erreicht werden, weswegen es hier keine Festlegung auf ein spezielles Vorgehen gibt.

4.3 Dienstbeschreibungsnetze

Dieser Abschnitt führt den Formalismus der Dienstbeschreibungsnetze ein. Zunächst werden die statischen Eigenschaften der Netze beschrieben. Hierbei wird in erster Linie die Syntax der Netze erläutert. Zum näheren Verständnis der Syntax sei auf das Beispiel aus Abschnitt 4.1 verwiesen. Nach der Beschreibung der Netzsyntax in Abschnitt 4.3.1 wird in Abschnitt 4.3.2 die Dynamik von Dienstbeschreibungsnetzen beschrieben. Dort folgt die Definition der Schaltregel und der Aktiviertheit.

4.3.1 Statische Netzeigenschaften

Dienstbeschreibungsnetze sind gefärbte Petrinetze mit speziellen Anweisungen zur Änderung eines Systemzustandes. Sie weichen vom Formalismus der algebraischen Netze im Wesentlichen an drei Stellen ab.

- Einerseits wird eine ordnungssortierte Signatur verwendet statt einer heterogenen Spezifikation, wie dies bei algebraischen Netzen der Fall ist. Die Ordnung auf den Sorten erlaubt es, Subtypen darzustellen. Die Verwendung einer Algebra statt der Verwendung einer Spezifikation ermöglicht es, die Veränderung der Sortenzugehörigkeit darzustellen, was bei einer Spezifikation so nicht möglich wäre, da die Sortenzugehörigkeit nicht über Gleichungen der Spezifikation festgelegt wird.

- Zweitens sind die Kantenanschriften lediglich Variablennamen und nicht beliebige Terme. Dies erlaubt einerseits eine leichtere Bindungssuche bei der Ausführung und andererseits eine leichtere Übertragung von Ergebnissen der Analyse des zugrundeliegenden ungefärbten Netzes auf das Dienstbeschreibungsnetz.

- Drittens besitzen Dienstbeschreibungsnetze eine Liste von Anweisungen, die angeben wie sich der Systemzustand ändern soll. Dieser Anweisung bilden eine Funktion, die vom aktuellen Systemzustand, kodiert als Algebra, auf den Folgezustand abbildet.

Ein Dienstbeschreibungsnetz besitzt die in Abbildung 4.5 dargestellten Elemente zusammen mit der Möglichkeit, Aktivierungsbedingungen für Transitionen zu definieren. Somit besitzt ein Dienstbeschreibungsnetz eine Zustandssignatur zur Darstellung der Daten und eine Menge von Variablen, die für die Kantenanschriften und die Anweisungen genutzt werden. Die Anweisungen werden den Transitionen zugeordnet. Zudem gibt es eine Aktivierungsbedingung, so dass sich folgende Netzdefinition ergibt, wobei eine boolesche Zustandssignatur wiederum eine Zustandssignatur ist, die die Sorte *bool* enthält.

Definition 4.16 (Dienstbeschreibungsnetz)
Ein *Dienstbeschreibungsnetz* (*SD-Netz*) $\mathcal{SDN} = (\Sigma^{\mathbf{Q}}, X, \mathcal{N}, d, \mathbf{cmd}, W, G)$ ist ein Tupel derart, dass

- $\Sigma^{\mathbf{Q}}$ eine boolesche Zustandssignatur ist,
- X eine S-indizierte Familie von Variablen ist,
- $\mathcal{N} = (P, T, F)$ ein Workflownetz ist,
- $d : P \to S$ eine *Stellentypisierung* ist,
- $\mathbf{cmd} : T \to \mathbf{Cmds}_{\Sigma^{\mathbf{Q}}}(X)$ jeder Transition einen wohlgeformten Anweisungsblock zuweist,
- $W : F \to X$ eine *Kantenbeschriftung* ist, so dass für alle $f \in F$ gilt
  - $W(f) \in (X_s \setminus \mathit{Bound}(\mathbf{cmd}(t)))$ mit $d(p) \preceq s$ wenn $f = (p, t) \in P \times T$,
  - $W(f) \in X_s$ mit $s \preceq d(p)$ wenn $f = (t, p) \in T \times P$,
  - $(p, t) \in F$ und $(p', t) \in F$ und $p \neq p'$ impliziert $W(p, t) \neq W(p', t)$ und
- $G : T \to \mathbb{T}_{\Sigma^{\mathbf{Q}}, \mathit{bool}}(X)$ jeder Transition eine *Aktivierungsbedingung* zuweist.

♦

Trotz der Nähe zu algebraischen Netzen existieren zwei wesentliche Unterschiede. Zum einen wird statt einer Spezifikation eine Signatur verwendet, da die Interpretation der Terme nicht über die Quotiententermalgebra erfolgt sondern über eine Algebra. Zum anderen sind an den ein- und ausgehenden Kanten einer Transition lediglich Variablen erlaubt. Sämtliche Terme werden also durch die Anschriften berechnet.

Ein Netzzustand ist eine Markierung des Netzes zusammen mit einem Zustand, der die Zustandssignatur und die Terme im Netz interpretiert.

Definition 4.17 (Markierung und Netzzustand)
Gegeben ein Dienstbeschreibungsnetz $\mathcal{SDN}$. Eine *Markierung* $\mathbf{m}$ von $\mathcal{SDN}$ in einem Zustand $\mathfrak{Q} = (S_Q, \Omega_Q)$ ist eine Abbildung $\mathbf{m} : P \to MS(S_Q)$, so dass $\mathbf{m}(p) \in MS(Q_s)$ mit $s \preceq d(p)$ für $p \in P$. Ein *Netzzustand* $Q = (\mathbf{m}, \mathfrak{Q})$ von $\mathcal{SDN}$ ist eine Markierung $\mathbf{m}$ von $\mathcal{SDN}$ zusammen mit einem Zustand $\mathfrak{Q}$, so dass $\mathbf{m} : P \to MS(S_Q)$ gilt. ♦

Wiederum lässt sich der Unterschied zu den algebraischen Netzen in Definition 4.17 erkennen, da eine Markierung nicht auf die Menge der Terme $\mathbb{T}_{\Gamma^{MS}}$ abbildet, wie in Definition 3.46, sondern auf die Menge der Träger der Algebra S_Q. Das Dienstbeschreibungsnetz nutzt also nicht die Quotiententermalgebra für die Markierungen sondern die Algebra des Zustands selbst. Um die Notation zu vereinfachen, wird dabei davon ausgegangen, dass $[\![true]\!]_\alpha^{\mathfrak{Q}} = true$ und $[\![false]\!]_\alpha^{\mathfrak{Q}} = false$ ist. Gleiches gilt für Zahlen und Zeichenketten.

Definition 4.18 (Initialer Netzzustand, Dienstbeschreibungsnetzsystem)
Ein Dienstbeschreibungsnetz $\mathcal{SDN}$ zusammen mit einem *initialen Netzzustand* $Q_0 = (\mathbf{m}_0, \mathfrak{Q}_0)$ wird *Dienstbeschreibungsnetzsystem* genannt. Dieser *initiale Netzzustand* besteht aus einer *Initialmarkierung* des Netzes $\mathbf{m}_0 : P \to MS(S_Q)$ und aus einem *initialen Zustand* $\mathfrak{Q}_0$, so dass $\mathbf{m}_0(p) = \mathbf{0}$ für alle $p \neq i$. (Nur die Anfangsstelle i des Workflownetzes ist initial markiert.) ♦

Damit das Verhalten jeder Transition eindeutig bestimmt ist, ist es meist sinnvoll, dass abgesehen von *new*-Anweisungen alle Anweisungen lediglich Variablen verwenden, die entweder einer Eingangskante zugewiesen wurden oder die durch Zuweisungsanweisungen definiert wurden[4]. Sei $\textsc{Var}^-(t) \stackrel{\text{def}}{=} \{W(p,t) \mid p \in {}^\bullet t\}$ und sei $\textsc{New}(\mathbf{Cmd})$ die Menge der *new*-Anweisungen in einem Anweisungsblock $\mathbf{Cmd}$, so stellt sich diese Forderung wie folgt dar:

$$\textsc{Var}^-(t) \cup \{v \in X \mid \exists s_\mathfrak{c} \in S^{stat} : \mathit{new}\ (v, s_\mathfrak{c}) \in \textsc{New}(\mathbf{cmd}(t))\} = \mathit{Free}(\mathbf{cmd}(t))$$

Diese Forderung führt dazu, dass alle freien Variablen eines Anweisungsblocks durch die Eingangsvariablen gebunden sind. Somit können lediglich die den *new*-Anweisungen zugewiesenen Objekte der Algebra variieren. Verwendet man diese Notation, so gilt zudem $\mathit{Bound}(\mathbf{cmd}(t)) \cap \textsc{Var}^-(t) = \emptyset$, was aus den Bedingungen an die Kantenbeschriftung W hervorgeht. Nach Definition 4.16 ließe sich auch eine Variable an einer Ausgangskante verwenden, die nicht durch eine Zuweisung oder eine Variable an einer Eingangskante gebunden ist. Dies ist in praktischen Fällen jedoch ebenfalls unüblich und ist daher in der Regel auch zu vermeiden, so

[4] *new*-Anweisungen benötigen lediglich in der ursprünglichen Form eine Variable, in der abkürzenden Schreibweise wird diese weggelassen.

dass $\textsc{Var}^+(t) \subseteq \mathit{Bound}(\mathbf{cmd}(t)) \cup \textsc{Var}^-(t)$ ist. Dienstbeschreibungsnetze, deren verwendete Variablen durch die Eingangskanten determiniert werden, werden als variablendeterminiert bezeichnet, was wie folgt definiert ist.

Definition 4.19 (Variablendeterminierte Dienstbeschreibungsnetze)
Ein Dienstbeschreibungsnetz $\mathcal{SDN}$ ist *variablendeterminiert* gdw. für jede Transition $t \in T$ gilt

$$\textsc{Var}(G(t)) \subseteq \textsc{Var}^-(t),$$

$$\textsc{Var}^+(t) \subseteq \mathit{Bound}(\mathbf{cmd}(t)) \cup \textsc{Var}^-(t) \text{ und}$$

$$\mathit{Free}(\mathbf{cmd}(t)) \subseteq \textsc{Var}^-(t) \cup \{v \in X \mid \exists s_\mathfrak{c} \in S^{stat} : \mathit{new}\ (v, s_\mathfrak{c}) \in \textsc{New}(\mathbf{cmd}(t))\}.$$

♦

Durch die Forderung, dass alle im Anweisungsblock $\mathbf{cmd}(t)$ verwendeten Variablensymbole (außer denjenigen in *new*-Anweisungen) entweder auf der linken Seite einer Zuweisungsanweisung stehen oder an genau einer der Eingangskanten als Beschriftung existieren, gestaltet sich die Bindungssuche einfach, da lediglich für die Marken auf einer Stelle entschieden werden muss, welche von ihnen an eine Variable gebunden werden soll.

Das Netz aus Abbildung 4.5 wurde noch einmal in Abbildung 4.8 wiederholt. Letztere Darstellung besitzt eine Markierung auf der Anfangsstelle. In diesem Fall ist $q \in Q_{PurchaseOrder}$ für den initialen Netzzustand $\mathfrak{Q}_0$ und es ist $\mathbf{m}_0(i) = q$, wobei q die verkürzte Darstellung der einelementigen Multimenge ist.

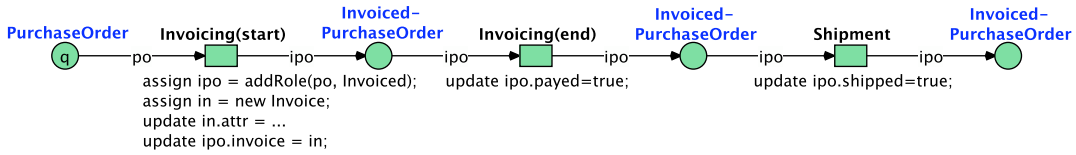


Abbildung 4.8: Darstellung der Initialmarkierung eines initialen Netzzustandes

Dadurch, dass lediglich Variablen an den Kanten eines Dienstbeschreibungsnetzes als Anschriften dienen, stellt das zugrundeliegende Netz eines Dienstbeschreibungsnetzes stets ein Workflownetz ohne Kantengewichte dar. Hierauf wird in Unterabschnitt 4.3.3 noch einmal eingegangen.

4.3.2 Aktiviertheit und Schalten

Das Entfernen von dynamischen Konzepten bedarf besonderer Aufmerksamkeit, da hierdurch Fernwirkungen entstehen können, die speziell behandelt werden müssen.

Aus diesem Grund wird zunächst eine Definition der Schaltmodi sowie der Aktiviertheit und des Feuerns für Dienstbeschreibungsnetze ohne *removeRole*-Anweisungen gegeben. Die Aktiviertheit und das Feuern wird dann im Anschluss auf Dienstbeschreibungsnetze mit solchen Anweisungen übertragen.

Dienstbeschreibungsnetze ohne *removeRole*-Anweisungen

Die Aktiviertheit einer Transition in einem Dienstbeschreibungsnetz ist ähnlich der Aktiviertheit einer Transition in einem algebraischen Netz, jedoch muss zudem gewährleistet sein, dass der Systemzustand nicht durch eine andere Transition verändert wird. Aus diesem Grund wird der Systemzustand während des Schaltens einer Transition reserviert.

Sind genügend Marken im Vorbereich einer Transition vorhanden und ist der Systemzustand nicht reserviert, so ist eine Transition in einem Schaltmodus aktiviert und kann feuern. Das Feuern einer Transition reserviert den Systemzustand für die Dauer des Feuerns. Dies bedeutet, dass zwei Transitionen eines Dienstbeschreibungsnetzes nur dann parallel (also gleichzeitig) feuern können, wenn sie unterschiedliche Lokationen verändern, was garantiert, dass stets alle Veränderungen des Zustands durch die Anschriften einer Transition atomar sind und nicht durch andere Transitionen gestört werden. Dies expliziert die Interpretation von $[\![\mathbf{Cmd}(\alpha)]\!]$ als Funktion auf den Zuständen, deren Anwendung auf einen Zustand stets atomar ist. Gibt es zwei parallel schaltende Transitionen, die die gleichen Lokationen verändern und lesen, so könnte es andernfalls in der praktischen Umsetzung zu den aus dem Bereich der Datenbanken bekannten Problemen von *Lost-Updates* oder *Dirty Reads* kommen (vgl. Kemper und Eickler, 1999).

Definition 4.20 (Schaltmodus)
Gegeben ein Dienstbeschreibungsnetz $\mathcal{SDN}$. Ein *Schaltmodus* von $\mathcal{SDN}$ in einem Zustand $\mathfrak{Q}$ ist eine Transition t zusammen mit einer Variablenbelegung $\alpha : (X \setminus \mathit{Bound}(\mathbf{cmd}(t))) \to S_Q$ notiert als Paar (t, α), so dass die Auswertung der Aktivierungsbedingung $[\![G(t)]\!]_\alpha^{\mathfrak{Q}} = \mathit{true}$ ist. ♦

Im Gegensatz zu algebraischen Netzen ist der Schaltmodus abhängig vom Zustand $\mathfrak{Q}$, da dieser die möglichen Variablenbelegungen bestimmt. Die Aktiviertheit und das Feuern ist wie für algebraische Netze definiert, nur dass in diesem Fall der Systemzustand reserviert werden muss. Dies entspricht dem Vorgehen aus Abbildung 4.7, bei dem der Systemzustand stets eine Nebenbedingung darstellt. Zur Vereinfachung des Umstandes wird eine globale boolesche Variable $b_{reserve}$ eingeführt, die angibt, ob der Systemzustand reserviert ist. Die Reservierung ist dabei streng genommen nicht notwendig, wenn man $\mathbf{cmd}(t)$ als Funktion auf den Zuständen betrachtet, da diese stets atomar sind. Dennoch hilft diese Darstellung, die

tatsächliche Realisierung zu explizieren.

Definition 4.21 (Aktiviertheit und Feuern ohne *removeRole*-Anweisung) Gegeben ein Dienstbeschreibungsnetz $\mathcal{SDN}$ ohne *removeRole*-Anweisungen. Ein Schaltmodus (t, α) von $\mathcal{SDN}$ unter einem Zustand $\mathfrak{Q}$ ist *aktiviert* in einem Netzzustand $Q = (\mathbf{m}, \mathfrak{Q})$, notiert als $Q \xrightarrow{t,\alpha}$, wenn

- der Systemzustand nicht reserviert ist ($b_{reserve} = \mathit{false}$ ist),
- für alle Stellen $p \in {}^\bullet t$ gilt, dass $\mathbf{m}(p) \geq \alpha(W(p,t))$.

Ein aktivierter Schaltmodus kann feuern. Das *Feuern* eines aktivierten Schaltmodus (t, α) wird als Übergang von Netzzuständen beschrieben und notiert als $Q \xrightarrow{t,\alpha} Q'$, wobei $Q = (\mathbf{m}, \mathfrak{Q})$ der Ausgangszustand und $Q' = (\mathbf{m}', \mathfrak{Q}')$ der Folgezustand ist. Der Folgezustand ist komponentenweise für einen Netzzustand definiert als Zustandsänderung durch Auswertung des belegungskonsistenten Anweisungsblocks $\mathfrak{Q}' = [\![\mathbf{Cmd}(\alpha)]\!]^{\mathfrak{Q}}$ mit $\mathbf{Cmd} = \mathbf{cmd}(t)$ und als $\mathbf{m}'$, wobei für jede Stelle $p \in P$ gilt

$$\mathbf{m}'(p) = \mathbf{m}(p) - [\alpha(W(p,t))] + [\bar{\alpha}^{\mathfrak{Q}}_{\mathbf{Cmd}}(W(t,p))]$$

wobei $\bar{\alpha}^{\mathfrak{Q}}_{\mathbf{Cmd}}$ die durch **Cmd** erfolgte Erweiterung der Zuweisung darstellt. Während des Feuerns einer Transition t reserviert sie den Systemzustand (es ist $b_{reserve} = \mathit{true}$). ⧫

Da es sich bei $\alpha(W(p,t))$ um Elemente der Algebra und nicht um Multimengen handelt, wird bei der Definition der Folgemarkierung in obiger Definition zunächst durch $[\cdot]$ die einelementige Multimenge erzeugt. In den Netzen werden, wie allgemein üblich, Variablen für die anonyme Marke • weglassen und die Kante stattdessen als unbeschriftet notiert. Ebenso wird die Stellentypisierung für die Sorte $s_{\mathbf{token}}$, deren einziges Element • ist, weggelassen. Bei der Reservierug von Lokationen werden Lokationen, die ausschließlich in Aktivierungsbedingungen auftreten, nicht reserviert, da sich ein ändern einer solchen Lokation nicht auf die Ausführung auswirken würde.

Wie für Petrinetze im Allgemeinen (vgl. Definition 3.32) sind auch für Dienstbeschreibungsnetze die Begriffe der Schaltfolge und der Erreichbarkeit definiert.

Definition 4.22 (Schaltfolge, Erreichbarkeit, Erreichbarkeitsmenge) Sei $\mathcal{SDN}$ eine Dienstbeschreibungsnetz. Eine Sequenz $\sigma_\alpha = t_1^\alpha \dots t_n^\alpha$ wird eine *Schaltfolge* von $\mathcal{SDN}$ genannt und als $Q_0 \xrightarrow{\sigma_\alpha} Q_n$ notiert, wenn $t_j^\alpha = (t_j, \alpha_j)$ und

$$Q_0 \xrightarrow{t_1,\alpha_1} Q_1 \xrightarrow{t_2,\alpha_2} \cdots \xrightarrow{t_n,\alpha_n} Q_n$$

gilt. Durch $Q \xrightarrow{*} Q'$ wird notiert, dass der Netzzustand Q' vom Netzzustand Q aus durch eine beliebige Schaltfolge σ *erreichbar* ist. $[Q\rangle_{\mathcal{SDN}} \stackrel{\text{def}}{=} \{Q' | \exists \sigma_\alpha : Q \xrightarrow{\sigma_\alpha} Q'\}$ definiert die *Menge aller von* Q *aus erreichbaren Netzzustände* in $\mathcal{SDN}$, wobei die Angabe des Netzes entfällt, wenn dieses aus dem Kontext hervorgeht. ⧫

Aufgrund der engen Verwandtschaft zur Schaltfolge in ungefärbten Netzen wird eine Folge von Schaltmodi ebenfalls als Schaltfolge bezeichnet und entsprechend durch σ_α dargestellt. Für die weiter unten diskutierten sicheren Dienstbeschreibungsnetze, die stets nur maximal eine Marke pro Stelle zulassen, determiniert die Schaltfolge der Transitionen sogar die Zuweisungen α_j, so dass die Angabe der Transitionen ausreicht, um die Schaltmodi zu rekonstruieren, da die Variablenbelegung eindeutig ist.

Dienstbeschreibungsnetze mit *removeRole*-Anweisungen

Bei der Nutzung der Konzeptänderungsanweisung *removeRole* lässt sich eine weitere Eigenart von Dienstbeschreibungsnetzen beobachten. Durch das Entfernen einer dynamischen Sorte ist ein Objekt der Algebra nicht mehr länger von dieser dynamischen Sorte. Markiert dieses Objekt jedoch eine Stelle, die durch diese Sorte getypt ist, so würde das Entfernen der Sorte dazu führen, dass die verbleibenden Marken im Netz nicht der Definition einer Markierung entsprechen, da sie die Typisierung verletzen. Das Problem ist in Abbildung 4.9 dargestellt. Nach dem Schalten von t_1 wird das Objekt q_s der Algebra durch die Folgemarkierung $\mathbf{m}$ den Stellen p_1 und p_2 zugewiesen. Schaltet nun die Transition t_3, so ist q_s nach dem Schalten nicht mehr von der Sorte *Student*. Folglich kann eine Marke q_s nicht mehr durch eine Markierung der Stelle p_1 oder der Stelle p_3 zugewiesen sein.

Dieses Verhalten ist zwar unschön, entspricht jedoch dem in praktischen Fällen notwendigen Verhalten, da ein ehemaliger Student nicht mehr die gleichen Rechte hat wie ein Student und somit nicht mehr so behandelt werden sollte wie ein Student. In den meisten praktischen Fällen würden die Stellen durch die Sorte Person getypt werden und dann durch die Transition abgefragt werden, ob die Marke auch die Sorte *Student* besitzt. Hat die Marke diese Sorte verloren, so kann eine andere Art der Bearbeitung erfolgen.

Zur Lösung dieser Problematik existieren prinzipiell drei Herangehensweisen. Als erste Möglichkeit kann das Schalten der Transition, die die Sorte entfernt, unterbunden werden, sofern diese Sorte noch verwendet wird. Als weitere Möglichkeit können die Marken auf den entsprechenden Stellen durch das Objekt $\perp\!\!\!\perp$ ersetzt werden, um darzustellen, dass diese Marke nicht mehr länger definiert ist. Schließlich existiert noch die Möglichkeit, den Systemzustand als ganzen als undefiniert zu markieren. Die zweite Möglichkeit entspricht dabei am wenigsten der Intuition, da sich hierdurch nicht nur die Sorte, sondern auch das Objekt selbst ändern würde. Vielmehr entspricht es dem in der Praxis sinnvollen Vorgehen, dass ein Objekt ein

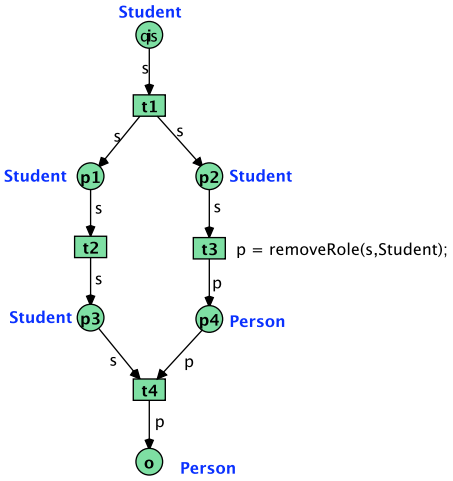


Abbildung 4.9: Problematik beim Entfernen einer dynamischen Sorte

dynamisches Konzept nicht verlieren darf, sofern die Beendigung eines Prozesses davon abhängig ist, dass das Objekt dieses Konzept besitzt. Verliert es das Konzept trotzdem, so stellt dies einen Fehler dar.

Den Systemzustand als ganzen als undefiniert zu markieren, bedeutet, dass es aufwendiger wird, wieder in einen gültigen Zustand zu gelangen. Zudem würde dies bedeuten, dass alle Anweisungen eines Anweisungsblock ausgeführt werden können, auch wenn hierbei ein ungültiger Zustand erreicht wird. Am ehesten der Intuition entspricht, dass ein Netz nicht schalten kann, wenn eine Rolle entfernt werden soll, die anderweitig benutzt wird.

Damit die Marken im Netz stets der Definition einer Markierung genügen, bleibt folglich die Möglichkeit, dass eine Transition im Zustand $\mathfrak{Q}$ unter der Variablenbelegung α nicht schalten kann, wenn sie eine *removeRole*$(\tau, s_\mathfrak{c})$-Anweisung aufweist und im Netz eine Stelle der Sorte $s_\mathfrak{c}$ mit $[\![\tau]\!]_\alpha^\mathfrak{Q}$ markiert ist. Dieser Problematik wird nun dadurch begegnet, dass die Sorten von Objekten der Algebra reserviert werden. Die Reservierung erfolgt über Sortenlokationen, die einem Objekt der Algebra $q \in S_Q$ zusammen mit einer Sorte $s \in S^{dyn}$ entsprechen. Ist ein $q \in S_Q$ auf einer Stelle p des Typs s, so ist (q, s) durch die Stelle p reserviert. Eine Reservierung kann beliebig häufig erfolgen, jedoch kann auf einer reservierten Lokation niemals gleichzeitig eine *removeRole*-Anweisung ausgeführt werden. Damit eine Transition mit einer *removeRole*-Anweisung schalten kann, müssen die Sorten aller *removeRole*-Anweisungen reservierbar sein. Dies verhindert, das genutzte Sorten entfernt werden. Um dies zu formalisieren, werden zunächst Sortenlokationen und ihre Reservierung definiert.

Definition 4.23 (Sortenlokation, Reservierung)
Gegeben ein Zustand $\mathfrak{Q}$ zu einer Zustandssignatur $\Sigma^{\mathbf{Q}}$. Eine Sortenlokation ist ein Paar $(q, s) \in S_Q \times S^{dyn}$ mit $q \in Q_s$. Die Menge der *reservierten Sortenlokationen* eines Netzzustandes $Q = (\mathbf{m}, \mathfrak{Q})$ für eine Transition $t \in T$ ist gegeben durch die Menge

$$\mathbf{Loc}^{\mathbf{I}}_{sort}(Q, t) \stackrel{\text{def}}{=} \{(q, s_{\mathfrak{c}}) \in S_Q \times S^{dyn} \mid q \in \mathbf{m}(p) \text{ und } d(p) = s_{\mathfrak{c}} \text{ für ein } p \in P \setminus {}^\bullet t\}$$

♦

Die Menge der veränderten Sortenlokationen $\mathbf{Loc}^{\mathbf{U}}_{sort}(Q, (t, \alpha))$ eines Schaltmodus (t, α) ist gegeben durch sämtliche Sortenlokationen, die entfernt worden sind: $\mathbf{Loc}^{\mathbf{U}}_{sort}(Q, (t, \alpha)) \stackrel{\text{def}}{=} \{(q, s'_{\mathfrak{c}}) \in S_Q \times S^{dyn} \mid (removeRole(\tau, s_{\mathfrak{c}})) \in \textsc{Alph}(\mathbf{cmd}(t)) \wedge q = [\![\tau]\!]^{\mathfrak{Q}}_{\alpha} \wedge s'_{\mathfrak{c}} \preceq s_{\mathfrak{c}}\}$. Offensichtlich sind bei diesen Betrachtungen nur dynamische Sorten relevant, da die anderen Sorten nicht entfernt werden können.

Die Reservierung erfolgt durch Veränderung eines Operators $\mathbf{r}^p_{sort}$. Dabei ist in jedem Netzzustand $(\mathbf{m}, \mathfrak{Q})$ dann $\mathbf{r}^p_{sort}(q, s_{\mathfrak{c}}) = \sum_{p \in P \wedge d(p) = s_{\mathfrak{c}}} \mathbf{m}(p)(q)$, wobei $\mathbf{m}(p)(q)$ die Multiplizität des Elements q in der Multimenge $\mathbf{m}(p)$ darstellt. Dies bedeutet, dass $\mathbf{r}^p_{sort}(q, s_{\mathfrak{c}})$ um 1 inkrementiert wird, bevor p markiert wird, und um 1 dekrementiert wird, wenn die Marke q von p entfernt wird.

Damit Stellen kein aktives Verhalten aufweisen, was der Intuition widerspräche, lassen sich die Veränderungen der Interpretation von $\mathbf{r}^p_{sort}$ durch Transitionen durchführen. Hierbei wird der Wert von $\mathbf{r}^p_{sort}(q, s_{\mathfrak{c}})$ inkrementiert, wenn die Transition eine Stelle vom Typ $s_{\mathfrak{c}}$ im Nachbereich besitzt, die durch q markiert wird. Das Schalten einer Transition ist dann eine Erweiterung von Definition 4.21.

Definition 4.24 (Aktiviertheit und Feuern mit *removeRole*-Anweisungen)
Gegeben ein Dienstbeschreibungsnetz $\mathcal{SDN}$. Ein Schaltmodus (t, α) von $\mathcal{SDN}$ unter einem Zustand $\mathfrak{Q}$ ist *aktiviert* in einem Netzzustand $Q = (\mathbf{m}, \mathfrak{Q})$, notiert als $Q \xrightarrow{t,\alpha}$, wenn

- der Systemzustand nicht reserviert ist ($b_{reserve} = \mathit{false}$ ist),
- für alle Stellen $p \in {}^\bullet t$ gilt, dass $\mathbf{m}(p) \geq \alpha(W(p, t))$ und
- $\mathbf{Loc}^{\mathbf{I}}_{sort}(Q, t) \cap \mathbf{Loc}^{\mathbf{U}}_{sort}(Q, (t, \alpha)) \neq \emptyset$ gilt, also keine der Sortenlokationen aus $\mathbf{Loc}^{\mathbf{U}}_{sort}(Q, (t, \alpha))$ reserviert ist.

Ein aktivierter Schaltmodus kann wie in Definition 4.21 definiert feuern. Mit dem Feuern werden sämtliche Sortenlokationen durch die Stellen reserviert, so dass $\mathbf{r}^p_{sort}(q, s_{\mathfrak{c}}) = \sum_{p \in P \wedge d(p) = s_{\mathfrak{c}}} \mathbf{m}(p)(q)$ gilt. ♦

Besitzt ein Dienstbeschreibungsnetz keine *removeRole*-Anweisungen, so ist für alle Transitionen t und alle Variablenbelegungen α entsprechend $\mathbf{Loc}^{\mathbf{U}}_{sort}(Q, (t, \alpha)) =$

$\emptyset$. In diesem Fall reduziert sich die Aktiviertheit aus Definition 4.24 auf die Aktiviertheit aus Definition 4.21.

Dienstbeschreibungsnetze besitzen eine recht große Nähe zu algebraischen Netzen. Dies wird im folgenden Unterabschnitt dargestellt. Hieraus folgt auch direkt, dass die zentralen Fragen wie die Erreichbarkeit einer Markierung und die Lebendigkeit eines Netzsystems für Dienstbeschreibungsnetze im Allgemeinen nicht entscheidbar sind.

4.3.3 Einordnung von Dienstbeschreibungsnetzen

Nachdem in den letzten Unterabschnitten Dienstbeschreibungsnetze definiert worden sind, ordnet dieser Unterabschnitt diese Netzklasse ein und stellt ihre Verwandtschaft zu algebraischen Netzen dar. Zunächst wird gezeigt, dass sich jedes algebraische Netz, dessen zugrundeliegendes Netz ein Workflownetz ist, durch ein Dienstbeschreibungsnetz simulieren lässt, indem sämtliche Anweisungsblöcke lediglich aus Zuweisungsanweisungen bestehen. Hieraus folgt direkt, dass sämtliche wichtigen Fragestellungen für Dienstbeschreibungsnetze unentscheidbar sind. Dennoch lässt sich eine Analyse der Dienstbeschreibungsnetze erreichen, wenn man das zugrundeliegende Workflownetz betrachtet. Dies wird im Anschluss diskutiert.

Dienstbeschreibungsnetze, algebraische Netze und Workflownetze

Die in Definition 3.45 definierten algebraischen Netze bieten eine recht starke Ähnlichkeit zu Dienstbeschreibungsnetzen. Daher soll an dieser Stelle gezeigt werden, dass jedes algebraische Netz, dessen zugrundeliegendes Netz ein Workflownetz ist, durch ein Dienstbeschreibungsnetz simuliert werden kann. Da das zugrundeliegende Netz ein Workflownetz ist, soll es nicht gestattet sein, Kantengewichte zu verwenden. Dies bedeutet, dass die Terme im algebraischen Netz keine Multimengen darstellen.

Sei also $\mathcal{APN}$ ein algebraisches Netz, dessen zugrundeliegendes Netz ein Workflownetz ist, und sei $\mathcal{SDN}$ ein Dienstbeschreibungsnetz, das $\mathcal{APN}$ simuliert. Die Zustandssignatur $\Sigma^{\mathbf{Q}}$ des Dienstbeschreibungsnetzes (vgl. Def. 4.3) besteht in diesem Fall lediglich aus der statischen Signatur $\Sigma^{stat} = (S^{stat}, \preceq^{stat}, \Omega^{stat})$, deren Ordnung $\preceq^{stat}$ gleich $\emptyset$ ist und die ansonsten der Spezifikation $\Gamma = (S, \Omega, E)$ des algebraischen Netzes entspricht. Die statische Algebra $\mathcal{A}^{stat} = (S_A, \Omega_A)$ ist die Quotiententermalgebra der Spezifikation Γ, so dass sämtliche Gleichungen der Spezifikation des algebraischen Netzes in dieser Algebra ebenfalls vorhanden sind. Die Menge der Variablen des Dienstbeschreibungsnetzes enthält alle Variablen des algebraischen Netzes. Das Netz $\mathcal{N}$ und die Stellentypisierung d können direkt übernommen werden. Somit bleiben noch die Kantenanschriften und die Aktivierungsbedingungen zu überführen.

Sei $t \in T$ eine Transition des algebraischen Netzes $\mathcal{APN}$. Jede Kantenanschrift, die eine Variable ist, lässt sich direkt übernehmen. Sind dabei die Kantenanschriften zweier Eingangsvariablen gleich, so wird einer der Kanten eine andere Variable, die noch ungenutzt ist, zugewiesen, und es werden beide Variablen über die Aktivierungsbedingung gleich gesetzt. Stellen alle Kantenanschriften der eingehenden Kanten von t lediglich Variablen dar, so lässt sich die weitere Aktivierungsbedingung dann direkt übernehmen. Da das zugrundeliegende Netz von $\mathcal{APN}$ ein (ungewichtetes) Workflownetz sein soll, ist sichergestellt, dass keine der Kantenanschriften mehr als eine Marke von einer Stelle abzieht oder zu einer Stelle hinzufügt. Somit stellen die Terme an ausgehenden Kanten keine Multimengen dar, und es kann den entsprechenden Kanten im Dienstbeschreibungsnetz eine Variable zugewiesen werden, die dann durch eine Zuweisungsanweisung mit der Auswertung des entsprechenden Terms gleich gesetzt wird.

Gibt es einen Term an einer eingehenden Kante, so muss ebenfalls über die Aktivierungsbedingung gewährleistet sein, dass der Term gleich einer Variable ist, die dann an dieser Eingangskante genutzt wird. Ist die Kantenanschrift etwa τ, so wird die neue Kantenanschrift v für eine ungenutzte Variable v und die Aktivierungsbedingung erhält zudem die Bedingung, dass auch $v = \tau$ gelten muss.

Durch dieses Vorgehen lässt sich jedes algebraische Petrinetz, dessen zugrundeliegendes Netz ein Workflownetz ist, in ein Dienstbeschreibungsnetz überführen. Andersherum lässt sich jedes Dienstbeschreibungsnetz ohne Update- und Konzeptänderungsanweisungen in ein algebraisches Netz überführen, indem die gebundenen Variablen der Zuweisungsanweisungen durch die Terme auf der rechten Seite der Zuweisung ersetzt werden.

> *Ein Dienstbeschreibungsnetz ohne Update- und Konzeptänderungsanweisungen lässt sich in ein äquivalentes algebraisches Petrinetz überführen, dessen zugrundeliegendes Netz ein Workflownetz ist, und umgekehrt.*

Vergegenwärtigt man sich nun noch einmal das Bild aus Abbildung 4.7, in dem auch der Zustand als Objekt der Algebra aufgefasst wurde, dann lässt sich das Dienstbeschreibungsnetz auch als ganzes in ein algebraisches Netz überführen. Hierbei stellt der Zustand ein entsprechend komplexes Element der Algebra dar, auf dem die Operatoren der Konzeptänderungs- und Update-Anweisungen definiert sind. Da diese Anweisungen jeweils Abbildungen auf der Klasse der Zustände sind, stellen sie Operatoren auf der Sorte der Zustände dar.

Ein Dienstbeschreibungsnetz wird *trivial* genannt, wenn $s_{\mathbf{token}}$ und *bool* die einzigen Sorten der Algebra darstellen, *bool* lediglich für die Aktivierungsbedingungen genutzt wird und $G(t) = \mathit{true}$ für alle Transitionen $t \in T$ gilt, $\bullet$, *true*, *false* die einzigen Operatoren der Signatur sind mit $\{\bullet\} = Q_{\epsilon,\bullet}$ und alle Variablen und Stellen die Sorte $s_{\mathbf{token}}$ besitzen.

Da alle Stellen und Variablen durch $s_{\mathbf{token}}$ getypt sind, sind sämtliche Anweisungsblöcke leer, da es aufgrund der fehlenden Operatoren auch keine Terme geben kann. Ebenso sind die Aktivierungsbedingungen alle stets wahr. Somit gibt es in einem solchen Netz nur anonyme Marken und das resultierende Netz ist ein Workflownetz. Dies wird in folgendem Lemma festgehalten.

Lemma 4.25 (Triviale Dienstbeschreibungsnetze sind Workflownetze)
Ein triviales Dienstbeschreibungsnetz ist äquivalent zu seinem zugrundeliegenden Workflownetz.

Dies bettet den Formalismus der Dienstbeschreibungsnetze in den bestehenden Formalismus der Workflownetze ein.

Entscheidbarkeit bei Dienstbeschreibungsnetzen

Aus den vorangegangenen Überlegungen dieses Abschnitts folgt, dass zentrale Fragen wie die Erreichbarkeit einer Markierung oder die Lebendigkeit eines Netzes unter einer Markierung für Dienstbeschreibungsnetze im Allgemeinen unentscheidbar sein müssen, da sie wie algebraische Netze bei hinreichend ausdrucksmächtiger Algebra Turing-mächtig sind. Reduziert man hingegen die Algebra auf die Sorte $s_{\mathbf{token}}$ mit der anonymen Marke $\bullet$ als einzigem Operatorsymbol (nullstellig), so ist das resultierende Netz ein Workflownetz (vgl. Lemma 4.25), für das diese Fragen entscheidbar sind (siehe Mayr (1984) und Esparza (1998a) für Petrinetze und van Hee u. a. (2004) speziell für Workflownetze). Es führt jedoch bereits die Verwendung einfacher Datentypen wie die Verwendung natürlicher Zahlen oder Zeichenketten zu Unentscheidbarkeiten bei diesen Problemen, was an den durch die Aktivierungsbedingungen ermöglichten Nulltests liegt, wodurch sich Zählerautomaten simulieren lassen. Zählerautomaten sind ein relativ einfaches Modell, in dem die Fragen der Erreichbarkeit und der Beschränktheit unentscheidbar sind, weswegen sie gerne zur Analyse von Petrinetzformalismen herangezogen werden, wie dies etwa bei Valk (1978) erfolgt. In Kummer (2002) werden sie zur Analyse von Objektnetzen verwendet. Durch die Verwendung einer Sorte *nat*, die die natürlichen Zahlen umfasst zusammen mit den Operatoren $+$ und $-$ und einem Gleichheitsoperator ist es stets möglich, einen Zählerautomaten zu simulieren, so dass sich Folgendes festhalten lässt. (Für Details zur Anwendung der Methode sei auf Kummer (2002), Kapitel 4 verwiesen.)

Lemma 4.26 (Unentscheidbarkeit von Erreichbarkeit und Lebendigkeit)
Die Erreichbarkeit einer Markierung in einem Dienstbeschreibungsnetz, die Beschränktheit und die Lebendigkeit eines Dienstbeschreibungsnetzes in einem initialen Netzzustand sind im Allgemeinen unentscheidbar.

Die Beschränktheit lässt sich dabei auf die Erreichbarkeit zurückführen, so dass ihre Unentscheidbarkeit aus der Unentscheidbarkeit der Erreichbarkeit folgt. Da an den Kanten lediglich Variablen erlaubt sind, ist ein Dienstbeschreibungsnetz beschränkt, wenn das zugrundeliegende Workflownetz beschränkt ist. Somit gibt es ein entscheidbares hinreichendes Kriterium für die Beschränktheit. Dies wird im Folgenden dargestellt.

Beschränktheit und Schaltverhalten des zugrundeliegenden Netzes

Dadurch, dass jede Kante lediglich durch eine Variable beschriftet ist, ist das Verhalten eines Dienstbeschreibungsnetzes eng verwandt mit dem Verhalten des zugrundeliegende Workflownetzes. Die Anzahl der Marken auf einer Stelle in diesem Workflownetz ist stets gleich der Anzahl der Marken im ursprünglichen Dienstbeschreibungsnetz. Die anonymisierte Markierung bildet die Markierung eines Dienstbeschreibungsnetzes auf die anonyme Markierung des zugrundeliegenden Workflownetzes ab.

Definition 4.27 (Anonymisierte Markierung)
Gegeben ein Dienstbeschreibungsnetz $\mathcal{SDN}$ und ein Netzzustand $Q = (\mathbf{m}, \mathfrak{Q})$. Die *anonymisierte Markierung* $\mathring{\mathbf{m}}$ von $\mathbf{m}$ ist für alle $p \in P$ gegeben durch $\mathring{\mathbf{m}}\ (p) = |\mathbf{m}(p)|$, wobei $|\mathbf{m}(p)|$ die Anzahl der Elemente in $\mathbf{m}(p)$ ist. ♦

Dass eine Transition in der anonymisierten Markierung im zugrundeliegenden Netz schalten kann, ist nun notwendige Bedingung für das Schalten einer Transition im Dienstbeschreibungsnetz, wie folgendes Lemma zeigt.

Lemma 4.28 (Feuern des zugrundeliegenden Netzes)
Gegeben ein Dienstbeschreibungsnetz $\mathcal{SDN}$ und ein Netzzustand $Q_1 = (\mathbf{m}_1, \mathfrak{Q}_1)$. Für eine Transition $t \in T$ und eine Variablenbelegung α gilt nun mit $Q_2 = (\mathbf{m}_2, \mathfrak{Q}_2)$ und $\mathring{\mathbf{m}}\ (p) = |\mathbf{m}(p)|$ für alle $p \in P$

$$Q_1 \xrightarrow[\mathcal{SDN}]{t,\alpha} Q_2 \quad \text{impliziert} \quad \mathring{\mathbf{m}}_1 \xrightarrow[\mathcal{N}]{t} \mathring{\mathbf{m}}_2$$

Beweis:
Sei $Q = (\mathbf{m}, \mathfrak{Q})$ ein Netzzustand, so weist die Markierung $\mathbf{m}$ jedem $p \in P$ eine Multimenge über den Elementen von S_Q zu. Da jede Kante des Dienstbeschreibungsnetzes $\mathcal{SDN}$ nur genau ein Element dieser Multimenge abzieht (da sie nur einfache Variablen als Anschrift besitzen kann) ist die Kardinalität der Multimenge nach dem Feuern gleich ihrer Kardinalität vor dem Feuern abzüglich der Anzahl der Kanten zur feuernden Transition. Analoges gilt für die Ausgangskanten der Tran-

sition. Dies entspricht genau dem Verhalten in $\mathcal{N}$ und es gilt $Q \xrightarrow[\mathcal{SDN}]{t,\alpha} Q'$ impliziert $\overset{\circ}{\mathbf{m}}_1 \xrightarrow[\mathcal{N}]{t} \overset{\circ}{\mathbf{m}}_2$. □

Korollar 4.29 (Notwendige Bedingung für die Erreichbarkeit)
Gegeben ein Dienstbeschreibungsnetz $\mathcal{SDN}$ mit einem initialen Netzzustand $Q_0 = (\mathbf{m}_0, \mathfrak{Q}_0)$. Ist $(\mathbf{m}_1, \mathfrak{Q}_1) \in [Q_0\rangle_{\mathcal{SDN}}$, so ist $\overset{\circ}{\mathbf{m}}_1 \in [\overset{\circ}{\mathbf{m}}_0\rangle_{\mathcal{N}}$.

Beweis:
Ist $(\mathbf{m}_1, \mathfrak{Q}_1) \in [Q_0\rangle_{\mathcal{SDN}}$, so gibt es eine Folge von Schaltmodi, die vom Netzzustand Q_0 zum Netzzustand $(\mathbf{m}_1, \mathfrak{Q}_1)$ führt. Dann existiert diese Schaltfolge auch in $\mathcal{N}$ und nach Lemma 4.28 gilt die Aussage. □

Andersherum gilt die Aussage offensichtlich nicht, da durch die Aktivierungsbedingung von t das Schalten unterbunden werden kann. Anders formuliert bedeutet dies, dass ein Dienstbeschreibungsnetz lediglich schalten kann, wenn das zugrundeliegende Workflownetz ebenfalls schalten kann, und dass die Zahl der Marken auf den Plätzen dabei in beiden Netzen gleich ist. Dies ist bei algebraischen Netzen ebenfalls der Fall. Zudem hat jede Schaltfolge im zugrundeliegenden Workflownetz eine Entsprechung im Dienstbeschreibungsnetz, die ebenfalls eine mögliche Schaltfolge darstellt, sofern die Aktivierungsbedingungen dies zulassen. Dies ist für algebraische Netze nicht mehr notwendigerweise der Fall, da es dort sein kann, dass Variablen unifiziert werden, da eine Variable an mehreren Eingangskanten verwendet wird.

Dies bedeutet auch, dass eine Transition ohne Aktivierungsbedingung[5] und ohne eine *removeRole*-Anweisung in einem Dienstbeschreibungsnetz stets schalten kann, wenn genügend Marken im Vorbereich vorhanden sind. Für die Simulation ist dies interessant, da lediglich überprüft werden muss, welche Transitionen im Workflownetz aktiviert sind, um dann die Aktivierungsbedingungen zu überprüfen, und so die aktivierten Transitionen des Dienstbeschreibungsnetzes herauszufinden. Insbesondere in Abgrenzung zu algebraischen Netzen gibt es einen stärkeren Bezug zwischen dem Dienstbeschreibungsnetz und dem zugrundeliegenden Workflownetz, was genutzt werden kann, um etwa Verklemmungsfreiheit oder Lebendigkeit nachzuweisen, da Dienstbeschreibungsnetze es nicht erlauben, gleiche Variablen an verschiedenen Eingangskanten einer Transition zu verwenden. Das Verhalten, das hierbei nicht auftreten kann, das jedoch in algebraischen Netzen möglich ist, ist in Abbildung 4.10 visualisiert, wobei x ein Variablensymbol und a, b Terme darstellen. Die dargestellten Netze stellen jeweils nur schematisch das Problem anhand eines algebraischen Netzes dar und weisen daher keine Workflowstruktur auf.

[5] Es wird im Folgenden wie allgemein üblich die Aktivierungsbedingung einer Transition t weggelassen, was dann $G(t) = \mathit{true}$ entspricht.

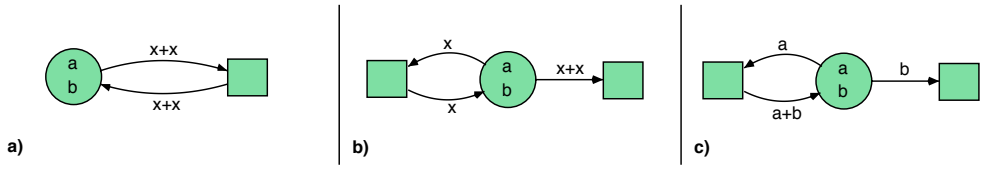


Abbildung 4.10: Unterschiedliche Eigenschaften eines algebraischen Netzes und seines zugrundeliegenden Netzes

Abbildung 4.10(a) zeigt einen Fall, in dem das zugrundeliegende Petrinetz lebendig ist, das algebraische Netz hingegen verklemmt. Dies ist für Dienstbeschreibungsnetze nicht möglich, da eine Variable lediglich einmal an einer Kante stehen darf. In Abbildung 4.10(b) führt das Schalten der rechten Transition im Petrinetz zur leeren Markierung, während diese Transition im algebraischen Netz unter dieser initialen Markierung niemals schalten kann. Abbildung 4.10(c) zeigt schließlich ein algebraisches Netz, welches lebendig ist unter dieser initialen Markierung, während es im Petrinetz eine Schaltfolge gibt, die zur leeren Markierung führen würde. Keiner dieser drei Fälle kann im Rahmen von Dienstbeschreibungsnetzen auftreten, da hier lediglich Variablen an den Kanten einer Transition erlaubt sind, die bei Eingangskanten zudem verschieden sein müssen.

Ähnlich geartete Fälle finden sich jedoch auch in Dienstbeschreibungsnetzen, da eine Transition aufgrund der Aktivierungsbedingungen niemals aktiviert sein kann, während sie im zugrundeliegenden Petrinetz schalten kann. Zwei mögliche Fälle sind in Abbildung 4.11 dargestellt. Abbildung 4.11(a) zeigt einen Ausschnitt eines Dienstbeschreibungsnetzes, der verklemmen würde, während das entsprechende zugrundeliegende Petrinetz lebendig ist. Abbildung 4.11(b) zeigt umgekehrt ein Dienstbeschreibungsnetz, das nicht verklemmt, während das zugrundeliegende Workflownetz dies tut.

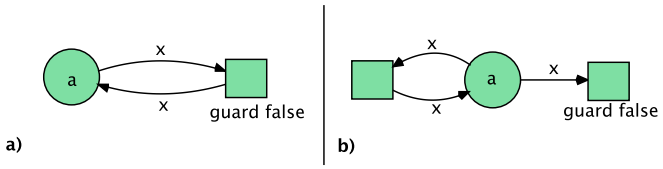


Abbildung 4.11: Unterschiedliche Eigenschaften eines Dienstbeschreibungsnetzes und seines Workflownetzes

Der wesentliche Unterschied zwischen Dienstbeschreibungsnetzen und algebraischen Netzen ist hierbei, dass ein unterschiedliches Schaltverhalten nur durch Aktivierungsbedingungen oder *removeRole*-Anweisungen auftritt. Gleichzeitig folgt aus

Lemma 4.28, dass ein Dienstbeschreibungsnetz beschränkt ist, wenn sein zugrundeliegendes Workflownetz beschränkt ist. Beides soll hier noch einmal festgehalten werden.

Definition 4.30 (Beschränktheit von Dienstbeschreibungsnetzen)
Ein Dienstbeschreibungsnetz $\mathcal{SDN}$ ist in einem Netzzustand $Q = (\mathbf{m}, \mathfrak{Q})$ k-beschränkt gdw. für alle $(\mathbf{m}', \mathfrak{Q}') \in [Q\rangle_{\mathcal{SDN}}$ gilt $\overset{\circ}{\mathbf{m}} < k$ für ein $k \in \mathbb{N}$. ⧫

Beschränktheit bezeichnet also analog zu ungefärbten Petrinetzen, dass stets weniger als k Marken auf einer Stelle vorhanden sind. Eine hinreichende Bedingung für die Beschränktheit ist die Beschränktheit des zugrundeliegenden Netzes.

Lemma 4.31 (Beschränktheitsbedingung)
Ein Dienstbeschreibungsnetz $\mathcal{SDN}$ ist in einem Netzzustand $Q = (\mathbf{m}, \mathfrak{Q})$ k-beschränkt, wenn für das zugrundeliegende Netz $\mathcal{N}$ entsprechend $(\mathcal{N}, \overset{\circ}{\mathbf{m}})$ k-beschränkt ist.

Beweis:
Sei $(\mathcal{N}, \overset{\circ}{\mathbf{m}})$ beschränkt. Nach Korollar 4.29 ist ein Netzzustand $(\mathbf{m}', \mathfrak{Q}') \in [Q\rangle_{\mathcal{SDN}}$, wenn $\overset{\circ}{\mathbf{m}'} \in [\overset{\circ}{\mathbf{m}}\rangle_{\mathcal{N}}$. Da $\overset{\circ}{\mathbf{m}'}$ die Schranke k besitzt, muss dies auch für $(\mathbf{m}', \mathfrak{Q}') \in [Q\rangle_{\mathcal{SDN}}$ gelten. □

Eine spezielle Klasse der k-beschränkten Netze sind die sicheren Netze, die auch als 1-beschränkt bezeichnet werden.

Definition 4.32 (Sichere Dienstbeschreibungsnetze)
Ein Dienstbeschreibungsnetz $\mathcal{SDN}$ ist in einem Netzzustand $Q_0 = (\mathbf{m}_0, \mathfrak{Q}_0)$ sicher gdw. es 1-beschränkt ist. ⧫

Korollar 4.33 (Sicherheit von Dienstbeschreibungsnetzen)
Ein Dienstbeschreibungsnetz $\mathcal{SDN}$ ist in einem Netzzustand $Q = (\mathbf{m}, \mathfrak{Q})$ sicher, wenn $\mathcal{N}$ unter $\overset{\circ}{\mathbf{m}}$ sicher ist mit $\overset{\circ}{\mathbf{m}}(p) = |\mathbf{m}(p)|$ für alle $p \in P$.

Beweis:
Die Aussage folgt direkt aus Lemma 4.31. □

Jede Transition, deren Aktivierungsbedingung stets wahr ist und deren Anweisungsblock keine *removeRole*-Anweisung besitzt, ist aktiviert in einem Netzzustand $(\mathbf{m}, \mathfrak{Q})$, wenn ihre Entsprechung im zugrundeliegenden Netz in der Markierung $\overset{\circ}{\mathbf{m}}$ aktiviert ist.

Lemma 4.34 (Aktiviertheit von Transitionen)
Sei $\mathcal{SDN}$ ein Dienstbeschreibungsnetz. Eine Transition $t \in T$, für die $G(t) = \mathit{true}$ gilt und die keine *removeRole*-Anweisung besitzt, ist aktiviert in einem Netzzustand $(\mathbf{m}, \mathfrak{Q})$ gdw. t in $\mathcal{N}$ in der Markierung $\mathring{\mathbf{m}}$ aktiviert ist und der Systemzustand nicht reserviert ist.

Beweis:
($\Rightarrow$) Sei $t \in T$ in $(\mathbf{m}, \mathfrak{Q})$ aktiviert. Nach Lemma 4.28 impliziert dies direkt, dass t in $\mathring{\mathbf{m}}$ in $\mathcal{N}$ aktiviert ist.
($\Leftarrow$) Sei t in $\mathcal{N}$ in der Markierung $\mathring{\mathbf{m}}$ aktiviert. Dann sind sämtliche Stellen im Vorbereich von t auch in $\mathcal{SDN}$ markiert, und es gibt eine Variablenbelegung α, die jeder Variable einer Eingangskante ein Objekt der Algebra zuweist, so dass $\mathbf{m}(p) \geq \alpha(W(p,t))$ für alle $p \in {}^\bullet t$. Nach Definition 4.21 ist t dann aktiviert, wenn der Systemzustand nicht reserviert ist, da stets $[\![G(t)]\!]_\alpha^{\mathfrak{Q}} = \mathit{true}$ gilt. □

Gibt es in einem Dienstbeschreibungsnetz keine Transitionen mit Aktivierungsbedingungen oder *removeRole*-Anweisungen, so ist eine Schaltfolge in einem Netzzustand $(\mathbf{m}, \mathfrak{Q})$ aktiviert, wenn ihre Entsprechung im zugrundeliegenden Netz in der Markierung $\mathring{\mathbf{m}}$ aktiviert ist.

Korollar 4.35 (Aktiviertheit von Schaltfolgen)
Sei $\mathcal{SDN}$ ein Dienstbeschreibungsnetz, bei dem für jede Transition $t \in T$ gilt $G(t) = \mathit{true}$ und bei dem keine Transition eine *removeRole*-Anweisung besitzt. Eine Schaltfolge σ_α von $\mathcal{SDN}$ ist in einem Netzzustand $(\mathbf{m}, \mathfrak{Q})$ aktiviert gdw. die Schaltfolge σ in der Markierung $\mathring{\mathbf{m}}$ in $\mathcal{N}$ aktiviert ist.

Beweis:
Dies folgt direkt aus Lemma 4.34, da jede Transition der Schaltfolge aktivierbar ist. □

Für sichere Dienstbeschreibungsnetze ist die Angabe einer Folge von Schaltmodi σ_α gleichbedeutend mit der Angabe einer Schaltfolge σ, da sich auf jeder Stelle im Vorbereich einer Transition ohnehin nur maximal ein Objekt befindet. Folglich ist die Zuweisung α durch die Eingangsvariablen der Transitionen bestimmt.

Da jede Schaltfolge des zugrundeliegenden Workflownetzes eines Dienstbeschreibungsnetzes nur durch Aktivierungsbedingungen eingeschränkt wird, sollte es folglich einen Netzzustand und eine Schaltfolge geben, die die entsprechenden Aktivierungsbedingungen im Dienstbeschreibungsnetz wahr werden lässt und somit die Transitionen aktiviert. Andernfalls würde diese Folge von Transitionen niemals aktiviert werden, was die Korrektheit des Modells in Frage stellt. Verschiedene Korrektheitskriterien werden daher im nächsten Unterabschnitt untersucht.

4.4 Eigenschaften von Dienstbeschreibungsnetzen

Nachdem im letzten Abschnitt das Verhältnis von Dienstbeschreibungsnetzen zu algebraischen Netzen und zu den ihnen zugrundeliegenden Workflownetzen betrachtet wurde, widmet sich dieser Abschnitt den Eigenschaften von Dienstbeschreibungsnetzen, die in erster Linie für eine Analyse von Interesse sind. Von den Workflownetzen wird hierzu die Korrektheitseigenschaft auf Dienstbeschreibungsnetze übertragen, die jedoch ausdifferenziert wird, um den Anforderungen an gefärbte Netze Rechnung zu tragen.

Anders als bei Workflownetzen muss bei der Betrachtung von Funktionalität unter der Berücksichtigung von Daten auch das Verhalten nebenläufig schaltendender Transitionen betrachtet werden. Eine wünschenswerte Eigenschaft von Dienstbeschreibungsnetzen ist dabei, dass für zwei nebenläufig aktivierte Transitionen unabhängig von der Reihenfolge ihres Schaltens stets derselbe Zustandsübergang erfolgt. Da die Reihenfolge für nebenläufige Transitionen in der Praxis zufällig ist, würde andernfalls der Zufall über das Ergebnis einer Workflow-Ausführung entscheiden, was in der Regel nicht wünschenswert ist. Dies wird im ersten Unterabschnitt diskutiert.

Hieran anschließend erfolgt die Betrachtung von Netzen, in denen die *removeRole*-Anweisungen nicht zu blockierenden Markierungen führen können. In solchen Netzen kann die Schaltregel für einfache Dienstbeschreibungsnetze auch für Dienstbeschreibungsnetze mit *removeRole*-Anweisungen verwendet werden.

Die Betrachtung verschiedener Korrektheitskriterien, die sich an den Kriterien für Workflownetze orientieren, die diese aber für Netze mit Dienstbeschreibungsnetze mit Daten anpassen, schließt die Betrachtung von Kriterien sinnvoller Modellierung ab. Abschließend erfolgt in diesem Abschnitt noch eine kurze Diskussion zur praktischen Umsetzung von Dienstbeschreibungsnetzen.

4.4.1 Nebenläufige Ausführung

Durch die nebenläufige Ausführung von Diensten können eine Reihe von unerwünschten Effekten auftreten. Hierzu zählen einmal Race-Conditions, bei denen zwei nebenläufig aktivierte Transitionen je nach Reihenfolge ihres Schaltens unterschiedliche Ergebnisse liefern, und hierzu zählen Verklemmungen, die dadurch entstehen, dass sich die Aktivierungsbedingungen zweier Transitionen gegenseitig blockieren. Dienstbeschreibungsnetze, die keine Transitionen mit solchen Sonderfällen aufweisen, werden als funktional ausführbar bezeichnet und im Folgenden eingeführt.

Probleme der nebenläufigen Ausführung

Die Eigenschaft, zwei Transitionen in beliebiger Reihenfolge schalten lassen zu können, ohne dass das Ergebnis von der Reihenfolge des Schaltens abhängig ist, wird als funktionale Ausführbarkeit bezeichnet. Zwei Transitionen in einem Dienstbeschreibungsnetze sind also funktional ausführbar, wenn das Ergebnis ihrer Ausführung unabhängig davon ist, in welcher Reihenfolge die Transitionen ausgeführt werden. Allgemein werden zwei Transitionen als nebenläufig aktiviert angesehen, wenn genügend Marken im Vorbereich vorhanden sind, so dass beide Transitionen feuern können und sie somit konfliktfrei sind. Für sichere Dienstbeschreibungsnetze, die in Abschnitt 4.3.3 eingeführt worden sind, ist dies gleichbedeutend damit, dass ihr Vorbereich disjunkt ist.

Bei Dienstbeschreibungsnetzen können jedoch aufgrund von Fernwirkungen über den Zustand Abhängigkeiten entstehen, die nicht mehr durch die Stellen im Vor- und Nachbereich der Transition beschrieben werden können. Aus diesem Grund soll der Begriff der funktionalen Nebenläufigkeit verwendet werden, um zu beschreiben, dass für zwei nebenläufige Transitionen gewährleistet bleibt, dass diese unabhängig voneinander schalten können und dass die Reihenfolge ihres Schaltens für den resultierenden Ergebniszustand unerheblich ist. Ist dies nicht gewährleistet, so wird häufig von Race-Conditions gesprochen, die dazu führen, dass, je nachdem welche Transition als erste feuert, die Ausführung in unterschiedlichen Ergebnissen resultiert.

Zwei Transitionen können unabhängig voneinander nebenläufig schalten, wenn sich ihre Anschriften nicht beeinflussen. Dies ist entweder der Fall, wenn sie unterschiedliche Lokationen verändern oder wenn die Veränderungen auf der gleichen Lokation kommutativ sind. Letzteres bedeutet, dass die Reihenfolge der Veränderung ohne Bedeutung für das Ergebnis ist. So ist es beispielsweise für das Ergebnis gleich, ob von einem Konto zuerst 100 Euro abgezogen werden und dann 50 Euro eingezahlt werden oder ob dies andersherum geschieht. Dennoch kann es Nebenbedingungen geben, die es notwendig machen, eine bestimmte Reihenfolge einzuhalten. So mag es notwendig sein, zunächst 50 Euro einzuzahlen, da andernfalls der Kontostand negativ werden würde, was nicht gestattet ist.

Ein entsprechendes Beispiel findet sich stark vereinfacht in Abbildung 4.12. Sei `a` eine Variable vom Typ Konto und sei `acc` ein Objekt der Trägermenge vom Typ Konto, welches das Attribut `balance` besitzt. Die Transition **`t1`** setzt dieses Attribut zunächst auf 0. Da Transition **`t2`** eine Aktivierungsbedingung besitzt, kann diese Transition folglich nur schalten, wenn das Attribut `balance` von `acc` durch Transition **`t3`** auf 200 gesetzt wurde. Somit ist das Schalten von **`t2`** abhängig vom Schalten von **`t3`** und die Lokalität von **`t2`** ist folglich verletzt. Dies entspricht aber der Abhängigkeit in der realen Welt und ist somit intendiert.

Wenn mehrere Transition nebenläufig den Wert einer Lokation durch einen festen Wert ersetzen, so ist dies ein Modellierungsfehler, da der Wert, den die Lokation

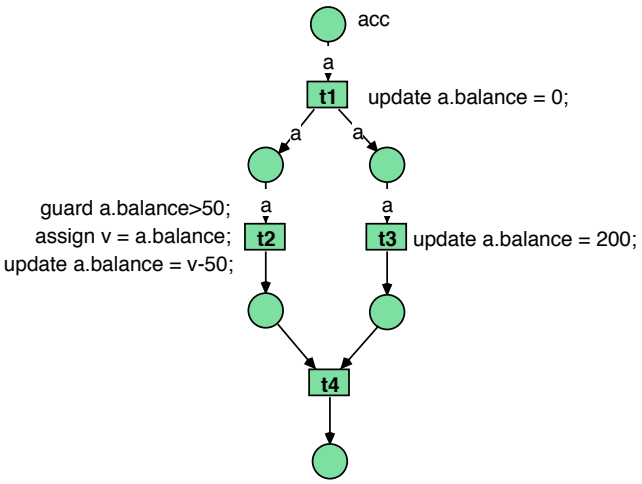


Abbildung 4.12: Abarbeitung abhängig vom Zustand

einnimmt in diesem Fall allein abhängig von der Ausführungsreihenfolge ist. Es ist in der Regel nicht sinnvoll, dass ein Wert nichtdeterministisch ersetzt wird, je nachdem welche Transition schneller schaltet, da dies mit der tatsächlichen Abarbeitungsgeschwindigkeit der Dienste keinerlei Zusammenhang aufweisen muss.

Anders verhält es sich bei relativen Updates, die den Wert einer Lokation in Abhängigkeit des vorherigen Wertes der Lokation ändern. Sind die Operatoren, die auf den vorherigen Wert der Lokation angewandt werden, kommutativ, so ergibt sich, dass die Reihenfolge nicht von Bedeutung ist, da das Ergebnis dasselbe ist. Ein Beispiel wäre etwa die oben dargestellte Verringerung des Kontostandes durch eine Transition und die Erhöhung desselben durch eine andere. Hier ist die Reihenfolge irrelevant, da die Addition kommutativ ist[6].

Funktionale Nebenläufigkeit

Funktionale Nebenläufigkeit beschreibt die Tatsache, dass zwei Transitionen ihr Schaltverhalten gegenseitig nicht beeinflussen, wenn sie nebenläufig schalten können. Es wird also verhindert, dass die Ausführungsreihenfolge das Ergebnis einer Schaltsequenz beeinflusst.

Diese Eigenschaft ist verwandt mit der Funktionalität wie sie von Jessen und Valk (1987) (S. 168) für Auftragssysteme beschrieben wird, da es auch bei funktionalen Auftragssystemen das Ziel ist, dass die möglichen Ausführungsfolgen zu

[6]Dies mag zunächst ungewöhnlich klingen, da es sich bei der Verringerung doch um eine Subtraktion handelt. Interpretieren man jedoch den Operator $-$ als Kurzschreibweise für die Addition der Negation, so dass $a-b=a+(-b)$ ist, so ist auch $a+((-b)+c)=a+(c+(-b))$. Gleiches gilt für die Division, wenn man sie als Multiplikation mit dem Kehrwert betrachtet.

dem selben Zustand führen, sofern dieselben Transitionen schalten. Da in Dienstbeschreibungsnetzen jedoch im Gegensatz zu Auftragssystemen Zyklen zugelassen sind, lässt sich die Definition nicht direkt übertragen.

Um zu beschreiben, dass sich zwei nebenläufige Updates nicht überschreiben können und dass das Feuern einer von zwei nebenläufigen Transitionen keinen Einfluss auf das Schaltverhalten der anderen Transition hat, soll hier funktionale Nebenläufigkeit definiert werden. Hierzu sollen zunächst die Begriffe der nebenläufigen Aktiviertheit von Transitionen und des Konflikts präzisiert werden. Für ein Netz $\mathcal{N} = (P, T, F)$ ist eine Multimenge von Transitionen $\mathbf{b} \in MS(T')$ über einer Menge $T' \subseteq T$ *nebenläufig aktiviert* in einer Markierung $\mathbf{m}$, wenn für alle $p \in P$ gilt

$$\mathbf{m}(p) \geq \sum_{t \in T'} (|{}^\bullet t \cap \{p\}| \cdot \mathbf{b}(t))$$

Dies wird als $\mathbf{m} \xrightarrow{\mathbf{b}}$ beziehungsweise für Mengen von Transitionen T' als $\mathbf{m} \xrightarrow{T'}$ notiert. Anders formuliert bedeutet dies, dass genug Marken im Vorbereich einer Multimenge von Transitionen vorhanden sein müssen, damit alle Elemente der Multimenge feuern können. Die Verwendung von Multimengen reflektiert die Tatsache, dass eine Transition durchaus mehrfach nebenläufig zu sich selbst aktiviert sein kann. Zum Teil findet sich auch die Formulierung, dass alle Permutationen von $\mathbf{b}$ in $\mathbf{m}$ als Schaltfolge aktiviert sein müssen. Obige Definition ist angelehnt an die Definition im ISO Standard ISO 15909 (2002), welcher auch eine Definition der nebenläufigen Aktiviertheit für algebraische Netze bietet.

Für Dienstbeschreibungsnetze sind Schaltmodi zu betrachten. Hierzu sei $\mathbf{Occ}(Q)$ die Menge der Schaltmodi in einem Netzzustand $Q = (\mathbf{m}, \mathfrak{Q})$. Eine Menge von Schaltmodi $\mathbf{Occ}_* \subseteq \mathbf{Occ}(Q)$ ist *nebenläufig aktiviert* in Q, wenn für alle $p \in P$ gilt

$$\mathbf{m}(p) \geq \sum_{(t,\alpha) \in \mathbf{Occ}_*} \alpha(W(p,t))$$

Dies wird als $\mathbf{m} \xrightarrow{\mathbf{Occ}_*}$ beziehungsweise für sichere Dienstbeschreibungsnetze auch als $\mathbf{m} \xrightarrow{T'}$ notiert, wobei T' die Menge der Transitionen der Schaltmodi aus $\mathbf{Occ}_*$ ist. Hierbei stellt das Summensymbol die Summe über Elementen einer Multimenge dar, wie dies in Definition 3.43 eingeführt wurde. Zudem wird für nicht vorhandene Kanten $W(f) = \mathbf{0}$ angenommen.

Es sei darauf hingewiesen, dass die Notation $\mathbf{m} \xrightarrow{\{t_1,t_2\}}$ nicht zu verwechseln mit der Notation $\mathbf{m} \xrightarrow{t_1 t_2}$ ist. Erstere bezeichnet die nebenläufige Aktiviertheit von t_1 und t_2, während letztere die Aktiviertheit der Schaltfolge $t_1 t_2$ beschreibt.

Zwei Transitionen $t, t' \in T$ stehen in der Markierung $\mathbf{m}$ zueinander in *Konflikt* gdw. $\mathbf{m} \xrightarrow{t}$, $\mathbf{m} \xrightarrow{t'}$ und nicht $\mathbf{m} \xrightarrow{\{t,t'\}}$ gilt. Der Begriff wird auf Dienstbeschreibungsnetze übertragen, indem statt Transitionen Schaltmodi verwendet werden.

Die nebenläufige Aktivierung zweier Schaltmodi bei Dienstbeschreibungsnetzen bedeutet nicht in jedem Fall, dass zwei Transitionen in beliebiger Reihenfolge schalten können, da durch die Veränderung von Lokationen Fernwirkungen auf die Aktivierungsbedingungen entstehen können, so dass die Aktivierungsbedingung einer Transition nach dem Schalten einer anderen Transition nicht mehr länger wahr ist. Dies wird durch funktionale Nebenläufigkeit verhindert, indem sie die von Bernstein (1966) geforderten Eigenschaften für die nebenläufige Verarbeitung von Abläufen umsetzt, wie folgende Definition darstellt.

Definition 4.36 (Funktionale Nebenläufigkeit)
Sei $\mathcal{SDN}$ ein Dienstbeschreibungsnetz, $Q = (\mathbf{m}, \mathfrak{Q})$ ein Netzzustand und seien (t_1, α_1) und (t_2, α_2) zwei Schaltmodi von $\mathcal{SDN}$ in Q. Die Schaltmodi (t_1, α_1) und (t_2, α_2) sind *funktional nebenläufig* in Q aktiviert gdw. sie nebenläufig aktiviert sind und gilt

- $Q \xrightarrow{(t_k,\alpha_k)} Q'$ impliziert $Q' \xrightarrow{(t_l,\alpha_l)}$ für $k, l \in \{1, 2\}, k \neq l$ und
- $Q \xrightarrow{(t_1,\alpha_1)} Q_1$ und $Q \xrightarrow{(t_2,\alpha_2)} Q_2$ impliziert $Q_1 \xrightarrow{(t_2,\alpha_2)} Q_*$ und $Q_2 \xrightarrow{(t_1,\alpha_1)} Q_*$.

♦

Die Forderungen entsprechen denen der gegenseitigen Unabhängigkeit der Aktivierungsbedingungen und der Unabhängigkeit der Ausführungsreihenfolge der Schaltmodi. Wiederum kann bei funktional nebenläufigen Transitionen jede Permutation der Transitionen schalten. Aus diesem Grund lässt sich die funktionale Nebenläufigkeit auf Mengen von Transitionen ausdehnen. Eine nebenläufig aktivierte Menge von Transitionen ist entsprechend funktional nebenläufig aktiviert, wenn je zwei Transitionen paarweise funktional nebenläufig aktiviert sind. Anders formuliert lässt sich festhalten, dass zwei nebenläufig aktivierte Transitionen kommutieren, wenn sie funktional nebenläufig aktiviert sind.

Sind alle Transitionen in allen erreichbaren Zuständen stets funktional nebenläufig aktiviert, so lassen sich nebenläufige Transitionen sequentialisieren. Dies wird in Kapitel 6 für elementare Dienstbeschreibungsnetze näher behandelt werden. Die Sequentialisierung bietet den Vorteil, dass das funktionale Verhalten eines Dienstbeschreibungsnetzes ohne Nebenläufigkeit anhand einer Zustandsmaschine betrachtet werden kann. Neben der funktionalen Nebenläufigkeit der Transitionen ist hierzu zudem die Typisierungskonformität notwendig, die im nächsten Unterabschnitt beschrieben wird.

4.4.2 Typisierungskonformität

Anders als algebraische Netze erlauben Dienstbeschreibungsnetze lediglich Variablen als Kantenanschrift und fordern zudem, dass jede Variable nur einmal an einer Eingangskante einer Transition steht, da andernfalls die gleichen Variablen verschiedener Kanten unifiziert werden müssten. Folglich sollte eine Transition prinzipiell schalten können, wenn genug Marken in ihrem Vorbereich existieren und wenn die Aktivierungsbedingung der Transition dies zulässt. Dies ist jedoch nicht immer der Fall, wenn Netze eine *removeRole*-Anweisung besitzen.

Die Reservierung der Sortenlokationen durch Stellen in der Definition des Schaltverhaltens von Dienstbeschreibungsnetzen mit *removeRole*-Anweisungen führt zu einem unerwarteten Verhalten. Netze, bei denen niemals eine Stelle eine Sortenlokation reserviert, die verändert wird, besitzen hingegen das gleiche Schaltverhalten wie Dienstbeschreibungsnetze ohne *removeRole*-Anweisungen, deren Schaltverhalten in Definition 4.21 angegeben wurde. Dies macht es sinnvoll, Dienstbeschreibungsnetze auf typisierungskonforme Dienstbeschreibungsnetze einzuschränken, die dieses Verhalten nicht aufweisen.

Definition 4.37 (Typisierungskonformität)
Ein Dienstbeschreibungsnetz $\mathcal{SDN}$ ist in einem initialen Netzzustand Q_0 *typisierungskonform* gdw. für jeden erreichbaren Netzzustand $(\mathbf{m}, \mathfrak{Q}) \in [Q_0\rangle_{\mathcal{SDN}}$ gilt: wenn der Schaltmodus (t, α) nicht in $(\mathbf{m}, \mathfrak{Q})$ aktiviert ist, gibt es auch kein $(\mathbf{m}', \mathfrak{Q})$ mit $\mathbf{m}' < \mathbf{m}$, in dem der Schaltmodus (t, α) aktiviert ist. ♦

Die markierten Stellen wirken für *removeRole*-Anweisungen wie Stellen mit Inhibitor-Kanten (vgl. Christensen und Hansen, 1993; Busi, 2002). Folglich ist es möglich, dass eine Transition in einer Markierung nicht schalten kann, wohl aber wenn man Marken aus dem Netz entfernt. Um dieses Verhalten zu vermeiden, wird die Typisierungskonformität eingeführt. Ein Dienstbeschreibungsnetz ist typisierungskonform, wenn es keine Stelle gibt, die für eine Marke als Inhibitor-Stellen fungiert. Diese Bedingung wird auch Monotonie-Bedingung für Petrinetzmarkierungen genannt, da eine größere Markierung stets alle Transitionen aktiviert, die eine Markierung mit weniger Marken aktiviert. Typisierungskonforme Dienstbeschreibungsnetze gehorchen also dieser Monotonie-Bedingung.

Eine hinreichende Bedingung hierfür ist bei sicheren Dienstbeschreibungsnetzen, dass keine Stelle p der Sorte s (oder einer Subsorte) zusammen mit einer Transition t existiert, so dass t im zugrundeliegenden Netz in einer Markierung aktivierbar und p markiert ist und t die Sorte entfernt, durch die p getypt ist. Dies folgt aus Lemma 4.28, da es in diesem Fall auch keine Schaltfolge im Dienstbeschreibungsnetz geben kann, die zu einer Markierung führt, die den Vorbereich der Transition t und die Stelle p gleichzeitig markiert. Von dieser Möglichkeit festzustellen, ob ein

sicheres Dienstbeschreibungsnetz typisierungskonform ist, wird im nächsten Kapitel zur Definition elementarer Dienstbeschreibungsnetze Gebrauch gemacht. Diese Netze werden in Abschnitt 5.2 näher beschrieben.

Für Dienstbeschreibungsnetze, die in jedem initialen Netzzustand typisierungskonform sind, lässt sich trotz der *removeRole*-Anweisungen die Aktiviertheit und das Feuern einer Transition auf die in Definition 4.21 gegebenen Bedingungen beziehungsweise auf das Schaltverhalten reduzieren. Dies bedeutet, dass Sortenlokationen während des Schaltens nicht betrachtet werden müssen.

Lemma 4.38 (Aktiviertheit bei Typisierungskonformität)
Sei $\mathcal{SDN}$ ein in einem initialen Netzzustand Q_0 typisierungskonformes Dienstbeschreibungsnetz. Eine Transition $t \in T$ ist in einem Netzzustand $Q = (\mathbf{m}, \mathfrak{Q}) \in [Q_0\rangle_{\mathcal{SDN}}$ aktiviert gdw. es eine Variablenbelegung α gibt, so dass $\mathbf{m}(p) \geq \alpha(W(p,t))$ für alle $p \in {}^\bullet t$ und $[\![G(t)]\!]_\alpha^{\mathfrak{Q}} = \mathit{true}$ gilt, und der Systemzustand nicht reserviert ist.

Beweis:
($\Rightarrow$) Sei $t \in T$ in $(\mathbf{m}, \mathfrak{Q})$ aktiviert. Dann gibt es einen Schaltmodus (t, α) für den $[\![G(t)]\!]_\alpha^{\mathfrak{Q}} = \mathit{true}$ ist. Nach der Definition der Aktiviertheit in Definition 4.24 ergibt sich dann die Aussage.
($\Leftarrow$) Sei t eine Transition und α eine Variablenbelegung, so dass $\mathbf{m}(p) \geq \alpha(W(p,t))$ für alle $p \in {}^\bullet t$ und $[\![G(t)]\!]_\alpha^{\mathfrak{Q}} = \mathit{true}$ gilt, und der Systemzustand nicht reserviert ist. Nach Definition 4.24 ist t dann aktiviert, wenn der Systemzustand nicht reserviert ist und wenn $\mathbf{Loc}^{\mathbf{I}}_{sort}(Q,t) \cap \mathbf{Loc}^{\mathbf{U}}_{sort}(Q,(t,\alpha)) = \emptyset$ ist, da $[\![G(t)]\!]_\alpha^{\mathfrak{Q}} = \mathit{true}$ gilt.

Wäre $\mathbf{Loc}^{\mathbf{I}}_{sort}(Q,t) \cap \mathbf{Loc}^{\mathbf{U}}_{sort}(Q,(t,\alpha)) \neq \emptyset$, so wäre es möglich durch Entfernen der Marke auf einer Stelle, die dafür sorgt, dass in $\mathbf{Loc}^{\mathbf{I}}_{sort}(Q,t)$ eine Sortenlokation enthält, die auch in $\mathbf{Loc}^{\mathbf{U}}_{sort}(Q,(t,\alpha))$ ist, eine Markierung zu erhalten, in der (t,α) aktiviert ist, da diese Stelle gemäß Definition nicht im Vorbereich von p sein kann. Dann wäre $\mathcal{SDN}$ jedoch nicht typisierungskonform im initialen Netzzustand Q_0. □

Korollar 4.39 (Schaltverhalten typisierungskonformer Netze)
Für typisierungskonforme Dienstbeschreibungsnetze kann die Reservierung der Sorten-Lokationen durch Stellen zur Beschreibung der Aktiviertheit und des Schaltverhaltens aus Definition 4.24 entfallen, so dass die Schaltregel der aus Definition 4.21 entspricht.

Es liegt nahe, dass Netze ohne *removeRole*-Anweisungen typisierungskonform sind, da sie keinerlei Anschriften aufweisen, durch die Markierungen mit weniger Marken ein anderes Schaltverhalten aufweisen. Dies hält folgendes Lemma fest.

Lemma 4.40 (Hinreichende Bedingung für Typisierungskonformität)
Sei $\mathcal{SDN}$ ein Dienstbeschreibungsnetz und Q_0 ein initialer Netzzustand. Gibt es

keinen erreichbaren Netzzustand $(\mathbf{m}, \mathfrak{Q}) \in [Q_0\rangle_{\mathcal{SDN}}$, so dass für ein $p \in P$ und ein $t \in T$ gilt $\mathbf{m}(p) \neq \mathbf{0}$, $d(p) \preceq s_\mathfrak{c}$ für ein $s_\mathfrak{c} \in S^{dyn}$ und $(removeRole(\tau, s_\mathfrak{c})) \in \textsc{Alph}(\mathbf{cmd}(t))$ und $(\mathbf{m}, \mathfrak{Q})$ die Transition t aktiviert, so ist $\mathcal{SDN}$ in $\mathfrak{Q}_0$ typisierungskonform.

Beweis:
Angenommen $\mathcal{SDN}$ wäre im initialen Netzzustand Q_0 nicht typisierungskonform. Dann gäbe es einen erreichbaren Netzzustand $(\mathbf{m}, \mathfrak{Q}) \in [Q_0\rangle_{\mathcal{SDN}}$, so dass ein Schaltmodus (t, α) nicht in $(\mathbf{m}, \mathfrak{Q})$ aktiviert ist, wohl aber im Netzzustand $(\mathbf{m}', \mathfrak{Q})$ mit $\mathbf{m}' < \mathbf{m}$. Gemäß der Schaltregel aus Definition 4.24 muss dann $\mathbf{Loc}^{\mathbf{I}}_{sort}(Q, t) \cap \mathbf{Loc}^{\mathbf{U}}_{sort}(Q, (t, \alpha)) \neq \emptyset$ gelten. Folglich ist $(removeRole(\tau, s'_\mathfrak{c})) \in \textsc{Alph}(\mathbf{cmd}(t))$ für ein $s_\mathfrak{c} \in S^{dyn}$ und es ist $q \in \mathbf{m}(p)$ und $d(p) \preceq s_\mathfrak{c}$ für ein $p \in P$. Dann würde $(\mathbf{m}, \mathfrak{Q}) \in [Q_0\rangle_{\mathcal{SDN}}$ jedoch die geforderte Bedingung verletzen. □

Korollar 4.41 (Typisierungskonformität ohne *removeRole*-Anweisungen)
Sei $\mathcal{SDN}$ ein Dienstbeschreibungsnetz, bei dem keine Transition eine *removeRole*-Anweisung besitzt. Dann ist $\mathcal{SDN}$ in jedem initialen Netzzustand typisierungskonform.

Beweis:
Dies folgt direkt aus Lemma 4.40, da keine *removeRole*-Anweisungen existieren. □

Besonders einfach stellt sich die Aktivierungsbedingung in typisierungskonformen Dienstbeschreibungsnetz für Transitionen dar, deren Aktivierungsbedingung stets wahr ist. Diese Transitionen verhalten sich genau so wie die zugrundeliegenden Netze. Dies lässt sich leicht aus Lemma 4.38 folgern.

Lemma 4.42 (Folgen von Schaltmodi ohne Aktivierungsbedingungen)
Sei $\mathcal{SDN}$ ein in einem initialen Netzzustand Q typisierungskonformes Dienstbeschreibungsnetz und sei σ_α eine Folge von Schaltmodi von $\mathcal{SDN}$, so dass für jede in σ_α vorkommende Transition t gilt, dass $G(t) = true$ ist. Dann ist σ_α in einem Netzzustand $(\mathbf{m}, \mathfrak{Q}) \in [Q\rangle_{\mathcal{SDN}}$ aktiviert gdw. die Schaltfolge σ, die der Reduktion von σ_α auf eine Folge von Transitionen entspricht, in der Markierung $\mathring{\mathbf{m}}$ in $\mathcal{N}$ aktiviert ist.

Beweis:
($\Rightarrow$) Dies folgt direkt aus Lemma 4.28.
($\Leftarrow$) Nach Lemma 4.38 ist zu zeigen, dass es für jede Transition t der Schaltfolge σ stets eine Variablenbelegung α gibt, so dass $\mathbf{m}(p) \geq \alpha(W(p, t))$ für alle $p \in {}^\bullet t$ gilt. Diese Variablenbelegung kann jedoch direkt dadurch erzeugt werden, dass sämtliche Marken der Eingangsstellen an die Variablen der Eingangskanten gebunden werden, was möglich ist, da alle Eingangsstellen markiert sind, wenn t in $\mathring{\mathbf{m}}$ markiert ist. □

4.4.3 Korrektheit

Zur Definition der Korrektheit wird das Konzept der initialbeschrifteten Dienstbeschreibungsnetze herangezogen. Dies sind Dienstbeschreibungsnetze, die neben der Typisierung der Anfangsstelle i noch weitere Anforderungen an diese Stelle besitzen, die durch einen Term $\tau_G \in \mathbb{T}_{\Sigma\mathfrak{Q},bool}(\{y\})$ repräsentiert werden, wobei y eine Variable ist, der der Initialwert des Netzes zugewiesen wird. Durch die Einschränkung auf Objekte, die dem Term τ_G genügen, ist es möglich anzugeben, dass das Netz für alle Objekte, die diesen Term zu wahr auswerten, korrekt ist. Ist dieser Term stets wahr, so fallen initialbeschriftete Dienstbeschreibungsnetze und herkömmliche Dienstbeschreibungsnetze zusammen.

Initialbeschriftete Dienstbeschreibungsnetze

Intialbeschriftete Dienstbeschreibungsnetze schränken die möglichen Initialmarkierungen eines Dienstbeschreibungsnetzes über die Typisierung der Stelle i hinausgehend ein und bieten so die Möglichkeit, Vorbedingungen für die Ausführung eines Dienstes anzugeben.

Definition 4.43 (Initialbeschriftetes Dienstbeschreibungsnetz)
Ein *initialbeschriftetes Dienstbeschreibungsnetz* $\mathcal{SDN}_{init} \stackrel{\text{def}}{=} (\mathcal{SDN}, y, \tau_G)$ ist ein Tupel bestehend aus einem Dienstbeschreibungsnetz $\mathcal{SDN}$ zusammen mit einer Variablen y mit $y \in X_s$ für $s \preceq d(i)$ sowie einem Term $\tau_G \in \mathbb{T}_{\Sigma\mathfrak{Q},bool}(\{y\})$, so dass y nicht in $\mathcal{SDN}$ verwendet wird.
Eine *Initialisierung* $\mathbf{In} = (\mathfrak{Q}_0, \alpha_0)$ eines initialbeschrifteten Dienstbeschreibungsnetzes $\mathcal{SDN}_{init}$ ist ein Paar bestehend aus einem Zustand $\mathfrak{Q}_0 = (S_Q, \Omega_Q)$ und einer Variablenbelegung $\alpha_0 : \{y\} \to S_Q$, so dass $[\![\tau_G]\!]_{\alpha_0}^{\mathfrak{Q}_0}$ zu wahr auswertet und $(\mathbf{m}_0, \mathfrak{Q}_0)$ ein initialer Netzzustand von $\mathcal{SDN}$ ist, für den $\mathbf{m}_0(i) = \alpha_0(y)$ ist. Der entsprechende Netzzustand $(\mathbf{m}_0, \mathfrak{Q}_0)$ von $\mathcal{SDN}$ wird als $Q^{\mathbf{In}}$ notiert. Die Gesamtheit aller Initialisierungen eines Netzes $\mathcal{SDN}_{init}$ wird als $\mathbf{Init}(\mathcal{SDN}_{init})$ notiert. ⧫

Ein initialbeschriftetes Dienstbeschreibungsnetz schränkt folglich die möglichen initialen Netzzustände eines Dienstbeschreibungsnetzes ein. Ist $\tau_G = \mathit{true}$, so fällt die Definition der initialbeschrifteten Dienstbeschreibungsnetze mit der Definition der Dienstbeschreibungsnetze zusammen.

Definition 4.44 (Funktional ausführbare initialbeschriftete SD-Netze)
Sei $\mathcal{SDN}_{init}$ ein initialbeschriftetes Dienstbeschreibungsnetz. Dann ist $\mathcal{SDN}_{init}$ unter der Initialisierung $\mathbf{In}$ *funktional ausführbar* gdw. für alle erreichbaren Netzzustände $Q = (\mathbf{m}, \mathfrak{Q}) \in [Q^{\mathbf{In}}\rangle$ gilt, dass die Existenz zweier nebenläufig in Q aktivierter Schaltmodi (t_1, α_1) und (t_2, α_2) unter $\mathfrak{Q}$ impliziert, dass diese in Q funktional

nebenläufig sind.
$\mathcal{SDN}_{init}$ ist *funktional ausführbar*, wenn es für alle Initialisierungen **In** funktional nebenläufig ausführbar ist. ⧫

In obiger Definition bezeichnet $[Q^{\mathbf{In}}\rangle$ die Menge aller von $Q^{\mathbf{In}}$ (vgl. Def. 4.43) erreichbaren Netzzustände. Die Definition besagt, dass alle Schaltmodi, die in einem initialbeschrifteten Netz überhaupt unter einem Zustand und einer Variablenbelegung nebenläufig aktiviert werden können, dann funktional nebenläufig aktiviert sein müssen. Dies verhindert, dass zwei nebenläufige Transitionen je nach ihrer Abarbeitung unterschiedliche Ergebnisse produzieren.

Funktional ausführbare initialbeschriftete Dienstbeschreibungsnetze verhalten sich bezüglich der Nebenläufigkeit so, dass nebenläufig aktivierte Transitionen auch stets in beliebiger Reihenfolge feuern können, ohne dass das Verhalten des Netzes ein anderes ist. Es entstehen somit keine unerwünschten Seiteneffekte, die dies verhindern. Zudem ist das Ergebnis eindeutig bestimmt und nicht von der Ausführungsreihenfolge abhängig.

Korrektheitsbegriffe für Dienstbeschreibungsnetze

In Anlehnung an die Korrektheit von Workflownetzen wird auch bei Dienstbeschreibungsnetzen gefordert, dass die Fortsetzbarkeit und die eindeutige Termination gegeben ist. Hierbei werden neben der Markierung jeweils auch die Zustände mit in die Betrachtung einbezogen. Weitere Korrektheitsforderungen betrachten die Endlichkeit der Ausführung und die Wahrung der Stellentypisierung, was etwa durch das Entfernen eines dynamischen Konzeptes verletzt werden könnte.

Für Dienstbeschreibungsnetze sollen verschieden strenge Korrektheitskriterien angegeben werden, da die Aktivierbarkeit von Transitionen nicht immer notwendig ist. Bei der Komposition von Diensten mag es etwa sein, dass eine Transition niemals aktivierbar ist, da der Dienst für einen allgemeineren Fall konzipiert wurde. Dies ist jedoch nicht unbedingt ein Fehler. Dennoch sollte die Transition im zugrundeliegenden Workflownetz aktivierbar sein, da sie andernfalls in keinem Aufruf dieses Dienstes aktivierbar ist.

In Anlehnung an die finale Markierung bei Workflownetzen ist ein finaler Netzzustand ein Netzzustand, der nur eine Marke auf der Stelle o besitzt. Dies wird unter Verwendung der anonymisierten Markierung $\mathring{\mathbf{m}}$ wie folgt definiert.

Definition 4.45 (Finaler Netzzustand)
Gegeben ein Dienstbeschreibungsnetz $\mathcal{SDN}$. Ein Netzzustand $Q_f = (\mathbf{m}_f, \mathfrak{Q}_f)$ von $\mathcal{SDN}$ ist *final* gdw. $\mathring{\mathbf{m}}_f = [o]$ gilt. ⧫

Mittels des finalen Netzzustandes lässt sich die schwache Korrektheit von Dienst-

beschreibungsnetzen definieren.

Definition 4.46 (Schwache Korrektheit von Dienstbeschreibungsnetzen) Ein initialbeschriftetes SD-Netz $\mathcal{SDN}_{init}$ ist *schwach korrekt* gdw. es *variablendeterminiert* (vgl. Def. 4.19) ist und für jede Initialisierung $\mathbf{In} = (\alpha_0, \mathfrak{Q}_0)$ mit dem initialen Netzzustand $Q^{\mathbf{In}} = (\mathbf{m}_0, \mathfrak{Q}_0)$ und für jeden Netzzustand $(\mathbf{m}_1, \mathfrak{Q}_1) \in [Q^{\mathbf{In}}\rangle$

1. ein finaler Netzzustand Q_f mit $(\mathbf{m}_1, \mathfrak{Q}_1) \xrightarrow{*} Q_f$ existiert (*Fortsetzbarkeit und Termination*),

2. für je zwei in $(\mathbf{m}_1, \mathfrak{Q}_1)$ nebenläufig aktivierte Schaltmodi (t_1, α_1) und (t_2, α_2) gilt, dass (t_1, α_1) und (t_2, α_2) funktional nebenläufig aktiviert sind (*Funktionale Nebenläufigkeit*), und

3. das Nicht-Aktiviertsein des Schaltmodus (t, α) in $(\mathbf{m}_1, \mathfrak{Q}_1)$ impliziert, dass es kein $(\mathbf{m}'_1, \mathfrak{Q}_1)$ mit $\mathbf{m}'_1 < \mathbf{m}_1$ gibt, in dem der Schaltmodus (t, α) aktiviert ist (*Typisierungskonformität*).

♦

Wie bei Workflownetzen fallen die Termination und die Fortsetzbarkeit zusammen. Zwar wäre es möglich gewesen, eine eigene Terminationsbedingung zu formulieren, wie beispielsweise $\overset{\circ}{\mathbf{m}}_1 \geq [o]$ impliziert, dass $\overset{\circ}{\mathbf{m}}_1 = [o]$ ist, jedoch ist diese Bedingung implizit, da o keinerlei Ausgangskanten besitzt und folglich $Q \xrightarrow{*} Q_f$ dies impliziert. Während die Korrektheit bei Workflownetzen neben der Termination und der Fortsetzbarkeit noch die Aktivierbarkeit fordert, wird hier stattdessen gefordert, dass je zwei Transitionen stets funktional nebenläufig aktivierbar sind. Die Aktivierbarkeit wird hier nur für die starke Korrektheit gefordert, die im Folgenden definiert wird.

Die schwache Korrektheit setzt nicht die Aktivierbarkeit von Transitionen voraus, so dass es möglich ist, dass in einem Netz niemals aktivierbare Transitionen existieren. Dies wird durch die starke Korrektheit eines Dienstbeschreibungsnetzes verhindert. Da es Transitionen geben kann, die nicht dem normalen Schaltverhalten des Netzes angehören, etwa weil sie der Fehlerbehandlung dienen, lässt sich die starke Korrektheit bezüglich einer Submenge von Transitionen definieren. Ist diese Menge leer, so fallen die schwache und die starke Korrektheit zusammen.

Definition 4.47 ((Starke) Korrektheit von Dienstbeschreibungsnetzen) Ein initialbeschriftetes SD-Netz $\mathcal{SDN}_{init}$ ist *(stark) korrekt* bezüglich einer Menge von Transitionen $T_C \subseteq T$ gdw.

1. $\mathcal{SDN}_{init}$ schwach korrekt ist und

2. zu jeder Transition $t \in T_C$ eine Initialisierung $\mathbf{In}$ mit dem initialen Netzzustand $Q^{\mathbf{In}}$ existiert, so dass es ein $Q \in [Q^{\mathbf{In}}\rangle$ gibt und $Q \xrightarrow{t}$ (*Aktivierbarkeit*).

♦

Der Korrektheitsbegriff für Dienstbeschreibungsnetze erweitert den Korrektheitsbegriff für Workflownetze von van der Aalst (2000) auf Dienstbeschreibungsnetze. Dabei ist jedoch zu beachten, dass das zugrundeliegende Workflownetz nicht korrekt sein muss, damit ein Dienstbeschreibungsnetz korrekt ist. Dies ist der Fall, weil durch Aktivierungsbedingungen Verklemmungen verhindert werden können, die in einem Workflownetz entstehen könnten. Für die Menge der Transitionen T_C ist es in der Regel sinnvoll zu verlangen, dass das Subnetz aus ${}^\bullet T_C$ und T_C ein Workflownetz ist, da es dann ein Workflownetz von garantiert aktivierbaren Transitionen gibt.

Reduziert man ein Dienstbeschreibungsnetz hingegen auf ein triviales Dienstbeschreibungsnetz (also ein Workflownetz) indem man als einzige Sorte der Algebra $s_{\mathbf{token}}$ zulässt und als einziges Operatorsymbol $\bullet$ erlaubt, so fallen nicht nur Dienstbeschreibungsnetze und Workflownetze zusammen, wie dies in Lemma 4.25 gezeigt wurde, sondern auch die Korrektheitsbegriffe.

Lemma 4.48 (Korrektheit von Workflow- und SD-Netzen)
Ein variablendeterminiertes triviales Dienstbeschreibungsnetz (also ein Workflownetz) $\mathcal{SDN}$ ist für $T_C = T$ und jede Initialisierung mit $\tau_G = \mathit{true}$ nach Definition 4.47 korrekt gdw. das zugrundeliegende Workflownetz $\mathcal{N}$ nach Definition 3.41 korrekt ist.

Beweis:
($\Rightarrow$) Sei $\mathcal{SDN}$ korrekt. Da $\mathcal{SDN}$ trivial ist besitzt es keinerlei Aktivierungsbedingungen und alle Variablen sind vom Typ $s_{\mathbf{token}}$. Nach Korollar 4.35 gilt dann die Aussage.
($\Leftarrow$) Da die Anweisungsblöcke trivialer Dienstbeschreibungsnetze leer oder trivial sind, ist der Systemzustand eines trivialen Dienstbeschreibungsnetzes stets gleich. Hieraus folgt die funktionale Ausführbarkeit und die Typisierungskonformität sofort. Nach Korollar 4.35 ist eine Schaltfolge in einem Netzzustand $(\mathbf{m}, \mathfrak{Q})$ genau dann aktiviert, wenn sie dies im zugrundeliegenden Netz in $\mathring{\mathbf{m}}$ ist. Hieraus folgt dann die Aussage. □

Die Korrektheit von Dienstbeschreibungsnetze ist somit eine Erweiterung des klassischen Korrektheitsbegriffs für Workflownetze, wie er in Definition 3.41 angegeben wurde. Für typisierungskonforme Dienstbeschreibungsnetze ohne Aktivierungsbedingungen folgt aus Korollar 4.35, dass diese Netze genau dann (stark) korrekt sind, wenn das zugrundeliegende Netz korrekt ist.

Lemma 4.49 (Korrektheit bei fehlenden Aktivierungsbedingungen)
Ein initialbeschriftetes variablendeterminiertes SD-Netz $\mathcal{SDN}_{init}$ ist (stark) korrekt, wenn $\mathcal{SDN}$ für jede Initialisierung $\mathbf{In} = (\alpha_0, \mathfrak{Q}_0)$ mit dem initialen Netzzustand $Q^{\mathbf{In}} = (\mathbf{m}_0, \mathfrak{Q}_0)$ typisierungskonform ist, $\mathcal{SDN}$ in $(\mathbf{m}_0, \mathfrak{Q}_0)$ funktional ausführbar ist, $G(t) = \mathit{true}$ für alle $t \in T$ ist und $\mathcal{N}$ ein korrektes Workflownetz ist.

Beweis:
Die Typisierungskonformität und die funktionale Ausführbarkeit folgen direkt aus den Bedingungen. Gemäß Korollar 4.35 lässt sich in jeder Initialisierung jede Schaltfolge des zugrundeliegenden Netzes ausführen. Dies gewährleistet die Fortsetzbarkeit, die Termination und die Aktivierbarkeit. □

Während die Korrektheit bei einem initialen Netzzustand verschiedene Schaltfolgen durch das Netz zulässt, solange nur die funktional nebenläufige Aktivierung gewahrt bleibt, lässt sich die Forderung nach der Vorhersagbarkeit des Verhaltens noch weiter verstärken. Dies ist bei deterministischen Dienstbeschreibungsnetzen der Fall, bei denen zudem gefordert wird, dass es zu jedem initialen Zustand nur genau einen Endzustand geben darf. Der Finalzustand ist also in diesem Fall eindeutig durch den Initialzustand bestimmt.

Definition 4.50 (Deterministische Dienstbeschreibungsnetze)
Ein schwach korrektes initialbeschriftetes SD-Netz $\mathcal{SDN}_{init}$ ist *deterministisch* gdw. für jede Initialisierung $\mathbf{In} = (\alpha_0, \mathfrak{Q}_0)$ mit initialem Netzzustand $Q^{\mathbf{In}} = (\mathbf{m}_0, \mathfrak{Q}_0)$ gilt, dass $(\mathbf{m}_1, \mathfrak{Q}_1), (\mathbf{m}_2, \mathfrak{Q}_2) \in [Q^{\mathbf{In}}\rangle$ und $\mathring{\mathbf{m}}_1 \geq [o]$ und $\mathring{\mathbf{m}}_2 \geq [o]$ impliziert, dass $\mathbf{m}_1 = \mathbf{m}_2$, $\mathfrak{Q}_1 = \mathfrak{Q}_2$ und $\mathring{\mathbf{m}}_1 = \mathring{\mathbf{m}}_2 = [o]$ gilt. ♦

In deterministischen Dienstbeschreibungsnetzen darf es keine unterschiedlichen Systemzustände geben, die durch die Ausführung des Netzes ausgehend von dem selben initialen Netzzustand erreicht werden können. Dies reflektiert die Tatsache, dass ein Dienst stets terminiert und dass sein Verhalten klar durch die Eingabe bestimmt ist. Dies bedeutet aber auch, dass für zwei alternative Pfade von i nach o für den selben initialen Netzzustand innerhalb des Netzes gilt, dass zwei Schaltfolgen, die diesen Pfaden folgen, die gleichen Veränderungen vornehmen müssen.

Abschließend sei noch das ebenfalls wichtige Kriterium der Endlichkeit definiert. Endlichkeit bedeutet, dass ein durch ein Dienstbeschreibungsnetz beschriebener Dienst nach endlicher Zeit mit der Bearbeitung fertig ist und einen finalen Netzzustand erreicht.

Definition 4.51 (Endliche Dienstbeschreibungsnetze)
Ein schwach korrektes initialbeschriftetes SD-Netz $\mathcal{SDN}_{init}$ ist *endlich* gdw. für jede Initialisierung $\mathbf{In} = (\alpha_0, \mathfrak{Q}_0)$ mit initialem Netzzustand $Q^{\mathbf{In}} = (\mathbf{m}_0, \mathfrak{Q}_0)$ gilt,

dass jede Schaltfolge in $\mathcal{SDN}$ endlich ist. ♦

Endlichkeit und Determinismus lassen sich nicht auf Workflownetze im Allgemeinen übertragen, da dort eine Invariante beliebig häufig realisiert werden kann, ohne dass das Netz dadurch fehlerhaft wäre, und da bei Workflownetzen verschiedene Pfade zur Markierung $[o]$ führen können, ohne dass dies einen Fehler darstellt. Für Netze mit Datentypen sind diese Eigenschaften jedoch in der Regel notwendige Kriterien für ein sinnvoll modelliertes System.

Sämtliche hier diskutierten Kriterien wie Korrektheit, Determinismus und Endlichkeit werden durch ein initialbeschriftetes Dienstbeschreibungsnetz erfüllt, für das $\tau_G = \mathit{false}$ ist, da es dann keine Initialisierung geben kann. Im Allgemeinen sind jedoch sämtliche Korrektheitskriterien unentscheidbar, da bereits die funktionale Ausführbarkeit einer Analyse der Erreichbarkeit bedarf. Dies führt zur Einschränkung der Ausdrucksmächtigkeit von Dienstbeschreibungsnetzen im nächsten Kapitel.

4.4.4 Exkurs: Parallelität bei der praktischen Realisierung

Dieser Unterabschnitt widmet sich der praktischen Umsetzung von Dienstbeschreibungsnetzen. Hierbei wird die Möglichkeit skizziert, eine stärker parallele Ausführung der Dienstbeschreibungsnetze zu gewährleisten, indem nicht stets der ganze Systemzustand reserviert wird, was dadurch geschieht, dass nur der Teil eines Systemzustandes reserviert wird, der tatsächlich benötigt wird. Weitergehende Möglichkeiten werden an dieser Stelle nicht diskutiert. Insbesondere wäre es für jeden Anweisungsblock möglich, die genutzten Lokationen bereits freizugeben, wenn keine Anweisungen folgen, die diese Lokationen nutzen. Für die in Unterabschnitt 4.4.1 angeschnittenen kommutativen Operatoren ließe sich ebenfalls die Reservierung optimieren, wobei sichergestellt sein muss, dass Lese- und Schreibvorgang stets aufeinander folgen. Näherer dazu, wie dies erreicht werden kann, findet sich bei Jessen und Valk (1987).

Reservierung von Lokationen

Während die parallele Verarbeitung von Aktionen bei der Modellierung und bei der Analyse keine Rolle spielt, ist sie in der praktischen Ausführung durchaus von Interesse, da die einzelnen Aktionen hier Zeit in Anspruch nehmen. Die Koordination der Anweisungen eines Anweisungsblocks kann hierbei eine gewisse Komplexität mit sich bringen, da Lese- und Schreibzugriffe der einzelnen Anweisungen koordiniert werden müssen. Dies wird beispielsweise von Jessen und Valk (1987) im Rahmen der schematischen Auftragssysteme behandelt.

Während es generell von Interesse ist, beim Schalten von Dienstbeschreibungsnetzen eine maximale Parallelität zuzulassen, soll an dieser Stelle ein Weg eingeschlagen

werden, der sämtliche genutzten Lokationen eines Anweisungsblocks reserviert, so dass ein Anweisungsblock in jedem Fall atomar ausgeführt wird. Hierzu werden Lokationen so reserviert, dass eine Transition nicht den Inhalt einer Lokation ändern kann, die von einer zweiten Transition genutzt wird. Ein gleichzeitiger Lesezugriff wird dabei jedoch zugelassen. Dies ist vom Verhalten her vergleichbar mit den für zeitbehaftete Netze (vgl. Merlin und Faber, 1976) definierten Lesekanten (Read-Arcs) von Vogler (1997, 2002). Der Vorteil dieser Vorgehensweise ist, dass sich das Netz so verhält, als würde stets nur maximal eine Transition zu einem Zeitpunkt schalten und der Systemzustand vollständig reserviert.

Um dieses Vorgehen zu präzisieren, wird einmal die *Menge der involvierten Lokationen* $\mathbf{Loc^I}(\mathbf{Cmd}, \mathfrak{Q}, \alpha)$ und einmal die *Menge der veränderten Lokationen* $\mathbf{Loc^U}(\mathbf{Cmd}, \mathfrak{Q}, \alpha)$ eines belegungskonsistenten Anweisungsblocks **Cmd** unter einer Variablenbelegung α in einem Zustand $\mathfrak{Q}$ benötigt. Diese nutzt die auf Seite 115 definierten Mengen der involvierten und der veränderten Lokationen $\mathbf{Loc^I}(\tau, \mathfrak{Q}, \alpha)$ und $\mathbf{Loc^U}(\mathbf{cmd}_U, \mathfrak{Q}, \alpha)$. Die Menge $\mathbf{Loc^U}(\mathbf{Cmd}, \mathfrak{Q}, \alpha)$ ist dabei die Vereinigung aller Mengen $\mathbf{Loc^U}(\mathbf{cmd}_U, \mathfrak{Q}, \alpha)$ von Update-Anweisungen $\mathbf{cmd}_U \in \textsc{Alph}(\mathbf{Cmd})$. Die Menge $\mathbf{Loc^I}(\mathbf{Cmd}, \mathfrak{Q}, \alpha)$ ist hingegen die Vereinigung aller $\mathbf{Loc^I}(\tau, \mathfrak{Q}, \alpha)$ für einen Term τ einer Anweisung $cmd \in \textsc{Alph}(\mathbf{Cmd})$. Erstere Menge ist eine Submenge der letzteren und beschreibt die veränderten Lokationen der Updates. Für einen belegungskonsistenten Anweisungsblock lassen sich diese Mengen bilden, da die Auswertung aller Terme bereits in dem Zustand möglich ist, in dem der Anweisungsblock ausgewertet wird.

Während des Schaltens einer Transition werden nun sämtliche Lokationen der Menge $\mathbf{Loc^I}(\mathbf{cmd}(t), \mathfrak{Q}, \alpha)$ *lesereserviert* (R-reserviert) und sämtliche Lokationen der Menge $\mathbf{Loc^U}(\mathbf{cmd}(t), \mathfrak{Q}, \alpha)$ *schreibreserviert* (W-reserviert). Die Reserviertheit ist eine globale Eigenschaft des Netzes. Eine Lokation ist reserviert, wenn sie von irgendeiner Transition reserviert ist. Sie kann dabei beliebig of R-reserviert, aber nur einmal W-reserviert werden.

Formaler lässt sich die Reservierung als Operatorpaar $(\mathbf{r}_r, \mathbf{r}_w)$ verstehen, deren Operatoren im Gegensatz zu den Operatoren der Systemsignatur sowohl vor als auch nach dem Schalten einer Transition verändert werden. Für eine Lokation $l^\Omega \in \mathbf{Loc^I}(\mathbf{cmd}(t), \mathfrak{Q}, \alpha)$ wird $\mathbf{r}_r(l^\Omega)$ vor dem Schalten von t um den Wert 1 inkrementiert. Nach dem Schalten wird der Wert dann entsprechend um 1 dekrementiert, so dass $\mathbf{r}_r(l^\Omega)$ stets der Zahl der Transitionen entspricht, die eine Lokation lesereserviert haben. Eine Lokation ist folglich lesereserviert, wenn $\mathbf{r}_r(l^\Omega) > 0$ ist. Analog lässt sich für Schreibreservierungen mit dem Operator $\mathbf{r}_w$ vorgehen. Bei einer Schreibreservierung wird der Operator vor dem Schalten der Transition ebenfalls um 1 inkrementiert und nach dem Schalten entsprechend dekrementiert. Da eine Schreibreservierung nur einmal vorkommen kann, ist $\mathbf{r}_w(l^\Omega) \in \{0, 1\}$. Eine Lokation l^Ω ist schreibreserviert, wenn $\mathbf{r}_w(l^\Omega) = 1$ ist.

Eine Transition entspricht somit drei Aktionen, die auch als drei Transitionen verstanden werden können. Dies ist in Abbildung 4.13 dargestellt. Zunächst werden

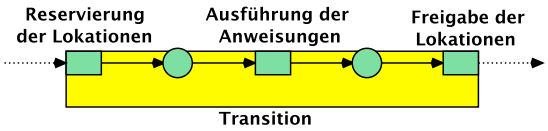


Abbildung 4.13: Die Aktionen einer Transition aufgeteilt in drei Transitionen

sämtliche Lokationen, die gelesen werden, R-reserviert und sämtliche Lokationen, die geschrieben werden, W-reserviert, indem die entsprechenden Operatoren verändert werden. Hieran schließt sich die eigentliche Ausführung der Transition mit den Anschriften an. Abschließend werden die reservierten Lokationen wieder freigegeben. Während die Lokationen reserviert sind, ist sichergestellt, dass sie durch keine andere Transition des Netzes verändert werden.

In der Schaltregel aus Definition 4.21 muss nun nicht mehr der gesamte Systemzustand reserviert werden, sondern es ist ausreichend, dass

- keine der Lokationen in $\mathbf{Loc^I}(\mathbf{cmd}(t), \mathfrak{Q}, \alpha)$ W-reserviert ist und
- keine der Lokationen in $\mathbf{Loc^U}(\mathbf{cmd}(t), \mathfrak{Q}, \alpha)$ R-reserviert ist.

Während des Feuerns einer Transition t ist durch die Transition t dann entsprechend die Lokationsmenge $\mathbf{Loc^I}(\mathbf{cmd}(t), \mathfrak{Q}, \alpha)$ R-reserviert und die Lokationsmenge $\mathbf{Loc^U}(\mathbf{cmd}(t), \mathfrak{Q}, \alpha)$ W-reserviert.

Während des Feuerns ist also die Menge der veränderten Lokationen schreibreserviert und die Menge der gelesenen Lokationen lese-reserviert ist. Zur Aktiviertheit muss die Menge der involvierten Lokationen nicht W-reserviert und die Menge der veränderten Lokationen nicht R-reserviert sein, so dass eine Transition nur aktiviert ist, wenn sie die Updates anderer Transitionen nicht „stört“. Dies entspricht dem transaktionalen Verhalten von Datenbanken, wo ebenfalls gleichzeitiges Lesen erlaubt ist, ein Schreibvorgang den weiteren Zugriff jedoch unterbindet. Es lässt sich leicht überprüfen, dass das Schaltverhalten so implementierter Netze dem Schaltverhalten aus Definition 4.21 entspricht.

Interpretation von Lokationen als Marken

Betrachtet man jede Lokation eines Dienstbeschreibungsnetzes als eigene Marke, die diese Lokation interpretiert, so erhält man in den meisten Fällen ein Netz mit unendlich vielen Marken. Bei diesem Vorgehen wird jedem Paar $(\omega, \overline{q}) \in \Omega^{sys} \times S_Q^n$ eine Marke $((\omega, \overline{q}), q)$ auf einer speziellen Stelle zugewiesen, die die Interpretation der Lokation $(\omega, \overline{q})$ darstellt, sofern dies eine gültige Lokation ist. Hierbei entspricht S_Q^n dann $S_Q \times \ldots \times S_Q$ also dem n-maligen Kreuzprodukt der Mengen und n der Stelligkeit des Operators ω, und es ist q der Wert der Lokation. Die Zahl der

notwendigen Marken ist somit abhängig von der Trägermenge der Algebra S_Q, und es ist zweckmäßig für die durch Netze dargestellten Beispiele in diesem Abschnitt eine entsprechend kleine Algebra zu verwenden.

Neben der Stelle, durch die die Interpretation einer Lokation repräsentiert wird, gibt es Stellen zur Repräsentation der Reservierung, die ebenfalls für jede Lokation eine Marke besitzen, die den Reserviertheitsstatus dieser Lokation angibt und die die Operatoren $\mathbf{r}_r$ beziehungsweise $\mathbf{r}_w$ repräsentiert. Fasst man jede Transition als das in Abbildung 4.13 dargestellte Tripel von Subtransitionen auf, so ergibt sich das in Abbildung 4.14 dargestellte Bild für die Reservierung von Lokationen. Der Übersichtlichkeit halber wurden mehrere Variablen als Tupel ($[x, y]$) dargestellt, anstatt unterschiedliche Stellen für Variablen zu verwenden. Gleichzeitig wurden Paare durch runde Klammern ((a, b)) dargestellt. Die Initialmarkierung des Netzes entspricht dann der Repräsentation des initialen Netzzustandes, wobei keine Lokation reserviert ist.

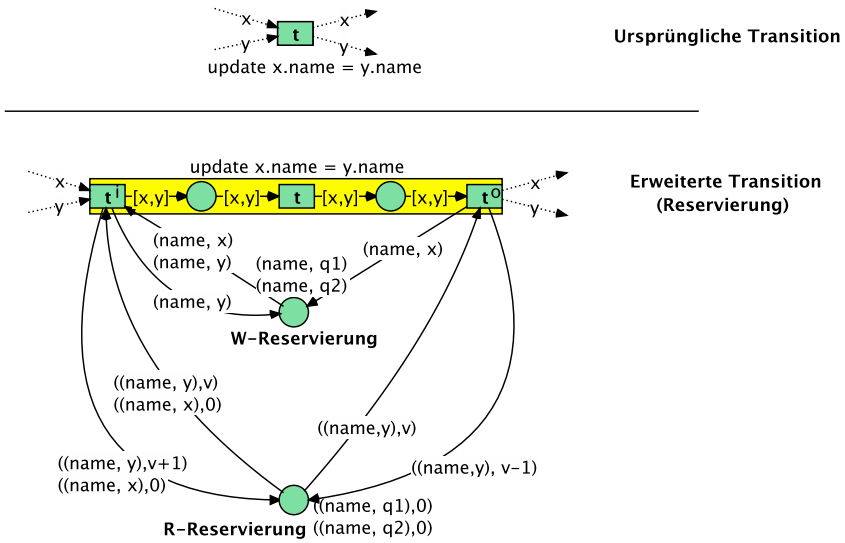


Abbildung 4.14: Reservierung der Lokationen einer Transition des Dienstbeschreibungsnetzes

In diesem Netz schaltet zunächst die Transition t^i, wodurch überprüft wird, ob die Lokation $(name, \alpha(x))$ R-reserviert ist, indem die Kantenanschrift $((name, x), 0)$ überprüft, dass der Wert des Operators $\mathbf{r}_r$ für $(name, \alpha(x))$ den Wert 0 besitzt. Da dieser Wert lediglich gelesen wird, kann er sofort wieder zurückgelegt werden. Ebenso wird überprüft, dass $(name, \alpha(x))$ und $(name, \alpha(y))$ nicht W-reserviert sind, indem auch dort überprüft wird, ob die Marke auf der Stelle vorhanden ist. Ist beides der Fall, so kann die Transition t^i schalten. In diesem Fall wird die Lokation $(name, \alpha(y))$ R-reserviert und die Lokation $(name, \alpha(x))$ W-reserviert. Dies führt

dazu das während der Ausführung der Anweisung *update* $x.name = y.name$ keine andere Transition die veränderten oder gelesenen Lokationen ändern kann, da $(name, \alpha(x))$ nicht auf der Stelle *W-Reservierung* ist und $((name, \alpha(y)), n)$ für ein $n > 0$ auf der Stelle *R-Reservierung* ist.

Im gegebenen Fall gibt es lediglich die beiden Elemente q_1 und q_2 der Algebra. Ist $\alpha(x) = q_1$ und $\alpha(y) = q_2$, so schaltet das Netz und die Lokationen werden reserviert. Ein Sonderfall entsteht, wenn $\alpha(x) = \alpha(y)$ ist, da dann die Eingangskanten von den Reservierungsstellen zu t^i zwei Marken abziehen würden, obwohl nur eine Marke vorhanden ist. Somit müssten diese Sonderfälle durch Aktivierungsbedingungen abgefangen werden, so dass es eine zweite Transition $t^{i,2}$ gibt, die schaltet, wenn $\alpha(x) = \alpha(y)$ ist. In diesem Fall können die Terme mit der Variablen y an den Kanten entfallen. Eine alternative Vorgehensweise wäre die Verwendung eines Operators, der die Multimenge $[\alpha(x)] + [\alpha(y)]$ bildet, wenn $\alpha(x) \neq \alpha(y)$ ist, und andernfalls die Multimenge $[\alpha(x)]$ bildet.

Das Schalten der Transition t führt zu einer Änderung der Lokationen, die ebenfalls durch Stellen repräsentiert werden kann. Besitzt t mehrere Anweisungen, so lassen sich diese durch die Hintereinanderausführung von Transitionen darstellen, was dann der Ausführung des Anweisungsblocks entspricht. Die Veränderung der Lokationen ist in Abbildung 4.15 dargestellt, wobei hier der Übersichtlichkeit halber auf die Darstellung der Reservierung der Lokationen verzichtet wurde, da diese bereits in Abbildung 4.14 dargestellt wurde.

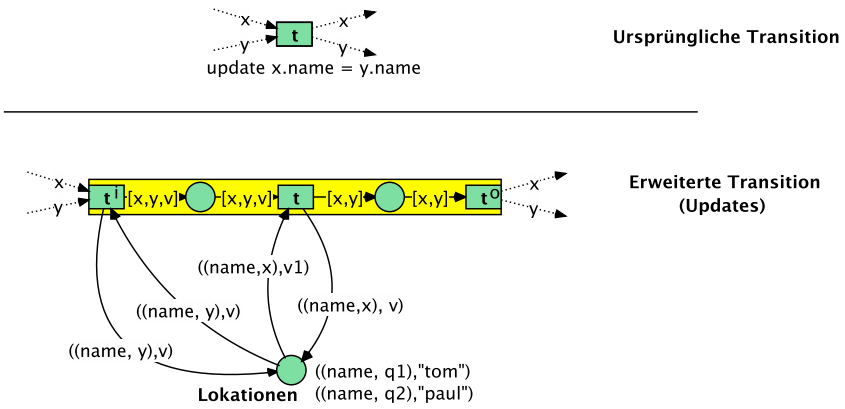


Abbildung 4.15: Update der Lokationen einer Transition in einem Dienstbeschreibungsnetz

Die Transition t^i liest zunächst sämtliche Lokationen aus, die nicht verändert werden. Aufgrund der in Unterabschnitt 4.2.3 diskutierten Belegungskonsistenz ist es stets möglich, sämtliche gelesenen Lokationen initial auszulesen. Hierbei müssen wiederum alternative Transitionen Sonderfälle abfangen, bei denen verschiedene

Terme zur gleichen Lokation auswerten. Das initiale Auslesen der gelesenen Transitionen stellt sicher, dass diese von anderen Transitionen während der Ausführung von t verwendet werden können. Die eigentliche Ausführung der Update-Anweisung erfolgt durch die Transition t. Diese ändert die Lokation $(name, \alpha(x))$. Betrachtet man nun ein reales System, in dem die Berechnung der Funktionen Zeit in Anspruch nimmt, so können die Transitionen t^i und t^o ohne Zeitverzögerung ausgeführt werden, da sie keinerlei Berechnungen durchführen müssen.

Besitzt die Transition t eine Aktivierungsbedingung, so muss diese ebenfalls auf die erste Transition t^i übertragen werden, so dass alle drei Transitionen nur schalten können, wenn die Aktivierungsbedingung erfüllt ist.

Betrachtung von *removeRole*-Amnweisungen

Bei der Verwendung von *removeRole*-Anweisungen müssen neben den Lokationen auch Sortenlokationen durch Transitionen reserviert werden. Die durch Transitionen reservierten Sortenlokationen können dabei so wie im Fall der herkömmlichen Lokationen modelliert werden, indem eine Stelle zur Reservierungen existiert, deren Zähler entsprechend inkrementiert und dekrementiert werden.

Die Reservierung durch Transitionen ist notwendig, da andernfalls ein Objekt der Algebra an eine Variable gebunden sein könnte, dieses Objekt jedoch während der Ausführung seinen Typ ändert, und somit der Typ der Variable ungültig wird. Um dies zu verhindern, reserviert eine Transition sämtliche genutzten Sorten der Eingangsvariablen. Dies ist ausreichend, da sich die Interpretation der Terme bereits durch die Reservierung der Lokationen nicht mehr ändern kann.

Damit Stellen kein aktives Verhalten aufweisen, wird die Reservierung der Sortenlokationen durch die Transition t^o übernommen. Dies ist beispielhaft in Abbildung 4.16 dargestellt. Die Reservierung durch Transitionen wurde durch die Stelle T-Reservierung dargestellt, die Reservierung durch Stellen wurde entsprechend durch die Stelle P-Reservierung dargestellt. Für die Initialmarkierung des Netzes ist zu beachten, dass die Stelle P-Reservierung mit den Stellentypen und ihrer initialen Markierung übereinstimmt.

Im Beispiel in Abbildung 4.16 reserviert t^i zunächst sämtliche dynamischen Sorten der Eingangsvariablen, so dass sich diese nicht mehr verändern können. Gleichzeitig werden die durch Stellen reservierten Sorten freigegeben, indem ihr Wert um 1 dekrementiert wird. Somit sind die Sorten nur noch durch die Transition reserviert. Nach dem Schalten werden die entsprechenden Sorten von der Transition wieder freigegeben. Da im Nachbereich lediglich eine Stelle durch die Sorte *Student* getypt ist, muss auch nur die Sortenlokation $(\alpha(y), \mathit{Student})$ reserviert werden.

Die Ausführung einer *removeRole*-Anweisung entfernt die Sortenlokationen die durch die Anweisung verändert werden, so dass keine andere Transition gleichzeitig auf sie zugreifen kann. Dies ist in Abbildung 4.17 dargestellt.

Auffällig an Abbildung 4.17 ist, dass die veränderten Lokationen nicht reserviert

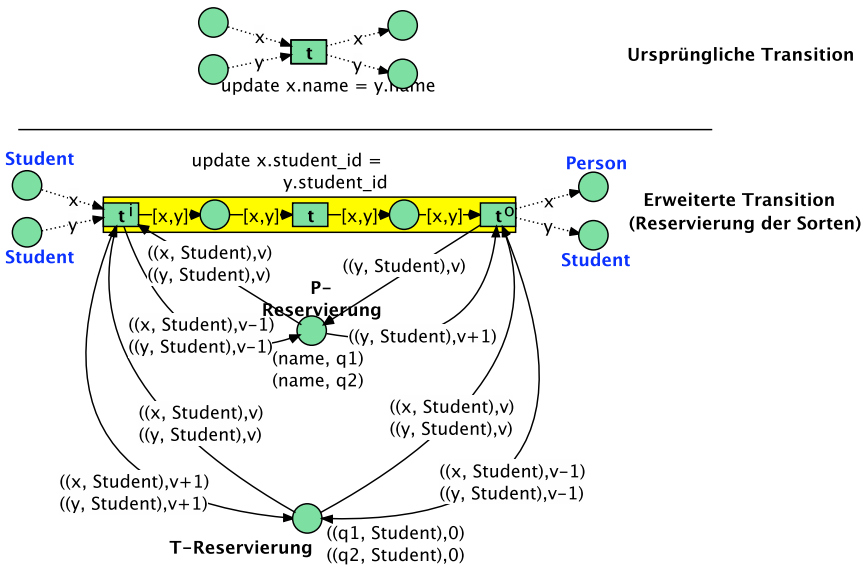


Abbildung 4.16: Update der Sortenlokationen einer Transition des Dienstbeschreibungsnetzes

werden, sondern lediglich überprüft wird, ob sie durch eine andere Stelle oder Transition reserviert sind. Die Reservierung ergibt sich jedoch implizit durch die Reservierung der Updates. Da sämtliche Lokationen $(\omega, \alpha(x))$, für die $\omega \in \Omega^{sys}_{Student,s}$ ist, reserviert worden sind und auch sämtliche Lokationen $(\omega, \overline{q})$, für die $\omega_Q(\overline{q}) = \alpha(x)$ ist, kann keine andere Transition diese Lokationen nutzen. Somit kann es auch keine Zuweisungsanweisungen geben, die einer Variablen vom Typ *Student* den Wert $\alpha(x)$ zuweisen, und es können keine Operatoren von der Sorte Student auf $\alpha(x)$ aufgerufen werden, weswegen diese Reservierung ausreichend ist. Eine explizite Reservierung der Sortenlokation durch das Abziehen der Marke $((x, Student), 1)$, die dann erst wieder durch t^o zurückgelegt wird, expliziert jedoch die Reservierung und ist daher intuitiver.

Nachdem $\alpha(x)$ die Sorte *Student* verloren hat, bleibt eine entsprechende Marke auf den Reservierungsstellen. Auf diese Weise bleibt die Zahl der Marken dort stets gleich. Denkbar wäre auch, diese Marke abzuziehen und erst wieder zurückzulegen, wenn die Sorte wieder zugewiesen wird. Da eine Sorte jedoch mehrfach zugewiesen werden kann, würde letzteres zusätzlichen Verwaltungsaufwand bedeuten, weswegen diese Lösung bevorzugt wurde.

Beispiel 4.4: Das Netz in Abbildung 4.18 nimmt das Netz aus Abbildung 4.5 wieder auf, nur dass diesmal die beiden Vorgänge der Auslieferung und der Rechnungslegung nebenläufig bearbeitet werden.

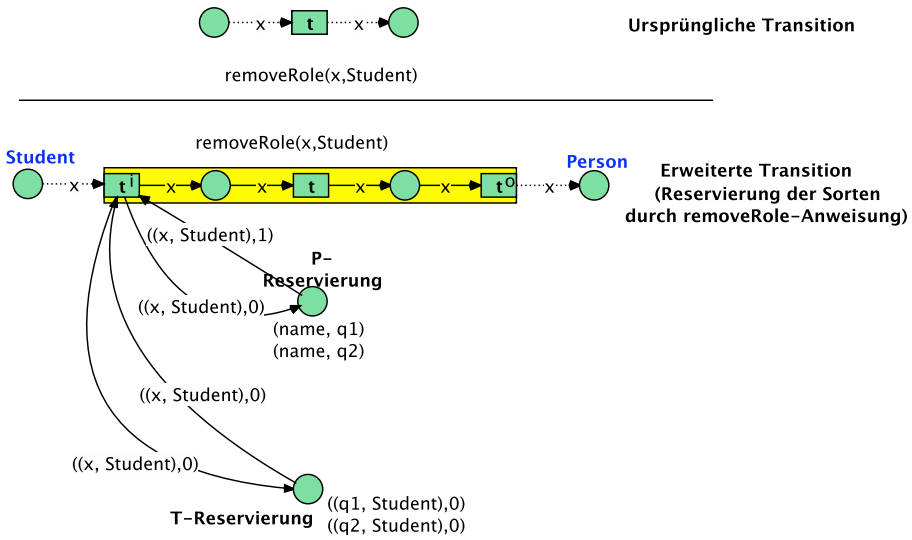


Abbildung 4.17: Die Sortenlokationen einer *removeRole*-Anweisung

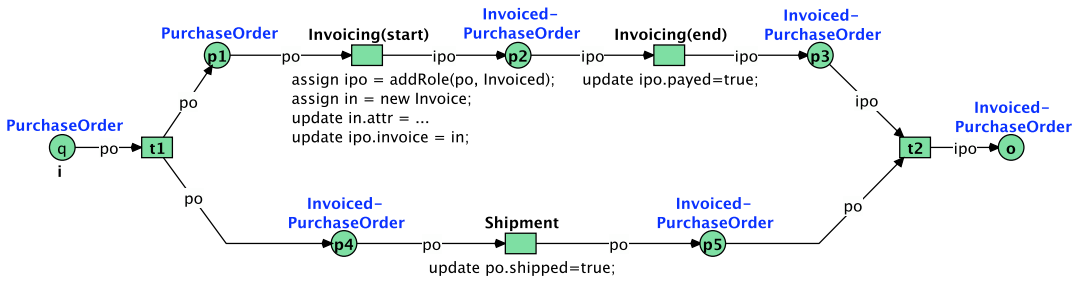


Abbildung 4.18: Ein Bestelleingang mit nebenläufiger Bearbeitung

Nach dem Schalten der Transition t_1, die keinerlei Anweisungen besitzt, markiert die Marke $q \in Q_{PurchaseOrder}$ die Stellen p_1 und p_4. Durch das Feuern der Transition *Invoicing(start)* werden sämtliche Lokationen W-reserviert, die von Anweisungen verändert werden, und sämtliche gelesenen Lokationen R-reserviert. In diesem Fall wird somit die Lokation $(invoice, q)$ W-reserviert. Diese Lokation ist zum Zeitpunkt des Feuerns der Transition im Netzzustand $Q = (\mathbf{m}, \mathfrak{Q})$ keine Lokation von $\mathfrak{Q}$, da q noch nicht von der Sorte *Invoiced* ist. Da es sich bei Lokationen jedoch lediglich um Operatorsymbole und Elemente der Algebra handelt, wird hiervon, wie in der Fußnote auf Seite 115 dargestellt, abstrahiert. Statt die Lokationen zu reservieren wäre es in einem solchen Fall auch möglich, das nebenläufige Hinzufügen der gleichen Sorten zu verbieten, was durch eine Reservierung der Sorten möglich wäre. In den meisten praktischen Fällen dürfte ein Objekt ohnehin nicht nebenläufig die

gleiche Sorte hinzugefügt bekommen, so dass die Art der Umsetzung nicht in dem Maße relevant sein dürfte.

Nach dem Schalten der Transition ist $q \in Q'_{Invoiced}$, da q an die Variable po gebunden wurde. Zudem ist $invoice(q) = q_2$ für ein $q_2 \in Q'_{Invoice}$. Dieses q_2 war in $Q_{\top_{\overline{use}}}$ und ist durch die *new*-Anweisung nun Element von $Q'_{Invoice}$.

Das Feuern der Transition *Shipment* führt zur W-Reservierung der Lokation $(shipped, q)$. Folglich kann es keine Transition geben, die nebenläufig dieses Attribut ausliest oder ändert, was im dargestellten Netz auch nicht der Fall ist. Das Resultat des Schaltens aller Transitionen des Netzes ist dasselbe wie beim Netz in Abbildung 4.5. ◇

4.5 Zusammenfassung

Dieses Kapitel beschreibt den von Köhler und Ortmann (2005) eingeführten Formalismus der Dienstbeschreibungsnetze. Diese Netze erweitern Workflownetze dadurch, dass sie erlauben, die Veränderungen auf einem Systemzustand darzustellen, was über spezielle Anweisungen erfolgt. Anweisungen dienen in Dienstbeschreibungsnetzen als Annotationen an den Transitionen, über die die Attribute von Objekten geändert werden können. Wie für Workflownetze sind auch für Dienstbeschreibungsnetze Kriterien von Interesse, die es ermöglichen zu entscheiden, ob ein Dienstbeschreibungsnetz korrekt modelliert wurde. In diesem Kapitel wurden Dienstbeschreibungsnetze eingeführt und diese Kriterien diskutiert.

Die Einführung von Dienstbeschreibungsnetzen in diesem Kapitel erfolgt zunächst über eine umgangssprachliche Betrachtung der grundlegenden Ideen der Dienstbeschreibungsnetze in Abschnitt 4.1. Die Verwendung eines Systemzustandes, auf dem durch Anweisungen operiert werden kann, macht es notwendig, die Begriffe des Zustandes und der Anweisung zu präzisieren, was in Abschnitt 4.2 erfolgt. Hierbei werden die Konzepte des Zustandes und des Systemzustandes in Unterabschnitt 4.2.1 dargestellt. Ein Zustand stellt eine Algebra dar, die eine Zustandssignatur interpretiert. Durch die Zustandssignatur werden die Typen im Netz als Sorten der Signatur und die Operator- und Konstantensymbole festgelegt, die dann von der Algebra interpretiert werden. Die in Unterabschnitt 4.2.2 eingeführten Anweisungen erlauben die Veränderung des Systemzustandes. Jede Anweisung stellt dabei eine Abbildung von einem Zustand auf einen Folgezustand dar, wie dies in Unterabschnitt 4.2.3 erläutert wird. Diese Interpretation erlaubt es, die Hintereinanderausführung mehrerer Anweisungen als Funktionskomposition zu begreifen. Ein Anweisungsblock stellt eine solche Hintereinanderausführung dar, die der Komposition der Zustandsübergänge der einzelnen Anweisungen entspricht.

Abschnitt 4.3 führt Dienstbeschreibungsnetze ein. Hierzu werden zunächst die statischen Netzeigenschaften in Unterabschnitt 4.3.1 eingeführt, um eine Definition von Dienstbeschreibungsnetzen und ihrem Aufbau geben zu können. Dienst-

beschreibungsnetze bestehen ähnlich wie algebraische Netze aus einer durch eine Signatur beschriebenen Algebra, die durch das Netz verwendet wird. Im Gegensatz zu algebraischen Netzen ist es hierbei jedoch möglich, die Algebra zu verändern. Aus diesem Grund verwenden Markierungen eines Dienstbeschreibungsnetzes keine Terme als Marken, wie dies bei algebraischen Netzen erfolgt. Stattdessen dienen die Objekte der Algebra als Marken auf den Stellen. Dies ermöglicht im Gegensatz zur Verwendung einer Spezifikation auch die Neuinterpretation der Sortenzugehörigkeit von Objekten der Algebra.

Unterabschnitt 4.3.2 widmet sich dem dynamischen Verhalten von Dienstbeschreibungsnetzen und betrachtet die Schaltregel für diese Netzklasse. Hierbei ist zwischen Dienstbeschreibungsnetzen mit *removeRole*-Anweisungen und Dienstbeschreibungsnetzen ohne diesen Anweisungen zu unterscheiden, da erstere ein einfacheres Schaltverhalten aufweisen. Das Entfernen einer Sorte durch eine *removeRole*-Anweisung kann dazu führen, dass die korrekte Typisierung von Stellen nicht mehr länger gewährleistet ist, so dass entsprechende Vorkehrungen notwendig sind, die das Schalten einer Transition nur zulassen, wenn die Stellentypisierung nicht verletzt wird.

Um Dienstbeschreibungsnetze einordnen zu können, setzt Unterabschnitt 4.3.3 das Verhalten von Dienstbeschreibungsnetzen zu anderen Netzklassen in Beziehung. Dies sind einerseits die algebraischen Netze und andererseits die einem Dienstbeschreibungsnetz zugrundeliegenden Workflownetze. Gleichzeitig wird in diesem Unterabschnitt aufgezeigt, dass die Fragen der Erreichbarkeit und der Lebendigkeit des Abschlusses für Dienstbeschreibungsnetze im Allgemeinen unentscheidbar sind. Dies motiviert die im nächsten Kapitel eingeführten elementaren Dienstbeschreibungsnetze, die hinreichende Kriterien für eine Vielzahl von Eigenschaften liefern.

Diese Eigenschaften werden in Abschnitt 4.4 diskutiert. Zu ihnen zählen einmal das Auftreten von Anomalien bei der nebenläufigen Ausführung von Transitionen und einmal das Verklemmen durch *removeRole*-Anweisungen. Unterabschnitt 4.4.1 führt den Begriff der funktionalen Ausführbarkeit ein, der beschreibt, dass für zwei nebenläufig aktivierte Transitionen die Reihenfolge ihres Schaltens für den resultierenden Systemzustand unerheblich ist. Ist ein Dienstbeschreibungsnetze funktional ausführbar, so sind für nebenläufige Transitionen alle Permutation der Ausführung bezüglich des Systemzustandes äquivalent. Unterabschnitt 4.4.2 führt dann die Typisierungskonformität ein, die die Anomalien im Zusammenhang mit *removeRole*-Anweisungen ausschließt. Hierzu wird verlangt, dass es keine von einem initialen Netzzustand erreichbare Markierung in einem Netz geben darf, die eine Transition nicht aktiviert, während eine kleinere Markierung diese Transition aktiviert.

Unterabschnitt 4.4.3 stellt verschiedene Korrektheitskriterien für Dienstbeschreibungsnetze dar, die sich an den Korrektheitskriterien von Workflownetzen orientieren. Die schwache Korrektheit fordert, dass jeder initiale Netzzustand in einen finalen Netzzustand übergehen kann, dass das Netz funktional ausführbar ist und

dass es keine Verletzung der Typisierungskonformität gibt. Die starke Korrektheit geht hierüber hinaus und verlangt zudem, dass sämtliche Transitionen im Dienstbeschreibungsnetz aktivierbar sein müssen. Dies ist jedoch nicht immer sinnvoll und analysierbar, weswegen die schwache Korrektheit diese Eigenschaften nicht besitzt.

Zum Abschluss wird in Unterabschnitt 4.4.4 kurz skizziert, wie sich Dienstbeschreibungsnetze praktisch realisieren lassen. Hierbei kann durch eine Reservierung von Teilen des Systemzustandes die Parallelisierung des Systems erhöht werden, was dem Ziel vieler dienstbasierter Systeme entspricht.

Die Unentscheidbarkeit der Kriterien aus Abschnitt 4.4 im Allgemeinen legt die Einführung einer Subklasse der Dienstbeschreibungsnetze dar, für die diese Eigenschaften überprüft werden können. Dies erfolgt im nächsten Kapitel mit der Einführung elementarer Dienstbeschreibungsnetze.

5 Elementare Dienstbeschreibungsnetze

Die im letzen Kapitel eingeführten Dienstbeschreibungsnetze bieten eine hohe Ausdrucksmächtigkeit, was jedoch dazu führt, dass die Analyse sämtlicher in Abschnitt 4.4 eingeführten Kriterien unentscheidbar ist. Viele der möglichen Ursachen für die Unentscheidbarkeit sind jedoch in praktischen Szenarien eher von untergeordneter Bedeutung. Als Kompromiss zwischen praktischer Nutzbarkeit auf der einen Seite und der Analysierbarkeit auf der anderen Seite werden in diesem Kapitel elementare Dienstbeschreibungsnetze eingeführt. Diese stellen eine Subklasse der Dienstbeschreibungsnetze dar, bei denen die dynamischen Eigenschaften aus Abschnitt 4.4 auf hinreichende Kriterien reduziert werden, die sich aufgrund der Terme an den Netzen und aufgrund des zugrundeliegenden Netzes analysieren lassen.

Nach der Einführung elementarer Dienstbeschreibungsnetze wird anhand von Free-Choice-Dienstbeschreibungsnetzen dargestellt, wie sich die Elementarität überprüfen lässt. Hierzu müssen zunächst die Voraussetzungen im Bereich der Free-Choice-Netze geschaffen werden, bevor dann auf die Umsetzung eingegangen wird.

5.1 Nutzung elementarer Dienstbeschreibungsnetze

Dieses Kapitel führt elementare Dientbeschreibungsnetze ein. Diese Netzklasse ist in vielen praktischen Szenarien ausreichend ausdrucksstark und erfüllt zudem stets die Kriterien der schwachen Korrektheit. Anhand von Dienstbeschreibungsnetzen, deren zugrundeliegendes Netz ein Free-Choice-Netz ist, wird gezeigt, wie sich diese Netzklasse analysieren lässt. Die Reduktion auf Free-Choice-Netze ist ebenfalls für viele praktische Szenarien ausreichend. So beschreiben van der Aalst und van Hee (2004), dass Free-Choice-Workflownetze, die zum Teil um wohlstrukturierte Netzteile ergänzt wurden, ausreichend sind, um die meisten praktisch relevanten Workflows zu modellieren. Dennoch gibt es bei der Behandlung von Ausnahmen Situationen, wo der Einsatz ausdrucksstärkerer Netzklassen sinnvoll sein kann.

Motivation für die Einführung elementarer Dienstbeschreibungsnetze

Elementare Dienstbeschreibungsnetze schränken die Möglichkeiten herkömmlicher Dienstbeschreibungsnetzen bezüglich der möglichen Transitionsanschriften ein. Im Gegenzug sind elementare Dienstbeschreibungsnetze stets schwach korrekt. Übertragen auf Dienste bedeutet dies, dass ein Dienst zu einer eingehenden Nachricht stets eine ausgehende Nachricht erzeugt. Die Einschränkungen, die hierzu getroffen werden, werden im Folgenden dargestellt.

Fortsetzbarkeit: Für die Fortsetzbarkeit wird gefordert, dass es bei Konflikten stets eine aktivierte Transition gibt. Da es nicht möglich ist, dies im Allgemeinen zu überprüfen, da das Problem der Überprüfung im Allgemeinen nicht entscheidbar ist, wird gefordert, dass es eine Transition gibt, die schalten kann, wenn keine andere Transition im Nachbereich einer Stelle schalten kann. Dies entspricht dem default-Fall der switch-Anweisungen vieler Programmiersprachen.
Motivation: Die Fortsetzbarkeit verhindert, dass ein Netz verklemmt, da es keine aktivierte Transition mehr gibt.
Auswirkung: Die Bedingung, dass es zu jeder Menge von Transitionen mit gleichem Vorbereich eine Default-Transition geben muss, die aktiviert ist, wenn keine andere Transition aktiviert ist, ist in vielen Modellierungsansätzen gegeben und stellt weniger eine Einschränkung als vielmehr eine Unterstützung für das Modell dar. Diese Erweiterung fängt Ausnahmesituationen ab und stellt sicher, dass stets ein finaler Netzzustand erreicht wird.

Funktional nebenläufige Ausführbarkeit: Die funktionale Ausführbarkeit wird durch die Einschränkung der veränderbaren Lokationen erreicht. So ist es nicht erlaubt eine Lokation zu nutzen, wenn eine Lokation mit gleichem Operator verändert wird. Hierdurch wird garantiert, dass die funktionale Ausführbarkeit gewährleistet ist.
Motivation: Es soll verhindert werden, dass zwei Transitionen nebenläufig dieselbe Lokation ändern, so dass der Wert der Lokation nach dem Feuern beider Transitionen ausschließlich von der Ausführungsreihenfolge der Transitionen abhängig ist.
Auswirkung: Die Einschränkungen stellen die stärksten unter den hier angegebenen Kritierien dar, da hierdurch nicht nebenläufig die gleichen Attribute zweier Objekte gleichen Typs geändert werden können. Dies ist jedoch die einzige Möglichkeit, die funktionale Ausführbarkeit ohne eine Analyse der Terme im Netz sicherzustellen. Eine Analyse, die die Terme symbolisch mit einbezieht, kann jedoch im schlechtesten Fall unentscheidbar werden, so dass diese Einschränkung in Kauf genommen wird.

Typisierungskonformität: Die Typisierungskonformität wird erreicht, indem es nicht gestattet ist, ein dynamisches Konzept von einem Objekt zu entfernen, wenn gleichzeitig eine markierte Stelle existiert, die dieses Konzept inne hat.
Motivation: Durch die Typisierungskonformität wird verhindert, dass ein Dienstbeschreibungsnetz verklemmt, wenn ein Objekt der Algebra eine dynamische Sorte verliert. Dies sorgt dafür, dass eine Transition schalten kann, wenn alle Stellen im Vorbereich markiert sind und gibt so ein intuitives Verhalten des Netzes wieder.
Auswirkung: Da das Entfernen dynamischer Konzepte in vielen Workflows eher die Ausnahme darstellt ist auch diese Einschränkung eher selten von Bedeutung. Zudem ist es möglich, dieses Problem durch die Nutzung des Cast-Operators zu umgehen, indem die Stelle einfach einen anderen Typ zugewiesen bekommt.

Endlichkeit: Für jede Schaltfolge in einem elementaren Dienstbeschreibungsnetz wird gefordert, dass sie endlich ist. Dies wird durch eine spezielle Endlichkeitsbedingung erreicht.
Motivation: Die Endlichkeit der Schaltfolgen bewirkt auch eine Endlichkeit des symbolischen Erreichbarkeitsgraphen, so dass alle wesentlichen Fragen zumindest für die symbolische Analyse entscheidbar werden. Dennoch ist eine Berechnung von Eigenschaften, die die Terme im Netz betreffen, wie etwa die Erreichbarkeit einer eine Transition aktivierenden Markierung, teilweise auch symbolisch nicht in Polynomialzeit möglich. Die tatsächliche Berechenbarkeit der Korrektheit hängt zudem von der Berechenbarkeit der verwendeten Interpretation der Operatorsymbole der Signatur ab.
Auswirkung: Vielfach werden in Schleifen eines Workflows Listen oder Mengen verarbeitet, so dass in einer Schleife alle Elemente einer Liste betrachtet werden müssen. Diese Schleifen werden dann nur so häufig durchlaufen, wie die Länge der Liste dies vorgibt und sind mithin endlich. Zudem können obere Schranken definiert werden. Ein Abbruch nach der doppelten Zahl von Durchläufen, die als maximal angenommen wird, simuliert eine unendliche Schleife in den meisten Szenarien hinreichend gut.

Diese Bedingungen decken eine Vielzahl der schwachen Korrektheitskriterien für Dienstbeschreibungsnetze aus Definition 4.46 direkt ab, lassen sich aber auf der anderen Seite recht leicht analysieren oder durch eine einfache Ergänzung des Netzes erzwingen, wie im nächsten Abschnitt dargestellt wird. Die Typisierungskonformität und die funktionale Ausführbarkeit erlauben es zudem, ein elementares Dienstbeschreibungsnetz zu sequentialisieren und in eine abstrakte Zustandsmaschine zu überführen. Dies ermöglicht es, die potentiell exponentielle Zahl verschiedener Abläufe zu reduzieren und so eine Analyse möglicher Schaltfolgen zu beschleunigen. Dies wird im nächsten Kapitel dargestellt. Aufgrund der funktionalen Ausführbarkeit ist das Ergebnis des Schaltens einer Schaltfolge von der Reihenfolge nebenläu-

figer Transitionen unabhängig. Daher lassen sich die Ergebnisse der Analyse der abstrakten Zustandsmaschine direkt auf das Dienstbeschreibungsnetz übertragen, was im nächsten Kapitel dargestellt wird.

Für elementare Dienstbeschreibungsnetze lässt sich die Frage der schwachen Korrektheit (Def. 4.46) direkt entscheiden, da die Definition dieser Netzklasse darauf ausgerichtet ist, Verhalten zu verhindern, dass nicht schwach korrekt ist. Zur Analyse starker Korrektheit (Def. 4.47) muss jedoch auch das Verhalten der Daten berücksichtigt werden

Nutzung von Free-Choice-Netzen

Die Überprüfung der Elementarität verlangt verschiedene Kriterien im zugrundeliegenden Workflownetz zu überprüfen, wie die nebenläufige Aktivierbarkeit zweier Transition oder die Korrektheit des Netzes. Diese Eigenschaften lassen sich jedoch für Workflownetze im Allgemeinen nicht mehr in Polynomialzeit überprüfen, so dass die Ausdrucksmächtigkeit der Workflownetze meist auf Subklassen eingeschränkt wird, bei denen sich die Korrektheit effizient analysieren lässt (vgl. van der Aalst (2000)). Aufgrund der Einschränkungen der zugrundeliegenden Workflownetze auf Free-Choice-Netze lassen sich Eigenschaften wie die Nebenläufigkeit zweier Transitionen analysieren, was in diesem Kapitel dargestellt wird. Die zu analysierenden Eigenschaften sind dabei wie folgt:

Korrektheit: Bei der Modellierung eines Ablaufs mit Daten sollte das zugrundeliegende Netz korrekt sein, so dass es innerhalb des Netzes nicht zu Verklemmungen kommen kann. Zwar lässt sich der Steuerfluss mit Hilfe der Daten so manipulieren, dass nicht korrekte Workflownetze ebenfalls nicht verklemmen, jedoch ist dies meist kontraintuitiv und wird daher von elementaren Dienstbeschreibungsnetzen nicht unterstützt.

Nebenläufiges Schalten zweier Transitionen: Das nebenläufige Schalten zweier Transitionen sollte nicht zu Race-Conditions führen, bei denen sich zwei nebenläufig ausgeführte Aktionen gegenseitig überschreiben. Um dies zu analysieren muss entschieden werden können, ob zwei Transitionen nebenläufig aktivierbar sind.

Gleichzeitige Markierbarkeit: Eng verwandt mit der nebenläufigen Aktivierbarkeit ist die Frage, ob eine Stelle markiert sein kann, wenn gleichzeitig eine Transition schaltet. Dies ist vor dem Hintergrund der Typisierungskonformität interessant.

Um diese Eigenschaften analysieren zu können, müssen Einschränkungen bei der Ausdruckskraft von Workflownetzen hingenommen werden, da bereits die Analyse der Korrektheit für Workflownetze im Allgemeinen nicht mehr in Polynomialzeit

möglich ist. Im Idealfall forcieren diese Einschränkungen gleichzeitig eine „sinnvolle Modellierung“. Dieser Begriff ist jedoch recht dehnbar und muss daher präzisiert werden. Bei van der Aalst und van Hee (2004) liegt der Fokus hierbei auf wohlstrukturierten und Free-Choice-Netzen (S. 286ff). Die Autoren gehen dort sogar so weit zu behaupten, dass ein Konstrukt, das keiner dieser beiden Netzklassen zuzuordnen ist, in der Regel eine Fehlerquelle darstellt (a.a.O. S. 286). Aus diesem Grund werden Free-Choice-Netze als zugrundeliegende Netze angenommen, auf denen dann analysiert wird, ob ein Netz ein elementares Dienstbeschreibungsnetz ist.

Gemäß Esparza (1998b) ist die Erreichbarkeit einer Markierung bereits für beschränkte Free-Choice-Netze im Allgemeinen nicht mehr in polynomieller Zeit entscheidbar. Für Free-Choice-Netze, bei denen die Initialmarkierung stets wieder erreichbar ist (Home Marking), ist die Erreichbarkeit hingegen in Polynomialzeit entscheidbar (vgl. (Desel und Esparza, 1995, Theorem 9.6)). Der Abschluss korrekter Workflownetze besitzt die Eigenschaft einer stets wieder erreichbaren Initialmarkierung, so dass zentrale Eigenschaften wie die Erreichbarkeit einer Markierung und die Korrektheit des Workflownetzes in Polynomialzeit entschieden werden können.

Analyse von Free-Choice-Dienstbeschreibungsnetzen

Die Analyse der Free-Choice-Workflownetze erfolgt über die minimalen T-Invarianten beziehungsweise über die T-Komponenten des Abschlusses. T-Komponenten und minimale T-Invarianten sind für Free-Choice-Workflownetze eng miteinander verwandt, da das Subnetz jeder minimalen T-Invariante des Abschlusses eines Free-Choice-Workflownetzes eine T-Komponente ist. Abschnitt 5.3 beginnt mit der Analyse der Korrektheit durch sukzessives Hinzufügen von T-Komponenten zu einem Free-Choice-Workflownetz. Gegenüber der Anwendung des Rang-Theorems hat dieses Vorgehen die Vorteile, dass zum einen eine Basis minimaler T-Invarianten während der Analyse erzeugt wird und zum anderen detaillierter zurückgekoppelt werden kann, wo sich ein Fehler im Modell befindet.

Das Verfahren bedient sich dabei einer Überdeckung durch linear unabhängige T-Komponenten beziehungsweise S-Komponenten, wie sie durch das Verfahren von Kemper (1993) für wohlgeformte Free-Choice-Netze effizient berechnet werden können. Dieses Verfahren berechnet zunächst nur minimale Siphons. Diese stellen jedoch in einem wohlgeformten Free-Choice-Netz S-Komponenten und somit auch minimale S-Invarianten dar. Da das duale Netz eines wohlgeformten Free-Choice-Netzes wiederum ein wohlgeformtes Free-Choice-Netz ist (vgl. Satz 3.62), lassen sich auf diesem Wege auch die T-Komponenten einer T-Überdeckung ermitteln.

Wendet man das in Abschnitt 5.3.1 vorgestellte Verfahren auf das duale Netz an, so kann ebenfalls die Wohlgeformtheit überprüft werden. Gleichzeitig stellt die hierdurch erzeugte Basis minimaler T-Invarianten des dualen Netzes eine Basis minimaler S-Invarianten des Ursprungsnetzes dar. Diese Basis kann dann dazu verwendet werden, zu überprüfen, ob der Abschluss eines Free-Choice-Workflownetzes

lebendig ist. Darüber hinaus kann die erzeugte Basis minimaler S-Invarianten im weiteren Verlauf für die Ermittlung der nebenläufigen Aktivierbarkeit zweier Transitionen genutzt werden. Zwar lässt sich eine Basis minimaler Invarianten auch direkt über die Inzidenzmatrix berechnen, jedoch existieren hierfür nur Verfahren, die allgemein auf Netzen arbeiten und daher unter Umständen nicht in polynomieller Zeit ausführbar sind, wie etwa das Verfahren von Martinez und Silva (1981).

Als Ergänzung zu einer Analyse über die Invarianten bietet auch die Reduktion des Netzes eine schnellere Analyse, da mehrere Operationen (notiert als Transitionsanschriften) zu einer Transition mit entsprechenden Anschriften zusammengefasst werden können. Entsprechende Reduktionsregeln für Petrinetze, die auf Murata (1989) basieren, werden in Abschnitt 5.3.3 diskutiert. Die Reduktionsregeln erlauben zudem elementare Teilnetze eines allgemeinen Dienstbeschreibungsnetzes zu reduzieren.

Aufgrund der gezeigten Eigenschaften von Free-Choice-Workflownetzen kann die Überprüfung der Elementarität von Free-Choice-Dienstbeschreibungsnetzen nun relativ einfach anhand der bereitgestellten Ergebnisse erfolgen. Hierzu wird mit den Mitteln aus Abschnitt 5.3 zunächst die Netzstruktur untersucht. Die Free-Choice-Eigenschaft und die Workfloweigenschaft können leicht durch bekannte Verfahren überprüft werden. Die Korrektheit kann entweder über das Rang-Theorem von Desel und Esparza (1995) oder über das Vorgehen aus Unterabschnitt 5.3.2 in Erfahrung gebracht werden. Eine vorherige Reduktion des Netzes durch die Reduktionsregeln aus Unterabschnitt 5.3.3 ermöglicht eine schnellere Realisierung dieser Überprüfung. Die Verwendung des Verfahrens aus Unterabschnitt 5.3.3 bietet dabei den Vorteil, dass gleich eine Basis minimaler S-Invarianten zur weiteren Analyse bereitsteht.

Die Typisierungskonformitätsbedingung und die Funktionalitätsbedingung aus Definition 5.10 lassen sich ebenfalls über die nebenläufige Aktivierbarkeit von Transitionen beziehungsweise über die nebenläufige Markierbarkeit von Stellen zeigen, deren Berechnung in Unterabschnitt 5.3.2 thematisiert wird. Die Endlichkeit kann über die Endlichkeitsbedingung nachgewiesen werden. Der Nachweis der Elementarität kann dann mit den bestehenden Ergebnissen erfolgen. Dies wird in Unterabschnitt 5.4.2 gezeigt.

5.2 Elementaritätskriterien für Dienstbeschreibungsnetze

Elementare Dienstbeschreibungsnetze schränken die Ausdrucksmächtigkeit allgemeiner Dienstbeschreibungsnetze ein und erlauben im Gegenzug eine weitergehende Analysierbarkeit. Hierzu werden einerseits die Typisierungskonformität und die funktionale Ausführbarkeit aufgrund der Anweisungen an den Transitionen ermit-

telt. Gleichzeitig wird die Fortsetzbarkeit und die Endlichkeit durch weitergehende Forderungen an die Dienstbeschreibungsnetze erzwungen. Dies führt dazu, dass elementare Dienstbeschreibungsnetze stets schwach korrekt, endlich und deterministisch, was am Ende dieses Abschnitts gezeigt wird, nachdem die Kriterien der Typisierungskonformität und funktionalen Ausführbarkeit sowie die Kriterien der Endlichkeit und der Fortsetzbarkeit eingeführt wurden.

5.2.1 Typisierungskonformität und funktionale Nebenläufigkeit

Typisierungskonformität und funktionale Ausführbarkeit sind beides Kriterien, für die hinreichende Bedingungen anhand der Erreichbarkeit im zugrundeliegenden Netz und anhand der Anweisungen an Transitionen angegeben werden können. Hierzu wird zunächst im zugrundeliegenden Netz ermittelt, ob überhaupt eine Markierung erreichbar sein kann, die zu einer Verletzung der Kriterien für Typisierungskonformität und funktionale Ausführbarkeit führen könnte. Schließt man sämtliche solchen Markierungen aus, so ist dies eine hinreichende Bedingung für die beiden Kriterien. Dies wird im Folgenden näher erläutert.

Hinreichende Kriterien für Typisierungskonformität

Eine hinreichende Bedingung für die Typisierungskonformität eines Dienstbeschreibungsnetzes in jedem initialen Netzzustand besteht darin, dass es nicht gestattet wird, dass im zugrundeliegenden Netz gleichzeitig eine Stelle der Sorte s markiert ist und eine Transition aktiviert ist, die diese Sorte von einem Objekt entfernt. Aus Lemma 4.40 folgt in diesem Fall, dass ein solches Netz in jedem Fall typisierungskonform ist. Dies soll hier präzisiert werden.

Definition 5.1 (Hinreichende Typisierungskonformitätsbedingung)
Ein Dienstbeschreibungsnetz $\mathcal{SDN}$ erfüllt die *hinreichende Typisierungskonformitätsbedingung* gdw. für $t \in T$ mit $(\mathit{removeRole}(\tau, s_{\mathfrak{c}})) \in \textsc{Alph}(\mathbf{cmd}(t))$ gilt, dass kein $\mathbf{m} \in [[i]\rangle_{\mathcal{N}}$ existiert, so dass $\mathbf{m}(p_*) > 0$ für ein $p_* \in P \setminus {}^\bullet t$ mit $d(p) \preceq s_{\mathfrak{c}}$ und für alle $p \in {}^\bullet t$ gilt $\mathbf{m}(p) > 0$. ⧫

Die hinreichende Typisierungskonformitätsbedingung ist tatsächlich hinreichend für die Typisierungskonformität, wie folgendes Lemma zeigt.

Lemma 5.2 (Hinreichende Typisierungskonformitätsbedingung)
Ein Dienstbeschreibungsnetz $\mathcal{SDN}$, das die hinreichende Typisierungskonformitätsbedingung erfüllt, ist in jedem initialen Netzzustand typisierungskonform.

Beweis:
Nach Lemma 4.40 ist zu zeigen, dass es für keinen initialen Netzzustand $\mathfrak{Q}_0$ einen er-

reichbaren Netzzustand $(\mathbf{m}, \mathfrak{Q}) \in [Q_0\rangle_{\mathcal{SDN}}$ gibt, so dass für ein $p \in P$ und ein $t \in T$ gilt $\mathbf{m}(p) \neq \mathbf{0}$, $d(p) \preceq s_\mathfrak{c}$ für ein $s_\mathfrak{c} \in S^{dyn}$ und $(removeRole(\tau, s_\mathfrak{c})) \in \textsc{Alph}(\mathbf{cmd}(t))$ und $(\mathbf{m}, \mathfrak{Q})$ die Transition t aktiviert, so ist $\mathcal{SDN}$ in $\mathfrak{Q}_0$ typisierungskonform. Wäre dies der Fall, so müsste nach Korollar 4.29 entsprechend die Markierung $\mathring{\mathbf{m}} \in [[i]\rangle_\mathcal{N}$ sein. Dies widerspricht jedoch der Bedingung aus Definition 5.1 und daher ist die Bedingung hinreichend. □

Hinreichende Kriterien für funktionale Ausführbarkeit

Dass die im letzten Kapitel in Unterabschnitt 4.4.1 beschriebene Eigenschaft der funktionalen Ausführbarkeit stets erfüllt ist, ist eines der Merkmale elementarer Dienstbeschreibungsnetze. Eine hinreichende Bedingungen dafür ist gegeben, wenn folgende Kriterien erfüllt sind, wobei angenommen wird, dass es zwei Schaltmodi (t_1, α_1) und (t_2, α_2) gibt, so dass die Transitionen t_1 und t_2 im Netzzustand Q aktiviert sind. Für $i, j \in \{1, 2\}$ mit $i \neq j$ gilt dann:

1. Wenn die Aktivierungsbedingung von t_i in Q wahr ist, muss nach dem Feuern von t_j die Aktivierungsbedingung von t_i noch immer wahr sein.

2. Wenn ein Term eine Lokation oder den Wert einer Ausgangsvariablen in $\mathbf{cmd}(t_i)$ bestimmt, darf die Auswertung dieses Terms durch t_j nicht verändert werden.

3. Wenn ein Term den Wert einer Lokation in $\mathbf{cmd}(t_i)$ bestimmt, darf die Auswertung dieses Terms durch t_j nicht verändert werden.

Wenn also keine der Lokationen, die gelesen werden, um eine Ausgangsvariable zu interpretieren oder um eine Lokation zu ändern, durch eine andere Transition verändert wird, so bleibt das Schaltverhalten einer Transition unabhängig davon, ob die zweite Transition zuvor feuert, gleich. Sind darüber hinaus die gelesenen und die veränderten Lokationen zweier Transitionen disjunkt, so können zwei Transitionen stets gleichzeitig feuern, wenn sie nebenläufig aktiviert sind.

Ein in jedem Fall hinreichendes Kriterium dafür, dass zwei Transitionen nicht dieselben Lokationen lesen und ändern, ist die Disjunktheit der gelesenen und der veränderten Operatorsymbole. In Abschnitt 4.2.2 wurde die Menge der gelesenen Lokationen für einen Term τ definiert als $\mathbf{Loc}^\mathbf{I}(\tau, \mathfrak{Q}, \alpha) \stackrel{\text{def}}{=} \{(\omega, \overline{q}) \mid \omega(\tau_1, \ldots, \tau_n) \in \textsc{Subterm}(\tau) \wedge \omega \in \Omega^{sys} \wedge \overline{q} = [\![\tau_1]\!]^\mathfrak{Q}_\alpha, \ldots, [\![\tau_n]\!]^\mathfrak{Q}_\alpha\}$. Wird nun keine Lokation mit dem Operatorsymbol ω in einer nebenläufig schaltenden Transition geändert, so ist die funktionale Ausführbarkeit gewährleistet, sofern sich nicht die Aktivierungsbedingung ändert. Somit muss diese ebenfalls als gelesener Term betrachtet werden.

Um diese Überlegungen zu formalisieren, werden die beiden Mengen der gelesenen und der veränderten Operatoren eingeführt. Hierzu wird zunächst die Menge aller

Terme benötigt, die gegeben ist durch:

$$\begin{aligned}
\textsc{Term}_{\textsc{Read}}(\mathit{update}\ \omega(\tau_1,\ldots,\tau_n)=\tau) &\stackrel{\text{def}}{=} \{\tau,\tau_1,\ldots,\tau_n\}\\
\textsc{Term}_{\textsc{Read}}(\mathit{addRole}(\tau,s_{\mathfrak{c}})) &\stackrel{\text{def}}{=} \{\tau\}\\
\textsc{Term}_{\textsc{Read}}(\mathit{removeRole}(\tau,s_{\mathfrak{c}})) &\stackrel{\text{def}}{=} \{\tau\}\\
\textsc{Term}_{\textsc{Read}}(\mathit{new}\ (v,s_{\mathfrak{c}})) &\stackrel{\text{def}}{=} \{\}\\
\textsc{Term}_{\textsc{Read}}(\mathit{assign}\ v=\tau) &\stackrel{\text{def}}{=} \{\tau\}
\end{aligned}$$

Für einen Anweisungsblock **Cmd** ist $\textsc{Term}_{\textsc{Read}}(\mathbf{Cmd})$ entsprechend die Vereinigung aller $\textsc{Term}_{\textsc{Read}}(\mathit{cmd})$ der Anweisung *cmd* der Folge **Cmd**. (Dies entspricht $\textsc{Term}_{\textsc{Read}}(\mathbf{Cmd}) \stackrel{\text{def}}{=} \bigcup_{\mathit{cmd}\in\textsc{Alph}(\mathbf{Cmd})} \textsc{Term}_{\textsc{Read}}(\mathit{cmd})$.) Für die Transition t ist $\textsc{Term}_{\textsc{Read}}(t) \stackrel{\text{def}}{=} \textsc{Term}_{\textsc{Read}}(\mathbf{Cmd}) \cup G(t)$ und berücksichtigt zudem noch die Aktivierungsbedingung. Bleibt die Interpretation aller Terme in $\textsc{Term}_{\textsc{Read}}(t)$ während des Schaltens jeder nebenläufig zu t aktivierten Transition gleich, so ist das Netz funktional ausführbar, wenn keine Lokation von beiden Transitionen geändert wird, da sowohl die Aktiviertheit weiterhin gegeben ist als auch das Ergebnis des Schaltens gleich bleibt. Diese Bedingung ist nicht in jedem Fall notwendig, ist jedoch in jedem Fall hinreichend[1].

Neben den verwendeten Termen muss auch für die veränderten Lokationen sichergestellt sein, dass die Veränderung unabhängig von der Reihenfolge des Feuerns ist. Eine hinreichende Bedingung hierfür ist, dass sämtliche geänderten Lokationen der beiden Transitionen ebenfalls verschieden sind und dass die Werte, mit denen die Lokationen verändert werden, gleich bleiben. Es wäre jedoch auch denkbar, dass Lokationen durch beide Transitionen geändert werden, wobei die Reihenfolge der Änderung für das Ergebnis unerheblich ist. Dies wird hier jedoch nicht weiter verfolgt.

Da allgemeine Konsistenzkriterien meist schwierig in der Praxis zu überprüfen sind, nutzen elementare Dienstbeschreibungsnetze Kriterien, die sich leichter anhand der Syntax des Netzes überprüfen lassen. Diese verlangen, dass ein geschriebener Operator nicht nebenläufig gelesen werden kann und dass ein veränderter Operator nur durch eine Transition nebenläufig verändert werden kann, was eine Disjunktheit der Lokationen garantiert. Die Kriterien orientieren sich an den von Bernstein (1966) dargestellten Kriterien für die Synchronisation von parallen Zugriffen auf eine Datenbank.

[1] Nicht notwendig wäre diese Bedingung beispielsweise, wenn ein Objekt nach Auswertung eines Terms eine Rolle hinzubekommt, die es bereits inne hat, und durch das Feuern einer nebenläufigen Transition der Term anders ausgewertet wird, und nun ein anderes Objekt diese Rolle erhält, und dieses Objekt ebenfalls bereits diese Rolle inne hat. In diesem Fall müssten jedoch beide Updates für alle erreichbaren Netzzustände stets trivial sein, und somit sind die Updates überflüssig.

Definition 5.3 (Mengen der gelesenen und veränderten Operatoren) Gegeben ein Dienstbeschreibungsnetz $\mathcal{SDN}$. Für eine Transition $t \in T$ ist die Menge der gelesenen Operatoren $\mathrm{R}_\Omega(t)$ definiert als

$$\mathrm{R}_\Omega(t) \stackrel{\text{def}}{=} \bigcup_{\tau \in \textsc{Term}_{\textsc{Read}}(t)} \left\{ \omega \in \Omega^{sys}_{s,s'} \mid \exists \tau_1, \ldots, \tau_n \in \mathbb{T}_{\Sigma^{\mathfrak{Q}},s}(X)) : (\omega(\tau_1, \ldots, \tau_n)) \in \textsc{Subterm}(\tau) \right\}$$

Die Menge der veränderten Operatoren $\mathrm{W}_\Omega(t)$ ist für $C = \textsc{Alph}(\mathbf{cmd}(t))$ definiert als

$$\begin{aligned} \mathrm{W}_\Omega(t) \stackrel{\text{def}}{=} & \left\{ \omega \in \Omega^{sys} \mid (\mathit{update}\ \omega(\tau_1, \ldots, \tau_n) = \tau) \in C \right\} \cup \\ & \left\{ \omega \in \Omega^{sys}_{s_1 \ldots s_n, s} \mid \mathit{removeRole}(\tau, s_{\mathfrak{c}}) \in C \wedge ((\bigvee_{j=1}^{n} s_j \preceq s_{\mathfrak{c}}) \vee s \preceq s_{\mathfrak{c}}) \right\} \cup \\ & \left\{ (s) \in \Omega^{sys}_{\perp^{\mathfrak{Q}}, bool} \mid \mathit{removeRole}(\tau, s_{\mathfrak{c}}) \in C \wedge s \preceq s_{\mathfrak{c}} \right\} \cup \\ & \left\{ (s) \in \Omega^{sys}_{\perp^{\mathfrak{Q}}, bool} \mid \mathit{addRole}(\tau, s_{\mathfrak{c}}) \in C \wedge s_{\mathfrak{c}} \preceq s \preceq s' \wedge \tau \in \mathbb{T}_{\Sigma^{\mathfrak{Q}},s'}(X) \right\} \end{aligned}$$

♦

Für eine Transition t stellt folglich $\mathrm{R}_\Omega(t)$ die Menge der von Anweisungen in $\mathbf{cmd}(t)$ gelesenen Operatorinterpretationen dar und $\mathrm{W}_\Omega(t)$ die Menge der von Anweisungen in $\mathbf{cmd}(t)$ veränderten Operatorinterpretationen. Letztere beinhalten auch diejenigen Lokationen, die durch eine Konzeptänderungsanweisung geändert wurden, da diese ebenfalls geändert wurden. Insbesondere sind auch die veränderten Cast-Operatoren Element dieser Menge.

Vereinfacht gesprochen sind zwei nebenläufig aktivierte Transitionen t_1 und t_2 nun funktional nebenläufig aktiviert, wenn kein von t_1 gelesener Operator von t_2 geändert wird, kein von t_2 gelesener Operator von t_1 geändert wird und wenn keine Operatoren von beiden Transitionen geändert werden. Wie am Ende von Unterabschnitt 4.2.2 behandelt wurde, werden Anweisungen, die die Sorte eines Objekts abfragen oder die diese überprüfen, durch eine Sortenänderung neu interpretiert, so dass eine separate Betrachtung der Sorten nicht notwendig ist, da jede Sorte mindestens einen veränderten Operator mit sich bringt: den Cast-Operator (s). Aus diesem Grund ist der Cast-Operator Element von $\mathrm{W}_\Omega(t)$, da so verhindert wird, dass eine Sorte eines Objektes nebenläufig hinzugefügt und entfernt wird. Diese Operatoren werden dann bereits durch die Disjunktheit der Mengen R_Ω und W_Ω überprüft.

Für ein elementares Dienstbeschreibungsnetz wird die Disjunktheit dieser Mengen gefordert, was sich wie folgt beschreiben lässt. Für zwei Transitionen $t_1, t_2 \in T$ impliziert die Existenz einer Markierung $\mathbf{m} \in [[i]\rangle_{\mathcal{N}}$, die beide Transitionen nebenläufig in $\mathcal{N}$ aktiviert, dass

- $\mathrm{R}_\Omega(t_1) \cap \mathrm{W}_\Omega(t_2) = \emptyset$,
- $\mathrm{R}_\Omega(t_2) \cap \mathrm{W}_\Omega(t_1) = \emptyset$ und
- $\mathrm{W}_\Omega(t_1) \cap \mathrm{W}_\Omega(t_2) = \emptyset$.

Dies ist ausreichend, da stets eine Markierung $\mathbf{m} \in [[i]\rangle_{\mathcal{N}}$ existieren muss, die beide Transitionen nebenläufig in $\mathcal{N}$ aktiviert, damit ein entsprechender Netzzustand in $\mathcal{SDN}$ erreichbar sein kann. Sind die entsprechenden Mengen dann stets disjunkt, so garantiert dies die funktionale Ausführbarkeit. Dies soll an einem Beispiel veranschaulicht werden.

Beispiel 5.1: Ist eine Transition t_1 mit einer Anweisung *assign* $v = preis(v_1)/2$ annotiert, so ist $preis(v_1) \in \textsc{Term}_{\textsc{Read}}(t_1)$ und $preis \in \mathrm{R}_\Omega(t_1)$, da der Operator *preis* auf den Subterm v_1 angewandt wurde. Gibt es jetzt eine nebenläufig aktivierbare Transition t_2 mit der Anschrift *update* $preis(v_2) = 12$, so ist $preis \in \mathrm{W}_\Omega(t_2)$. Folglich ist $\mathrm{R}_\Omega(t_1) \cap \mathrm{W}_\Omega(t_2) \neq \emptyset$ und somit die hinreichende Bedingung für die funktionale Ausführbarkeit verletzt. Es kann nun zwar v_1 und v_2 unterschiedlich interpretiert werden, so dass verschiedene Lokationen geändert werden, jedoch ist das Kriterium in jedem Fall hinreichend. Abbildung 5.1 stellt die Situation anhand eines Netzes dar.

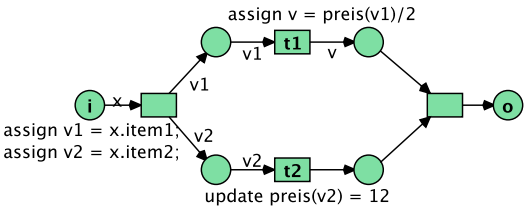


Abbildung 5.1: Zugriff auf Lokationen mit gleichem Operatorsymbol

◇

Definition 5.4 (Hinreichende Funktionalitätsbedingung)
Ein initialbeschriftetes Dienstbeschreibungsnetz $\mathcal{SDN}_{init}$ erfüllt die *hinreichende Funktionalitätsbedingung*, wenn es variablendeterminiert ist und für je zwei Transitionen $t_1, t_2 \in T$ die Existenz einer Markierung $\mathbf{m} \in [[i]\rangle_{\mathcal{N}}$ mit $\mathbf{m} \xrightarrow[\mathcal{N}]{\{t_1,t_2\}}$ impliziert, dass

- $\mathrm{R}_\Omega(t_1) \cap \mathrm{W}_\Omega(t_2) = \emptyset$,
- $\mathrm{R}_\Omega(t_2) \cap \mathrm{W}_\Omega(t_1) = \emptyset$ und

- $\mathrm{W}_\Omega(t_1) \cap \mathrm{W}_\Omega(t_2) = \emptyset$.

⧫

Wiederum soll gezeigt werden, dass die hinreichende Funktionalitätsbedingung tatsächlich hinreichend ist, damit ein Dienstbeschreibungsnetz in jedem initialen Netzzustand funktional ausführbar ist.

Lemma 5.5 (Hinreichende Funktionalitätsbedingung)
Ein initialbeschriftetes Dienstbeschreibungsnetz $\mathcal{SDN}_{init}$ ist in jeder Initialisierung funktional ausführbar, wenn es die hinreichende Funktionalitätsbedingung aus Definition 5.4 erfüllt.

Beweis:
Erfülle $\mathcal{SDN}_{init}$ die hinreichende Funktionalitätsbedingung aus Definition 5.4 und sei $Q^{\mathbf{In}}$ eine Initialisierung von $\mathcal{SDN}_{init}$. Wäre $\mathcal{SDN}_{init}$ nicht in $Q^{\mathbf{In}}$ funktional ausführbar, so müsste für ein $Q = (\mathbf{m}, \mathfrak{Q}) \in [Q^{\mathbf{In}}\rangle$ gelten, dass zwei in Q nebenläufig aktivierte Schaltmodi (t_1, α_1) und (t_2, α_2) existieren, ohne dass diese in Q funktional nebenläufig aktivierbar sind. Dann wäre eine der beiden Bedingungen aus Definition 4.36 verletzt. Würde $Q \xrightarrow{(t_k,\alpha_k)} Q'$ impliziert $Q' \xrightarrow{(t_l,\alpha_l)}$ für $k, l \in \{1, 2\}, k \neq l$ nicht gelten, so wäre eine Transition nach dem Schalten der anderen nicht mehr aktiviert. Da aufgrund der nebenläufigen Aktiviertheit die Zahl der Marken im Vorbereich ausreichend ist, müsste sich die Auswertung der Aktivierungsbedingung geändert haben. Dies würde bei gleicher Variablenbelegung jedoch bedeuten, dass sich der Wert eines Operators geändert haben muss. Dann würde jedoch ein in der Aktivierungsbedingung vorkommender Operator neu interpretiert, und da dieser dann in $\mathrm{R}_\Omega(t_l)$ wäre, wäre $\mathrm{R}_\Omega(t_l) \cap \mathrm{W}_\Omega(t_k) \neq \emptyset$ und somit die Bedingung verletzt. Da die Cast-Operatoren ebenfalls Teil der Mengen sind, gilt dies auch für neu hinzugewonne und entfernte Sorten.

Würde hingegen $Q \xrightarrow{(t_1,\alpha_1)} Q_1$ und $Q \xrightarrow{(t_2,\alpha_2)} Q_2$ impliziert $Q_1 \xrightarrow{(t_2,\alpha_2)} Q_*$ und $Q_2 \xrightarrow{(t_1,\alpha_1)} Q_*$ nicht gelten, so wäre zwar die jeweils andere Transition aktiviert, jedoch wäre unter gleicher Variablenbelegung das Ergebnis der Schaltfolge abhängig von der Schaltreihenfolge. Dann müsste sich ebenfalls die Auswertung der Terme und somit die Interpretation eines Operators ändern, was nach obiger Argumentation die Bedingungen verletzt. Somit ist auch die Folgemarkierung gleich, da den Variablen die gleichen Werte zugewiesen werden. □

Die hinreichende Funktionalitätsbedingung liefert also eine hinreichende Bedingung dafür, dass ein Dienstbeschreibungsnetz funktional ausführbar ist. Zusammen mit der hinreichenden Bedingung für die Typisierungskonformität ermöglicht dies bereits zwei der Kriterien für schwache Korrektheit allein aufgrund der zugrundeliegenden Netze und der Anweisungen der Transitionen zu überprüfen. Fortsetzbarkeit

und Termination werden im Folgenden betrachtet.

5.2.2 Endlichkeit und Fortsetzbarkeit

Neben der Bedingung, dass ein Dienstbeschreibungsnetz funktional ausführbar ist und dass es typisierungskonform ist, sind die Fortsetzbarkeit und die Termination die nach Definition 4.46 notwendigen Kriterien für die schwache Korrektheit eines Dienstbeschreibungsnetzes. Beide Kriterien lassen sich erreichen, indem spezielle Bedingungen an die Netze gestellt werden. Dabei werden beide Eigenschaften über die Korrektheit des zugrundeliegenden Netzes zusammen mit der Endlichkeit und der Fortsetzbarkeit erreicht, da dann gewährleistet ist, dass das Netz stets in einen finalen Netzzustand übergeht. Die Endlichkeit wird dadurch erzwungen, dass jede Schleife nur endlich oft durchlaufen werden kann. Die Fortsetzbarkeit wird durch das Einführen einer Default-Transition erreicht. Dies wird im Folgenden dargestellt.

Definition 5.6 (Endlichkeitsbedingung)
Ein Dienstbeschreibungsnetz $\mathcal{SDN}$ erfüllt die *Endlichkeitsbedingung* gdw. sein zugrundeliegendes Netz $\mathcal{N}$ ein korrektes Workflownetz ist und jede minimale T-Invariante von $\mathcal{N}$ nur endlich oft in $\mathcal{SDN}$ durch entsprechende Schaltmodi realisiert werden kann. ⧫

Die Fortsetzbarkeit elementarer Dienstbeschreibungsnetze wird durch die Verwendung einer Default-Transition erreicht, die aktiviert sein muss, wenn keine andere Transition aktiviert ist. Für eine Menge von Transitionen mit gleichem Vorbereich muss es stets eine ausgewählte Transition geben, deren Aktivierungsbedingung der Konjunktion der Negationen aller anderen Transitionen der Menge entspricht. Diese Transition kann dann nur schalten, wenn keine andere Transition schalten kann. Hierdurch ist unter Umständen die starke Korrektheit aller Transitionen verletzt, da diese Transition niemals aktiviert werden kann. Aus diesem Grund lässt sich die starke Korrektheit auch auf Transitionen einschränken.

Definition 5.7 (Alternativtransitionen, Default-Transition)
Gegeben ein Dienstbeschreibungsnetz $\mathcal{SDN}$. Sei $T \times T$ eine Äquivalenzrelation, so dass $t_1 \sim_a t_2$ gdw. ${}^\bullet t_1 = {}^\bullet t_2$. Die Äquivalenzklasse $[t]_a$ zu einer Transition wird Klasse der *Alternativtransitionen* genannt, sofern $|[t]_a| > 1$ ist. Die Menge $[t]_a$ besitzt eine *Default-Transition* $t_d \in [t]_a$ gdw. $G(t_d) = \bigwedge_{t \in [t]_a \setminus \{t_d\}} \neg G(t)$ ist. ⧫

Die Default-Transition ist also diejenige Transition, die nur schalten kann, wenn keine andere Transition aus $[t]_a$ schalten kann. Da die Reihenfolge der Transitionen in der Menge $t \in [t]_a \setminus \{t_d\}$ nicht festgelegt ist, ist die Gleichheit von $G(t_d)$ und $\bigwedge_{t \in [t]_a \setminus \{t_d\}} \neg G(t)$ als Gleichheit im Sinne einer Spezifikation zu verstehen, die

der statischen Algebra $\mathcal{A}^{stat}$ entspricht, so dass die Reihenfolge der durch die Konjunktion verbundenen Terme beliebig sein kann. Welche der Transitionen aus $[t]_a$ der Default-Transition entspricht ist dabei im Prinzip beliebig. Häufig werden diese Transitionen jedoch durch das spezielle Schlüsselwort `default` ausgezeichnet.

Die Forderung nach einer Default-Transition findet sich beispielsweise auch für die BPMN-Notation (vgl. BPMN 1.2 (2008) (B.11.9)). Eine allgemeine Überprüfung, ob stets eine Transition aktivierbar ist, erfordert, die Erfüllbarkeit der entsprechenden Aktivierungsbedingungen zu überprüfen, was im Allgemeinen dem SAT-Problem für aussagenlogische Formeln entspricht. Dieses ist NP-vollständig (vgl. Cook, 1971), so dass hier der Weg der Default-Transition gewählt wurde, um in Polynomialzeit entscheiden zu können, ob ein Dienstbeschreibungsnetz elementar ist.

Definition 5.8 (Hinreichende Fortsetzbarkeitsbedingung)
Ein Dienstbeschreibungsnetz $\mathcal{SDN}$ erfüllt die *hinreichende Fortsetzbarkeitsbedingung* gdw.

- das zugrundeliegende Netz $\mathcal{N}$ ein korrektes Workflownetz ist,
- für jede Transition $t \in T$, für die nicht $G(t) = \mathit{true}$ ist, eine Transition mit gleichem Vorbereich existiert und
- jede Klasse von Alternativtransitionen (vgl. Def. 5.7) eine Default-Transition besitzt, die durch die Anschrift `default` ausgezeichnet ist.

⧫

Lemma 5.9 (Hinreichende Fortsetzbarkeitsbedingung)
In einem typisierungskonformen Dienstbeschreibungsnetz $\mathcal{SDN}$, das die hinreichende Fortsetzbarkeitsbedingung erfüllt, ist für jeden initialen Netzzustand Q_0 und für jeden erreichbaren Netzzustand $Q \in [Q_0\rangle_{\mathcal{SDN}}$ eine Transition aktiviert, sofern Q nicht finaler Netzzustand ist.

Beweis:
Da das zugrundeliegende Workflownetz $\mathcal{N}$ korrekt ist, ist nach Lemma 4.28 im zugrundeliegenen Netz für jeden in $\mathcal{SDN}$ erreichbaren Netzzustand $(\mathbf{m}, \mathfrak{Q})$ eine Transition t in $\mathring{\mathbf{m}}$ aktiviert. Dann existiert eine Variablenbelegung α, die jeder Eingangsvariable einen Wert zuweist. Wäre nun keine Transition in $\mathcal{SDN}$ aktiviert, so wäre die Aktivierungsbedingung jeder Transition in $[t]_a$ falsch. Da jede Transition mit einer Aktivierungsbedingung jedoch eine Klasse von Alternativtransitionen besitzt und da in dieser stets eine Default-Transition existiert, gilt die Aussage. □

Diese Eigenschaften führen zu elementaren Dienstbeschreibungsnetzen, in denen die hinreichenden Bedingungen für die Endlichkeit, für die Fortsetzbarkeit, für die Typisierungskonformität und für die funktionale Ausführbarkeit gefordert werden.

5.2.3 Elementarität

Die vorangegangenen Unterabschnitte lieferten hinreichende Bedingungen für verschiedene Eigenschaften von Dienstbeschreibungsnetzen. Diese Eigenschaften werden hier unter dem Begriff der Elementarität zusammengefasst. Elementare Dienstbeschreibungsnetze erfüllen sämtliche der oben angegebenen Bedingungen, woraus sich ergibt, dass diese Netze unter jedem initialen Netzzustand schwach korrekt sind. Dies wird in diesem Unterabschnitt gezeigt.

Definition 5.10 (Elementares Dienstbeschreibungsnetz)
Ein Dienstbeschreibungsnetz $\mathcal{SDN}$ ist elementar gdw. die folgenden Bedingungen für $\mathcal{SDN}$ erfüllt sind:

- die Variablendeterminiertheit (Definition 4.19),
- die hinreichende Typisierungskonformitätsbedingung (Definition 5.1),
- die hinreichende Funktionalitätsbedingung (Definition 5.4),
- die Endlichkeitsbedingung (Definition 5.6) und
- die hinreichende Fortsetzbarkeitsbedingung (Definition 5.8).

♦

Die Variablendeterminiertheit wird an dieser Stelle noch einmal explizit gefordert, obwohl sie bereits implizit durch die hinreichende Funktionalitätsbedingung gefordert wird. Sie besagt, dass im Netz keine Variablen existieren, die nicht durch Eingangskanten oder durch Zuweisungsanweisungen gebunden sind. Aus diesen Forderungen folgt direkt die zentrale Aussage für elementare Dienstbeschreibungsnetze, die in folgendem Satz wiedergegeben ist.

Satz 5.11 (Schwache Korrektheit elementarer Dienstbeschreibungsnetze)
Ein elementares initialbeschriftetes Dienstbeschreibungsnetz $\mathcal{SDN}_{init}$ ist unter jeder Initialisierung schwach korrekt und endlich.

Beweis:
[*Endlichkeit*] Gäbe es eine unendliche Schaltfolge, so müsste im zugrundeliegenden Netz eine T-Invariante unendlich häufig realisiert werden, da das zugrundeliegende

Netz korrekt und somit beschränkt ist und folglich eine Markierung unendlich oft erreicht werden müsste. Aufgrund von Lemma 4.28 muss jede Schaltfolge in $\mathcal{SDN}$ auch eine Schaltfolge in $\mathcal{N}$ bei entsprechender Abbildung der Markierung sein. Somit müsste eine minimale T-Invariante unendlich oft realisiert werden, da jede T-Invariante eine Linearkombination minimaler T-Invarianten ist (vgl. Satz 3.55). Dies widerspricht der Endlichkeitsbedingung aus Definition 5.6.
[*Fortsetzbarkeit* und *Termination*] Dass stets eine Transition aktiviert ist, folgt aus der hinreichenden Fortsetzbarkeitsbedingung und aus Lemma 5.8. Aufgrund der Endlichkeit wird jede Transition nur endlich oft geschaltet, so dass nach einer endlichen Schaltsequenz ein finaler Netzzustand Q_f erreichbar ist. Für diesen muss aufgrund der Korrektheit des zugrundeliegenden Netzes und aufgrund von Lemma 4.28 $\mathring{\mathbf{m}}_f = [o]$ gelten.
[*Funktionale Nebenläufigkeit* und *Typisierungskonformität*] Diese Eigenschaften folgen aus der Erfüllung der hinreichenden Typisierungskonformitätsbedingung und der hinreichenden Funktionalitätsbedingung mit Lemma 5.2 und Lemma 5.5. □

Die Überprüfung des Determinismus aus Definition 4.50 verlangt zudem, dass in jedem erreichbaren Netzzustand gilt, dass für zwei im Konflikt stehende Transitionen niemals beide Aktivierungsbedingungen zu wahr auswerten. Dieses Problem kann im Allgemeinen nicht entschieden werden, da hierzu sämtliche möglichen Belegungen und sämtliche Terme ausgewertet werden müssten. Bereits bei lediglich aussagenlogischen Verknüpfungen ist das Problem NP-vollständig (SAT-Problem (vgl. Cook, 1971)). Dennoch dürfte es in vielen praktischen Szenarien durchaus möglich sein, diese Eigenschaft zu überprüfen, da die durch einen menschlichen Modellierer beschriebenen Aktiverungsbedingungen in der Regel nicht zu komplex sind.

Zur Überprüfung der starken Korrektheit muss die Aktivierbarkeit einer Transition nachgewiesen werden. Diese lässt sich jedoch nur über symbolische Betrachtungen oder über den Erreichbarkeitsgraphen analysieren. So lässt sich beispielsweise über die Betrachtung der minimalen T-Invarianten überprüfen, welche minimalen T-Invarianten überhaupt eine gegebene Transition überdecken, um dann symbolisch zurückzurechnen, ob ein Anfangszustand existiert, der zu einer Aktivierung dieser Transition führt. Ist ein Dienstbeschreibungsnetz funktional ausführbar und typisierungskonform, wie dies für elementare Dienstbeschreibungsnetze der Fall ist, so ist zur Analyse der deterministischen Korrektheit und zur Analyse der Korrektheit die Betrachtung der Sequentialisierung ausreichend, wie im nächsten Kapitel ausgeführt werden wird. Diese Analyse ist jedoch auch dann im Allgemeinen noch immer unentscheidbar.

Elementare Dienstbeschreibungsnetze erlauben eine kompaktere Schaltregel, da keine Lokationen existieren können, die gleichzeitig von zwei Transition genutzt werden, ohne dass beide Transitionen lesen. Die Definition eines Schaltmodus aus

Definition 4.20 bleibt hierbei erhalten. Es ändert sich jedoch die Bedingung der Aktiviertheit und das Feuern einer Transition, da hierfür nicht mehr länger die Reservierung des Systemzustandes notwendig ist. Ein Schaltmodus (t, α) ist somit in einem Netzzustand $Q = (\mathbf{m}, \mathfrak{Q})$ aktiviert, wenn für alle Stellen $p \in P$ gilt, dass $\mathbf{m}(p) \geq \alpha(W(p,t))$. Das Feuern ist dann analog zu Definition 4.21, wobei auf die Reservierung des Zustandes verzichtet werden kann.

Im Unterschied zu Dienstbeschreibungsnetzen im Allgemeinen entfällt die Reservierung des Systemzustandes, da sie für elementare Dienstbeschreibungsnetze durch die strenge Forderung der hinreichenden Typisierungskonformitätsbedingung und der hinreichenden Funktionalitätsbedingung nicht mehr notwendig ist.

Die Aktiviertheit und die Schaltregel elementarer Dienstbeschreibungsnetze besitzt somit eine deutlich stärkere Nähe zur Aktiviertheit und zur Schaltregel herkömmlicher algebraischer Netze, wie sie in Definition 3.46 angegeben wurde. Dies erlaubt es, Überlegungen zur Analyse aus dem Bereich der algebraischen Netze auf elementare Dienstbeschreibungsnetze zu übertragen, was in Unterabschnitt 6.3.3 noch einmal skizziert wird.

Für die Überprüfung der Elementarität sind neben der Überprüfung der Korrektheit des zugrundeliegenden Netzes auch die Überprüfung der Erreichbarkeit einer Markierung im Workflownetz notwendig, da für die hinreichende Typisierungskonformitätsbedingung und die hinreichenden Funktionalitätsbedingung festgestellt werden muss, ob eine bestimmte Markierung oder eine größere Markierung als diese erreichbar sind. Aus diesem Grund wird die Analyse auf Free-Choice-Netze reduziert, da eine solche Analyse für diese Netzklasse in Polynomialzeit möglich ist.

5.3 Korrektheit und Nebenläufigkeit in Free-Choice-Workflownetzen

Zur Überprüfung der Elementarität eines Dienstbeschreibungsnetzes sind neben der Betrachtung der Terme in erster Linie Eigenschaften des zugrundeliegenden Netzes von Interesse. Dieses zugrundeliegende Workflownetz muss zum einen korrekt sein, zum anderen muss ermittelt werden können, ob es eine erreichbare Markierung gibt, die zwei Transitionen nebenläufig aktiviert beziehungsweise die eine Stelle markiert und gleichzeitig eine Transition aktiviert, wie dies für die hinreichende Funktionalitätsbedingung (Definition 5.4) und die hinreichende Typisierungskonformitätsbedingung (Definition 5.1) notwendig ist.

Da diese Fragen im Allgemeinen für Workflownetze NP-vollständig sind (vgl. Esparza (1998a)), ist es sinnvoll sich auf eine Netzklasse zu beschränken, die hinreichend ausdrucksstark ist und die gleichzeitig eine Analyse in Polynomialzeit erlaubt. Eine solche Klasse stellen die Workflownetze mit der Free-Choice-Eigenschaft (*FCWF-Netze*) dar, die in diesem Abschnitt zunächst näher untersucht werden,

bevor die Ergebnisse dieser Untersuchung dann im nächsten Abschnitt zur Überprüfung elementarer Dienstbeschreibungsnetze genutzt werden.

Mittels Satz 3.58 und Satz 3.59 lässt sich die Korrektheit durch Berechnung einer Basis minimaler S-Invarianten und durch die Kriterien des Rang-Theorems in Satz 3.58 zeigen. Dieser Ansatz bietet jedoch den Nachteil, dass eine detailliertere Angabe zur Fehlerquelle nur möglich ist, wenn eine Transition nicht durch eine T-Invariante überdeckt wird oder wenn eine Stelle unbeschränkt ist. Eine Analyse mittels der Handle (also der Paare gleicher öffnender und schließender Knoten) im Netz ist hingegen für Netze mit der Free-Choice-Eigenschaft im Allgemeinen nicht möglich, sondern lediglich für wohlstrukturierte Netze. Neben dieser ungenauen Diagnose fehlt ein Verfahren, um schnell entscheiden zu können, ob zwei Transitionen nebenläufig aktivierbar sind, was für die Ermittlung der Elementarität notwendig ist.

Das hier vorgestellte Vorgehen nutzt eine Basis minimaler T-Invarianten, um das Verhalten eines Netzes zu analysieren und orientiert sich hierbei an der Vorgehensweise von Desel und Esparza (1993), wo eine Überdeckung von S-Komponenten genutzt wird. (Aufgrund der Eigenschaften des dualen Netzes (vgl. Satz 3.62) lassen sich Teile der Vorgehensweise übertragen.) Ein Algorithmus, der die S-Komponenten eines Free-Choice-Netzes zur Berechnung der Wohlgeformtheit des Netzes ausnutzt, findet sich bei Kovalyov (1996), jedoch wird dort nicht die Korrektheit des Algorithmus gezeigt.

Ein weiterer Vorteil des hier vorgestellten Verfahrens ist, dass bei der Ermittlung der Wohlgeformtheit gleich eine Basis minimaler S-Invarianten erzeugt wird, die dann dazu genutzt werden kann, die Lebendigkeit des Abschlusses eines Free-Choice-Workflownetzes zu ermitteln. Die Basis minimaler S-Invarianten dient dann auch dazu zu ermitteln, welche Stellen überhaupt gleichzeitig markierbar sind. Dies wird später für die Ermittlung der funktionalen Ausführbarkeit benötigt.

Zur Veranschaulichung des Vorgehens wird ein Beispiel in Beispiel 5.2 eingeführt, an dem dann die verschiedenen Verfahren erläutert werden.

5.3.1 Eigenschaften von Free-Choice-Netzen

Die Analyse eines Free-Choice-Netzes (FC-Netz) findet im Folgenden über seine minimalen T-Invarianten statt. Es wird gezeigt, dass bei einem wohlgeformten Free-Choice-Netz die Vereinigung der Subnetze beliebiger minimaler T-Invarianten stets ein wohlgeformtes Free-Choice-Netz darstellt, sofern diese Vereinigung zusammenhängend ist. Auf diese Weise lässt sich durch sukzessives Hinzufügen der Subnetze der minimalen T-Invarianten einer Basis des T-Invariantenvektorraums erkennen, ob es zu einer Verklemmung kommen kann. Dabei wird davon ausgegangen, dass das zu analysierende Netz stets zusammenhängend ist und mindestens zwei Knoten besitzt. Nach Theorem 2.40 in Desel und Esparza (1995) ist es dann auch stark zusammenhängend, wenn es wohlgeformt ist. Das Vorgehen soll hier zunächst an

einem Beispiel motiviert werden. Dass dieses Beispiel kein Sonderfall ist, wird dann im weiteren Verlauf dieses Abschnitts gezeigt.

Beispiel 5.2: Am Beispiel des Netzes in Abbildung 5.2 lässt das Vorgehen verdeutlichen. Das dargestellte Netz besitzt eine positive S-Invariante und beispielsweise die minimalen T-Invarianten $\vec{\iota_1} = [t_1, t_2, t_3, t_4, t_9, t^*]$, $\vec{\iota_2} = [t_1, t_4, t_5, t_6, t_9, t^*]$ und $\vec{\iota_3} = [t_1, t_6, t_7, t_8, t_9, t^*]$, die gemeinsam das Netz überdecken[2].

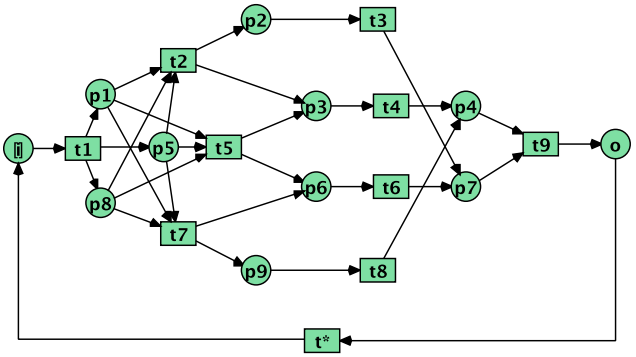


Abbildung 5.2: Ein wohlgeformtes Free-Choice-Netz

Die grundlegende Vorgehensweise ist in Abbildung 5.3 dargestellt. Hierbei wird ein Netz $\mathcal{N}^{\#}$ ausgehend von einer minimalen T-Invariante sukzessive aufgebaut. Die Subnetze der minimalen T-Invarianten werden in der Reihenfolge $\vec{\iota_1}$, $\vec{\iota_2}$ und $\vec{\iota_3}$ hinzugefügt. Initial besteht das Netz $\mathcal{N}^{\#}$ aus dem in Abbildung 5.3a) dargestellten markierten Graphen $\mathcal{N}_{\vec{\iota_1}}$. Fügt man nun das Subnetz von $\vec{\iota_2}$ zu $\mathcal{N}^{\#}$ hinzu, so erhält man das Netz in Abbildung 5.3b). Im Anschluss hieran werden die Kanten und Knoten des Subnetzes der letzten minimalen T-Invariante $\vec{\iota_3}$ hinzugefügt. Ein entsprechendes Netz findet sich in Abbildung 5.3c).

◇

Während des Hinzufügens können die Eigenschaften des Rang-Theorems in jedem Schritt überprüft werden, da sich zeigen wird, dass das Netz nach jedem Hinzufügen ein wohlgeformtes Free-Choice-Netz sein muss, wenn das Gesamtnetz ein wohlgeformtes Free-Choice-Netz ist. Dies bedeutet, dass diese Eigenschaft in jedem Schritt überprüft werden kann, was es erlaubt, direkt zurückzukoppeln, welche minimalen T-Invarianten zu einem fehlerhaften Netz führen. Zudem lässt sich durch dieses Verfahren aus einer Überdeckung durch T-Komponenten eine Basis minimaler T-

[2]Für Vektoren über $\{0, 1\}$ werden dabei nur die Werte angegeben, die verschieden von 0 sind. Dies geschieht in der Form $[a, b]$, so dass $[a, b](a) = [a, b](b) = 1$ und $[a, b](c) = 0$ für $c \notin \{a, b\}$ (vgl. Abschnitt 3.1).

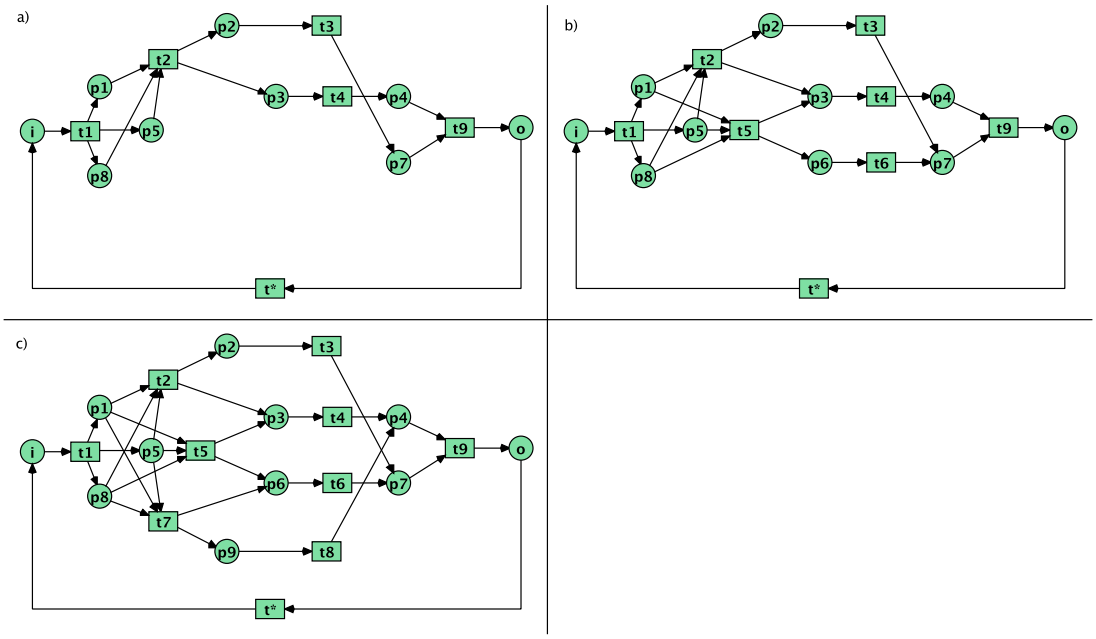


Abbildung 5.3: Das Verfahren dieses Abschnitts angewandt auf das Netz aus Abbildung 5.2

Invarianten erzeugen, was für die weitere Analyse relevant ist. Wie eine solche Basis aufgebaut ist, wird im Folgenden behandelt.

Basen der Invariantenvektorräume und minimale Invarianten

Die Menge aller T-Invarianten und die Menge aller S-Invarianten eines Netzes $\mathcal{N}$ wird kürzer als $\textsc{Sinv}(\mathcal{N})$ beziehungsweise als $\textsc{Tinv}(\mathcal{N})$ notiert, so dass im Folgenden

- $\textsc{Sinv}(\mathcal{N}) \stackrel{\text{def}}{=} \{\vec{\kappa} \in \mathbb{Q}^{|P|} \mid \vec{\kappa}$ ist S-Invariante von $\mathcal{N}\}$ die *Menge der S-Invarianten*
- $\textsc{Tinv}(\mathcal{N}) \stackrel{\text{def}}{=} \{\vec{\iota} \in \mathbb{Q}^{|T|} \mid \vec{\iota}$ ist T-Invariante von $\mathcal{N}\}$ die *Menge der T-Invarianten*

eines Netzes $\mathcal{N}$ ist.

Betrachtet man die Inzidenzmatrix eines Petrinetzes als lineare Abbildung $[\mathcal{N}] : \mathbb{Q}^{|T|} \to \mathbb{Q}^{|P|}$, so stellen die T-Invarianten den Kern der Abbildung dar[3]. Sie bilden den Untervektorraum, dessen Vektoren stets auf den Nullvektor $\vec{\mathbf{0}}$ abgebildet werden. (Invarianten wurden in Definition 3.35 über $\mathbb{Q}$ definiert, so dass der Begriff

[3] Auch wenn alle Einträge der Inzidenzmatrix ganze Zahlen aus $\mathbb{Z}$ sind, kann die Abbildung natürlich als lineare Abbildung von Vektoren über $\mathbb{Q}$ aufgefasst werden.

des Vektorraums an dieser Stelle korrekt gewählt ist.) Nach Satz 3.55 gibt es für jedes Netz mit einer positiven T-Invariante eine Menge minimaler T-Invarianten, so dass jede T-Invariante des Netzes eine Linearkombination dieser minimalen T-Invarianten ist. Da die minimalen T-Invarianten ein Erzeugendensystem des Vektorraums der T-Invarianten darstellen, gibt es auch eine Basis dieses Vektorraums, die aus minimalen T-Invarianten besteht. Dies lässt sich leicht anhand folgender Überlegung sehen.

Sei $\vec{\iota} = \sum_{j=1}^{n} a_j \cdot \vec{\iota_j}$, wobei $\textsc{Tinv}_{\min}(\mathcal{N}) = \{\vec{\iota_1}, \ldots, \vec{\iota_n}\}$ die Menge der minimalen T-Invarianten ist. Wenn die minimalen T-Invarianten $\vec{\iota_1}$ bis $\vec{\iota_n}$ nicht sämtlich linear unabhängig sind, so kann eine von ihnen durch die anderen erzeugt werden, und es ist o.B.d.A. $\vec{\iota_n} = \sum_{j=1}^{n-1} b_j \cdot \vec{\iota_j}$. Folglich lässt sich auch in der Summe einer jeden T-Invariante statt $\vec{\iota_n}$ diese Summe einsetzen. Wird dies wiederholt, so erhält man eine Basis des Netzes aus minimalen T-Invarianten. Nach der gleichen Argumentation und ebenfalls nach Satz 3.55 gilt Analoges natürlich auch für S-Invarianten, so dass sich folgende Definition ergibt.

Definition 5.12 (Basis des Invariantenvektorraums)
Zu einem Netz $\mathcal{N}$ mit positiver S-Invariante wird eine *Basis des S-Invariantenvektorraums* aus minimalen S-Invarianten notiert als $\mathbf{B}_S(\mathcal{N}) \subseteq \textsc{Sinv}(\mathcal{N})$. Zu einem Netz $\mathcal{N}$ mit positiver T-Invariante wird eine *Basis des T-Invariantenvektorraums* aus minimalen T-Invarianten notiert als $\mathbf{B}_T(\mathcal{N}) \subseteq \textsc{Tinv}(\mathcal{N})$. ♦

Da es im Allgemeinen mehr als $|\mathbf{B}_T(\mathcal{N})|$ minimale T-Invarianten beziehungsweise mehr als $|\mathbf{B}_S(\mathcal{N})|$ minimale S-Invarianten in einem Netz geben kann, sind die Invariantenbasen im Allgemeinen nicht eindeutig. Gleichzeitig stellt eine Überdeckung durch minimale linear unabhängige Invarianten im Allgemeinen keine Basis dar, da hierzu nicht immer alle Invarianten einer Basis benötigt werden.

Sei $\dim(\cdot)$ die *Dimension* eines Vektorraums und sei $\mathrm{rg}(\cdot)$ der *Rang* einer Matrix. Die Dimensionsformel für lineare Abbildungen (vgl. (Jänich, 1998, S. 87) oder (Meyer, 2001, S. 199)) besagt, dass für die lineare Abbildung $[\mathcal{N}] : \mathbb{Q}^{|T|} \to \mathbb{Q}^{|P|}$ die Formel $\dim(kern[\mathcal{N}]) + \mathrm{rg}([\mathcal{N}]) = |T|$ gilt, wobei die Dimension des Kerns $\dim(kern[\mathcal{N}])$ von $[\mathcal{N}]$ genau der Dimension $\dim(\textsc{Tinv}(\mathcal{N}))$ von $\textsc{Tinv}(\mathcal{N})$ entspricht. Somit gilt $\mathrm{rg}([\mathcal{N}]) = |T| - \dim(\textsc{Tinv}(\mathcal{N}))$. Für die transponierte Matrix gilt entsprechend $\mathrm{rg}([\mathcal{N}^t]) = |P| - \dim(\textsc{Sinv}(\mathcal{N}))$. Da der Rang der transponierten Matrix gleich dem Rang der Ursprungsmatrix ist, folgt dann

$$|P| - \dim(\textsc{Sinv}(\mathcal{N})) = |T| - \dim(\textsc{Tinv}(\mathcal{N})) = \mathrm{rg}([\mathcal{N}]).$$

Die zentrale Aussage soll in einem Lemma festgehalten werden. Dabei gilt zudem $|\mathbf{B}_T(\mathcal{N})| \leq |T|$ und $|\mathbf{B}_S(\mathcal{N})| \leq |P|$. Dies ergibt sich, da der Kern einer linearen

Abbildung $[\mathcal{N}] : \mathbb{Q}^{|T|} \to \mathbb{Q}^{|P|}$ nicht größer sein kann als die Dimension des Vektorraums $\mathbb{Q}^{|T|}$ und da $\dim(\mathbb{Q}^{|T|}) = |T|$ ist. Gleiches gilt für die Dimension einer Basis $\mathbf{B}_S(\mathcal{N})$ und $|P|$.

Lemma 5.13 (Kardinalität der Basen der Invariantenvektorräume)
Zu einem Netz $\mathcal{N}$ mit einer positiven S-Invariante und einer positiven T-Invariante gilt für jede Basis minimaler S-Invarianten $\mathbf{B}_S(\mathcal{N}) \subseteq \textsc{Sinv}(\mathcal{N})$ und für jede Basis minimaler T-Invarianten $\mathbf{B}_T(\mathcal{N}) \subseteq \textsc{Tinv}(\mathcal{N})$ der Invariantenvektorräume, dass alle S-Invarianten von $\mathcal{N}$ eine Linearkombination der Vektoren aus $\mathbf{B}_S(\mathcal{N})$ und alle T-Invarianten von $\mathcal{N}$ eine Linearkombination der Vektoren aus $\mathbf{B}_T(\mathcal{N})$ sind und dass

$$|\mathbf{B}_T(\mathcal{N})| - |\mathbf{B}_S(\mathcal{N})| = |T| - |P| \text{ sowie } |\mathbf{B}_T(\mathcal{N})| \leq |T| \text{ und } |\mathbf{B}_S(\mathcal{N})| \leq |P|.$$

Lemma 5.13 erlaubt es, eine Abschätzung zu treffen, wieviele minimale linear unabhängige Invarianten ein Netz besitzt. Ist etwa bekannt, dass ein Netz lediglich eine minimale T-Invariante besitzt, so ist auch bekannt, dass die maximale Anzahl linear unabhängiger minimaler S-Invarianten gleich $|P| - |T| + 1$ ist.

Minimale Invarianten von Free-Choice-Netzen

Für Free-Choice-Netze haben die minimalen Invarianten besondere Eigenschaften. So ist das Subnetz jeder minimalen T-Invariante eine T-Komponente, wie das folgende Lemma zeigt. Ebenso ist das Subnetz jeder minimalen S-Invariante eine S-Komponente.

Lemma 5.14 (Minimale Invarianten wohlgeformter Free-Choice-Netze)
Gegeben ein wohlgeformtes Free-Choice-Netz $\mathcal{N}$.
Eine T-Invariante von $\mathcal{N}$ ist minimal gdw. ihr Subnetz eine T-Komponente ist.
Eine S-Invariante von $\mathcal{N}$ ist minimal gdw. ihr Subnetz eine S-Komponente ist.

Beweis:
[T-Invariante] Nach Theorem 5.17 in Desel und Esparza (1995) ist das Subnetz jeder minimalen T-Invariante eines wohlgeformten Free-Choice-Netzes $\mathcal{N}$ eine T-Komponente. Nach Satz 3.64 ist für eine T-Komponente $\mathcal{N}_T$ der Vektor $\vec{\iota} = \chi[T_T]$ eine minimale T-Invariante von $\mathcal{N}$.
[S-Invariante] Nach Satz 3.62 ist das duale Netz $\mathcal{N}^d$ ebenfalls ein wohlgeformtes Free-Choice-Netz. Folglich ist auch das Subnetz jeder minimalen T-Invariante von $\mathcal{N}^d$ eine T-Komponente. Da jede minimale T-Invariante in $\mathcal{N}^d$ eine minimale S-Invariante in $\mathcal{N}$ ist und jede T-Komponente in $\mathcal{N}^d$ eine S-Komponente in $\mathcal{N}$ ist, gilt die Aussage (vgl. Proposition 6.20 in Desel und Esparza (1995)). □

Es sei darauf hingewiesen, dass nicht jede T-Invariante, die auf $\{0,1\}$ abbildet auch minimal ist, da es sich hierbei um die Summe mehrerer minimaler T-Invarianten handeln kann, deren Subnetze lediglich Stellen gemeinsam haben. In Anlehnung an Verbeek u. a. (2001) werden hier uniforme Invarianten angeführt, wobei dieses Konzept sowohl für S- als auch für T-Invarianten verwendet wird.

Definition 5.15 (Uniforme Invarianten)
Gegeben ein Netz $\mathcal{N}$. Eine S-Invariante $\vec{\kappa}$ (T-Invarianten $\vec{\iota}$) von $\mathcal{N}$ ist *uniform* gdw. $\vec{\kappa} : P \to \{0,1\}$ ($\vec{\iota} : T \to \{0,1\}$) gilt. ♦

Jede S-Komponente und jede T-Komponente ist das Subnetz einer uniformen S- bzw. T-Invariante (vgl. Satz 3.63 und Satz 3.64). Aufgrund von Lemma 5.14 folgt, dass jede minimale S-Invariante und jede minimale T-Invariante eines wohlgeformten Free-Choice-Netzes uniform ist[4]. Da das Subnetz einer minimalen S- bzw. T-Invarianten eine S- bzw. T-Komponente ist und da die charakteristische Funktion einer solchen eine Invariante ist, liegt die Aussage nahe. Somit besitzt ein wohlgeformtes Free-Choice-Netz eine Basis aus uniformen minimalen S- bzw. T-Invarianten, da es nach Satz 3.58 eine positive S- und eine positive T-Invariante besitzt.

Das Subnetz jeder minimalen T-Invariante eines wohlgeformten Free-Choice-Netzes ist nach Lemma 5.14 eine T-Komponente. Jede T-Komponente $\mathcal{N}_T$ ist wiederum ein Free-Choice-Netz, da stets $|p^\bullet| = 1$ für alle Stellen $p \in P_T$ gilt, was nach Definition 3.37 direkt die Free-Choice-Eigenschaft impliziert. Betrachtet man nun die T-Komponenten der Subnetze einer Basis minimaler T-Invarianten eines wohlgeformten Free-Choice-Netzes, so lassen sich ausgehend von einer T-Komponente sukzessive alle weiteren T-Komponenten hinzufügen, so dass das resultierende Netz stets stark zusammenhängend ist. Ein solches Netz ist in jedem dieser Schritte ein Free-Choice-Netz. Dies wird in Lemma 5.16 gezeigt, wobei von dem in Definition 3.36 definierten Subnetz eines T-Vektors $\mathcal{N}_{\vec{\iota}}$ Gebrauch gemacht wird.

Lemma 5.16 (Free-Choice-Eigenschaft der T-Invariantensubnetze)
Gegeben ein wohlgeformtes Free-Choice-Netz $\mathcal{N}$ und eine Basis minimaler T-Invarianten $\mathbf{B}_T(\mathcal{N})$. Sei $\vec{\iota_*} = \sum_{\vec{\iota} \in \mathbf{B}_T(\mathcal{N})} a_{\vec{\iota}} \cdot \vec{\iota}$ mit $a_{\vec{\iota}} \in \{0,1\}$ für alle $\vec{\iota} \in \mathbf{B}_T(\mathcal{N})$ und sei $\vec{\iota_*} \neq \vec{\mathbf{0}}$. Dann besitzt $\mathcal{N}_{\vec{\iota_*}}$ die Free-Choice-Eigenschaft.

[4] Zu jeder minimalen S- oder T-Invariante gibt es nach Lemma 5.14 eine S- oder T-Kompnonente mit gleichem Subnetz, deren charakteristischer Vektor eine minimale S- bzw. T-Invariante darstellt, die uniform ist. Diese ließe sich von einer minimalen S- bzw. T-Invariante, die nicht uniform ist, abziehen bis entweder eine Invariante daraus resultiert, die weniger Knoten im Träger besitzt, oder bis die resultierende Invariante uniform ist. In beiden Fällen wäre die ursprüngliche Invariante nicht uniform gewesen (siehe hierzu auch die Erläuterungen im Anschluss an Definition 3.35).

Beweis:
Der Beweis erfolgt per Induktion über $n = \sum_{\vec{\tau} \in \mathbf{B}_T(\mathcal{N})} a_{\vec{\tau}}$. Ist $n = 1$, so liegt nur eine minimale T-Invariante in $\mathbf{B}_T(\mathcal{N})$ vor, und es ist $\mathcal{N}_{\vec{\iota_*}}$ ein markierter Graph (eine T-Komponente). Dann ist $|p^\bullet| = 1$ für alle $p \in P_{\vec{\iota_*}}$ und $\mathcal{N}_{\vec{\iota_*}}$ ist Free-Choice.

Sei die Aussage also für n bewiesen und sei die Aussage nun für $n+1$ zu zeigen. Dann gibt es eine T-Invariante $\vec{\iota_\#} = \vec{\iota_*} - \vec{\tau}$, wobei $\vec{\tau} \in \mathbf{B}_T(\mathcal{N})$ und $a_{\vec{\tau}} = 1$ ist, so dass für $\mathcal{N}_{\vec{\iota_\#}}$ die Behauptung gilt (Induktionsannahme) und für $\mathcal{N}_{\vec{\tau}}$ die Behauptung (initiale Behauptung) gilt. Da für jede T-Komponente alle Stellen im Vorbereich einer Transition ebenfalls Teil der T-Komponente sind, würden zwei Transitionen, die die Free-Choice-Eigenschaft verletzen, diese auch in $\mathcal{N}$ verletzen, da nach dem Hinzufügen einer Transition durch die Konstruktion niemals mehr eine Ein- oder Ausgangskante dieser Transition entfernt oder hinzugefügt wird. Somit gilt die Aussage. □

Interessant ist nun, dass bei dem Vorgehen aus Lemma 5.16 für $\mathcal{N}_{\vec{\iota_*}}$ nicht nur die Free-Choice-Eigenschaft erhalten bleibt, sondern auch die Wohlgeformtheit des Subnetzes $\mathcal{N}_{\vec{\iota_*}}$. Dies wird in folgendem Lemma gezeigt. Dabei wird das Rang-Theorem aus Satz 3.58 zur Hilfe genommen. Formt man dieses um, so gilt für wohlgeformte Free-Choice-Netze, dass $|T| - (|C_\mathcal{N}| - 1) = |T| - |C_\mathcal{N}| + 1 = \dim(\textsc{Sinv}(\mathcal{N}))$ ist, da $\mathrm{rg}([\mathcal{N}]) = |C_\mathcal{N}| - 1$ ist. Fügt man nun die Knoten und Kanten einer weiteren T-Komponente eines Free-Choice-Netzes zu einer existierenden Vereinigung von T-Komponenten hinzu, so muss diese Bedingung weiterhin gelten, wenn das Netz weiterhin wohlgeformt ist. Für jede T-Komponente $\mathcal{N}_T$ ist $\dim(\textsc{Sinv}(\mathcal{N}_T)) = 1$ und somit gilt $|T_T| = |C_{\mathcal{N}_T}|$, wovon in folgendem Beweis Gebrauch gemacht wird. Im Anschluss hieran wird das Ergebnis in Satz 5.19 verallgemeinert und gezeigt, dass sogar das Subnetz jeder semi-positiven T-Invariante eines wohlgeformten Free-Choice-Netzes ein wohlgeformtes Free-Choice-Netz ist.

Lemma 5.17 (Wohlgeformtheit der T-Invariantensubnetzvereingigung)
Gegeben ein wohlgeformtes Free-Choice-Netz $\mathcal{N}$ und eine Basis minimaler T-Invarianten $\mathbf{B}_T(\mathcal{N})$. Sei $\vec{\iota_*} = \sum_{\vec{\tau} \in \mathbf{B}_T(\mathcal{N})} a_{\vec{\tau}} \cdot \vec{\tau}$ mit $a_{\vec{\tau}} \in \{0, 1\}$ für alle $\vec{\tau} \in \mathbf{B}_T(\mathcal{N})$ und sei $\vec{\iota_*} \neq \vec{\mathbf{0}}$.
Ist $\mathcal{N}_{\vec{\iota_*}}$ zusammenhängend, so ist $\mathcal{N}_{\vec{\iota_*}}$ ein wohlgeformtes Free-Choice-Netz.

Beweis:
Der Beweis erfolgt per Induktion über $n = \sum_{\vec{\tau} \in \mathbf{B}_T(\mathcal{N})} a_{\vec{\tau}}$. Ist $n = 1$, so ist $\mathcal{N}_{\vec{\iota_*}}$ ein markierter Graph und nach Satz 3.66 ein wohlgeformtes Free-Choice-Netz.

Sei die Aussage also für n bewiesen, und sei sie nun für $n+1$ zu zeigen. Dann gibt es eine T-Invariante $\vec{\iota_\#} = \vec{\iota_*} - \vec{\tau}$, wobei $\vec{\tau} \in \mathbf{B}_T(\mathcal{N})$ und $a_{\vec{\tau}} = 1$ ist, so dass für $\mathcal{N}_{\vec{\iota_\#}}$ die Induktionsannahme gilt. Sei zudem nicht $\vec{\iota_\#} > \vec{\tau}$, da andernfalls $\mathcal{N}_{\vec{\iota_\#}} = \mathcal{N}_{\vec{\iota_*}}$ ist und die Aussage sofort gilt. Da $\mathcal{N}_{\vec{\iota_*}}$ aufgrund der Konstruktion eine

positive T-Invariante besitzt, soll nun gezeigt werden, dass $\mathcal{N}_{\vec{\iota_*}}$ auch eine positive S-Invariante besitzt, da dann das Rang-Theorem angewandt werden kann.

Jede minimale S-Invariante von $\mathcal{N}$ ist auch eine minimale S-Invariante in $\mathcal{N}_{\vec{\iota_*}}$, da bei der Konstruktion der Vor- und der Nachbereich jeder Transition erhalten bleibt, und somit auch dort $\sum_{p\in {}^\bullet t}(\vec{\kappa}(p)) = \sum_{p\in t^\bullet}(\vec{\kappa}(p)) = 1$ für jede minimale S-Invariante $\vec{\kappa}$ gilt (vgl. Satz 3.52). Somit ist $\mathcal{N}_{\vec{\iota_*}}$ durch uniforme S-Invarianten überdeckbar und besitzt eine positve S-Invariante. Folglich besitzt $\mathcal{N}_{\vec{\iota_*}}$ eine positive S-Invariante (aufgrund der S-Überdeckbarkeit), eine positive T-Invariante (aufgrund der Konstruktion) und nach Lemma 5.16 die Free-Choice-Eigenschaft, und es lässt sich das Rang-Theorem aus Satz 3.58 auf $\mathcal{N}_{\vec{\iota_*}}$ anwenden, wonach $\mathcal{N}_{\vec{\iota_*}}$ wohlgeformt ist, wenn $\mathrm{rg}([\mathcal{N}_{\vec{\iota_*}}]) = |C_{\mathcal{N}_{\vec{\iota_*}}}| - 1$ gilt.

Sei nun $T_A \stackrel{\text{def}}{=} \|\vec{\iota}\| \setminus \|\vec{\iota_\#}\|$ die Menge der Transitionen, die in $\mathcal{N}_{\vec{\iota_*}}$, nicht aber in $\mathcal{N}_{\vec{\iota_\#}}$ existieren. Da $\mathcal{N}_{\vec{\iota_*}}$ ein Free-Choice-Netz ist, gilt für jede Transition $t \in T_A$, dass ${}^\bullet t \cap P_{\vec{\iota_\#}} \neq \emptyset$ die Existenz einer Transition $t' \in T_{\vec{\iota_\#}}$ impliziert mit ${}^\bullet t = {}^\bullet t'$. Sei $T_X = \{t \in T_A \mid {}^\bullet t \cap P_{\vec{\iota_\#}} \neq \emptyset\}$. Dann nimmt die Zahl der Cluster um $|T_A| - |T_X|$ zu, da das Subnetz einer minimalen T-Invariante eine T-Komponente (ein markierter Graph) ist und somit zu jeder Transition ein eigenes Cluster besitzt (da $|p^\bullet| = 1$ für alle Stellen gilt). Es ist zudem $\mathrm{rg}([\mathcal{N}_{\vec{\iota_*}}]) = |T_{\vec{\iota_*}}| - |\mathbf{B}_T(\mathcal{N}_{\vec{\iota_*}})|$ und gleiches gilt für $\mathrm{rg}([\mathcal{N}_{\vec{\iota_\#}}])$. Dann ergibt sich $\mathrm{rg}([\mathcal{N}_{\vec{\iota_*}}]) = \mathrm{rg}([\mathcal{N}_{\vec{\iota_\#}}]) + |T_A| - (|\mathbf{B}_T(\mathcal{N}_{\vec{\iota_*}})| - |\mathbf{B}_T(\mathcal{N}_{\vec{\iota_\#}})|)$, was sich zu $\mathrm{rg}([\mathcal{N}_{\vec{\iota_*}}]) - \mathrm{rg}([\mathcal{N}_{\vec{\iota_\#}}]) = |T_A| - (|\mathbf{B}_T(\mathcal{N}_{\vec{\iota_*}})| - |\mathbf{B}_T(\mathcal{N}_{\vec{\iota_\#}})|)$ umformen lässt. Die Differenz im Rang entspricht aber in einem wohlgeformten Netz genau der Zahl der neuen Cluster, so dass $\mathcal{N}_{\vec{\iota_*}}$ genau dann wohlgeformt ist, wenn $|T_A| - (|\mathbf{B}_T(\mathcal{N}_{\vec{\iota_*}})| - |\mathbf{B}_T(\mathcal{N}_{\vec{\iota_\#}})|) = |T_A| - |T_X|$ ist, was sich als $|\mathbf{B}_T(\mathcal{N}_{\vec{\iota_*}})| - |\mathbf{B}_T(\mathcal{N}_{\vec{\iota_\#}})| = |T_X|$ ergibt. Ausgehend von der Induktionsannahme, dass $\mathcal{N}_{\vec{\iota_\#}}$ wohlgeformt ist, ist also $\mathcal{N}_{\vec{\iota_*}}$ wohlgeformt gdw. $|\mathbf{B}_T(\mathcal{N}_{\vec{\iota_*}})| - |\mathbf{B}_T(\mathcal{N}_{\vec{\iota_\#}})| = |T_X|$ gilt.

Wäre diese Eigenschaft verletzt, so müsste es durch Hinzufügen der weiteren T-Komponenten aus $\mathbf{B}_T(\mathcal{N})$ möglich sein, soviele Cluster zu erzeugen, dass für das gesamte Netz $|T| - |C_{\mathcal{N}}| + 1 = |\mathbf{B}_T(\mathcal{N})|$ gilt, da $\mathcal{N}$ wohlgeformt ist. Folglich müsste für jede minimale T-Invariante, für die $|\mathbf{B}_T(\mathcal{N}_{\vec{\iota_*}})| - |\mathbf{B}_T(\mathcal{N}_{\vec{\iota_\#}})| > |T_X|$ gilt, auch eine minimale T-Invariante existieren, für die $|\mathbf{B}_T(\mathcal{N}_{\vec{\iota_*}})| - |\mathbf{B}_T(\mathcal{N}_{\vec{\iota_\#}})| < |T_X|$ wäre und andersherum.

Sei $\Delta_{\mathbf{B}_T} = |\mathbf{B}_T(\mathcal{N}_{\vec{\iota_*}})| - |\mathbf{B}_T(\mathcal{N}_{\vec{\iota_\#}})|$ und sei $\Delta_{\mathbf{B}_S} = |\mathbf{B}_S(\mathcal{N}_{\vec{\iota_*}})| - |\mathbf{B}_S(\mathcal{N}_{\vec{\iota_\#}})|$, so ist zu zeigen, dass $\Delta_{\mathbf{B}_T} > |T_X|$ nicht möglich ist. Dies wiederum ist nach Lemma 5.13 äquivalent damit, zu zeigen, dass $\Delta_{\mathbf{B}_T} = \Delta_{\mathbf{B}_S} + |T_A| - |P_A| > |T_X|$ nicht möglich ist, wobei $P_A = P_{\vec{\iota_*}} \setminus P_{\vec{\iota_\#}}$ ist. Weiterhin gilt für alle $p \in P_A$, dass $|{}^\bullet p| = |p^\bullet| = 1$ ist, da das Subnetz jeder minimalen T-Invariante eine T-Komponente ist. Betrachtet man lediglich das Subnetz aus P_A und T_A, so muss es sich dabei um ein oder mehrere markierte Graphen handeln, da diese Netze nach der Konstruktion jeweils transitionsberandete Teilnetze des Subnetzes der minimalen T-Invariante $\vec{\iota}$ (einer T-Komponente) darstellen. Die Unmöglichkeit soll über $|P_A|$ gezeigt werden. Ist $|P_A| = 0$, so muss $|T_A| = |T_X|$ sein. Da dann $\Delta_{\mathbf{B}_S} = 0$ sein muss, gilt die Aussage.

Sei die Aussage also für $|P_A| = n$ bewiesen und sei nun $|P_A| = n + 1$. Entfernt man eine Stelle $p \in P_A$ aus dem Netz, so gibt es für einen markierten Graphen drei Möglichkeiten, da sich ein solcher durch die Regeln b), c), e) aus Abbildung 5.7 auf eine Transition reduzieren lässt[5]:
(1) Es gibt eine Stelle $p' \in P_A$ mit ${}^\bullet p = {}^\bullet p'$ und $p^\bullet = p'^\bullet$. In diesem Fall ist $\Delta_{\mathbf{B}_S}$ um 1 kleiner (da $\vec{\kappa}(p) = 1$ und $\vec{\kappa}(p') = -1$ eine S-Invariante darstellt, die entfällt) und $|P_A|$ um 1 kleiner, so dass $\Delta_{\mathbf{B}_T} = \Delta_{\mathbf{B}_S} + |T_A| - |P_A|$ konstant bleibt. Da $|T_X|$ sich hierdurch nicht verändern kann, gilt die Aussage.
(2) Es ist ${}^\bullet p = p^\bullet$. In diesem Fall ist ebenfalls $\Delta_{\mathbf{B}_S}$ um 1 kleiner (da $\vec{\kappa}(p) = 1$ eine S-Invariante darstellt, die entfällt) und $|P_A|$ um 1 kleiner, so dass $\Delta_{\mathbf{B}_T} = \Delta_{\mathbf{B}_S} + |T_A| - |P_A|$ konstant bleibt. Da $|T_X|$ sich hierdurch nicht verändern kann, gilt die Aussage.
(3) Es kann Regel b) aus Abbildung 5.7 angewandt werden. In diesem Fall verringert sich die Zahl der Transitionen $|T_A|$ und die Zahl der Stellen $|P_A|$ um 1, während $\Delta_{\mathbf{B}_S}$ gleich bleibt, so dass $\Delta_{\mathbf{B}_T} = \Delta_{\mathbf{B}_S} + |T_A| - |P_A|$ ebenfalls konstant bleibt. $|T_X|$ kann hierdurch nicht größer, sondern lediglich kleiner werden, wenn die entfernte Transition auch in T_X war. Somit kann in diesem Fall nicht einmal mehr $\Delta_{\mathbf{B}_T} = |T_X|$, sondern nur $\Delta_{\mathbf{B}_T} < |T_X|$ gelten. (In diesem Fall ist das Netz nicht wohlgeformt.) Somit gilt stets $\Delta_{\mathbf{B}_T} \leq |T_X|$. (Dies gilt für alle zusammenhängenden Netze, die eine Basis minimaler T-Invarianten besitzen, deren Subnetze T-Komponenten sind, da die Free-Choice-Eigenschaft im Beweis nicht benutzt wurde.) Folglich gilt auch die Aussage dieses Satzes. □

Eine wesentliche Aussage des Beweises ist, dass stets $|\mathbf{B}_T(\mathcal{N}_{\vec{\iota}_*})| - |\mathbf{B}_T(\mathcal{N}_{\vec{\iota}_\#})| = |T_X|$ gilt. Dies soll in einem separaten Korollar festgehalten werden, da dies bedeutet, dass es zu jeder minimalen T-Invariante eine Transition gibt, die dafür sorgt, dass alle benötigten Marken zur Realisierung des T-Vektors der Transitionen, die nur dieser minimalen T-Invariante, nicht aber anderen T-Invarianten angehören, abgezogen werden.

Korollar 5.18 (Randtransitionen minimaler T-Invarianten)
Gegeben ein wohlgeformtes Free-Choice-Netz $\mathcal{N}$ und eine Basis minimaler T-Invarianten $\mathbf{B}_T(\mathcal{N})$. Sei $\vec{\iota_\Sigma} = \sum_{\vec{\tau} \in \mathbf{B}_T(\mathcal{N})} a_{\vec{\tau}} \cdot \vec{\tau}$ mit $a_{\vec{\tau}} \in \{0, 1\}$ für alle $\vec{\tau} \in \mathbf{B}_T(\mathcal{N})$, so dass $\mathcal{N}_{\vec{\iota_\Sigma}}$ zusammenhängend ist. Sei $\vec{\iota_1} \in \mathbf{B}_T(\mathcal{N})$ mit $a_{\vec{\iota_1}} = 0$ und $T_A \stackrel{\text{def}}{=} \|\vec{\iota_1}\| \setminus \|\vec{\iota_\Sigma}\|$ und $T_A \neq \emptyset$, und sei $\vec{\iota_*} = \vec{\iota_\Sigma} + \vec{\iota_1}$. Dann gilt für Basen $\mathbf{B}_T(\mathcal{N}_{\vec{\iota_*}})$ und $\mathbf{B}_T(\mathcal{N}_{\vec{\iota_\Sigma}})$

$$|\mathbf{B}_T(\mathcal{N}_{\vec{\iota_*}})| - |\mathbf{B}_T(\mathcal{N}_{\vec{\iota_\Sigma}})| = |\{t \in T_A \mid {}^\bullet t \cap P_{\vec{\iota_\Sigma}} \neq \emptyset\}|$$

Die Aussage aus Lemma 5.17 lässt sich nun noch deutlich knapper darstellen, da

[5] Dies wird etwa in Murata (1989) gezeigt, ein Nachweis findet sich jedoch auch bereits bei Ullrich (1977). Markierungen können bei diesem Vorgehen ignoriert werden, da lediglich die Wohlgeformtheit betrachtet wird.

jede semi-positive T-Invariante eines wohlgeformten Free-Choice-Netzes eine Linearkombination minimaler T-Invarianten einer Basis ist. Indem gezeigt wird, dass es zu jeder semi-positiven T-Invariante eine Basis gibt, die die Kriterien aus Lemma 5.17 erfüllt, lässt sich folgende Aussage zeigen.

Satz 5.19 (Wohlgeformtheit der T-Invariantensubnetze)
Gegeben ein wohlgeformtes Free-Choice-Netz $\mathcal{N}$ und eine semi-positive T-Invariante $\vec{\iota_*}$ von $\mathcal{N}$ mit $\vec{\iota_*} \neq \vec{\mathbf{0}}$.
Ist $\mathcal{N}_{\vec{\iota_*}}$ zusammenhängend, so ist $\mathcal{N}_{\vec{\iota_*}}$ ein wohlgeformtes Free-Choice-Netz.

Beweis:
Nach Lemma 5.17 ist zu zeigen, dass es eine Basis minimaler T-Invarianten $\mathbf{B}_T(\mathcal{N})$ gibt, so dass $\vec{\iota_x} = \sum_{\vec{\iota} \in \mathbf{B}_T(\mathcal{N})} a_{\vec{\iota}} \cdot \vec{\iota}$ mit $a_{\vec{\iota}} \in \{0, 1\}$ für alle $\vec{\iota} \in \mathbf{B}_T(\mathcal{N})$ ist und $\mathcal{N}_{\vec{\iota_*}} = \mathcal{N}_{\vec{\iota_x}}$ gilt. Aufgrund der Konstruktion besitzt $\mathcal{N}_{\vec{\iota_*}}$ eine positive T-Invariante, und nach Satz 3.55 ist jede T-Invariante eine Linearkombination minimaler T-Invarianten, so dass es eine Basis $\mathbf{B}_T(\mathcal{N}_{\vec{\iota_*}})$ gibt. Da die T-Invarianten aus $\mathbf{B}_T(\mathcal{N}_{\vec{\iota_*}})$ nur Transitionen aus $T_{\vec{\iota_*}}$ im Träger besitzen können, ist für $\vec{\iota_x} = \sum_{\vec{\iota} \in \mathbf{B}_T(\mathcal{N}_{\vec{\iota_*}})} \vec{\iota}$ dann $\mathcal{N}_{\vec{\iota_*}} = \mathcal{N}_{\vec{\iota_x}}$.

Sei $\textsc{Inv}_x$ die Erweiterung der Vektoren $\vec{\iota} \in \mathbf{B}_T(\mathcal{N}_{\vec{\iota_*}})$ durch 0-Einträge, so dass sie Vektoren von $\mathcal{N}$ darstellen. Es bleibt zu zeigen, dass sich $\textsc{Inv}_x$ zu einer Basis von $\mathcal{N}$ erweitern lässt, da dies dann die gesuchte Basis ist. Da $\mathbf{B}_T(\mathcal{N})$ eine Basis von $\mathcal{N}$ ist und der durch $\textsc{Inv}_x \cup \mathbf{B}_T(\mathcal{N})$ aufgespannte Vektorraum gleich dem durch $\mathbf{B}_T(\mathcal{N})$ aufgespannten ist, folgt dies direkt aus dem Basisergänzungssatz der linearen Algebra (vgl. Jänich, 1998). □

Ein interessanter Aspekt wohlgeformter Free-Choice-Netze ist, dass sämtliche Aussagen dieses Unterabschnitts auch auf das duale Netz eines wohlgeformten Free-Choice-Netzes angewandt werden können, da dieses ebenfalls ein wohlgeformtes Free-Choice-Netz ist. Somit gilt Satz 5.19 auch für S-Invarianten, da diese ja gerade die T-Invarianten des dualen Netzes darstellen. Hieraus folgt sofort der folgende Satz.

Satz 5.20 (Wohlgeformtheit der S-Invariantensubnetze)
Gegeben ein wohlgeformtes Free-Choice-Netz $\mathcal{N}$ und eine semi-positive S-Invariante $\vec{\kappa_*}$ von $\mathcal{N}$ mit $\vec{\kappa_*} \neq \vec{\mathbf{0}}$.
Ist $\mathcal{N}_{\vec{\kappa_*}}$ zusammenhängend, so ist $\mathcal{N}_{\vec{\kappa_*}}$ ein wohlgeformtes Free-Choice-Netz.

Beweis:
Dies folgt direkt aus Satz 3.62 und aus Satz 5.19 sowie aus der Beobachtung das T-Invarianten die S-Invarianten des dualen Netzes sind und umgekehrt (vgl. Desel und Esparza (1995), Proposition 6.20). □

Netzanalyse durch Betrachtung minimaler Invarianten

Die bislang dargestellten Ergebnisse sind stets von einer Basis minimaler T-Invarianten ausgegangen. Eine solche Basis ermitteln zu müssen, ist jedoch nicht immer vorteilhaft. Während zur Ermittlung einer Überdeckung eines Free-Choice-Netzes durch T-Komponenten effiziente Algorithmen existieren (vgl. Kemper, 1993), erfolgt die Ermittlung einer Basis meist durch Methoden der linearen Algebra. Es lässt sich jedoch eine Überdeckung durch T-Komponenten zu einer Basis erweitern. Dies wird im Folgenden dargestellt. Die durch die T-Komponenten induzierten minimalen T-Invarianten entsprechen einer Überdeckung durch minimale T-Invarianten. Eine solche Überdeckung durch minimale Invarianten ist analog zur Überdeckung durch Komponenten wie folgt definiert.

Definition 5.21 (Überdeckung durch minimale Invarianten)
Eine *Überdeckung* eines Netzes $\mathcal{N}$ *durch minimale T-Invarianten (S-Invarianten)* ist eine Menge minimaler T-Invarianten $\textsc{Tinv}$ (S-Invarianten $\textsc{Sinv}$), so dass gilt $\|\sum_{\vec{\iota}\in\textsc{Tinv}} \vec{\iota}\| = T \quad (\|\sum_{\vec{\kappa}\in\textsc{Sinv}} \vec{\kappa}\| = P)$. ♦

Aus einer Überdeckung durch minimale T-Invarianten lässt sich in einem wohlgeformten Free-Choice-Netz eine Basis erzeugen. Dies geschieht, indem die Subnetze der minimalen T-Invarianten der Überdeckung sukzessive zusammengefügt werden. Ist bei diesem Vorgehen die Differenz der Dimensionen der T-Invariantenvektorräume größer 1, so werden die entsprechend fehlenden minimalen T-Invarianten „nachberechnet“. Dies geschieht, indem eine der Transitionen, die eine Stelle mit dem bereits bestehenden Netz gemeinsam hat, ausgewählt wird und zu ihr eine T-Invariante berechnet wird. Diese minimale T-Invariante ist linear unabhängig und somit kann die Überdeckung um diese T-Invariante ergänzt werden, um eine Basis zu erhalten. Hiervon wird in dem im Anschluss an das folgende Lemma vorgestellten Verfahren 5.23 Gebrauch gemacht, so dass nicht nur die Korrektheit überprüft wird, sondern auch eine Basis von T-Komponenten erzeugt wird. Das folgende Lemma zeigt zunächst die Existenz einer solchen minimalen T-Invariante.

Lemma 5.22 (Minimale T-Invarianten von Randtransitionen)
Gegeben ein wohlgeformtes Free-Choice-Netz $\mathcal{N}$ und eine Überdeckung von $\mathcal{N}$ durch minimale T-Invarianten $\textsc{Tinv}$. Sei $\vec{\iota_\Sigma} = \sum_{\vec{\tau}\in\textsc{Tinv}} a_{\vec{\tau}} \cdot \vec{\tau}$ mit $a_{\vec{\tau}} \in \{0,1\}$ für alle $\vec{\tau} \in \textsc{Tinv}$, so dass $\mathcal{N}_{\vec{\iota_\Sigma}}$ zusammenhängend ist.
Zu einer minimalen T-Invariante $\vec{\iota_a} \in \textsc{Tinv}$ mit $a_{\vec{\iota_a}} = 0$ und $T_A \stackrel{\text{def}}{=} \|\vec{\iota_a}\| \setminus \|\vec{\iota_\Sigma}\|$ mit $T_A \neq \emptyset$ sei $\vec{\iota_*} = \vec{\iota_\Sigma} + \vec{\iota_a}$. Dann gilt für $T_X \stackrel{\text{def}}{=} \{t \in T_A \mid {}^\bullet t \cap P_{\vec{\iota_\Sigma}} \neq \emptyset\}$, dass $t \in T_X$ impliziert, dass es in $\mathcal{N}_{\vec{\iota_*}}$ eine minimale T-Invariante $\vec{\iota_t}$ gibt mit $\|\vec{\iota_t}\| \cap T_X = \{t\}$.

Beweis:

Es sind $\mathcal{N}_{\vec{\iota}_*}$ und $\mathcal{N}_{\vec{\iota}_\Sigma}$ nach Satz 5.19 zusammenhängende wohlgeformte Free-Choice-Netze und besitzen entsprechend eine Basis minimaler T-Invarianten. Sei $\mathbf{B}_T(\mathcal{N}_{\vec{\iota}_\Sigma})$ eine Basis minimaler T-Invarianten von $\mathcal{N}_{\vec{\iota}_\Sigma}$. Da $\vec{\iota_a}$ linear unabhängig zu den T-Vektoren in $\mathbf{B}_T(\mathcal{N}_{\vec{\iota}_\Sigma})$ ist, gibt es eine Basis von $\mathcal{N}_{\vec{\iota}_*}$ der Form $\mathbf{B}_T(\mathcal{N}_{\vec{\iota}_\Sigma}) \cup \{\vec{\iota_a}\} \cup \textsc{Inv}^*$ für ein $\textsc{Inv}^* = \{\vec{\iota_1}, \ldots, \vec{\iota_m}\}$. Dabei ist $m = |\mathbf{B}_T(\mathcal{N}_{\vec{\iota}_*})| - |\mathbf{B}_T(\mathcal{N}_{\vec{\iota}_\Sigma})| - 1$ die Differenz der Dimensionen der T-Invariantenvektorräume abzüglich 1 (für $\vec{\iota_a}$). Da $\mathcal{N}_{\vec{\iota}_*}$ nach Satz 5.19 ein wohlgeformtes Free-Choice-Netz ist, gilt nach Korollar 5.18 stets $|T_X| = m + 1$.

Der Beweis erfolgt über $|T_X|$. Ist $|T_X| = 1$, so ist $\vec{\iota_a}$ der gesuchte Vektor und $m = 0$. Sei die Aussage also für $|T_X| = n$ bewiesen und sei nun $|T_X| = n+1$. Würden die minimalen T-Invarianten aus $\textsc{Inv}^*$ ebenfalls alle Transitionen aus $\|\vec{\iota_a}\| \setminus \|\vec{\iota_\Sigma}\|$ in ihrem Träger besitzen, so wäre $\mathbf{B}_T(\mathcal{N}_{\vec{\iota}_\Sigma}) \cup \{\vec{\iota_a}\} \cup \textsc{Inv}^*$ keine Basis. Andererseits ist aus dem gleichen Grund $\|\vec{\iota_a}\| \cap \|\vec{\iota_j}\| \neq \emptyset$ und $\vec{\iota_j} < \vec{\iota_*}$ für $j \in \{1, \ldots, m\}$. Somit lässt sich die Induktionsannahme auf alle minimalen T-Invarianten aus $\textsc{Inv}^*$ anwenden, wenn diese statt $\vec{\iota_a}$ zu $\vec{\iota_\Sigma}$ hinzuaddiert werden. Ist $t \in \|\vec{\iota_j}\|$ für ein $j \in \{1, \ldots, m\}$, so folgt die Aussage aus der Induktionsannahme.

Andernfalls ist $\vec{\iota_n} = \vec{\iota_a} - \vec{\iota_1}$ eine T-Invariante, die n oder weniger Transitionen aus T_X in ihrem Träger besitzt und für die $t \in \|\vec{\iota_n}\|$ ist. Ist $\vec{\iota_n}$ nicht semi-positiv, so kann dies durch Hinzuaddieren minimaler T-Invarianten aus $\mathbf{B}_T(\mathcal{N}_{\vec{\iota}_\Sigma})$ erreicht werden. Ist das Resultat nicht minimal, so gibt es eine minimale T-Invariante von $\mathcal{N}_{\vec{\iota}_\Sigma}$, deren Träger eine Teilmenge der T-Invariante ist und die somit kleiner ist als die gegebene T-Invariante. Somit kann diese minimale T-Invariante subtrahiert werden, bis aus $\vec{\iota_n}$ eine minimale T-Invariante geworden ist, die n oder weniger Transitionen aus T_X im Träger besitzt und die t im Träger besitzt. Auf diese minimale T-Invariante lässt sich dann die Induktionsannahme anwenden.

Fügt man dem Netz $\mathcal{N}_{\vec{\iota}_\Sigma}$ also statt $\vec{\iota_a}$ die minimale T-Invariante $\vec{\iota_n}$ hinzu, so lässt sich nach Induktionsannahme für jede Transition aus dem dann resultierenden T_X eine minimale T-Invariante finden, die nur diese Transition überdeckt. Da t sich auch in dieser Menge befindet, gilt die Aussage. □

Korollar 5.18 legt eine Analyse nahe, bei der sukzessive minimale T-Invarianten zu einem Netz hinzugefügt werden. Wenn man eine Basis minimaler T-Invarianten zusammen mit einer Ordnung auf den T-Invarianten hat, so dass stets nur eine Transition in $\{t \in T_A \mid {}^\bullet t \cap P_{\vec{\iota}_\Sigma} \neq \emptyset\}$ ist, dann lassen sich diese minimalen T-Invarianten gemäß der Ordnung zu einem Netz zusammenfügen, wobei in jedem Schritt gewährleistet ist, dass das Netz wohlgeformt ist. Lemma 5.22 legt ein ähnliches Verfahren zur Bildung einer Basis nahe. Beides soll in folgendem Verfahren zusammen umgesetzt werden. Dabei bezeichnet $\mathcal{N}_1 \oplus \mathcal{N}_2$ die in Definition 3.29 eingeführte Vereinigung zweier Netze. Das Färben in folgendem Algorithmus stellt das Markieren einer T-Invariante als benutzt dar.

Verfahren 5.23 (Wohlgeformtheit eines T-überdeckbaren FC-Netzes)
Gegeben ein zusammenhängendes Free-Choice-Netz $\mathcal{N}$.

(I) Ermittle eine Überdeckung durch minimale T-Invarianten TINV und ermittle eine positive S-Invariante.
Weise das Netz als nicht wohlgeformt aus, wenn (1) $\mathcal{N}$ keine positive S-Invariante besitzt, wenn (2) für $\mathcal{N}$ keine Überdeckung durch minimale T-Invarianten existiert oder wenn (3) die Subnetze der T-Invarianten aus TINV nicht allesamt T-Komponenten sind.

(II) Wähle eine minimale T-Invariante $\vec{\iota} \in$ TINV, färbe $\vec{\iota}$ und initialisiere $j = 1$ und $\mathcal{N}_1^\# = \mathcal{N}_{\vec{\iota}}$.

(1) Gibt es keine ungefärbten T-Invarianten in TINV, so ist $\mathcal{N}$ wohlgeformt und die T-Invarianten aus TINV stellen eine Basis minimaler T-Invarianten dar.

(2) Wähle eine ungefärbte minimale T-Invariante $\vec{\iota} \in$ TINV derart, dass $\mathcal{N}_{\vec{\iota}}$ und $\mathcal{N}_j^\#$ mindestens eine Stelle gemeinsam haben.
Ist $\|\vec{\iota}\| \subseteq T_j^\#$, so entferne $\vec{\iota}$ aus TINV und gehe zu (1).

(3) Setze $T_A = \|\vec{\iota}\| \setminus T_j^\#$ und $T_X = \{t \in T_A \mid {}^\bullet t \cap P_j^\# \neq \emptyset\}$.

(3.1) Ist $|T_X| = 1$, so färbe $\vec{\iota}$ und setze $\mathcal{N}_{j+1}^\# = \mathcal{N}_j^\# \oplus \mathcal{N}_{\vec{\iota}}$ (vgl. Def. 3.29).
Inkrementiere j und gehe zu (1).

(3.2) Andernfalls ($|T_X| > 1$)

- Wähle ein $t \in T_X$ und berechne eine minimale T-Invariante $\vec{\iota_a}$ von $\mathcal{N}_j^\# \oplus \mathcal{N}_{\vec{\iota}}$, die t, aber keine andere Transition aus T_X überdeckt.
- Gibt es eine solche T-Invariante $\vec{\iota_a}$, so setze $\vec{\iota} = \vec{\iota_a}$ und füge $\vec{\iota}$ zu TINV hinzu. Gehe dann zu (3).
- Andernfalls weise die minimale T-Invariante $\vec{\iota}$ als fehlerhaft aus und breche ab.

■

Definition und Satz 5.24 (Korrektheit des Verfahrens 5.23)
Das Verfahren 5.23 ist korrekt gdw. die Ausgabe der Wohlgeformtheit von $\mathcal{N}$ durch Verfahren 5.23 impliziert, dass $\mathcal{N}$ wohlgeformt ist und dass TINV eine Basis minimaler T-Invarianten darstellt. Das Verfahren 5.23 ist vollständig gdw. die Wohlgeformtheit von $\mathcal{N}$ impliziert, dass das Verfahren 5.23 die Wohlgeformtheit von $\mathcal{N}$ ausgibt.
Das Verfahren 5.23 terminiert und ist vollständig und korrekt. ♦

Beweis:
Das Verfahren soll zunächst betrachtet werden, ohne dass auf die Berechnung der T-Komponente in Schritt 4.2 detaillierter eingegangen wird.
[*Termination*] Es muss die Termination der Schleife aus Schritt 1 bis Schritt 3 gezeigt werden. Ist $|T_X| = 1$ in Schritt 3.1, so wird in jedem Durchgang eine T-Invariante der endlichen Menge TINV gefärbt oder aus der Menge entfernt. Da der Algorithmus abbricht, wenn es keine ungefärbten T-Invarianten mehr in TINV gibt, gilt die Aussage.

Ist $|T_X| > 1$, so wird eine minimale T-Invariante von $\mathcal{N}^{\#}_{j+1}$ berechnet, die t, aber keine andere Transition aus T_X überdeckt. Diese Berechnung soll hier zunächst als terminierend angenommen werden, da sie später ausführlicher behandelt wird. Ist diese Berechnung nicht erfolgreich, so bricht das Verfahren ab und terminiert folglich. Andernfalls fährt das Verfahren mit Schritt 3 fort und färbt die neu berechnete T-Invariante. Aufgrund der Konstruktion ist $|T_X| = 1$, da nur eine Transition aus T_X und ansonsten nur Transitionen aus $T^{\#}_j$ im Träger von $\vec{\iota}$ sind. (Wäre $|T_X| \neq 1$, so müsste dann eine Transition existieren, die weder in $T^{\#}_j$ noch in $\|\vec{\iota_a}\|$ aber in $\|\vec{\iota}\|$ ist, was nicht sein kann, da $\vec{\iota}$ eine T-Invariante von $\mathcal{N}^{\#}_{j+1}$ ist.) Somit terminiert das Verfahren auch in diesem Fall.
[*Korrektheit*] Angenommen Verfahren 5.23 gibt für ein zusammenhängendes Free-Choice-Netz „wohlgeformt“ aus. Dann besitzt dieses Netz eine positive S- und eine positive T-Invariante und es lässt sich nach Satz 3.58 das Rang-Theorem anwenden, wonach $\mathrm{rg}([\mathcal{N}]) = |C_{\mathcal{N}}| - 1$ hinreichend ist, damit $\mathcal{N}$ ein wohlgeformtes Free-Choice-Netz ist. Es soll gezeigt werden, dass alle Netze $\mathcal{N}^{\#}_j$ wohlgeformt sein müssen, damit Verfahren 5.23 für ein zusammenhängendes Free-Choice-Netz „wohlgeformt“ ausgibt. Der Beweis erfolgt über $|\mathbf{B}_T(\mathcal{N})|$.

Ist $|\mathbf{B}_T(\mathcal{N})| = 1$, so gibt das Verfahren in Schritt 1 richtigerweise die Wohlgeformtheit aus. Zudem sind die gefärbten minimalen T-Invarianten von TINV eine Basis der minimalen T-Invarianten von $\mathcal{N}$. Sei die Annahme also für $|\mathbf{B}_T(\mathcal{N})| = n$ bewiesen und sei nun $|\mathbf{B}_T(\mathcal{N})| = n + 1$. Damit das Verfahren in Schritt 1 „wohlgeformt“ angibt, muss zuvor in Schritt 3.1 eine minimale T-Invariante aus TINV gefärbt worden sein. Entfernt man die Knoten aus $\mathcal{N}$, die durch die letzte T-Invariante hinzugekommen sind, so erhält man das Netz $\mathcal{N}^{\#}_{j-1}$, das eine Basis der Dimension n besitzt und somit nach Induktionsannahme wohlgeformt ist. Wäre $\mathcal{N}$ nun nicht wohlgeformt, so dürfte es nach Korollar 5.18 keine minimale T-Invariante in TINV mit $|T_X| = 1$ geben. Dann wird allerdings in Schritt 3.2 die minimale T-Invariante als fehlerhaft ausgewiesen.

Somit bleibt zu zeigen, dass die gefärbten minimalen T-Invarianten aus TINV eine Basis von $\mathcal{N}^{\#}_{j+1}$ darstellen. Da $|T_X| = 1$ ist und $\mathcal{N}^{\#}_{j+1}$ wohlgeformt ist, muss eine minimale linear unabhängige T-Invariante zu den gefärbten T-Invarianten von TINV in jedem Schritt hinzukommen. Somit sind sämtliche gefärbten minimalen T-Invarianten in TINV linear unabhängig, und es bleibt zu zeigen, dass ihre Zahl

$|\mathbf{B}_T(\mathcal{N})|$ ist. Eine minimale T-Invariante wird einmal zur Initialisierung und jedesmal dann gefärbt, wenn $|T_X| = 1$ ist. Somit ist die Zahl der gefärbten minimalen T-Invarianten in Tinv gleich j. Andererseits ist die Zahl der Cluster in $\mathcal{N}$ gleich $|T| - (j-1)$, da jede Transition aus T_X einem bestehenden Cluster angehört und Schritt 3.1 $j-1$ mal ausgeführt wurde. Nach dem Rang-Theorem ist $|T| - |\mathbf{B}_T(\mathcal{N})| = |T| - (j-1) - 1$ und somit $|\mathbf{B}_T(\mathcal{N})| = j$. Dann sind die j gefärbten linear unabhängigen minimalen T-Invarianten in Tinv eine Basis.
[*Vollständigkeit*] Sei $\mathcal{N}$ wohlgeformt. Die Gültigkeit der Überprüfungen aus Schritt I folgt direkt aus dem Rang-Theorem in Satz 3.58, nach dem es eine positive S- und eine positive T-Invariante geben muss. Folglich gibt es auch eine Überdeckung durch minimale T-Invarianten, deren Subnetze nach Lemma 5.14 T-Komponenten sein müssen. Schritt II ist dann stets möglich.

Nach Satz 5.19 ist jedes Netz $\mathcal{N}_j^\#$ wohlgeformt. Die Aussage folgt somit durch den Nachweis, dass das Verfahren mit der Wohlgeformtheit von $\mathcal{N}_j^\#$ und der Wohlgeformtheit von $\mathcal{N}_{j+1}^\#$ korrekt ausgibt, dass $\mathcal{N}_{j+1}^\#$ wohlgeformt ist. Da die Wohlgeformtheit von $\mathcal{N}_1^\#$ korrekt ermittelt wird, lässt sich die Aussage dann induktiv schließen.

Ist $\mathcal{N}_j^\# = \mathcal{N}$, so terminiert das Verfahren korrekt, da alle minimalen T-Invarianten aus Tinv gefärbt oder entfernt wurden. Seien also $\mathcal{N}_j^\#$ und $\mathcal{N}_{j+1}^\#$ wohlgeformte Free-Choice-Netze und seien die gefärbten T-Invarianten aus Tinv eine Basis minimaler T-Invarianten von $\mathcal{N}_j^\#$ (Induktionsannahme). Da $\mathcal{N}$ zusammenhängend ist und da Tinv eine Überdeckung ist, gibt es in Schritt 2 eine entsprechende minimale T-Invariante.

Da $\mathcal{N}$ zusammenhängend ist, ist $|T_X| \geq 1$. Ist $|T_X| = 1$, so wird $\mathcal{N}_{j+1}^\#$ als wohlgeformtes Free-Choice-Netz korrekt identifiziert. Andernfalls wird zunächst für ein $t \in T_X$ eine minimale T-Invariante $\vec{\iota_a}$ von $\mathcal{N}_j^\# \oplus \mathcal{N}_{\vec{\iota}}$ berechnet, die t, aber keine andere Transition aus T_X überdeckt. Eine solche T-Invariante existiert nach Lemma 5.22. Somit ist für $\vec{\iota}$ in Schritt 3 dann $|T_X| = 1$, und es wird $\vec{\iota}$ gefärbt und $\mathcal{N}_{j+1}^\# = \mathcal{N}_j^\# \oplus \mathcal{N}_{\vec{\iota}}$ gesetzt. Auch in diesem Fall wird $\mathcal{N}_{j+1}^\#$ korrekt als wohlgeformt identifiziert. Somit ist das Verfahren vollständig. □

Das Verfahren soll an einem Beispiel erläutert werden, was im Folgenden geschieht. Beispiel 5.3 stellt den Fall dar, dass keine weitere T-Invarianten berechnet werden müssen, während Beispiel 5.4 den Ablauf darstellt, wenn weitere minimale T-Invarianten berechnet werden müssen.

Beispiel 5.3: Dieses Beispiel greift die Netze aus Beispiel 5.2 wieder auf und erläutert Verfahren 5.23 an ihnen. Wie bereits in Beispiel 5.2 beschrieben, besitzt das Netz in Abbildung 5.2 eine positive S-Invariante und die minimalen T-Invarianten $\vec{\iota_1} = [t_1, t_2, t_3, t_4, t_9, t^*]$, $\vec{\iota_2} = [t_1, t_4, t_5, t_6, t_9, t^*]$ und $\vec{\iota_3} = [t_1, t_6, t_7, t_8, t_9, t^*]$. Diese

T-Invarianten stellen eine Überdeckung (sogar eine Basis) minimaler T-Invarianten Tinv von $\mathcal{N}$ dar (*Schritt I*).

Als erste minimale T-Invariante wird $\vec{\iota_1} \in \mathbf{B}_T(\mathcal{N})$ ausgewählt und gefärbt. Es folgt die Initialisierung mit $j = 1$ und $\mathcal{N}_1^\# = \mathcal{N}_{\vec{\iota_1}}$ (*Schritt II* und *Schritt 1*). Dies ist in Abbildung 5.3a) dargestellt.

In *Schritt 2* wird nun das Subnetz $\mathcal{N}_{\vec{\iota_2}}$ der ungefärbten T-Invariante $\vec{\iota_2} \in \mathbf{B}_T(\mathcal{N})$ ausgewählt. Es ist nun $\mathcal{N}_2^\# = \mathcal{N}_1^\# \oplus \mathcal{N}_{\vec{\iota_2}}$, was in Abbildung 5.3b) dargestellt ist. Zudem gilt $T_A = \|\vec{\iota_2}\| \setminus T_1^\# = \{t_5, t_6\}$. Beim Hinzufügen von $\mathcal{N}_{\vec{\iota_2}}$ ist $|\{t \in T_A \mid {}^\bullet t \cap P^\# \neq \emptyset\}| = 1$, da $P^\# = \{i, o, p_1, p_2, p_3, p_4, p_7, p_8\}$ und somit $\{t \in T_A \mid {}^\bullet t \cap P^\# \neq \emptyset\} = \{t_5\}$ gilt. Folglich wird j zu 1 inkrementiert, $\vec{\iota_2}$ gefärbt und Schritt 3 mit der nächsten minimalen T-Invariante wiederholt, da das Netz $\mathcal{N}_2^\#$ wohlgeformt ist.

Als nächstes wird wiederum in *Schritt 3* das Subnetz der T-Invariante $\vec{\iota_3} \in \mathbf{B}_T(\mathcal{N})$ hinzugefügt. Hierbei ist $\{t \in T_A \mid {}^\bullet t \cap P^\# \neq \emptyset\} = \{t_7\}$, und somit $|\{t \in T_A \mid {}^\bullet t \cap P^\# \neq \emptyset\}| = 1$. Folglich muss wiederum keine ungefärbte T-Invariante gefunden werden und das Netz $\mathcal{N}_2^\#$ ist weiterhin wohlgeformt. Da nach dem Färben von $\vec{\iota_3}$ keine weitere ungefärbte minimale T-Invariante aus Tinv mehr existiert, ist $\mathcal{N}$ wohlgeformt.

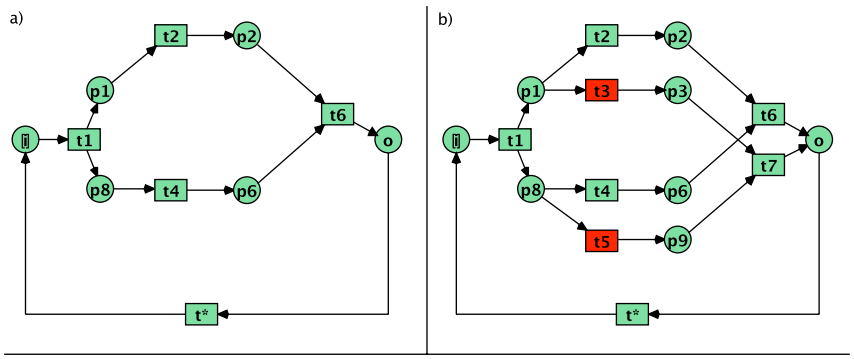


Abbildung 5.4: Verfahren 5.23 bei einem nicht korrekten Free-Choice-Netz

Ein Beispiel eines nicht wohlgeformten Free-Choice-Netzes findet sich in Abbildung 5.4b). Das dort dargestellte Netz besitzt wiederum eine positive S-Invariante und besitzt zudem die minimalen T-Invarianten $\vec{\iota_1} = [t_1, t_2, t_4, t_6, t^*]$ und $\vec{\iota_2} = [t_1, t_3, t_5, t_7, t^*]$. Das Netz $\mathcal{N}_1^\#$ besteht aus dem Subnetz $\mathcal{N}_{\vec{\iota_1}}$, was in Abbildung 5.4a) dargestellt ist. Beim Hinzufügen des Subnetzes $\mathcal{N}_{\vec{\iota_2}}$ ist nun jedoch $T_A = \{t_3, t_5, t_7\}$ und es ist $\{t \in T_A \mid {}^\bullet t \cap P^\# \neq \emptyset\} = \{t_3, t_5\}$. Somit ist $|\{t \in T_A \mid {}^\bullet t \cap P^\# \neq \emptyset\}| = 2$, und es müsste eine minimale T-Invariante berechenbar sein, die nur t_3, nicht aber t_5 überdeckt. Dies ist jedoch nicht der Fall, so dass das Netz nicht wohlgeformt sein kann. ◇

Die einzelnen Schritte des Verfahrens lassen sich wie im Folgenden dargestellt interpretieren. Gibt es keine Basis uniformer T-Invarianten, so kann $\mathcal{N}$ nicht wohl-

geformt sein. Entsprechend können die Transitionen als fehlerhaft ausgewiesen werden, die nicht durch eine minimale T-Invariante überdeckt werden. Gibt es zu einer Stelle keine positive S-Invariante, so kann diese Stelle ebenfalls nicht Teil eines wohlgeformten Free-Choice-Netzes sein und kann als fehlerhaft ausgewiesen werden. In vielen Fällen ist die Stelle in diesem Fall nicht sicher.

Sind die ersten beiden Bedingungen überprüft, so ist bereits jede minimale T-Invariante in einer Markierung realisierbar. Es bleibt nun festzustellen, ob es zu einer Verklemmung kommen kann, indem zwei Transitionen feuern, die nicht derselben minimalen T-Invariante angehören. Dies wird durch das Verfahren 5.23 erreicht. Ist eine minimale T-Invariante fehlerhaft, so bedeutet dies, dass das Netz bei den Transitionen $\{t \in T_A \mid {}^\bullet t \cap P^\# \neq \emptyset\}$ verklemmen kann. Entsprechend sind diese Transitionen in Abbildung 5.4d) rot markiert.

Abschließend soll noch ein Beispiel für die Anwendung des Verfahrens 5.23 folgen, wenn die Kardinalität von Tinv kleiner ist als die Dimension des T-Invariantenvektorraums. Dies ist in folgendem Beispiel gegeben.

Beispiel 5.4:

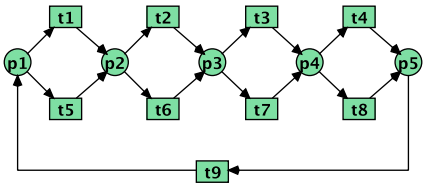


Abbildung 5.5: Ein wohlgeformtes Free-Choice-Netz zur Darstellung der Ordnung auf T-Invarianten

Das in Abbildung 5.5 dargestellte Netz besitzt eine Überdeckung durch minimale T-Invarianten Tinv und eine positive S-Invariante. Tinv umfasst dabei beispielsweise die folgenden T-Invarianten: $\vec{\iota_1} = [t_1, t_2, t_3, t_8, t_9]$ und $\vec{\iota_2} = [t_5, t_6, t_7, t_4, t_9]$.

Es ist Tinv $= \{\vec{\iota_1}, \vec{\iota_2}\}$ offensichtlich eine Überdeckung durch minimale T-Invarianten. In Schritt II wählt das Verfahren die minimale T-Invariante $\vec{\iota_1}$ aus. In Schritt 2 wird entsprechend die minimale T-Invariante $\vec{\iota_2}$ ausgewählt. In Schritt 3 ist $T_A = \{t_5, t_6, t_7, t_4\}$ und $T_X = T_A$ und folglich $|T_X| > 1$. Somit wird in Schritt 3.2 für t_5 eine minimale T-Invariante $\vec{\iota_3} = [t_5, t_2, t_3, t_8, t_9]$ berechnet. Für diese ist dann $T_A = \{t_5\}$ und $T_X = T_A$. Somit umfasst $\mathcal{N}_2^\#$ die Transitionen $t_1, t_2, t_3, t_5, t_8, t_9$, und es ist $\mathbf{B}_T(\mathcal{N}_2^\#) = \{\vec{\iota_1}, \vec{\iota_3}\}$. Beim nächsten Durchlauf von Schritt 2 wird wieder $\vec{\iota_2}$ ausgewählt. In Schritt 3 ist $T_A = \{t_6, t_7, t_4\}$ und $T_X = T_A$ und folglich $|T_X| > 1$. Somit wird in Schritt 3.2 für t_6 eine minimale T-Invariante $\vec{\iota_4} = [t_5, t_6, t_3, t_8, t_9]$ berechnet. Für diese ist dann $T_A = \{t_6\}$ und $T_X = T_A$. Somit umfasst $\mathcal{N}_3^\#$ die Transitionen $t_1, t_2, t_3, t_5, t_6, t_8, t_9$, und es ist $\mathbf{B}_T(\mathcal{N}_3^\#) = \{\vec{\iota_1}, \vec{\iota_3}, \vec{\iota_4}\}$.

Dies wird fortgeführt, so dass abschließend $\textsc{Tinv} = \{\vec{\iota_1}, \vec{\iota_2}, \vec{\iota_3}, \vec{\iota_4}, \vec{\iota_5}\}$ mit $\vec{\iota_5} = [t_5, t_6, t_7, t_8, t_9]$ eine Basis von $\mathcal{N}$ darstellt. Gleichzeitig wird das Netz als wohlgeformt ausgegeben. ◇

Ungeklärt ist bislang die Berechnung der T-Komponente, die genau eine Transition aus T_X überdeckt in Schritt 3.2 von Verfahren 5.23. Dies kann mit dem Verfahren von Kemper (1993) erfolgen, das es erlaubt, zu einer gegebenen Stelle p in einem wohlgeformten Netz eine S-Komponente zu ermitteln. Wiederum kann das Verfahren über die Nutzung des dualen Netzes leicht angepasst werden, um T-Komponenten in einem wohlgeformten Netz zu ermitteln[6]. Gibt es keine S-Komponente, so scheitert das Verfahren von Kemper (1993), was zum Abbruch von Verfahren 5.23 führt.

Um sicherzustellen, dass auch wirklich nur eine Transition der Menge T_X durch eine minimale T-Invariante überdeckt wird, wird ein Netz konstruiert, das das bestehende Netz um diese eine Transition aus T_X und um alle weiteren Knoten erweitert, die notwendig sind, um die minimale T-Invariante zu berechnen. Dabei wird nur genau eine Transition aus T_X zum Netz hinzugefügt. Das hieraus resultierende Netz ist das wie folgt definierte T-Invariantenkonstruktionsnetz.

Definition 5.25 (T-Invariantenkonstruktionsnetz)
Gegeben ein zusammenhängendes Free-Choice-Netz $\mathcal{N}$ mit positiver S-Invariante und eine Überdeckung von $\mathcal{N}$ durch minimale T-Invarianten $\textsc{Tinv}$. Sei weiterhin $\vec{\iota_\Sigma} = \sum_{\vec{\iota} \in \textsc{Tinv}} a_{\vec{\iota}} \cdot \vec{\iota}$ mit $a_{\vec{\iota}} \in \{0, 1\}$ für alle $\vec{\iota} \in \textsc{Tinv}$, so dass $\mathcal{N}_{\vec{\iota_\Sigma}}$ ein zusammenhängendes wohlgeformtes Free-Choice-Netz ist.
Zu einer minimalen T-Invariante $\vec{\iota_a} \in \textsc{Tinv}$ mit $a_{\vec{\iota_a}} = 0$ sei $T_A \stackrel{\text{def}}{=} \|\vec{\iota_a}\| \setminus \|\vec{\iota_\Sigma}\|$ mit $T_A \neq \emptyset$ und $\vec{\iota_*} = \vec{\iota_\Sigma} + \vec{\iota_a}$ sowie $T_X = \{t \in T_A \mid {}^\bullet t \cap P_{\vec{\iota_\Sigma}} \neq \emptyset\}$.
Dann ist für ein $t_A \in T_X$ das *T-Invariantenkonstruktionsnetz* zu $\mathcal{N}_{\vec{\iota_\Sigma}}$ und $\vec{\iota_a}$ gegeben durch das Netz $\mathcal{N}^T(\vec{\iota_\Sigma}, t_A, \vec{\iota_A}) \stackrel{\text{def}}{=} (P^T, T^T, F^T)$ mit

$$\begin{array}{lcll} P^T & \stackrel{\text{def}}{=} & P_{\vec{\iota_\Sigma}} & \cup \quad (E \cap P) \\ T^T & \stackrel{\text{def}}{=} & T_{\vec{\iota_\Sigma}} & \cup \quad (E \cap T) \\ F^T & \stackrel{\text{def}}{=} & F_{\vec{\iota_*}} & \cap \quad (P^T \times T^T \cup T^T \times P^T) \end{array}$$

Hierbei ist die Menge E definiert als

$$E \stackrel{\text{def}}{=} \{n \in P_{\vec{\iota_*}} \cup T_{\vec{\iota_*}} \mid \exists \pi \in E_d^{\mathcal{N}}(t_A, n) \wedge \textsc{Alph}(\pi) \cap P_{\vec{\iota_\Sigma}} = \emptyset\}.$$

⧫

[6] Der Algorithmus in Kemper (1993) findet minimale Siphons in einem Free-Choice-Netz. Diese stellen jedoch in einem wohlgeformten Free-Choice-Netz S-Komponenten dar (vgl. Satz 3.61), so dass lediglich überprüft werden muss, ob das Siphon eine S-Komponente ist. Ist dies nicht der Fall, ist das Netz ohnehin fehlerhaft.

Ausgehend von t_A werden also alle Knoten in $\mathcal{N}^T(\vec{\iota_\Sigma}, t_A, \vec{\iota_A})$ aufgenommen, die von t_A auf einem Pfad ohne Knoten aus $P_{\vec{\iota_\Sigma}}$ erreichbar sind. Dies entsprich stets einem Teilnetz von $\mathcal{N}_{\vec{\iota_a}}$. Für dieses T-Invariantenkonstruktionsnetz gelten nun die folgenden Aussagen.

Satz 5.26 (Eigenschaften des T-Invariantenkonstruktionsnetzes)
Gegeben ein zusammenhängendes Free-Choice-Netz $\mathcal{N}$ mit positiver S-Invariante und eine Überdeckung von $\mathcal{N}$ durch minimale T-Invarianten $\textsc{Inv}$. Sei weiterhin $\vec{\iota_\Sigma} = \sum_{\vec{\tau} \in \textsc{Inv}} a_{\vec{\tau}} \cdot \vec{\tau}$ mit $a_{\vec{\tau}} \in \{0, 1\}$ für alle $\vec{\tau} \in \textsc{Inv}$, so dass $\mathcal{N}_{\vec{\iota_\Sigma}}$ ein zusammenhängendes wohlgeformtes Free-Choice-Netz ist.
Zu einer minimalen T-Invariante $\vec{\iota_a} \in \textsc{Inv}$ mit $a_{\vec{\iota_a}} = 0$ und $T_A \stackrel{\text{def}}{=} \|\vec{\iota_a}\| \setminus \|\vec{\iota_\Sigma}\|$ mit $T_A \neq \emptyset$ sei $\vec{\iota_*} = \vec{\iota_\Sigma} + \vec{\iota_a}$ und $T_X = \{t \in T_A \mid {}^\bullet t \cap P_{\vec{\iota_\Sigma}} \neq \emptyset\}$. Zudem sei für ein $t_A \in T_X$ das T-Invariantenkonstruktionsnetz zu $\mathcal{N}_{\vec{\iota_\Sigma}}$ und $\vec{\iota_a}$ gegeben durch das Netz $\mathcal{N}^T(\vec{\iota_\Sigma}, t_A, \vec{\iota_A}) \stackrel{\text{def}}{=} (P^T, T^T, F^T)$. Ist $\mathcal{N}_{\vec{\iota_*}}$ wohlgeformt, so gilt:

1. Jede Transition $t \in T^T$ hat in $\mathcal{N}^T(\vec{\iota_\Sigma}, t_A, \vec{\iota_A})$ und in $\mathcal{N}$ den gleichen Vor- und Nachbereich.
2. Jede minimale T-Invariante von $\mathcal{N}^T(\vec{\iota_\Sigma}, t_A, \vec{\iota_A})$ ist eine minimale T-Invariante von $\mathcal{N}_{\vec{\iota_*}}$ (jede nicht im ersten Netz existierende Transition erhält den Eintrag 0).
3. $\mathcal{N}^T(\vec{\iota_\Sigma}, t_A, \vec{\iota_A})$ besitzt eine minimale T-Invariante $\vec{\iota}$, für die $\|\vec{\iota}\| \cap T_X = \{t_A\}$ ist.
4. $\mathcal{N}^T(\vec{\iota_\Sigma}, t_A, \vec{\iota_A})$ ist ein wohlgeformtes Free-Choice-Netz.

Beweis:
Sei $P^A = P^T \setminus P_{\vec{\iota_\Sigma}}$ die Menge der hinzugefügten Stellen. Dann ist $P^A \subseteq (P_{\vec{\iota_*}} \setminus P_{\vec{\iota_\Sigma}})$. Somit ist t_A die einzige Transition aus T_X, die dem Netz hinzugefügt wird, da alle anderen Transitionen aus T_X Stellen aus $P_{\vec{\iota_\Sigma}}$ im Vorbereich besitzen. Da für alle hinzugefügten Stellen $|p^\bullet| = 1$ gilt ($\mathcal{N}_{\vec{\iota_a}}$ ist eine T-Komponente), ist $\mathcal{N}^\Sigma$ ein zusammenhängendes Free-Choice-Netz und das Subnetz aus den hinzugefügten Transitionen und den hinzugefügten Stellen ein markierter Graph.
(1) Gäbe es ein $t_b \in T^T$, für das die Aussage nicht gilt, so gäbe es ein $p \in {}^\bullet_{(\mathcal{N})} t_b \cap (P \setminus P^T)$ und somit ein $p \in {}^\bullet t_b \cap (P_{\vec{\iota_*}} \setminus P^T)$, da jede Transition aus $T_{\vec{\iota_*}}$ in $\mathcal{N}_{\vec{\iota_*}}$ und in $\mathcal{N}$ den gleichen Vor- und Nachbereich aufweist. Da das Netz $\mathcal{N}_{\vec{\iota_*}}$ stark zusammenhängend ist, gibt es von einer Stelle $p' \in P_{\vec{\iota_*}}$ einen Pfad nach p. Dann muss es jedoch auch eine Transition $t' \in T_X$ geben, die auf diesem Pfad liegt. Da diese verschieden von t_A ist, kann die nach Lemma 5.22 notwendige minimale T-Invariante, für die $\|\vec{\iota}\| \cap T_X = \{t_A\}$ gilt, nicht in $\mathcal{N}_{\vec{\iota_*}}$ existieren, da im Widerspruch zu Satz 3.52 $\sum_{t \in {}^\bullet p}(\vec{\iota}(t)) \neq \sum_{t \in p^\bullet}(\vec{\iota}(t))$ wäre.
(2) Sei $\vec{\iota}$ eine minimale T-Invariante von $\mathcal{N}^T(\vec{\iota_\Sigma}, t_A, \vec{\iota_A})$. Für jede hinzugefügte

Transition t ist nach (1) auch ${}^\bullet t$ und $t^\bullet$ Teil des konstruierten Netzes. Wäre $\vec{\iota}$ keine minimale T-Invariante von $\mathcal{N}_{\vec{\iota_*}}$, so wäre $\sum_{t\in {}^\bullet p}(\vec{\iota}(t)) \neq \sum_{t\in p^\bullet}(\vec{\iota}(t))$ für ein $p \in P_{\vec{\iota_*}}$. Da jedoch sämtliche in $\mathcal{N}_{\vec{\iota_*}}$ hinzukommenden Kanten zu Transitionen führen und von Transitionen t' ausgehen, für die $\vec{\iota}(t') = 0$ ist, gilt die Aussage nach wie vor. Somit ist $\vec{\iota}$ auch eine minimale T-Invariante von $\mathcal{N}_{\vec{\iota_*}}$.
(3) Die Existenz einer solchen Invariante folgt für $\mathcal{N}_{\vec{\iota_*}}$ aus Lemma 5.22. Sie ist hier für $\mathcal{N}^T(\vec{\iota_\Sigma}, t_A, \vec{\iota_A})$ zu zeigen. Der Beweis erfolgt analog zum Beweis von Satz 5.19 per Induktion über die Kardinalität der Menge der Stellen, die in $\mathcal{N}_{\vec{\iota_a}}$ nicht aber in $\mathcal{N}_{\vec{\iota_\Sigma}}$ sind, also über $P_A^* = P_{\vec{\iota_a}} \setminus P_{\vec{\iota_\Sigma}}$, wobei wiederum die Regeln b), c), e) aus Abbildung 5.7 zur Reduktion des markierten Graphen der hinzugefügten Knoten verwendet werden. Ist $|P_A^*| = 0$, so besitzt $\mathcal{N}_{\vec{\iota_*}}$ nur zusätzliche Transitionen. Sei $\vec{\iota}$ die minimale T-Invariante in $\mathcal{N}_{\vec{\iota_*}}$ mit $\|\vec{\iota}\| \cap T_X = \{t_A\}$, die es nach Lemma 5.22 gibt. Wäre $\vec{\iota}$ keine minimale T-Invariante von $\mathcal{N}^\Sigma$, so wäre $\sum_{t\in {}^\bullet_{(\mathcal{N}^\Sigma)} p}(\vec{\iota}(t)) \neq \sum_{t\in p^\bullet_{(\mathcal{N}^\Sigma)}}(\vec{\iota}(t))$ für eine Stelle $p \in P^T$. Da jedoch alle Transitionen, die nicht in T^T sind dann in T_X sind, ist dies nicht möglich. Somit gibt es in diesem Fall die gesuchte T-Invariante $\vec{\iota}$.

Sei die Aussage also für $|P_A^*| = n$ bewiesen und sei nun $|P_A^*| = n + 1$. Dann tritt wiederum eine der folgenden Fälle auf, wie dies bereits im Beweis zu Satz 5.19 gezeigt wurde:
(1) Es gibt eine Stelle $p' \in P_A^*$ mit ${}^\bullet p = {}^\bullet p'$ und $p^\bullet = p'^\bullet$.
(2) Es ist ${}^\bullet p = p^\bullet$.
(3) Es kann Regel b) aus Abbildung 5.7 angewandt werden.
Wie sich anhand der Gleichung $\sum_{t\in {}^\bullet_{(\mathcal{N}^\Sigma)} p}(\vec{\iota}(t)) = \sum_{t\in p^\bullet_{(\mathcal{N}^\Sigma)}}(\vec{\iota}(t))$ aus Satz 3.52 überprüfen lässt, ändern sich die T-Invarianten für die beteiligten Transitionen in den ersten beiden Fällen nicht. Im Fall (3) existiert weiterhin eine T-Invariante, in der die entfernte Transition dann den Wert 0 hat. Dies wird in Unterabschnitt 5.3.3 noch einmal detaillierter dargestellt. Somit lässt sich ein $\vec{\iota_a}$ konstruieren, für das $|P_A^*| = n$ ist, und es gilt die Aussage nach der Induktionsannahme.
(4) $\mathcal{N}^T(\vec{\iota_\Sigma}, t_A, \vec{\iota_A})$ ist ein zusammenhängendes Free-Choice-Netz mit positiver T-Invariante (wegen 3) und positiver S-Invariante ($\sum_{p\in {}^\bullet t}(\vec{\kappa}(p)) = \sum_{p\in t^\bullet}(\vec{\kappa}(p))$ bleibt erhalten). Somit ist das Netz wohlgeformt, wenn die Zahl der zusätzlichen Cluster in $\mathcal{N}^T(\vec{\iota_\Sigma}, t_A, \vec{\iota_A})$ gleich der Zahl der hinzugefügten Transitionen weniger der Zahl des Dimensionszuwachs des T-Invariantenvektorraums ist. Da $|T_X| = 1$ ist und die Dimension um 1 zunimmt, gilt die Aussage. □

Dies erlaubt es, die in Schritt 3.2 von Verfahren 5.23 geforderte minimale T-Invariante zu berechnen, indem dazu zunächst das T-Invariantenkonstruktionsnetz $\mathcal{N}^T(\vec{\iota_\Sigma}, t_A, \vec{\iota_A})$ zu $\mathcal{N}_{\vec{\iota_\Sigma}}$, $\vec{\iota_a}$ und t_A gebildet wird. In diesem kann dann mit dem Verfahren von Kemper (1993) eine T-Komponente zu einer Transition ermittelt werden. Die charakteristische Funktion dieser T-Komponente entspricht dann einer

minimalen T-Invariante.

Bei der Bildung des T-Invariantenkonstruktionsnetzes kann es vorkommen, dass die erste Eigenschaft aus Satz 5.26 verletzt wird, wenn das Netz nicht wohlgeformt ist. In diesem Fall ist das gesamte Netz folglich nicht wohlgeformt. Dies kann dann entsprechen ausgegeben werden. Somit ergibt sich folgendes Verfahren zur Berechnung der minimalen T-Invariante in Schritt 3.2 in Verfahren 5.23.

Verfahren 5.27 (Invariantenberechnung für Schritt 3.2 in Verfahren 5.23)
Sei $t \in T_X$ die in Schritt 3.2 von Verfahren 5.23 gewählte Transition, zu der eine minimale T-Invariante berechnet werden soll, und sei $\vec{\iota_\Sigma}$ die Summe der gefärbten minimalen T-Invarianten aus Tinv.

(1) Erzeuge das T-Invariantenkonstruktionsnetz $\mathcal{N}^T(\vec{\iota_\Sigma}, t, \vec{\iota})$ zu $\mathcal{N}_{\vec{\iota_\Sigma}}$, $\vec{\iota}$ und t.

(2) Berechne mit dem Verfahren von Kemper (1993) ein minimales Siphon des dualen Netzes $\mathcal{N}^T(\vec{\iota_\Sigma}, t, \vec{\iota})^d$. Ist dies keine T-Komponente von $\mathcal{N}^T(\vec{\iota_\Sigma}, t, \vec{\iota})$, so breche ab, da das Netz nicht wohlgeformt sein kann.

(3) Gib die charakteristische Funktion dieser T-Komponente als gesuchte minimale T-Invariante aus.

■

Dies schließt die Überprüfung der Wohlgeformtheit eines gegebenen Free-Choice-Netzes durch Verfahren 5.23 ab. Im nächsten Unterabschnitt werden die Ergebnisse auf Workflownetze angewandt.

5.3.2 Anwendung der Ergebnisse auf Free-Choice-Workflownetze

Nachdem der letzte Unterabschnitt die Grundlagen zur Analyse von Free-Choice-Workflownetzen gegeben hat, werden die Ergebnisse in diesem Unterabschnitt auf Workflownetze angewandt. Dies dient sowohl dazu, eine alternative Korrektheitscharakterisierung zu liefern als auch dazu, die gleichzeitige markierbarkeit von Stellen zu ermitteln.

Alternative Korrektheitscharakterisierung

Die Wohlgeformtheit des Abschlusses eines Free-Choice-Workflownetzes ist eine notwendige, jedoch keine hinreichende Bedingung an die Korrektheit des Netzes. Vielmehr muss zudem gewährleistet sein, dass die Stelle i sämtliche Siphons überdeckt. Dies kann auf zwei mögliche Herangehensweisen festgestellt werden. Entweder kann nach Satz 3.59 für sämtliche minimalen S-Invarianten einer Basis der

S-Invarianten überprüft werden, ob i im Träger jeder minimalen S-Invariante ist, oder es kann die Stelle i aus dem Netz entfernt werden und das maximale Siphon berechnet werden. Ist dieses Siphon leer, so überdeckt i jedes Siphon, was mit Satz 3.61 und Satz 3.59 ebenfalls ein hinreichendes Kriterium ist Ein entsprechender Algorithmus findet sich bei Desel und Esparza (1995) (S.87) und wird in Algorithmus 5.1 auf Seite 218 dieser Arbeit detaillierter erläutert, wobei beachtet werden muss, dass dieser Algorithmus Traps ermittelt und daher das inverse Netz verwendet werden muss.

Aufgrund der Tatsache, dass das duale Netz eines Free-Choice-Workflownetzes nach Satz 3.62 ebenfalls ein wohlgeformtes Free-Choice-Worflownetz ist, lässt sich das Verfahren 5.23 auf das duale Netz anwenden, so dass die erzeugte Basis minimaler T-Invarianten einer Basis minimaler S-Invarianten des Ursprungsnetzes entspricht. Diese kann dann nach Satz 3.59 dazu verwendet werden, die Lebendigkeit des Netzes zu überprüfen, da diese S-Invarianten ja genau den S-Komponenten entsprechen. Da es sich zudem um eine Basis handelt, sind sämtliche anderen S-Komponenten die Subnetze einer Linearkombination dieser minimalen S-Invarianten, so dass die Betrachtung dieser S-Komponenten ausreicht. Somit ergibt sich folgendes Verfahren, um festzustellen, ob ein Free-Choice-Workflownetz korrekt ist.

Verfahren 5.28 (Wohlgeformtheit eines T-überdeckbaren FC-Netzes)
Gegeben ein zusammenhängendes Free-Choice-Workflownetz $\mathcal{N}$ mit Abschluss $\overline{\mathcal{N}}$.

(1) Wende das Verfahren 5.23 auf das duale Netz $\overline{\mathcal{N}}^d$ an.

(2) Ist $\overline{\mathcal{N}}^d$ nicht wohlgeformt, dann breche ab.

(3) Überprüfe, ob $i^d \in \|\vec{\iota}\|$ für alle $\vec{\iota} \in \textsc{Inv}^d$ ist, wobei $\textsc{Inv}^d$ die Basis minimaler T-Invarianten des dualen Netzes $\overline{\mathcal{N}}^d$ ist.

(4) War dies erfolgreich, so ist $\mathcal{N}$ korrekt.

■

Durch die Betrachtung des dualen Netzes erlaubt Verfahren 5.28 das gleichzeitige Erzeugen einer Basis minimaler S-Invarianten. Somit lassen sich die Bedingung der Wohlgeformtheit und der Markierung sämtlicher S-Komponenten in einem Schritt überprüfen. Die Korrektheit des Verfahrens folgt aus den Ergebnissen des letzten Unterabschnitts. In Schritt 1 wird ermittelt, ob $\overline{\mathcal{N}}^d$ wohlgeformt ist, was nach Satz 3.62 impliziert, dass $\overline{\mathcal{N}}$ wohlgeformt ist. Gleichzeitig existiert mit $\textsc{Inv}$ eine Basis minimaler T-Invarianten von $\overline{\mathcal{N}}^d$. Dies ist gleichzeitig eine Basis minimaler S-Invarianten von $\overline{\mathcal{N}}$ (vgl. Proposition 6.20 in Desel und Esparza (1995)). Da i im Träger jeder dieser minimalen S-Invarianten ist, ist i im Träger aller minimalen

S-Invarianten, da sich keine Linearkombination aus den Vektoren aus TINV erstellen lässt, deren Wert für i gleich 0 ist, ohne dass dieser Vektor negative Einträge besitzt. Da jede S-Komponente nach Satz 5.14 eine minimale S-Invariante ist, folgt mit Satz 3.59 die Aussage. Dies wird in folgendem Satz festgehalten.

Definition und Satz 5.29 (Korrektheit von Verfahren 5.28)
Verfahren 5.28 ist korrekt, wenn die Ausgabe der Korrektheit für ein zusammenhängendes Free-Choice-Workflownetz $\mathcal{N}$ bedeutet, dass $\mathcal{N}$ korrekt ist.
Verfahren 5.28 ist vollständig, wenn für ein zusammenhängendes korrektes Free-Choice-Workflownetz $\mathcal{N}$ die Ausgabe der Korrektheit erfolgt.
Verfahren 5.28 terminiert, ist korrekt und vollständig. ⧫

Beweis:
Verfahren 5.23 terminiert und ist korrekt und vollständig nach Satz 5.24. Hieraus folgt direkt die Termination. Gleichzeitig ist garantiert, dass in Schritt 3 von Verfahren 5.28 $\overline{\mathcal{N}}$ ein wohlgeformtes zusammenhängendes Free-Choice-Workflownetz ist. Somit bleibt nach Satz 3.59 zu zeigen, dass alle S-Komponenten von $\overline{\mathcal{N}}$ die Stelle i überdecken. Dies ist gleichbedeutend mit der Überprüfung, ob $i^d \in \|\vec{\iota}\|$ für alle $\vec{\iota} \in \text{TINV}^d$ ist, wobei TINV^d die Basis minimaler T-Invarianten des dualen Netzes $\overline{\mathcal{N}}^d$ ist, die in Schritt 3 erfolgt, da die S-Komponenten den Subnetzen der minimalen T-Invarianten des dualen Netzes entsprechen. Da es sich bei TINV^d zudem um eine Basis handelt, sind sämtliche minimalen T-Invarianten von $\overline{\mathcal{N}}^d$ eine Linearkombination der minimalen T-Invarianten aus TINV^d. Somit überdecken alle diese minimalen T-Invariantne i^d und im Ursprungsnetz $\overline{\mathcal{N}}$ überdecken alle S-Komponenten die Stelle i. Dies zeigt die Vollständigkeit und die Korrektheit. □

Die bei diesem Verfahren berechneten minimalen S-Invarianten werden im nächsten Unterabschnitt genutzt, um die Erreichbarkeit von Teilmarkierungen zu ermitteln. Sie sind somit auch für die weitere Analyse notwendig.

Nebenläufigkeit

Nachdem im letzten Unterabschnitt die Korrektheit und die Ermittlung einer Basis minimaler S-Invarianten beziehungsweise T-Invarianten für ein Free-Choice-Workflownetz betrachtet wurde, widmet sich dieser Unterabschnitt der Frage, wann eine Menge von Stellen in einem Free-Choice-Workflownetz gleichzeitig markierbar ist. Hierbei wird von der Basis minimaler S-Invarianten des letzten Unterabschnitts Gebrauch gemacht.

Aus Satz 3.60 ist bekannt, dass eine Markierung $\mathbf{m}$ eines korrekten Free-Choice-Workflownetzes genau dann erreichbar ist, wenn $\mathbf{m}$ und $[i]$ auf allen minimalen S-Invarianten übereinstimmen und $\mathbf{m}$ jede ordentliche Trap von $\overline{\mathcal{N}}$ markiert. Ersteres lässt sich mittels Satz 3.51 ermitteln, letzteres lässt sich überprüfen, indem

sämtliche in $\mathbf{m}$ markierten Stellen aus dem Netz entfernt werden und überprüft wird, ob das verbleibende Netz keine ordentliche Falle besitzt.

Dieser Ansatz funktioniert jedoch nur für Markierungen, die mit $[i]$ auf allen S-Komponenten übereinstimmen. Um zu entscheiden, ob zwei Transitionen in einer erreichbaren Markierung nebenläufig aktiviert (vgl. Def. 3.32) sind, ist es jedoch notwendig zu entscheiden, ob es zu einer Stellenmenge eine Markierung gibt, die all diese Stellen markiert und die erreichbar ist. Dies ist mit obigem Verfahren ebenfalls möglich, wie sich im Folgenden zeigen wird. Dazu muss das Netz jedoch auf diejenigen S-Komponenten reduziert werden, die in jedem Fall markiert sein müssen.

Um zu zeigen, dass lediglich die relevanten S-Komponenten betrachtet werden müssen, wird im Folgenden zunächst von der Eigenschaft Gebrauch gemacht, dass jede Kombination von S-Komponenten des Abschlusses eines korrekten Free-Choice-Workflownetzes ebenfalls ein lebendiges und sicheres Free-Choice-Netz darstellt und ebenfalls als Abschluss eines Free-Choice-Workflownetzes interpretiert werden kann. Dies zeigt folgender Satz, der sich auf die Ergebnisse aus Satz 5.20 bezieht.

Satz 5.30 (Lebendigkeit der S-Komponenten eines FCWF-Netzes)

Gegeben ein korrektes FCWF-Netz $\mathcal{N}$ mit Abschluss $\overline{\mathcal{N}}$ und einer S-Invariantenbasis $\mathbf{B}_S(\overline{\mathcal{N}})$. Sei $\vec{\kappa_*} = \sum_{\vec{\kappa} \in \mathbf{B}_S(\overline{\mathcal{N}})} a_{\vec{\kappa}} \cdot \vec{\kappa}$ mit $\vec{\kappa_*} \neq \vec{\mathbf{0}}$ und $a_{\vec{\kappa}} \in \{0, 1\}$ für alle $\vec{\kappa} \in \mathbf{B}_S(\overline{\mathcal{N}})$.

Dann ist $\mathcal{N}_{\vec{\kappa_*}}$ der Abschluss eines Free-Choice-Workflownetzes und $(\mathcal{N}_{\vec{\kappa_*}}, [i])$ lebendig und sicher.

Beweis:
Da $\vec{\kappa_*} > \vec{\mathbf{0}}$ und nach Satz 3.59 jedes Subnetz einer S-Invariante aus $\mathbf{B}_S(\overline{\mathcal{N}})$ die Stelle i überdeckt, ist $\mathcal{N}_{\vec{\kappa_*}}$ zusammenhängend. Nach Satz 5.20 ist $\mathcal{N}_{\vec{\kappa_*}}$ ein wohlgeformtes Free-Choice-Netz. Da die Stellen i und o erhalten bleiben, ist $\mathcal{N}_{\vec{\kappa_*}}$ der Abschluss eines Workflownetzes. Da alle S-Invarianten $\vec{\kappa} \in \mathbf{B}_S(\overline{\mathcal{N}})$ die Stelle i im Träger haben, bleibt zu zeigen, dass dann alle S-Komponenten i überdecken. Wäre dies nicht der Fall, so müsste in $\mathcal{N}_{\vec{\kappa_*}}$ eine S-Komponenten existieren, die nicht in $\overline{\mathcal{N}}$ existiert, da sie nicht i überdeckt. Dann gäbe es auch eine entsprechende minimale S-Invariante. Da jedoch nach Satz 3.52 für jede minimale S-Invariante $\sum_{p \in {}^\bullet t}(\vec{\kappa}(p)) = \sum_{p \in t^\bullet}(\vec{\kappa}(p))$ gilt, ist dies nicht möglich, da für jede Stelle auch ihr Vor- und Nachbereich gleich dem in $\overline{\mathcal{N}}$ ist. Damit gilt die Aussage nach Satz 3.59. □

Ebenso aufgrund der Dualitätseigenschaft folgt die folgende Aussage, die die Randstellen minimaler S-Invarianten betrachtet.

Lemma 5.31 (Minimale S-Invarianten von Randstellen)
Gegeben ein wohlgeformtes Free-Choice-Netz $\mathcal{N}$ und eine Überdeckung von $\mathcal{N}$ durch minimale S-Invarianten SINV. Sei $\overrightarrow{\kappa_\Sigma} = \sum_{\vec{\kappa} \in \text{SINV}} a_{\vec{\kappa}} \cdot \vec{\kappa}$ mit $a_{\vec{\kappa}} \in \{0, 1\}$ für alle $\vec{\kappa} \in$ SINV, so dass $\mathcal{N}_{\overrightarrow{\kappa_\Sigma}}$ zusammenhängend ist.
Zu einer minimalen S-Invariante $\overrightarrow{\kappa_a} \in$ SINV mit $a_{\overrightarrow{\kappa_a}} = 0$, $P_A \stackrel{\text{def}}{=} \|\overrightarrow{\kappa_a}\| \setminus \|\overrightarrow{\kappa_\Sigma}\|$ und $P_A \neq \emptyset$ sei $\overrightarrow{\kappa_*} = \overrightarrow{\kappa_\Sigma} + \overrightarrow{\kappa_a}$. Dann gilt für $P_X \stackrel{\text{def}}{=} \{p \in P_A \mid p^\bullet \cap T_{\overrightarrow{\kappa_\Sigma}} \neq \emptyset\}$, dass $p \in P_X$ impliziert, dass es in $\mathcal{N}_{\overrightarrow{\kappa_*}}$ eine minimale S-Invariante $\overrightarrow{\kappa_p}$ gibt mit $\|\overrightarrow{\kappa_p}\| \cap P_X = \{p\}$.

Beweis:
Dies folgt aus Satz 3.62 und Lemma 5.22. □

Diese Aussage lässt sich dazu verwenden, eine Ordnung auf den minimalen S-Invarianten zu zeigen, wie sie bereits bei den T-Invarianten in Verfahren 5.23 genutzt wurde.

Lemma 5.32 (Ordnung minimaler S-Invarianten)
Gegeben ein wohlgeformtes Free-Choice-Netz $\mathcal{N}$ und eine Basis minimaler S-Invarianten $\mathbf{B}_S(\mathcal{N})$. Sei $\overrightarrow{\kappa_\Sigma} = \sum_{\vec{\kappa} \in \mathbf{B}_S(\mathcal{N})} a_{\vec{\kappa}} \cdot \vec{\kappa}$ mit $a_{\vec{\kappa}} \in \{0, 1\}$ für alle $\vec{\kappa} \in \mathbf{B}_S(\mathcal{N})$, so dass $\mathcal{N}_{\overrightarrow{\kappa_\Sigma}}$ zusammenhängend ist.
Es lassen sich nun S-Invarianten $\overrightarrow{\kappa_1}, \ldots, \overrightarrow{\kappa_n}$ von $\mathcal{N}$ finden, so dass für $\overrightarrow{\kappa_{\Sigma,j}} = \overrightarrow{\kappa_\Sigma} + \sum_{j=1}^{n} \overrightarrow{\kappa_j}$ gilt

$$|\mathbf{B}_S(\mathcal{N}_{\overrightarrow{\kappa_{\Sigma,\,j+1}}})| - |\mathbf{B}_S(\mathcal{N}_{\overrightarrow{\kappa_{\Sigma,\,j}}})| = 1$$

Beweis:
Dies folgt aus Lemma 5.31, aus der Dualitätseigenschaft von Satz 3.62 und aus Korollar 5.18, wobei durch die Dualitätseigenschaft die S- und die T-Invarianten vertauscht werden können. □

In dem oben in Satz 5.30 gegebenen Netzsystem $(\mathcal{N}_{\overrightarrow{\kappa_*}}, [i])$ ist – wie in jedem Workflownetz mit der Free-Choice-Eigenschaft – eine Markierung erreichbar, wenn sie auf allen S-Komponenten mit $[i]$ übereinstimmt und wenn sie jede Falle markiert. Um festzustellen, ob eine Markierung erreichbar ist, die eine gegebene Stellenmenge von $\overline{\mathcal{N}}$ markiert, ist dies folglich eine notwendige Voraussetzung. Im Folgenden soll gezeigt werden, dass diese Bedingung auch hinreichend ist. Hierzu muss gezeigt werden, dass es dann Stellen gibt, die, wenn sie zusätzlich markiert sind, zu einer Markierung führen, die einerseits alle S-Komponenten markiert und andererseits alle Fallen markiert.

Satz 5.33 (Markierbarkeit in korrekten FCWF-Netzen)
Gegeben ein korrektes FCWF-Netz $\mathcal{N}$ mit Abschluss $\overline{\mathcal{N}}$ und einer S-Invariantenbasis $\mathbf{B}_S(\overline{\mathcal{N}})$ sowie eine Menge von Stellen $P_\| \subseteq P$. Sei $\text{SINV}_\| \subseteq \mathbf{B}_S(\overline{\mathcal{N}})$ gegeben

durch $\textsc{Sinv}_{\|} \stackrel{\text{def}}{=} \{\vec{\kappa} \in \mathbf{B}_S(\overline{\mathcal{N}}) \mid (\|\vec{\kappa}\| \cap P_{\|}) \neq \emptyset\}$ und sei $\vec{\kappa_*} = \sum_{\vec{\kappa} \in \textsc{Sinv}_{\|}} \vec{\kappa}$.
Dann existiert eine Markierung $\mathbf{m}_1 \in [[i]\rangle_{\overline{\mathcal{N}}}$ mit $P_{\|} \subseteq \|\mathbf{m}_1\|$ gdw. eine Markierung $\mathbf{m}_2 \in [[i]\rangle_{\mathcal{N}_{\vec{\kappa_*}}}$ existiert, so dass $P_{\|} = \|\mathbf{m}_2\|$ ist.

Beweis:
($\Rightarrow$) Es soll gezeigt werden, dass jede Schaltfolge σ in $\overline{\mathcal{N}}$ reduziert auf die Transitionen von $T_{\vec{\kappa_*}}$ eine Schaltfolge in $\mathcal{N}_{\vec{\kappa_*}}$ ist, da sich dann die Aussage ergibt. Sei n die Länge der Schaltfolge σ und sei $\sigma|_{T_{\vec{\kappa_*}}}$ deren Reduktion auf die Transitionen aus $T_{\vec{\kappa_*}}$. Ist $n = 0$, so ist die Aussage trivial. Sei die Aussage also für eine Schaltfolge der Länge $n = m$ bewiesen und sei nun $n = m + 1$. Dann ist $\sigma = \sigma' t$, und es ist $[i] \xrightarrow[\overline{\mathcal{N}}]{\sigma'} \mathbf{m}'$ und somit nach Induktionsannahme $[i] \xrightarrow[\mathcal{N}_{\vec{\kappa_*}}]{\sigma'|_{T_{\vec{\kappa_*}}}} \mathbf{m}'|_{P_{\vec{\kappa_*}}}$. Ist $t \notin T_{\vec{\kappa_*}}$, so wäre keine Stelle aus dem Vor- oder Nachbereich von t in $\mathcal{N}_{\vec{\kappa_*}}$ und somit würde die Transition keinerlei Veränderung in $\mathcal{N}_{\vec{\kappa_*}}$ bewirken, und es folgt die Aussage.

Wäre $t \in T_{\vec{\kappa_*}}$ und in $\mathcal{N}_{\vec{\kappa_*}}$ nicht aktiviert, so gäbe es eine Stelle im Vorbereich von t, die in $\mathbf{m}'|_{T_{\vec{\kappa_*}}}$ nicht markiert wäre. Folglich gäbe es eine Transition in $\textsc{Alph}(\sigma')$, die diese Stelle in $\overline{\mathcal{N}}$, nicht aber in $\mathcal{N}_{\vec{\kappa_*}}$ im Nachbereich besitzt. Da jedoch zu jeder Stelle der Vor- und Nachbereich ebenfalls in $\mathcal{N}_{\vec{\kappa_*}}$ ist, wäre dann auch die Stelle nicht in $P_{\vec{\kappa_*}}$ und folglich nicht im Vorbereich von t. Somit gilt die Aussage.
($\Leftarrow$) Nach Satz 5.30 ist $\mathcal{N}_{\vec{\kappa_*}}$ ein wohlgeformtes Free-Choice-Netz und besitzt eine Basis minimaler S-Invarianten $\mathbf{B}_S(\mathcal{N}_{\vec{\kappa_*}})$. Nach Lemma 5.32 lässt sich diese Basis durch minimale S-Invarianten von $\overline{\mathcal{N}}$ dergestalt ergänzen, dass $\mathbf{B}_S(\mathcal{N}_{\vec{\kappa_*}}) \cup \{\vec{\kappa_1}, \ldots, \vec{\kappa_n}\}$ eine Basis minimaler S-Invarianten von $\overline{\mathcal{N}}$ ist, und für $\vec{\kappa_1}, \ldots, \vec{\kappa_n}$ mit $\vec{\kappa_{*,j}} = \vec{\kappa_*} + \sum_{j=1}^{n} \vec{\kappa_j}$ gilt $|\mathbf{B}_S(\mathcal{N}_{\vec{\kappa_{j,j}}})| - |\mathbf{B}_S(\mathcal{N}_{\vec{\kappa_{j,j-1}}})| = 1$. Da jede dieser minimalen S-Invarianten eine Stelle besitzt, die von keiner anderen minimalen S-Invariante der Basis überdeckt wird (P_X aus Lemma 5.31 enthält mindestens ein Element), lässt sich diese Stelle durch $\mathbf{m}_1$ markieren. Auf diese Weise entsteht eine Markierung, die jede minimale S-Invariante mit genau einer Marke versieht und daher auf allen minimalen S-Invarianten der Basis $\mathbf{B}_S(\mathcal{N}_{\vec{\kappa_*}}) \cup \{\vec{\kappa_1}, \ldots, \vec{\kappa_n}\}$ mit $[i]$ übereinstimmt. Dann stimmt $\mathbf{m}_1$ auch auf allen minimalen S-Invarianten mit $[i]$ überein, und es bleibt zu zeigen, dass zudem sämtliche ordentlichen Fallen von $\overline{\mathcal{N}}$ durch $\mathbf{m}_1$ markiert sind, da dann die Aussage aufgrund von Satz 3.60 gilt.

Der Beweis erfolgt über die Zahl der zur Ergänzung der Basis notwendigen S-Invarianten $\{\vec{\kappa_1}, \ldots, \vec{\kappa_n}\}$. Ist keine S-Invariante notwendig, so ist $\mathcal{N}_{\vec{\kappa_*}} = \overline{\mathcal{N}}$ und die Aussage gilt. Sie die Aussage also für m zu ergänzende S-Invarianten bewiesen und seinen nun $m + 1$ minimale S-Invarianten notwendig. Wäre eine ordentliche Falle S mit $S^\bullet \subseteq {}^\bullet S$ von $\overline{\mathcal{N}}$ in $\mathbf{m}_1$ nicht markiert, so müsste diese durch das Hinzufügen der Knoten von $\mathcal{N}_{\vec{\kappa_{m+1}}}$ entstanden sein. Da die hinzugefügten Knoten eine Zustandsmaschine mit einer Endstelle (wegen $|P_X| = 1$) darstellen, kann eine Falle nur entstehen, wenn nicht diese Stelle aus P_X markiert ist. Da diese Stelle jedoch durch obiges Vorgehen markiert wurde, gilt die Aussage. □

Es ist $\mathrm{rg}(\mathcal{N}_{\vec{\kappa}_*}) = |C_{\mathcal{N}_{\vec{\kappa}_*}}| - 1$, da $\mathcal{N}_{\vec{\kappa}_*}$ ein wohlgeformtes Free-Choice-Netz ist. Wie in Lemma 5.13 gezeigt wurde, ist dann $|P_{\vec{\kappa}_*}| - (|C_{\mathcal{N}_{\vec{\kappa}_*}}| - 1) = |\mathbf{B}_S(\mathcal{N}_{\vec{\kappa}_*})|$ und somit ist die Zahl der S-Komponenten aus $\mathbf{B}_S(\overline{\mathcal{N}})$, die eine Stelle aus $P_\|$ markieren müssen, gleich $|P_{\vec{\kappa}_*}| - |C_{\mathcal{N}_{\vec{\kappa}_*}}| + 1$. Es gilt also $|\textsc{Sinv}_\|| = |P_{\vec{\kappa}_*}| - |C_{\mathcal{N}_{\vec{\kappa}_*}}| + 1$.

Die Aussage, dass alle Transitionen in $T_\|$ nebenläufig aktivierbar sind, ist äquivalent mit der Aussage, dass für $P_\| = {}^\bullet T_\|$ dann $P_\| \subseteq \|\mathbf{m}_1\|$ ist für eine Markierung $\mathbf{m}_1 \in [[i]\rangle_{\overline{\mathcal{N}}}$. Der folgende Satz gibt zwei notwendige und hinreichende Kriterien für die Erreichbarkeit einer Markierung, die eine Menge von Stellen überdeckt.

Satz 5.34 (Erreichbare Teilmarkierungen in FCWF-Netzen)
Gegeben ein korrektes FCWF-Netz $\mathcal{N}$ mit Abschluss $\overline{\mathcal{N}}$ und einer S-Invariantenbasis $\mathbf{B}_S(\overline{\mathcal{N}})$ sowie eine Menge von Stellen $P_\| \subseteq P$. Sei $\textsc{Sinv}_\| \subseteq \mathbf{B}_S(\overline{\mathcal{N}})$ gegeben durch $\textsc{Sinv}_\| \stackrel{\text{def}}{=} \{\vec{\kappa} \in \mathbf{B}_S(\overline{\mathcal{N}}) \mid (\|\vec{\kappa}\| \cap P_\|) \neq \emptyset\}$. Sei weiterhin $\vec{\kappa}_* \stackrel{\text{def}}{=} \sum_{\vec{\kappa} \in \textsc{Sinv}_\|} \vec{\kappa}$. Dann gibt es eine Markierung $\mathbf{m} \in [[i]\rangle_{\overline{\mathcal{N}}}$ mit $P_\| \subseteq \|\mathbf{m}\|$ gdw.

- in $\mathcal{N}_{\vec{\kappa}_*}$ alle Fallen eine Stelle aus $P_\|$ überdecken und
- $[i] + [\mathcal{N}_{\vec{\kappa}_*}] \cdot X = \chi[P_\|]$ für einen T-indizierten Spaltenvektor $X \in \mathbb{Q}^{|T|}$.

Beweis:
($\Rightarrow$) Nach Satz 5.30 ist $\mathcal{N}_{\vec{\kappa}_*}$ ein wohlgeformtes Free-Choice-Netz und nach Satz 5.33 ist $\mathbf{m} = \chi[P_\|]$ in $\mathcal{N}_{\vec{\kappa}_*}$ von $[i]$ aus erreichbar. Dann gilt nach Satz 3.60 und nach Satz 3.51 die Aussage.
($\Leftarrow$) Nach Satz 3.51 und Satz 3.60 ist $\mathbf{m} = \chi[P_\|]$ in $\mathcal{N}_{\vec{\kappa}_*}$ von $[i]$ aus erreichbar. Aus Satz 5.33 folgt dann die Aussage. □

Die erste Bedingung lässt sich in Polynomialzeit berechnen, wie in Desel und Esparza (1995) dargestellt wird. Das dort vorgestellte Verfahren ist in Algorithmus 5.1 auf Seite 218 dargestellt, worauf in Abschnitt 5.4.2 noch einmal eingegangen wird. Die erste Bedingung impliziert bereits, dass jede S-Komponente mindestens mit einer Marke markiert ist, da andernfalls eine unmarkierte Falle existieren würde. Es muss jedoch überprüft werden, ob jede S-Komponente auch mit genau einer Marke markiert ist. Dies kann mit den Mitteln der linearen Algebra ebenfalls in Polynomialzeit erfolgen.

Eine weitere notwendige Bedingung ist, dass $|\textsc{Sinv}_\|| = |P_\||$ sein muss, da andernfalls offensichtlich eine minimale S-Invariante $\vec{\kappa} \in \textsc{Sinv}_\|$ existieren muss, für die $|\|\vec{\kappa}\| \cap P_\|| \geq 2$ ist. Eine Markierung, die dies erfüllt, stimmt dann offensichtlich nicht mit $[i]$ auf der minimalen S-Invariante $\vec{\kappa}$ überein und ist nach Satz 3.60 nicht erreichbar. Die minimalen S-Invarianten aus $\textsc{Sinv}_\|$ sind jedoch im Allgemeinen keine Basis, so dass eine Überprüfung auf $|\textsc{Sinv}_\|| = |P_\||$ notwendig, aber nicht hinreichend ist.

Beispiel 5.5: Das Vorgehen aus Satz 5.34 soll anhand eines Beispiels erläutert werden. Betrachtet man das Netz in Abbildung 5.6 so soll ermittelt werden, ob die Transitionen t_3 und t_8 nebenläufig schalten können. Es ist somit $T_{\|} = \{t_3, t_8\}$ und $P_{\|} = \{p_2, p_7\}$. Die S-Invariantenbasis $\mathbf{B}_S(\overline{\mathcal{N}})$ des Netzes ergibt sich als $\mathbf{B}_S(\overline{\mathcal{N}}) = \{\overrightarrow{\kappa_1}, \overrightarrow{\kappa_2}, \overrightarrow{\kappa_3}\}$ mit $\overrightarrow{\kappa_1} = [i, p_1, p_3, p_5, p_7, o]$, $\overrightarrow{\kappa_2} = [i, p_1, p_8, o]$ und $\overrightarrow{\kappa_3} = [i, p_1, p_2, p_4, p_6, o]$.

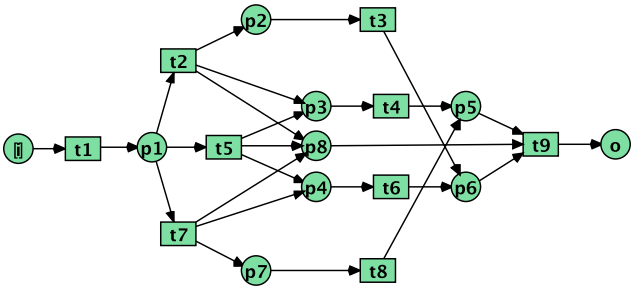


Abbildung 5.6: Nebenläufige Aktiviertheit von t_3 und t_8

Da p_2 und p_7 nicht im Träger derselben minimalen S-Invariante sind, gilt $\textsc{Sinv}_{\|} = \{\overrightarrow{\kappa_1}, \overrightarrow{\kappa_3}\}$ und somit $|\textsc{Sinv}_{\|}| = |P_{\|}| = 2$. Entfernt man die Stellen p_2 und p_7 aus dem Netz $\mathcal{N}_{\overrightarrow{\kappa_*}}$, so existiert jedoch mit $\{i, p_1, p_3, p_4, p_5, p_6, o\}$ eine ordentliche Falle. Diese Falle von $\mathcal{N}_{\overrightarrow{\kappa_*}}$ ist somit nicht durch Stellen aus $P_{\|}$ überdeckt. Somit können die Stellen aus $P_{\|}$ niemals gleichzeitig in einer Markierung markiert sein und somit auch die Transitionen aus $T_{\|} = \{t_3, t_8\}$ niemals nebenläufig schalten, obwohl $\mathbf{m} = \chi[P_{\|}]$ in $\mathcal{N}_{\overrightarrow{\kappa_*}}$ auf allen S-Invarianten mit $[i]$ übereinstimmt. (Wähle $X = [1, 1, 0, 1, 0, 0, 0, -1, 0]$, wobei die Reihenfolg der Einträge der Nummerierung der Transitionen entspricht.)

Betrachtet man hingegen die Transitionen t_3 und t_4 unter Verwendung derselben Basis $\mathbf{B}_S(\overline{\mathcal{N}})$, so ergibt sich $P_{\|} = \{p_2, p_3\}$. Wiederum gilt $\textsc{Sinv}_{\|} = \{\overrightarrow{\kappa_1}, \overrightarrow{\kappa_3}\}$ und $|\textsc{Sinv}_{\|}| = |P_{\|}| = 2$. Entfernt man die Stellen p_2 und p_3 aus dem Netz $\mathcal{N}_{\overrightarrow{\kappa_*}}$, so besitzt $\mathcal{N}_{\overrightarrow{\kappa_*}}$ keine ordentliche Falle und das erste Kriterium aus Satz 5.34 ist erfüllt. Gleichzeitig stimmt $\mathbf{m} = \chi[P_{\|}]$ in $\mathcal{N}_{\overrightarrow{\kappa_*}}$ auf allen S-Invarianten mit $[i]$ überein, da mit $X = [1, 1, 0, 0, 0, 0, 0, 0, 0]$ ein entsprechender Vektor existiert und die zweite Bedingung erfüllt ist. (Hierbei entspricht die Reihenfolg der Einträge wiederum der Nummerierung der Transitionen.) Somit sind t_3 und t_4 nebenläufig aktivierbar. $\diamond$

5.3.3 Reduktionsregeln für Netze

Die Analyse von Workflownetzen mit der Free-Choice-Eigenschaft ist in Polynomialzeit möglich. Dennoch ist es meist sinnvoll die Größe des Netzes durch Reduktion der Knotenmenge zu reduzieren. Hierzu haben sich eine Reihe von Reduktionsregeln etabliert, die hier noch einmal zusammengefasst werden sollen und die dann um eine weitere Regel ergänzt werden sollen. Zudem sollen die Auswirkungen der

Regeln nicht nur bezüglich der Beschränktheit und der Lebendigkeit, sondern auch bezüglich der Möglichkeit der nebenläufigen Aktiviertheit untersucht werden.

Um die Komplexität der Analyse von Netzen zu reduzieren, gibt es eine Reihe von Reduktionsregeln, die zentrale Eigenschaften wie die Lebendigkeit, die S- und die T-Überdeckbarkeit und die Sicherheit eines Netzes nicht verändern. Verwandt mit dem Vorgehen hier sind beispielsweise die Ergebnisse von Hack (1974), der die Statemachine Allocatable-Eigenschaft (SMA-Eigenschaft) von Netzen beschreibt. Diese wird durch eine Reduktion der Knoten nach vorheriger Auswahl einer Überdeckung durch Zustandsmaschinen (SM-Reduktion) überprüft. Führt jede solche SM-Reduktion zu einem durch S-Komponenten überdeckten Netz, so besitzt dieses Netz die SMA-Eigenschaft. Ein Free-Choice-Netz ist wohlgeformt gdw. es die SMA-Eigenschaft besitzt (a.a.O. S. 5-6).

Bekannte Reduktionsregeln

Es gibt eine Reihe von Vorschlägen zu Reduktionsregeln für Petrinetze. So diskutiert Kwong (1977) die Reduktion von Transitionssystemen im Allgemeinen, Valette (1979) beschreibt die schrittweise Verfeinerung, indem Transitionen durch sogenannte Blocks, also Teilnetze mit genau einer Start- und einer Endtransition ersetzt werden. Ein ähnliches Vorgehen beschreiben Johnsonbaugh und Murata (1981) und Suzuki und Murata (1983), wobei dort auch die Verfeinerung von Stellen diskutiert wird. Eine Stelle wird dabei in zwei Stellen und eine Transition aufgeteilt und die Transition wird verfeinert. Vogler (1987) generalisiert diesen Ansatz dahingehend, dass auch Teilnetze mit mehr als einer Ein- und Ausgangstransition bei der Verfeinerung zugelassen werden, was jedoch die Verfeinerung komplizierter gestaltet. Eine Übersicht zu Reduktionsregeln im Kontext von Petrinetzsprachen liefern Brauer u. a. (1989).

Einen allgemeineren Ansatz liefern Berthelot u. a. (1980), der dann von Berthelot (1986) erweitert wird. Dieser Ansatz diskutiert auch die Transformationsregeln, die von Murata (1989) graphisch zusammengefasst wurden. Diese Zusammenfassung ist in Abbildung 5.7 dargestellt. Dabei sind mögliche weitere Knoten im Vorbereich oder Nachbereich eines Knotens schematisch durch Pfeile dargestellt. In der oberen Zeile werden Knoten mit genau einer Ein- und einer Ausgangskante ersetzt. In der mittleren Zeile werden Knoten mit identischem Vor- und Nachbereich ersetzt und in der unteren Zeile werden Schleifen entfernt, wenn nur ein Knoten im Vor- und im Nachbereich vorhanden ist, der zudem derselbe ist. Die Regel e) kann dabei in einem Workflownetz nicht angewandt werden, da i die einzige markierte Stelle darstellt. Dennoch ist sie der Vollständigkeit halber in Abbildung 5.7 aufgenommen worden.

Reduktionsregeln für Free-Choice-Netze werden ebenfalls in Desel und Esparza (1995) diskutiert. Diese bieten jedoch den Nachteil, dass sie nicht auf Netze angewendet werden können, die nicht die Free-Choice-Eigenschaft besitzen, ohne dass

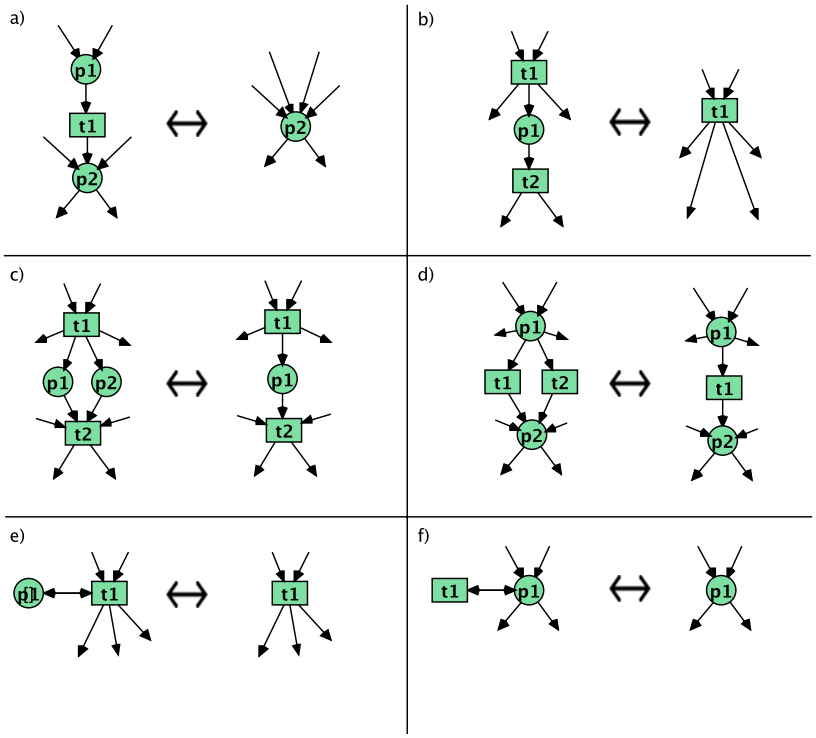


Abbildung 5.7: Die Reduktionsregeln von Murata (1989)

zentrale Eigenschaften verloren gehen können.

Wie in Murata (1989) dargestellt, erhält jede der Transformationsregeln in Abbildung 5.7 die Lebendigkeit und die Sicherheit eines Netzes. Berthelot (1986) zeigt darüber hinaus, das auch die Überdeckung durch S-und T-Invarianten erhalten bleibt. Wendet man die dargestellten Regeln von Links nach Rechts an, so müssen die Ursprungsnetze nicht die Free-Choice-Eigenschaft besitzen, während die resultierenden Netze diese Eigenschaft nach der Transformation besitzen können. Somit lässt sich mit Hilfe dieser Regeln nicht nur die Analyse beschleunigen, sondern auch die Ausdruckmächtigkeit der Netze erhöhen.

Sei $\mathcal{N}$ ein zusammenhängendes Netz und sei $\mathcal{N}'$ ein Netz, das aus $\mathcal{N}$ durch eine der in Abbildung 5.7 dargestellten Transformationen entstanden ist und das mindestens eine Stelle und mindestens eine Transition besitzt. Nach Berthelot (1986) und Murata (1989) lassen sich dann folgende Eigenschaften für die beiden Netze $\mathcal{N}$ und $\mathcal{N}'$ und eine Markierung $\mathbf{m}_0$, die keine Stelle markiert, die nicht in einer Reduktionsregel markiert ist, festhalten[7]:

[7] In Berthelot (1986) wird zwar lediglich die S-Invariante nachgewiesen, jedoch lässt sich das Bestehen der postiven T-Invariante sehr leicht über das duale Netz zeigen, da in diesem jede T-Invariante eine S-Invariante des Originalnetzes ist und umgekehrt (vgl. Desel und Esparza,

- $\mathcal{N}$ besitzt eine *positive S-Invariante* gdw. dies für $\mathcal{N}'$ gilt,
- $\mathcal{N}$ besitzt eine *positive T-Invariante* gdw. dies für $\mathcal{N}'$ gilt,
- $\mathcal{N}$ ist in $\mathbf{m}_0$ *beschränkt* gdw. dies für $\mathcal{N}'$ gilt,
- $\mathcal{N}$ ist in $\mathbf{m}_0$ *sicher* gdw. dies für $\mathcal{N}'$ gilt und
- $\mathcal{N}$ ist in $\mathbf{m}_0$ *lebendig* gdw. dies für $\mathcal{N}'$ gilt.

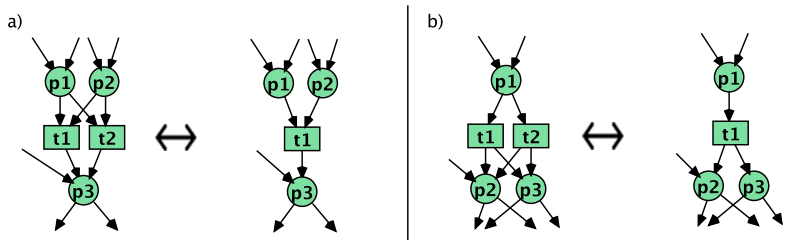


Abbildung 5.8: Ersetzung von Transitionen mit gleichem Vor- und Nachbereich gemäß Berthelot (1986)

Eine positive Invariante bezieht sich dabei auf das gesamte Netz und ist eine Invariante, in der alle Elemente der Invariante einen Wert größer als 0 besitzt. Berthelot (1986) generalisiert zudem die Regeln in der mittleren Zeile von Abbildung 5.7 auf Transitionen und Stellen mit gleichem Vor- und Nachbereich im Allgemeinen, so dass auch die in Abbildung 5.8 dargestellten Transitionen verfeinert werden können. Auch hierbei bleiben die oben beschriebenen Eigenschaften erhalten.

Weitere Reduktionsregel

Neben diesen Regeln soll im weiteren Verlauf noch eine weitere Regel verwendet werden, die in Abbildung 5.9 schematisch dargestellt ist. Da diese Regel in den oben genannten Quellen nicht mit angegeben wird, soll sie hier formalisiert werden und gezeigt werden, dass auch für diese Regel die oben genannten Eigenschaften erhalten bleiben. Dabei kann der Zyklus γ eine beliebige Länge aufweisen, solange für alle Transitionen $t \in \textsc{Alph}(\gamma) \cap T$ dann $|{}^\bullet t| = |t^\bullet| = 1$ gilt.

Definition 5.35 (Reduktion einfacher Zyklen)
Gegeben ein Netz $\mathcal{N}$. Dann lässt sich folgende Transformation zu einem Netz $\mathcal{N}'$ durchführen, sofern die Vorbedingungen eingehalten werden.

1995). Zudem sind im dualen Netz die Regeln der linken und die der rechten Spalte vertauscht, woraus sich die Aussage ergibt, wenn $T' \neq \emptyset$ ist.

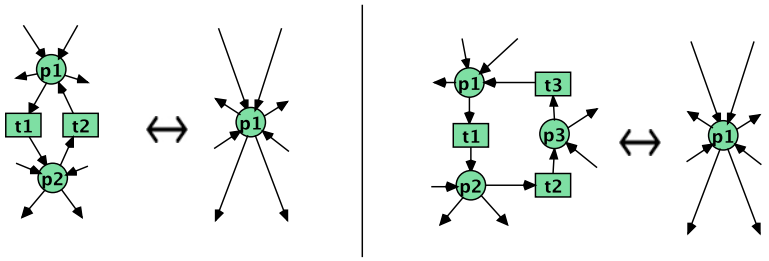


Abbildung 5.9: Die Reduktionsregel aus Definition 5.35

Vorbedingung:
Es gibt einen Zyklus γ, so dass für alle Transitionen $t \in \textsc{Alph}(\gamma) \cap T$ gilt $|{}^\bullet t| = |t^\bullet| = 1$.
Regel:
Es ist $\mathcal{N}' = (P', T', F')$ mit

$$\begin{aligned} P' &\stackrel{\text{def}}{=} P \cup \{p_*\} \setminus \textsc{Alph}(\gamma) \\ T' &\stackrel{\text{def}}{=} T \setminus \textsc{Alph}(\gamma) \\ F' &\stackrel{\text{def}}{=} F \cap (P' \times T' \cup T' \times P') \\ &\quad \cup \{(p_*, t) \in (P' \times T') \mid (p, t) \in F \wedge p \in (P \setminus P')\} \\ &\quad \cup \{(t, p_*) \in (T' \times P') \mid (t, p) \in F \wedge p \in (P \setminus P')\} \end{aligned}$$

⧫

Insbesondere gilt auch der Sonderfall, dass es einen Zyklus mit nur zwei Knoten gibt, so dass die Regel in Abbildung 5.7f) lediglich einen Sonderfall dieser Regel darstellt. Ebenso sind in Abbildung 5.9 nur zwei Fälle dargestellt. Eine ähnliche Regel findet sich bei Ramos u. a. (2006) (Def. 2.18). Dort ist die Menge der Kanten jedoch nicht ganz korrekt definiert worden. Auch für diese Regel bleiben wesentliche Eigenschaften erhalten. Wiederum bezieht sich eine positive Invariante auf das gesamte Netz und ist eine Invariante, in der alle Elemente der Invariante einen Wert größer 0 besitzen.

Satz 5.36 (Eigenschaften der Reduktion einfacher Zyklen)
Sei $\mathcal{N}$ ein Netz, auf das sich die in Definition 5.35 gegebene Transformationsregel anwenden lässt. Dann gilt für das Resultat der Transformation $\mathcal{N}'$ (mit $P' \neq \emptyset$ und $T' \neq \emptyset$):

- $\mathcal{N}$ besitzt eine *positive S-Invariante* gdw. dies für $\mathcal{N}'$ gilt,
- $\mathcal{N}$ besitzt eine *positive T-Invariante* gdw. dies für $\mathcal{N}'$ gilt,

- $\mathcal{N}$ ist in $\mathbf{m}_0$ *beschränkt* gdw. dies für $\mathcal{N}'$ gilt,
- $\mathcal{N}$ ist in $\mathbf{m}_0$ *sicher* gdw. dies für $\mathcal{N}'$ gilt und
- $\mathcal{N}$ ist in $\mathbf{m}_0$ *lebendig* gdw. dies für $\mathcal{N}'$ gilt.

Beweis:
(Positive Invarianten) Nach Satz 3.52 ist eine Abbildung $\vec{\iota} : T \to \mathbb{Q}$ eine T-Invariante gdw. für alle $p \in P$ gilt $\sum_{t \in {}^\bullet p}(\vec{\iota}(t)) = \sum_{t \in p^\bullet}(\vec{\iota}(t))$ und eine Abbildung $\vec{\kappa} : P \to \mathbb{Q}$ eine S-Invariante gdw. für alle $t \in T$ gilt $\sum_{p \in {}^\bullet t}(\vec{\kappa}(p)) = \sum_{p \in t^\bullet}(\vec{\kappa}(p))$. Somit müssen alle Stellen des Zyklus den selben S-Invarianten angehören, die dann genau denjenigen von p_* nach der Transformation entsprechen und umgekehrt. Jede T-Invariante, die keine der Transitionen des Zyklus überdeckt, ist in beiden Netzen existent. Jede T-Invariante, die nur einen Teil der Transitionen des Zyklus überdeckt existiert ebenfalls in beiden Netzen, wobei diese Transitionen nach der Reduktion nicht mehr Teil des Netzes sind. Es bleibt jedoch die Bedingung an die T-Invarianten erfüllt. Da durch die Transitionen des Zyklus eine eigene T-Invariante entsteht, gilt die Aussage auch für diejenigen T-Invarianten, die alle Transitionen des Zyklus überdecken, wobei dann diese minimale T-Invariante nicht mehr für $\mathcal{N}'$ existiert. Da $T' \neq \emptyset$ ist, gibt es eine T-Invariante in $\mathcal{N}'$.
(Beschränktheit und Sicherheit) Ist in $\mathcal{N}$ eine Markierung $\mathbf{m}$ erreichbar, die eine Stelle $p \in P$ mit mehr als k Marken markiert, so ist diese Schaltfolge auch in $\mathcal{N}'$ möglich, wenn man die Transitionen des Zyklus γ aus dieser Schaltfolge streicht, da jede Transition im Nachbereich einer Stelle des Zyklus in $\mathcal{N}'$ im Nachbereich von p_* ist. Umgekehrt ist jede Schaltfolge in $\mathcal{N}'$ auch in $\mathcal{N}$ möglich, wobei dieser Schaltfolge die entsprechenden Transitionen des Zyklus hinzugefügt werden müssen, wenn Transitionen im Vorbereich des Zyklus und im Nachbereich des Zyklus nacheinander schalten, die jeweils nicht Teil des Zyklus sind.
(Lebendigkeit) Wie für die Beschränktheit, so ist auch für die Lebendigkeit jede Schaltfolge in $\mathcal{N}$ durch eine entsprechende Schaltfolge in $\mathcal{N}'$ realisierbar und umgekehrt. Folglich ist jede Markierung, die keine Stelle des Zyklus markiert, in beiden Netzen erreichbar. Da von einer Markierung, die eine Stelle des Zyklus in $\mathcal{N}$ markiert, zu jeder Transition im Nachbereich von p_* stets eine Markierung erreichbar ist, die ebenfalls die entsprechende Stelle im Vorbereich dieser Transition markiert, gilt die Aussage. □

In den meisten praktischen Fällen lässt sich mit Hilfe dieser Reduktionsregeln für eine Vielzahl von Workflownetzen die Zahl der Knoten drastisch reduzieren. Da dies effizient möglich ist, lässt sich hierdurch die Analyse auch für größere Netze noch effizient durchführen. So lässt sich etwa der Abschluss des Netzes aus Abbildung 5.6 bis auf einen Knoten reduzieren. Zunächst können mit der Regel aus Abbildung 5.7 a) oder b) die Stellen p_2 und p_7 zusammen mit den Transitionen t_3 und t_8 reduziert werden. Durch die gleichen Regeln können auch die Stellen p_5 und p_6 zusammen

mit den Transitionen t_4 und t_6 reduziert werden. Dann haben die Transitionen t_2, t_5 und t_7 den gleichen Vor- und den gleichen Nachbereich, so dass zwei davon ebenfalls reduziert werden können. Hierauf folgen die Stellen p_3, p_4 und p_8, die dann ebenfalls den gleichen Vor- und Nachbereich haben. Abschließend bleibt lediglich eine Sequenz, die dann durch die Regeln in Abbildung 5.7 a) oder b) auf eine Stelle und ein Transition reduziert werden kann. Übrig bleibt die Anwendung der Regel aus Abbildung 5.7 f) oder die einzige im Abschluss eines korrekten Workflownetzes mögliche Anwendung der Regel aus Abbildung 5.7 e).

Bei Ullrich (1977) werden wohlstrukturierte Netze anders definiert, da sie keine Zyklen besitzen, die die Form der Zyklen aus Definition 5.35 besitzen. Aus diesem Grund kann dann auch auf die Reduktionsregel aus Definition 5.35 verzichtet werden. Diese Netzklasse wird im nächsten Kapitel als elementar-wohlstrukturiert bezeichnet und dort zur Modellierung von Objekten verwendet.

Reduktionsregeln zur weiteren Analyse

Neben der Möglichkeit, die Korrektheit und die Sicherheit eines Netzes durch die Reduktionsregeln zu überprüfen existiert zudem die Möglichkeit, die nebenläufige Aktiviertheit ebenfalls über Reduktionsregeln zu erkennen. Es lässt sich relativ leicht erkennen, dass die Stellen in Abbildung 5.7 c) gleichzeitig markierbar sind, wenn das zugehörige Netz korrekt ist. Repräsentieren diese Stellen ganze Subnetze, so sind die Transitionen dieser Subnetze offensichtlich nebenläufig aktivierbar.

Reduktionsabbildungen Zu jeder Reduktion lässt sich eine Abbildung auf Knoten und Kanten definieren. Dabei werden die Regeln aus Abbildung 5.7 a) und b) auf Kanten abgebildet, während die Regeln aus Abbildung 5.7 c), d) und f) auf Knoten abgebildet werden. Die Regel aus Abbildung 5.9 ist hingegen etwas aufwendiger abzubilden.

Die aus den Reduktionsregeln abgeleiteten Ersetzungsregeln haben die allgemeine Form $(P_1, T_1, F_1) \overset{\mathcal{N}}{\leftrightarrow} (P_2, T_2, F_2)$. Eine solche Regel besagt, dass alle Elemente aus (P_1, T_1, F_1) aus einem Netz $\mathcal{N}$ entfernt werden sollen und durch die Elemente von (P_2, T_2, F_2) ersetzt werden sollen. (P_1, T_1, F_1) ist im Allgemeinen kein Subnetz von $\mathcal{N}$, da F_1 auch Kanten zu Knoten beinhaltet, die nicht in P_1 oder T_1 sind. Die Ersetzungsregeln für die Reduktionsregeln aus Abbildung 5.7 sind in der Tabelle 5.1 dargestellt. Dort beschreibt die linke Seite die anzuwendende Reduktionsregel und die rechte Seite die resultierende Ersetzungsregel, mit der das ursprüngliche Netz später wieder rekonstruiert werden kann.

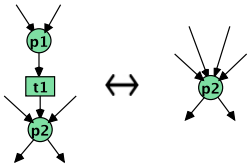
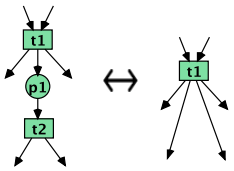
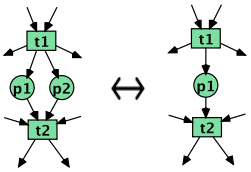
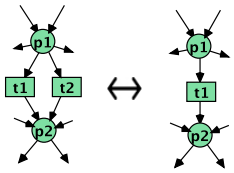
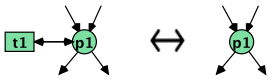
	Reduktionsregel	Ersetzungsregel $(P_1, T_1, F_1) \overset{\mathcal{N}}{\leftrightarrow} (P_2, T_2, F_2)$
a		$P_1 = \{p_1\},\ T_1 = \{t_1\},$ $F_1 = {}^\bullet p_1 \times \{p_1\} \cup \{(p_1, t_1), (t_1, p_2)\}$ $P_2 = \emptyset,\ T_2 = \emptyset,\ F_2 = {}^\bullet p_1 \times \{p_2\}$
b		$P_1 = \{p_1\},\ T_1 = \{t_2\},$ $F_1 = \{t_2\} \times t_2^\bullet \cup \{(t_1, p_1), (p_1, t_2)\}$ $P_2 = \emptyset,\ T_2 = \emptyset,\ F_2 = \{t_1\} \times t_2^\bullet$
c		$P_1 = \{p_2\},\ T_1 = \emptyset,$ $F_1 = \{(t_1, p_2), (p_2, t_2)\}$ $P_2 = \emptyset,\ T_2 = \emptyset,\ F_2 = \emptyset$
d		$P_1 = \emptyset,\ T_1 = \{t_2\},$ $F_1 = \{(p_1, t_2), (t_2, p_2)\}$ $P_2 = \emptyset,\ T_2 = \emptyset,\ F_2 = \emptyset$
f		$P_1 = \emptyset,\ T_1 = \{t_1\},$ $F_1 = \{(p_1, t_1), (t_1, p_2)\}$ $P_2 = \emptyset,\ T_2 = \emptyset,\ F_2 = \emptyset$

Tabelle 5.1: Reduktionsanweisungen mit korrespondierenden Ersetzungsregeln

Bei der Anwendung der Reduktionsregeln lässt sich eine Folge von Ersetzungsregeln bilden, die es erlaubt, vom Ergebnis der Reduktion zurück zum Ursprungsnetz

zu gelangen, indem man die Ersetzungsregeln in der umgekehrten Richtung anwendet. Während der Reduktion wird eine Ersetzungsregeln $r = [(P_1, T_1, F_1) \overset{\mathcal{N}}{\leftrightarrow} (P_2, T_2, F_2)]$ auf ein Netz $\mathcal{N} = (P, T, F)$ mit dem Resultat $\mathcal{N}' = (P', T', F')$ wie folgt angewandt:

$$\begin{aligned} P' &= (P \setminus P_1) \cup P_2 \\ T' &= (T \setminus T_1) \cup T_2 \\ F' &= (F \setminus F_1) \cup F_2 \end{aligned}$$

Entsprechend wird die Ersetzungsregel beim rekursiven Aufbau des Netzes dann andersherum angewandt, so dass wiederum aus dem Netz $\mathcal{N} = (P, T, F)$ das Netz $\mathcal{N}' = (P', T', F')$ entsteht:

$$\begin{aligned} P' &= (P \setminus P_2) \cup P_1 \\ T' &= (T \setminus T_2) \cup T_1 \\ F' &= (F \setminus F_2) \cup F_1 \end{aligned}$$

Hierbei werden nicht die Vorbedingungen der Reduktionsregeln überprüft, was separat geschehen muss. Vielmehr wird jeder Reduktionsregel eine Ersetzungsregel zugeteilt, so dass diese Rückgängig gemacht werden kann. Eine Reduktion eines Netzes auf ein kleineres Netz generiert somit eine Folge von Ersetzungsregeln der Art $r_1 \dots r_n$ mit $r_j = [(P_1, T_1, F_1) \overset{\mathcal{N}}{\leftrightarrow} (P_2, T_2, F_2)]$. Es lässt sich leicht überprüfen, dass für jede Ersetzungsregel die Anwendung der Regel in beide Richtungen zum Ursprungsnetz führt.

Nebenläufige Aktiviertheit Anhand der Ersetzungsregeln lassen sich nun auch nebenläufige Teile eines Netzes identifizieren. Lässt sich ein Teilnetz mit Hilfe der Reduktionsregeln auf eine Stelle oder eine Transition reduzieren, so sind zwei Transitionen genau dann nebenläufig aktivierbar, wenn sie in Teilnetzen sind, die durch die Verfeinerung zweier Stellen entstanden sind, die wiederum aus der Regel c) hervorgegangen sind. Dies lässt sich in einem Satz festhalten.

Satz 5.37 (Nebenläufigkeit und Reduktionsregeln)
Sei $\mathcal{N}$ ein sicheres korrektes Workflownetz und sei $\mathcal{N}'$ ein Subnetz von $\mathcal{N}$. Lässt sich $\mathcal{N}'$ durch die Reduktionsregeln in Tabelle 5.1 auf einen Knoten reduzieren und ist $r_1 \dots r_n$ die hierzu notwendige Folge von Ersetzungsregeln, so sind zwei Transition t_1 und t_2 in $\mathcal{N}'$ nebenläufig aktivierbar gdw. es ein r_m mit $1 < m < n$ gibt, so dass

- r_m die Regel c) aus Tabelle 5.1 umsetzt und
- t_1 in einem Subnetz $\mathcal{N}_A$ ist, dass durch die Regeln $r_1 \dots r_{m-1}$ zu der Stelle p_1 der Regel reduziert wird und
- t_2 in einem Subnetz $\mathcal{N}_B$ ist, dass durch die Regeln $r_1 \dots r_{m-1}$ zu der Stelle p_2 der Regel reduziert wird.

Beweis:
($\Rightarrow$) Seien t_1 und t_2 nebenläufig aktivierbar. Da $\mathcal{N}$ sicher ist müssen die Vorbereiche von t_1 und t_2 disjunkt und in einer erreichbaren Markierung gleichzeitig markierbar sein. Da eine solche Markierung auf allen S-Invarianten mit $[i]$ übereinstimmen muss, dürfen die Vorbereich keine gemeinsame minimale S-Invariante einer Basis aufweisen. Dann muss es eine Transition t geben, die zu diesen beiden minimalen S-Invarianten führt und für die folglich $p_1, p_2 \in t^\bullet$ ist für $p_1 \neq p_2$. Eine solche Transition lässt sich jedoch nur durch Regel c) reduzieren, so dass die beiden Transitionen in den Subnetzen der verschiedenen minimalen S-Invarianten durch Ersetzungsregeln aus den beiden Stellen dieser Regel hervorgegangen sein müssen. Diese Ersetzungsregeln müssen vor der Elimination der Stellen $"p_1$ und p_2 auftreten.
($\Leftarrow$) Setze r_m die Regel c) aus Tabelle 5.1 um, so dass obige Angaben gelten. Da die beiden aus p_1 und p_2 hervorgegangenen Teilnetze keine Verbindung zueinander besitzen, können sie sich nicht beeinflussen. Als Verfeinerung der Stellen p_1 und p_2 muss entweder Regel a) oder Regel f) zuerst angewandt worden sein. Da sich leicht überprüfen lässt, dass bei jeder Anwendung der Regeln b) und c), in der die erste Transition der rechten Seite der Regeln aktivierbar ist, auch die Transitionen der linken Seite aktivierbar und die Stellen markierbar sind, und dass bei jeder Anwendung der Regeln a), d) und f), in der die erste Stelle der rechten Seite der Regeln markierbar ist, auch alle Transitionen der linken Seite aktivierbar und alle Stellen markierbar sind, folgt die Aussage dann über die Zahl der nach der ersten Verfeinerung angewandten Regeln $m - 2$. □

Anhand dieser Regeln ist es möglich, die Analyse der Workflownetze auf kleinere Netze zu begrenzen. Somit ist es möglich, ein Netz zunächst auf ein kleineres Netz zu reduzieren und dann zu analysieren. Zudem lassen sich auch die Anschriften eines Dienstbeschreibungsnetzes zusammenziehen, was im nächsten Kapitel betrachtet wird.

5.4 Elementarität endlicher Free-Choice-Dienstbeschreibungsnetze

Die Möglichkeit der Analyse ist die Hauptmotivation hinter der Einführung elementarer Dienstbeschreibungsnetze. Dabei zeigte sich, dass die schwache Korrektheit direkt für elementare Dienstbeschreibungsnetze entschieden werden kann, wie dies in Satz 5.11 gezeigt wurde. Für Dienstbeschreibungsnetze, deren zugrundeliegendes Netz ein Free-Choice-Netz ist, lässt sich auch die Elementarität in Polynomialzeit zeigen. Hierzu muss jedoch die Endlichkeit gegeben sein, was durch eine hinreichende Endlichkeitsbedingung erreicht wird. Diese wird zunächst vorgestellt. Im Anschluss hieran wird die Ermittlung der Elementarität für Dienstbeschreibungs-

netze mit Free-Choice-Charakter diskutiert.

5.4.1 Endliche Free-Choice-Dienstbeschreibungsnetze

Elementare Dienstbeschreibungsnetze wurden in Definition 5.10 definiert und weisen gegenüber herkömmlichen Dienstbeschreibungsnetzen zahlreiche Einschränkungen auf. Diese Eigenschaften lassen sich für Free-Choice-Workflownetze in Polynomialzeit zeigen. Mittels der hinreichenden Endlichkeitsbedingung dieses Unterabschnitts wird dabei dafür gesorgt, dass die Endlichkeit der Schaltfolgen des Netzes gewährleistet ist. Dies führt zur Einführung endlicher Free-Choice-Dienstbeschreibungsnetze, die in diesem Unterabschnitt betrachtet werden.

Elementare Dienstbeschreibungsnetze erlauben lediglich eine endliche Zahl von Wiederholungen in Schleifen. Eine hinreichende Bedingung, um dies zu gewährleisten, ist durch die Verwendung eines Zählers gegeben, der angibt, wie häufig eine Schleife durchlaufen werden soll. In Anlehnung an programmiersprachliche Konstrukte sind also nur FOR-Schleifen, nicht hingegen WHILE-Schleifen erlaubt. Für erstere wird dabei angenommen, dass ein einmal gesetzter Zähler bei jedem Schleifendurchlauf um 1 dekrementiert wird. Bei der Formalisierung dieser Eigenschaft ist darauf zu achten, dass die Verschachtelung verschiedener Schleifen nicht dazu führt, dass zwei Schleifen ihre Zähler gegenseitig zurücksetzen und so eine unendliche Schaltfolge entsteht. Dies wird durch folgende Definition der Endlichkeitsbedingung vermieden. Die Endlichkeitsbedingung ist dabei hinreichend für die Endlichkeit, sie ist jedoch im Allgemeinen nicht notwendig.

Definition 5.38 (Hinreichende Endlichkeitsbedingung)
Ein Dienstbeschreibungsnetz $\mathcal{SDN}$ erfüllt die *hinreichende Endlichkeitsbedingung* gdw.

- das zugrundeliegende Netz ein korrektes Free-Choice-Workflownetz ist,
- für jede minimale T-Invariante $\vec{\iota}$ von $\mathcal{N}$ eine Transition $t_{\vec{\iota}} \in \|\vec{\iota}\|$ und eine Stelle $p_{\vec{\iota}} \in P$ existiert, so dass $p_{\vec{\iota}} \in {}^\bullet t_{\vec{\iota}} \cap t_{\vec{\iota}}^\bullet$ und $d(p_{\vec{\iota}}) = nat$,
- $W(p_{\vec{\iota}}, t_{\vec{\iota}}) = c_{i1}$, $W(t_{\vec{\iota}}, p_{\vec{\iota}}) = c_{i2}$, so dass $G(t_{\vec{\iota}}) = \tau$ impliziert $\tau = (c_{i1} > 0)$ oder $\tau = \tau' \wedge (c_{i1} > 0)$ und $(assign\ c_{i2} = c_{i1} - 1) \in \mathbf{cmd}(t)$, und
- für $R = \{(\vec{\iota_1}, \vec{\iota_2}) \mid \vec{\iota_1} \neq \vec{\iota_2} \wedge p_{\vec{\iota_2}} \in {}^\bullet\|\vec{\iota_1}\|\}$ die Hülle R^+ zyklenfrei ist.

♦

Die Endlichkeitsbedingung soll anhand von Abbildung 5.10 illustriert werden. In diesem Fall besteht $\vec{\iota_1}$ aus den Transitionen t_2, t_4 und t_5 und es ist $p_{\vec{\iota_1}} = p_2$ und $t_{\vec{\iota_1}} = t_2$. Nach einer endlich Zahl von Realisierungen der T-Invariante $\vec{\iota_1}$ ist

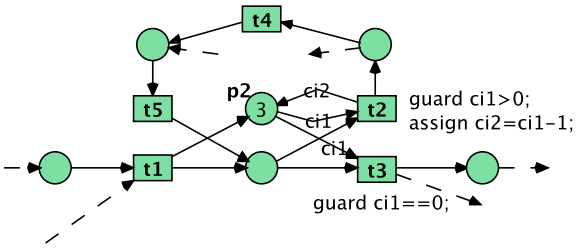


Abbildung 5.10: Umsetzung der Endlichkeitsbedingung

die Aktivierungsbedingung von $t_{\vec{\iota_1}} = t_2$ nicht mehr wahr, und es muss t_3 schalten. Die letzte Anforderung in Definition 5.38 sorgt dafür, dass sich zwei minimale T-Invarianten nicht gegenseitig die Zähler auf $p_{\vec{\iota}}$ zurücksetzen, und es so zu einer unendlichen Schaltfolge kommt.

Aufgrund der Free-Choice-Eigenschaft ist $t_{\vec{\iota}}$ die einzige Transition der T-Invariante, die den Wert der Marke auf der Stelle $p_{\vec{\iota}}$ verändert. Somit existiert zu jeder minimalen T-Invariante eine Transition $t_{\vec{\iota}}$, die nur endlich oft schalten kann, und es muss nach einer endlichen Zahl von Realisierungen einer T-Invariante eine weitere T-Invariante schalten. Diese kann wiederum nur endlich oft schalten, da auch diese T-Invariante eine entsprechende Stelle und eine entsprechende Transition besitzt. Es besteht jedoch die Möglichkeit, dass eine T-Invariante den Zähler einer anderen T-Invariante zurücksetzt oder umgekehrt, so dass es in einer Gruppe von T-Invarianten zu einer unendlichen Schaltfolge kommt. Dies wird durch die Forderung verhindert, dass R^+ zyklenfrei sein soll, da mit $p_{\vec{\iota_2}} \in {}^\bullet\|\vec{\iota_1}\|$ auch $p_{\vec{\iota_2}} \in \|\vec{\iota_1}\|^\bullet$ gilt. Folglich kann die Realisierung einer minimalen T-Invariante durch eine Transition einer anderen minimalen T-Invariante nur dann aktiviert werden, wenn die T-Invarianten in der Relation R stehen. Da R^+ zyklenfrei ist, kann es somit keine unendlichen Schaltfolgen geben. Aus diesen Überlegungen ergibt sich das folgende Lemma.

Satz 5.39 (Hinreichende Endlichkeitsbedingung garantiert Endlichkeit)
Erfüllt ein Dienstbeschreibungsnetz $\mathcal{SDN}$ die hinreichende Endlichkeitsbedingung aus Definition 5.38, so ist jede Schaltfolge endlich.

Beweis:
Angenommen es gäbe eine unendliche Schaltfolge σ_α in $\mathcal{SDN}$. Da nach Lemma 4.28 jede Schaltfolge in $\mathcal{SDN}$ auch im zugrundeliegenden Netz $\mathcal{N}$ möglich ist, muss es in $\mathcal{N}$ eine T-Invariante geben, die unendlich oft realisiert wird. Wäre dies nicht der Fall, so müssten alle erreichten Markierungen verschieden voneinander sein. Da $\mathcal{N}$ jedoch aufgrund der Sicherheit maximal $2^{|P|}$ verschiedene Markierungen besitzt, ist

dies nicht möglich. (Jedes korrekte Free-Choice-Workflownetz ist sicher.)

Sei $\vec{\iota}_*$ die semi-positive T-Invariante von $\mathcal{N}$, die unendlich oft realisiert wird. Nach Satz 3.55 ist $\vec{\iota}_*$ eine Summe minimaler T-Invarianten. Jede dieser minimalen T-Invarianten $\vec{\iota}$ kann aufgrund der Stelle $p_{\vec{\iota}}$ und der Transition $t_{\vec{\iota}}$ nur endlich oft realisiert werden, bevor eine weitere minimale T-Invariante realisiert werden muss, da der Wert von c_{i2} mit jedem Schalten dekrementiert wird, denn dieser Wert kann aufgrund der Konsistenz des Anweisungsblocks nur einmal zugewiesen werden und $G(t_{\vec{\iota}})$ wertet nur zu wahr aus, wenn c_{i1} größer 0 ist. Eine weitere minimale T-Invariante kann aus den gleichen Gründen ebenfalls nur endlich oft schalten, sofern der Zähler einer T-Invariante nicht zurückgesetzt wird, die dann wiederum einen Zähler einer anderen T-Invariante zurücksetzt usw. bis es zu einer unendlichen Schaltfolge kommt. Dies wird durch die Forderung verhindert, dass R^+ zyklenfrei sein soll, da mit $p_{\vec{\iota}_2} \in {}^\bullet\|\vec{\iota}_1\|$ auch $p_{\vec{\iota}_2} \in \|\vec{\iota}_1\|^\bullet$ gilt. Hierdurch kann die Realisierung einer minimalen T-Invariante durch eine Transition einer anderen minimalen T-Invariante nur dann aktiviert werden, wenn die T-Invarianten in der Relation R stehen. Da R^+ zyklenfrei ist, kann es somit keine unendlichen Schaltfolgen geben. □

Dies bedeutet jedoch nicht, dass jede Schaltfolge zu einem finalen Netzzustand führt. Vielmehr ist es im Allgemeinen durchaus möglich, dass ein Netz verklemmt. Um dies zu verhindern sind weitere Bedingungen notwendig, die in diesem Kapitel im Rahmen der Korrektheitsanalyse elementarer Dienstbeschreibungsnetze in Abschnitt 5.2 diskutiert worden sind. Die Endlichkeitsbedingung wird dabei durch die hinreichende Endlichkeitsbedingung erfüllt.

Korollar 5.40 (Hinreichende Endlichkeitsbedingung)
Ein Dienstbeschreibungsnetz $\mathcal{SDN}$, das die hinreichende Endlichkeitsbedingung aus Definition 5.38 erfüllt, erfüllt auch die Endlichkeitsbedingung aus Definition 5.6.

Beweis:
Dies folgt direkt aus Definition 5.38 und aus Satz 5.39. □

Wie in folgender Definition wiedergegeben, sind endliche Free-Choice-Dienstbeschreibungsnetze dadurch gekennzeichnet, dass das ihnen zugrundeliegende Workflownetz die Free-Choice-Eigenschaft besitzt und dass sie die hinreichende Endlichkeitsbedingung erfüllen.

Definition 5.41 (Endliches Free-Choice-Dienstbeschreibungsnetze)
Ein endliches Free-Choice-Dienstbeschreibungsnetze ist ein Dienstbeschreibungsnetz, dessen zugrundeliegendes Netz $\mathcal{N}$ ein Free-Choice-Workflownetz ist und das die hinreichende Endlichkeitsbedingung aus Definition 5.38 erfüllt. ⧫

Um die relativ aufwendige Struktur zu verbergen sollte die hinreichende Endlichkeitsbedingung in einer graphischen Notation durch ein eigenes Konstrukt gekapselt werden. Dies kann etwa durch eine spezielle Schleifen-Transition erfolgen, die dann entsprechend verfeinert wird, wie dies bei BPMN anhand einer Loop-Task erfolgt. Dies wird in Kapitel 7 noch einmal angeschnitten.

5.4.2 Elementarität endlicher Dienstbeschreibungsnetze

Zwei notwendige Kriterien zur Ermittlung der Elementarität waren die Korrektheit des zugrundeliegenden Workflownetzes und die Möglichkeit, Markierungen zu identifizieren, die Transitionen nebenläufig aktivieren oder eine bestimmte Stelle markieren, während eine bestimmte Transition aktiviert ist. Diesen Fragen aufbauend auf den Ergebnissen des letzten Abschnitts für Free-Choice-Netze nachzugehen ist Gegenstand dieses Unterabschnitts. Free-Choice-Workflownetze bieten dabei den Vorteil, dass diese Fragen sämtlich in Polynomialzeit entschieden werden können.

Korrektheit von Free-Choice-Workflownetzen

Um Nachzuweisen, dass ein Free-Choice-Workflownetz korrekt ist, muss nach Verfahren 5.23 eine Überdeckung durch minimale T-Invarianten berechnet werden, deren Subnetze jeweils T-Komponenten sind. Da die Analyse mit Verfahren 5.28 auf dem dualen Netz erfolgt, müssen statt der T-Komponenten dann S-Komponenten ermittelt werden.

Ein effizienter Ansatz eine Überdeckung durch S-Komponenten für ein Free-Choice-Netz zu berechnen findet sich bei Kemper (1994). Dort werden die Cluster des Netzes anhand einer Tiefensuche in eine Baumstruktur gebracht, auf der dann die minimalen Siphons ermittelt werden. Sind diese Siphons gleichzeitig S-Komponenten, so werden sie der Menge der S-Komponenten hinzugefügt. Andernfalls ist das Netz nicht wohlgeformt und der Algorithmus terminiert mit der Aussage, dass ein minimales Siphon keine S-Komponente ist (vgl. Satz 3.61). Die Termination des Algorithmus mit einer S-Überdeckung ist also ein notwendiges, aber kein hinreichendes Kriterium für die Wohlgeformtheit eines Free-Choice-Netzes. Dabei wird nicht notwendigerweise die kleinste Zahl von S-Komponenten ermittelt, die benötigt wird, um ein Netz zu überdecken. Da jedoch auf den S-Komponenten eine Ordnung existiert, so dass bei der Vereinigung aller S-Komponenten jede S-Komponente mindestens eine Stelle überdeckt, die in der Vereinigung noch nicht existiert, sind alle S-Komponenten der Konstruktion linear unabhängig. Sie stellen jedoch im Allgemeinen keine Basis dar.

Da der Algorithmus aus Kemper (1994) mit der Zeitkomplexität $O(|T||P|)$ arbeitet, lässt sich so mit einem Zeitaufwand von $O(|T||P|)$ eine Überdeckung durch S-Komponenten und durch T-Komponenten ermitteln. Dies entspricht dem ersten Schritt in Verfahren 5.23. In der Schleife dieses Verfahrens muss dann im schlech-

testen Fall jede minimale T-Invariante nachberechnet werden. Zudem muss für jede minimale T-Invariante die Vereinigung zweier Netze erfolgen, und es müssen Mengenoperationen auf den Mengen der Transitionen durchgeführt werden. Das Berechnen der T-Komponenten kann nach dem Verfahren von Kemper (1993) mit einem Zeitaufwand von $O(|P| + |T| + |F|)$ erfolgen[8]. Das Erstellen des T-Invariantenkonstruktionsnetz aus Definition 5.25 kann in Linearzeit zur Zahl der Kanten in einem Netz durch eine Tiefensuche erfolgen. Die Mengenoperationen können hingegen bei effizienter Implementierung in Linearzeit zu $|T|$ oder schneller erfolgen. Im schlechtesten Fall wäre die Komplexität somit $O((|P|+|T|+|F|)\cdot|T|)$. Somit ist die Analyse der Korrektheit mit quadratischem Zeitaufwand möglich, wenn man $O(|F|) = O(|P|)$ annimmt, was für die meisten praktischen Fälle gelten dürfte, da man stets die Zahl eingehender Kanten eines Knoten auf eine Konstante c beschränken kann.

Nebenläufigkeit in Free-Choice-Workflownetzen

Im ersten Teil dieses Unterabschnitt wurde dargestellt, wie mittels des Algorithmus von Kemper (1994) und mittels des Verfahrens 5.28 die Korrektheit eines Workflownetzes effizient überprüft werden kann. Gleichzeitig liefert dieses Verfahren eine Basis minimaler S-Invarianten, die nun dazu genutzt werden kann, mittels Satz 5.34 die nebenläufige Aktiviertheit einer Transition respektive die Erreichbarkeit einer Markierung zu ermitteln. Stellt $P_{\|}$ die Menge der markierten Stellen dar, so müssen hierzu nach Satz 5.34 folgende Eigenschaften überprüft werden, wobei $\textsc{Sinv}_{\|} \stackrel{\text{def}}{=} \{\vec{\kappa} \in \mathbf{B}_S(\overline{\mathcal{N}}) \mid (\|\vec{\kappa}\| \cap P_{\|}) \neq \emptyset\}$ und $\vec{\kappa_*} \stackrel{\text{def}}{=} \sum_{\vec{\kappa}\in\textsc{Sinv}_{\|}} \vec{\kappa}$ ist.

- In $\mathcal{N}_{\vec{\kappa_*}}$ müssen alle Fallen eine Stelle aus $P_{\|}$ überdecken und
- $[i] + [\mathcal{N}_{\vec{\kappa_*}}] \cdot X = \chi[P_{\|}]$ muss für einen T-indizierten Spaltenvektor $X \in \mathbb{Q}^{|T|}$ gelten.

Vor der Überprüfung kann zur Steigerung der Effizienz überprüft werden, ob $|\textsc{Sinv}_{\|}| = |P_{\|}|$ erfüllt ist. Dies lässt sich leicht überprüfen und liefert bereits ein hinreichendes Ausschlusskriterium, wie im Anschluss an Satz 5.34 dargestellt wurde.

Die Überprüfung der ersten Bedingung, dass in $\mathcal{N}_{\vec{\kappa_*}}$ alle Fallen eine Stelle aus $P_{\|}$ überdecken, kann durch das im Beweis zu Theorem 8.12 von Desel und Esparza (1995) vorgestellte Verfahren überprüft werden. Hierzu wird die Menge $R = P_{\vec{\kappa_*}} \setminus P_{\|}$

[8] Hierzu wird das Verfahren von Kemper (1993) zur Ermittlung minimaler Siphons in Linearzeit zur Zahl der Kanten verwendet. Dem dort geschilderten Verfahren wird eine Stelle übergeben, woraufhin ein minimales Siphon berechnet wird, dass diese Stelle überdeckt. Da jedes Siphon in einem wohlgeformten Free-Choice-Netz eine S-Komponente ist (vgl. Satz 3.61), lassen sich auf diese Weise S-Komponenten der Basis „nachberechnen“. Da weiterhin jede S-Komponente des dualen Netzes eine T-Komponente des Netzes ist, lassen sich so auch T-Komponenten ermitteln.

INPUT: Ein Netz $\mathcal{N} = (P, T, F)$ und eine Menge $R \subseteq P$
OUTPUT: Die maximale Falle R_{max} in R mit $R_{max} \subseteq R$

0 Initialisierung:
  $R_{max} = R$

while es gibt ein $p \in R_{max}$ und ein $t \in p^\bullet$ mit $t \notin {}^\bullet R_{max}$ **do**
  $R_{max} := R_{max} \setminus \{p\}$
5 **end while**;

Listing 5.1: Berechnung der maximalen Falle (Desel und Esparza, 1995)

gebildet. Besitzt R eine ordentliche Falle, so können nicht alle ordentlichen Fallen von $P_{\|}$ überdeckt sein. Somit müssen alle ordentlichen Fallen von R berechnet werden, was durch den Algorithmus zur Berechnung der Fallen von Desel und Esparza (1995) (S.87) überprüft werden kann. Dieser soll hier kurz wiederholt werden.

Der Algorithmus 5.1 berechnet die maximale Falle. Diese muss für $R = P_{\overrightarrow{\kappa_*}} \setminus P_{\|}$ leer sein, da ja gerade jede ordentliche Falle Stellen aus $P_{\|}$ überdecken soll. Ist das Ergebnis R_{max} von Algorithmus 5.1 nicht leer, so sind die Stellen entsprechend nicht erreichbar. Der Algorithmus arbeitet in quadratischer Komplexität zur Zahl der Knoten in einem Netz.

Aufwendiger gestaltet sich hingegen die Überprüfung der zweiten Bedingung. Hierzu muss ein Gleichungssystem gelöst werden, was im Allgemeinen kubische Komplexität hat. Diese Komplexität dominiert somit das Verfahren, so dass folgende Aussage gilt.

> *Die nebenläufige Ausführbarkeit zweier Transitionen in einem korrekten Free-Choice-Workflownetzes lässt sich mit kubischem Zeitaufwand zur Zahl der Knoten im Netz ermitteln.*

Bei der Analyse der Elementarität muss nun im schlechtesten Fall für je zwei Transitionen entschieden werden, ob diese nebenläufig aktivierbar sind. Gleichzeitig muss im schlechtesten Fall für jede Transition und jede Stelle überprüft werden, ob der Vorbereich der Transition und die Stelle in einer erreichbaren Markierung gleichzeitig markierbar sind. Dies bringt dann eine Zeitkomplexität von $O((|P| + |T|)^5)$ mit sich, die jedoch praktisch nicht erreicht wird. Somit ist die Analyse stets in Polynomialzeit möglich.

Überprüfung der Elementarität

Für endliche Free-Choice-Dienstbeschreibungsnetze kann die Elementarität nun direkt durch die Überprüfung sämtlicher Eigenschaften aus Definition 5.10 ermittelt werden. Die Free-Choice-Eigenschaft und die Workflow-Eigenschaft des zugrun-

deliegenden Netzes lassen sich zuvor mit bekannten Verfahren überprüfen. Nach Definition 5.10 sind dann die folgenden Eigenschaften zu analysieren:

- die Variablendeterminiertheit (Definition 4.19),
- die Endlichkeitsbedingung (Definition 5.6),
- die hinreichende Fortsetzbarkeitsbedingung (Definition 5.8),
- die hinreichende Typisierungskonformitätsbedingung (Definition 5.1) und
- die hinreichende Funktionalitätsbedingung (Definition 5.4).

Diese Eigenschaften sollen im Folgenden unter Bezugnahme auf die Ergebnisse dieses Kapitels für Free-Choice-Dienstbeschreibungsnetze überprüft werden. Hierbei stellt sich heraus, dass die Elementarität eines Free-Choice-Dienstbeschreibungsnetzes in Polynomialzeit gezeigt werden kann.

Variablendeterminiertheit Die in Definition 4.19 eingeführte Variablendeterminiertheit lässt sich über die Überprüfung der an den Kanten und in den Anweisungen verwendeten Variablen erreichen.

Endlichkeitsbedingung und hinreichende Fortsetzbarkeitsbedingung Eine Überprüfung der Free-Choice-Eigenschaft ist in linearer Zeit zur Zahl der Knoten möglich (vgl. Desel und Esparza, 1995). Ebenso kann die Workflow-Eigenschaft eines Netzes durch eine Tiefensuche in Linearzeit zur Anzahl der Kanten überprüft werden. Die Korrektheit kann dann mittels der Überlegungen aus Unterabschnitt 5.4.2 in quadratischer Zeit ermittelt werden. Für ein korrektes Free-Choice-Workflownetz kann über die Bedingungen in Definition 5.38 die hinreichende Endlichkeitsbedingung ermittelt werden. Die Überprüfung der Anschriften und die Zyklenfreiheit von R^+ kann dabei ebenfalls in Polynomialzeit erfolgen. Hieraus folgt nach Korollar 5.40 direkt die Erfüllung der Endlichkeitsbedingung. Hierbei ist zu beachten, dass die Überprüfung der minimalen T-Invarianten einer Basis ausreichend ist, da jede minimale T-Invariante dann eine Linearkombination dieser Basis ist.

Für die hinreichende Fortsetzbarkeitsbedingung ist nach Definition 5.8 ergänzend zu zeigen, dass

- für jede Transition $t \in T$, für die nicht $G(t) = \mathit{true}$ ist, eine Transition mit gleichem Vorbereich existiert und
- jede Klasse von Alternativtransitionen (vgl. Def. 5.7) eine Default-Transition besitzt, die durch die Anschrift `default` ausgezeichnet ist.

Die erste Bedingung lässt sich sehr leicht über die Menge der Transitionen feststellen, wobei zunächst alle Transitionen, die Transitionen mit gleichem Vorbereich besitzen, gefärbt werden, und dann für die übrigbleibenden Transitionen überprüft wird, ob für sie $G(t) = \mathit{true}$ ist. Anschließend wird überprüft, ob jede Menge von Alternativtransitionen, die während der obigen Überprüfung gebildet werden kann, eine Default-Transition besitzt. Durch die Verwendung des Schlüsselwortes `default` ist es möglich, die Aktivierungsbedingung dieser Transition zu berechnen, so dass eine Überprüfung auf Äquivalenz der Terme entfällt, da diese nicht immer in Polynomialzeit möglich sein muss.

Hinreichende Typisierungskonformitätsbedingung Nach Definition 5.1 ist zu zeigen, dass es zu einer Transition $t \in T$ mit einer Anweisung $(\mathit{removeRole}(\tau, s_\mathfrak{c})) \in \textsc{Alph}(\mathbf{cmd}(t))$ kein $\mathbf{m} \in [[i]\rangle_\mathcal{N}$ gibt, so dass $\mathbf{m}(p_*) > 0$ für ein $p_* \in P \setminus {}^\bullet t$ mit $d(p) \preceq s_\mathfrak{c}$ und $\mathbf{m}(p) > 0$ für alle $p \in {}^\bullet t$ gilt.

Die Überprüfung der Bedingung zur Sicherung der Typisierungskonformität erfolgt unter Verwendung der Ergebnisse aus Satz 5.34 und aus Unterabschnitt 5.4.2. Hierbei muss für $t \in T$ mit $\mathbf{Cmd} = \mathbf{cmd}(t)$ und $(\mathit{removeRole}(\tau, s_\mathfrak{c})) \in \mathbf{Cmd}$ überprüft werden, dass $p_* \in P \setminus {}^\bullet t$ mit $d(p) \preceq s_\mathfrak{c}$ impliziert, dass es kein $\mathbf{m} \in [[i]\rangle_\mathcal{N}$ gibt, für das $\mathbf{m}(p_*) > 0$ und für alle $p \in {}^\bullet t$ auch $\mathbf{m}(p) > 0$ ist. Bildet man die Menge $P_\| = \{p_*\} \cup {}^\bullet t$, so lässt sich Satz 5.34 anwenden. Somit ist die Analyse gemäß der Diskussion in Unterabschnitt 5.4.2 in Polynomialzeit möglich.

Hinreichende Funktionalitätsbedingung Gilt für je zwei Transitionen $t_1, t_2 \in T$ nicht, dass

- $\mathrm{R}_\Omega(t_1) \cap \mathrm{W}_\Omega(t_2) = \emptyset$,
- $\mathrm{R}_\Omega(t_2) \cap \mathrm{W}_\Omega(t_1) = \emptyset$ und
- $\mathrm{W}_\Omega(t_1) \cap \mathrm{W}_\Omega(t_2) = \emptyset$,

so ist nach Definition 5.4 zu zeigen, dass keine Markierung $\mathbf{m} \in [[i]\rangle_\mathcal{N}$ mit $\mathbf{m} \xrightarrow[\mathcal{N}]{\{t_1,t_2\}}$ existiert.

Die Überprüfung der Funktionalitätsbedingung erfolgt wiederum unter Verwendung der Ergebnisse aus Satz 5.34 und aus Unterabschnitt 5.4.2. Gilt für zwei Transitionen $t_1, t_2 \in T$, dass $\mathrm{R}_\Omega(t_1) \cap \mathrm{W}_\Omega(t_2) \neq \emptyset$, dass $\mathrm{R}_\Omega(t_2) \cap \mathrm{W}_\Omega(t_1) \neq \emptyset$ oder dass $\mathrm{W}_\Omega(t_1) \cap \mathrm{W}_\Omega(t_2) \neq \emptyset$ ist, so lässt sich für die Stellen $P_\| = {}^\bullet t_1 \cup {}^\bullet t_2$ mit Satz 5.34 überprüfen, ob es eine Markierung $\mathbf{m} \in [[i]\rangle_{\overline{\mathcal{N}}}$ mit $P_\| \subseteq \|\mathbf{m}\|$ gibt. Aufgrund der Ergebnisse aus Unterabschnitt 5.4.2 ist dies in Polynomialzeit möglich. Die Ermittlung der Mengen R_Ω und W_Ω kann ebenfalls in Polynomialzeit zur Anzahl der Symbole der Terme erfolgen.

Analyse der Elementarität Aufgrund der vergangenen Abschnitte lässt sich festhalten, dass die Elementarität eines Free-Choice-Dienstbeschreibungsnetzes in Polynomialzeit entschieden werden kann, da sich alle notwendigen Eigenschaften in Polynomialzeit berechnen lassen.

5.4.3 Anwendung von Reduktionsregeln

Die in Unterabschnitt 5.3.3 dargestellten Reduktionsregeln können lediglich auf ungefärbte Netze angewandt werden und dienen somit nur zur Reduktion des einem Dienstbeschreibungsnetz zugrundeliegenden Workflownetzes. Es lassen sich jedoch darüber hinaus auch einfache Konstrukte in elementaren Dienstbeschreibungsnetzen reduzieren. Zwei mögliche Reduktionsregeln werden in diesem Unterabschnitt dargestellt. Sie entsprechen der Übertragung der Regeln b) und c) aus Abbildung 5.7 auf elementare Dienstbeschreibungsnetze und sind in Tabelle 5.2 dargestellt.

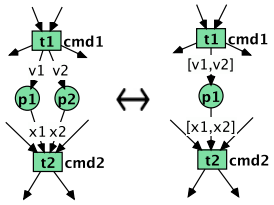
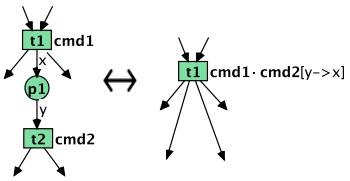
	Reduktionsregel	Anmerkungen
a		Sind Tupel oder Paare Teil der algebraischen Signatur, so lassen sich Stellen mit gleichem Vor- und Nachbereich zusammenfassen, indem die Variablen in einer Anschrift als Tupel dargestellt werden.
b		Die Variablenmengen der Anweisungsblöcke $\mathbf{cmd}_1$ und $\mathbf{cmd}_2$ müssen disjunkt sein. Sämtliche Vorkommen der Eingangsvariable y von t_2 werden in der Hintereinanderausführung $\mathbf{cmd}_1 \circ \mathbf{cmd}_2$ durch die Variable x ersetzt.

Tabelle 5.2: Reduktionsanweisungen für elementare Dienstbeschreibungsnetze

Die in Tabelle 5.2 dargestellten Regeln umfassen einmal die Reduktion zweier Datenstellen auf eine Stelle, sofern die Verwendung von Tupeln durch entsprechende Sorten in der Signatur gestattet ist (Regel a), und einmal die Reduktion der sequentiellen Ausführung zweier Transitionen zu einer Transition (Regel b). Im Fall der zweiten Reduktionsregel muss stets gewährleistet sein, dass nach jedem Schalten der ersten Transition auch die zweite Transition schaltet, was dadurch gegeben ist, dass

die Stelle p_1 der Regel nur eine Ein- und eine Ausgangskante besitzt und die Transition t_2 nur p_1 im Vorbereich besitzt. Aufgrund der Fortsetzbarkeit elementarer Dienstbeschreibungsnetze muss gewährleistet sein, dass die Aktivierungsbedingung von t_2 nach jedem Schalten von t_1 zu wahr auswertet, so dass stets t_2 nach dem Schalten von t_1 schaltet. Bei der Komposition der Anweisungsblöcke müssen dann die Eingangsvariablen von t_2 durch die entsprechenden Ausgangsvariablen von t_1 ersetzt werden, was als $\mathbf{cmd}_2[y \to x]$ notiert wird.

Dass die in Tabelle 5.2 wiedergegebenen Reduktionsregeln allesamt die Korrektheit des zugrundeliegenden Workflownetzes erhalten folgt direkt aus den Beobachtungen aus Unterabschnitt 5.3.3. Aufgrund der Elementarität des Netzes lassen sich die Regeln dann auch auf Dienstbeschreibungsnetze übertragen. Dabei sind die Ein- und Ausgangsvariablen der ersten und letzten Transition nicht in den Abbildungen in Tabelle 5.2 angegeben. Sie müssen als unverändert vor und nach dem Anwenden der Regel angenommen werden.

Regel a) in Tabelle 5.2 ist offensichtlich korrekt, da beide Transitionen die gleichen Ein- und Ausgangsvariablen besitzen und auch das Schaltverhalten des Netzes gleich bleibt.

Für Regel b) in Tabelle 5.2 müssen zwei Fälle unterschieden werden. Ist $t_1^\bullet = \{p_1\}$, so entspricht das Schalten von t_1 und t_2 der Hintereinanderausführung der beiden Anweisungsblöcke. Sind die Variablen in beiden Anweisungsblöcken disjunkt, so können sie direkt an einer Transition hintereinander ausgeführt werden, wobei die Eingangsvariablen der Transition t_2 durch die Ausgangsvariablen der Transition t_1 ersetzt werden müssen. (Bei mehreren Eingangsvariablen muss entsprechend jede separat ersetzt werden.) Da die weiteren Variablen der Anweisungsblöcke als disjunkt angenommen werden, was leicht durch eine Umbenennung erreicht werden kann, entspricht das Schalten von $t_1 t_2$ dann dem Schalten von t_1 mit dem Anweisungsblock $\mathbf{cmd}_1\mathbf{cmd}_2[y \to x]$, der genau der Hintereinanderausführung von $\mathbf{cmd}_1$ und $\mathbf{cmd}_2$ entspricht, wobei zuvor in $\mathbf{cmd}_2$ die Variablen entsprechend umbenannt werden mussten.

Der zweite Fall von Regel b) in Tabelle 5.2 entsteht, wenn $|t_1^\bullet| > 1$ ist und somit eine Transition nebenläufig aktiviert ist. Aufgrund der Elementarität des Netzes und der daraus resultierenden funktionalen Ausführbarkeit ist die Schaltreihenfolge zweier nebenläufig aktivierter Transitionen unerheblich, so dass angenommen werden kann, dass stets t_2 nach dem Schalten von t_1 schaltet. Dann können die Anweisungen beider Transitionen jedoch nach dem Umbenennen der Eingangsvariablen von t_2 zusammengefasst werden, so dass sich wiederum die Regel ergibt.

Satz 5.42 (Korrektheit der Reduktionsregeln)
Gegeben eine elementares Dienstbeschreibungsnetz $\mathcal{SDN}$. Die Anwendung der Regeln aus Tabelle 5.2 ändert die Menge der von einem initialen Netzzustand erreichbaren finalen Netzzustände von $\mathcal{SDN}$ nicht.

Beweis:
(a) Die Regel a) aus Tabelle 5.2 ändert weder die Eingangsvariablen noch das Verhalten des zugrundeliegenden Netzes. Somit bleibt jede Folge von Schaltmodi nach der Reduktion aktiviert, die vor der Reduktion aktiviert war und umgekehrt.
(b) Aufgrund der Elementarität und der daraus folgenden Fortsetzbarkeit muss t_2 stets in einem Schaltmodus aktivierbar sein, wenn t_1 geschaltet hat. Somit werden die Anweisungsblöcke $\mathbf{cmd}_1$ und $\mathbf{cmd}_2$ stets nacheinander ausgeführt. Da die Menge der Variablen disjunkt ist und da andere Transitionen des Netzes aufgrund der funktionalen Ausführbarkeit das Verhalten der Transition t_2 nicht beeinflussen, kann davon ausgegangen werden, das t_2 stets direkt nach t_1 schaltet, ohne dass eine weitere Transition schaltet. Dann können nach Umbenennung der Eingangsvariablen auch die beiden Anweisungsblöcke direkt hintereinander ausgeführt werden. □

Diese Reduktionsregeln erlauben es in einfachen Fällen bereits eine Reduktion des Dienstbeschreibungsnetzes vorzunehmen, die dann weiter analysiert werden kann. Durch die Entfernung von Nebenläufigkeit durch die Regeln aus Tabelle 5.2 ist es zudem möglich, schneller eine Sequentialisierung zu erlangen, auf deren Verwendung im nächsten Kapitel näher eingegangen wird.

5.5 Zusammenfassung

Dieses Kapitel hat elementare Dienstbeschreibungsnetze eingeführt. Diese Netzklasse reduziert die Ausdrucksmächtigkeit von Dienstbeschreibungsnetzen, erlaubt im Gegenzug jedoch eine schnellere Analyse der Netze. Hierzu werden für verschiedene Eigenschaften der Netze hinreichende Kriterien analysiert, die sich für endliche Free-Choice-Dienstbeschreibungsnetze in Polynomialzeit entscheiden lassen.

In Abschnitt 5.2 werden elementare Dienstbeschreibungsnetze in Definition 5.10 eingeführt. Diese Netzklasse ist aufgrund hinreichender Bedingungen für die Typisierungskonformität, für die funktionale Ausführbarkeit, für die Endlichkeit und für die Fortsetzbarkeit stets schwach korrekt, was in Satz 5.11 gezeigt wird. Die hinreichenden Bedingungen für die vier Eigenschaften werden in Abschnitt 5.2 nacheinander eingeführt. Ihre Überprüfung wird jedoch zunächst noch nicht diskutiert, da hierzu Eigenschaften des zugrundeliegenden Netzes überprüft werden müssen, wie die Korrektheit und die gleichzeitige Markierbarkeit von Stellen in einer erreichbaren Markierung.

Für Free-Choice-Workflownetze können diese Eigenschaften des zugrundeliegenden Netzes in Polynomialzeit überprüft werden, was in Abschnitt 5.3 gezeigt wird. Aufbauend auf den bekannten Ergebnissen für Free-Choice-Netze und Netze allgemein aus Abschnitt 3.4 werden in Abschnitt 5.3.1 zunächst Eigenschaften von Free-Choice-Netzen betrachtet. Zentral ist hier einmal die Tatsache, dass das Sub-

netz einer semi-postiven S- oder T-Invariante eines wohlgeformten Free-Choice-Netzes wiederum ein wohlgeformtes Free-Choice-Netz ist, wie dies in Satz 5.19 gezeigt wurde. Dies erlaubt die Einführung von Verfahren 5.23 zur Überprüfung der Wohlgeformtheit eines Netzes. Gleichzeitig lässt sich hierdurch eine Basis von S-Invarianten ermitteln, anhand derer dann wie in Unterabschnitt 5.3.2 dargestellt die Korrektheit eines Workflownetzes ermittelt werden kann oder anhand derer mittels der Kriterien aus Satz 5.34 die Erreichbarkeit von Markierungen ermittelt werden kann. Letztere Überprüfung wird für die hinreichende Bedingung der funktionalen Ausführbarkeit und für die hinreichende Typisierungskonformitätsbedingung benötigt. Reduktionsregeln erlauben es, das zu analysierende Netz zu verkleinern, wodurch sich die Analyse schneller durchführen lässt. Dieses Vorgehen wird in Unterabschnitt 5.3.3 dargestellt.

Die Anwendung der Ergebnisse führt in Abschnitt 5.4 zur Analyse endlicher Free-Choice-Dienstbeschreibungsnetze. Endliche Free-Choice-Dienstbeschreibungsnetze sind die in Unterabschnitt 5.4.1 eingeführten Dienstbeschreibungsnetze, die die hinreichende Endlichkeitsbedingung aus Definition 5.38 erfüllen und deren zugrundeliegendes Workflownetz die Free-Choice-Eigenschaft besitzt. Mittels der Ergebnisse aus Abschnitt 5.3 lässt sich für diese Netze die Elementarität in Polynomialzeit überprüfen. Dies wird in Unterabschnitt 5.4.2 gezeigt. Unterabschnitt 5.4.3 widmet sich abschließend der Möglichkeit, durch Reduktionsregeln Vereinfachungen des Netzes bereits direkt an einem elementaren Dienstbeschreibungsnetz vorzunehmen.

Die schwache Korrektheit liefert zwar ein wichtiges Kriterium dafür, ob ein Ablauf sinnvoll modelliert wurde, jedoch wird eine vollständige Analyse des Verhaltens eines Netzes erst durch die Überprüfung erreichbarer Zustände möglich. Hierzu lässt sich jedoch in den meisten Fällen durch eine Sequentialisierung von der Nebenläufigkeit im Netz abstrahieren, was im nächsten Kapitel dargestellt wird.

6 Analyse elementarer Dienstbeschreibungsnetze

Kapitel 5 betrachtete elementare Dienstbeschreibungsnetze und die Möglichkeit, schwache Korrektheit in ihnen zu analysieren. Häufig ist die schwache Korrektheit jedoch kein ausreichendes Kriterium, um zu gewährleisten, dass ein Dienst sich wie beim Entwurf intendiert verhält. Folglich ist eine weitere Analyse der Eigenschaften des Dienstes notwendig, die jedoch selbst für endliche Netze zu einem exponentiell großen Zustandsraum führen kann. Aus diesem Grund wird in diesem Kapitel untersucht, wie sich dieser Zustandsraum durch Sequentialisierung verkleinern lässt. Im Anschluss hieran erfolgt eine Analyse der Objektzustände der in einem Dienstbeschreibungsnetz verwendeten Objekte. Diese ermöglicht es zu überprüfen, ob ein Objekt in einem Dienst so genutzt wird, wie dies beim Entwurf des Objektes intendiert war.

6.1 Möglichkeiten der Analyse

Die Klasse der elementaren Dienstbeschreibungsnetze stellt einen Kompromiss zwischen Ausdrucksmächtigkeit und Analysierbarkeit dar. Für endliche Free-Choice-Dienstbeschreibungsnetze lässt sich die Elementarität dabei relativ einfach ermitteln, wie im letzten Kapitel dargestellt wurde. Dennoch ist für eine detailliertere Analyse des Verhaltens häufig die Betrachtung der erreichbaren Zustände notwendig. Durch Methoden der symbolischen Analyse (vgl. Jhala und Majumdar, 2009) lassen sich zwar auch Systeme mit unendlichen Mengen von Konstanten analysieren, jedoch besteht hierbei stets die Gefahr, dass durch die Nebenläufigkeit im System exponentiell viele Zustände betrachtet werden müssen. Besitzt bereits der Zustandsraum des ungefärbten Workflownetzes exponentiell viele Zustände, so kann der Zustandsraum des gefärbten Netzes offensichtlich nicht kleiner sein. Da bereits ein markierter Graph als Abschluss eines korrekten Workflownetzes mit 100 paarweise nebenläufig aktivierten Transition über 2^{100} erreichbare Zustände besitzt (es kann jeweils entweder der Vor- oder Nachbereich jeder Transition und es können die Stellen i und o markiert sein), liegt es nahe, dass eine erschöpfende Analyse des Zustandsraums in großen Systemen nicht mehr möglich ist.

Kann jedoch garantiert werden, dass jede Permutation von Schaltfolgen der 100 Transitionen des obigen Beispiels zum gleichen Ergebnis führt, so muss lediglich

eine Schaltfolge betrachtet werden. Statt über 2^{100} erreichbaren Zuständen sind dann nur noch etwas über 100 Zustände zu betrachten. Dies ist die Motivation für die Sequentialisierung von Dienstbeschreibungsnetzen, die für elementare Free-Choice-Dienstbeschreibungsnetze in diesem Kapitel betrachtet werden soll.

Ein anderer Aspekt liegt in der Betrachtung des Verhaltens von Objekten. Häufig soll für Objekte nicht jeder Zustandsübergang erlaubt sein. So mag es nicht gewollt sein, dass ein Auftrag den Zustand storniert oder bezahlt einnimmt, bevor er überhaupt geordert wurde. Um dies zu verhindern, werden Objekte in Anlehnung an Valk (1986, 1991, 1998) mit einem Zustand und Zustandsübergängen versehen. Abweichend zu Valk (1998) kann eine Zustandsänderung jedoch nur durch das Systemnetz, also das Dienstbeschreibungsnetz, veranlasst werden.

Sequentialisierung und ihre Anwendung

Die Sequentialisierung von Dienstbeschreibungsnetzen erlaubt es, die in einer nebenläufigen Ausführung mögliche Explosion des Zustandsraums einzudämmen (vgl. Valmari, 1996; Kristensen und Valmari, 1998). Für elementare Dienstbeschreibungsnetze sind die wichtigen Voraussetzungen der Typisierungskonformität und der funktionalen Ausführbarkeit gegeben, so dass sich Transitionen, die sich im zugrundeliegenden Netz sequentialisieren lassen, auch im Dienstbeschreibungsnetz sequentialisieren lassen.

Für elementare Dienstbeschreibungsnetze liegt es somit nahe, zunächst das zugrundeliegende Workflownetz zu betrachten. Hier schränkt sich die Betrachtung wiederum auf Free-Choice-Netze ein, da sich diese in Polynomialzeit untersuchen lassen und sie gleichzeitig eine hinreichende Ausdrucksstärke bieten. Die Sequentialisierung von Free-Choice-Workflownetzen wird in Abschnitt 6.2 dargestellt. Hierzu wird zunächst definiert, wann ein Workflownetz sequentialisierbar ist, bevor dann darauf eingegangen wird, wie sich ein Free-Choice-Workflownetz aufbauend auf den Ergebnissen des letzten Kapitels sequentialisieren lässt. Das Vorgehen zur Sequentialisierung wird zunächst in Verfahren 6.8 eingeführt und dann in den sich hieran anschließenden Unterabschnitten an einem Beispiel illustriert, bevor dann seine Korrektheit gezeigt wird.

Die Sequentialisierung der Free-Choice-Workflownetze in Abschnitt 6.2 bietet die Grundlage für die Sequentialisierung elementarer Free-Choice-Dienstbeschreibungsnetze in Abschnitt 6.3. Durch die funktionale Ausführbarkeit hat jede Permutation einer Schaltfolge funktional die gleichen Auswirkungen, so dass die Sequentialisierung des zugrundeliegenden Netzes auf das Dienstbeschreibungsnetz übertragen werden kann. Hierbei müssen jedoch auch zwischen Transitionen ausgetauschte Daten betrachtet werden. Aus diesem Grund kann ein Dienstbeschreibungsnetz nicht direkt in eine Zustandsmaschine überführt werden, sondern wird stattdessen in eine Quasi-Zustandsmaschine überführt, in der alle Transitionen von einer Zustandsmaschine überdeckt werden, in der aber weitere Stellen im Netz erlaubt sind. Diese

Quasi-Zustandsmaschine stellt ein Dienstbeschreibungsnetz ohne nebenläufig aktivierbare Transitionen dar.

Der Einsatz der Sequentialisierung wird dann in Unterabschnitt 6.3.3 diskutiert. Hier sind vor allem die Analyse des (symbolischen) Zustandsraums und Testszenarien zu nennen. Zudem ist es häufig zur Auswahl passender Dienste bei der Modellierung oder beim Durchlauf von Test-Szenarien entscheidend zu wissen, wie sich ein Dienst bei bestimmten Eingaben verhalten kann. Ist ein Dienstbeschreibungsnetz korrekt, so ist sichergestellt, dass der entsprechende Dienst zumindest einen Wert zurückliefert. Es ist jedoch möglich, dass das Verhalten des Dienstes nicht mit dem gewünschten Verhalten übereinstimmt. Das funktionale Verhalten eines Dienstbeschreibungsnetzes kann durch die Sequentialisierung in verschiedenen Szenarien getestet werden, ohne dass gleiche Zustände, die durch unterschiedliche Reihenfolgen nebenläufiger Transitionen entstehen, mehrfach berechnet werden.

Das sequentialisierte Netz lässt sich dann zum Testen oder zur Analyse erreichbarer Zustände verwenden. Eine Möglichkeit der Analyse bietet die Überführung eines elementaren Dienstbeschreibungsnetzes in ein algebraisches Netz. Dies wird im Anschluss an die Sequentialisierung kurz skizziert. Der Zustand wird in diesem Fall durch eine Konstante der Signatur repräsentiert, auf der entsprechende Operatoren definiert sind, so dass ein Dienstbeschreibungsnetz in ein klassisches algebraisches Netz überführt werden kann.

Verwandt ist die Sequentialisierung mit der Reduktion des Zustandsraums im Bereich des Model Checking durch Partial-Order Reduction, wie dies etwa von Clarke Jr. u. a. (2001) oder Godefroid und Wolper (1994) erläutert wird. Hierbei wird der Zustandsraum einer Kripke-Struktur reduziert, indem nebenläufige Vorgänge in Äquivalenzklassen mit gleichem Verhalten gruppiert werden, die dann durch eine geringere Zahl von Zuständen repräsentiert werden können. Eine Übersicht verschiedener Ansätze zur Partial-Order Reduction findet sich bei Willems und Wolper (1996).

Im Bereich der Petrinetze finden die auf Valmari (1989) zurückgehenden sturen Mengen (engl. stubborn sets) Verwendung zur Reduktion des Zustandsraumes. Diese wurden von Schmidt (1999, 2000) und von Kristensen und Valmari (2000) näher erforscht und etwa in dem Petrinetz-Analysewerkzeug LoLa implementiert (vgl. Schmidt, 2003). Die Sweep-Line-Methode zur Exploration des Zustandsraums (vgl. Christensen u. a., 2001) nutzt hingegen eine Analyse der Zustände zur Laufzeit und entscheidet, ob diese wieder auftreten können. Diese Methode wird von Mitchell u. a. (2007) auch zur Analyse gefärbter Workflows verwendet.

Objektzustände und Dienstinteraktion

Neben den Abläufen besitzen auch Objekte Zustände, die über die Rollen, die sie annehmen können, modelliert werden können. Auch bei diesen Zuständen gibt es erlaubte und nicht gestattete Übergänge im Modell. Soll etwa eine Konsistenzkon-

trolle eines Auftragseingangs stattfinden, so ist es nicht sinnvoll, dass diese stattfindet, nachdem der Auftrag ausgeliefert wurde. Ein Auftragsobjekt würde also einen Zustandsübergang von *konsistent* zu *ausgeliefert* erlauben, nicht jedoch einen Übergang in die andere Richtung.

Die zugrundeliegenden Konzepte dieses Ansatzes bauen auf den elementaren Objektnetzen (Elementary Object Systems) von Valk (1998) auf. Sie werden genutzt, um die Zustände eines Objektes zu modellieren. Im Unterschied zu Valk (1998) besitzt das Objektnetz jedoch keinerlei eigenes Verhalten. Vielmehr können Zustandsübergänge nur extern durch ein Systemnetz, das Workflownetz, erfolgen. Dieses Vorgehen erlaubt es, das beide Netze über die Transitionen miteinander zu verschmelzen, so dass sich die Analyse wiederum auf ein Workflownetz beschränkt. Dieses Vorgehen wird in Abschnitt 6.4 näher erläutert.

Vorgehen und Ergebnisse

Die wesentlichen Ergebnisse dieses Kapitels sind die Sequentialisierung elementarer Free-Choice-Dienstbeschreibungsnetze und die Betrachtung von Objektzuständen. Die Sequentialisierung eines Free-Choice-Workflownetzes erfolgt über das in Unterabschnitt 6.2.2 angegebene Verfahren 6.8. Dieses Verfahren wird in Unterabschnitt 6.3.1 in Verfahren 6.16 erweitert, so dass die Stellen im Vor- und Nachbereich einer Transition erhalten bleiben. Dies ermöglicht die Übertragung von Daten in einem Dienstbeschreibungsnetz in der Sequentialisierung zu übernehmen, so dass die in Definition 6.18 eingeführte Sequentialisierung eines Dienstbeschreibungsnetzes hierauf aufbaut.

In Abschnitt 6.4 wird die Überprüfung der Einhaltung von Objektzuständen durch Abläufe diskutiert. Zustandsübergänge von Objekten ermöglichen die Analyse, ob ein Ablauf dem Verhalten des Objektes entsprechend modelliert ist, was häufig ein guter Indikator für fehlerfreies Verhalten in einem Ablauf ist. Zudem wird erläutert, wie sich Dienstbeschreibungsnetze dazu verwenden lassen, komplexere Interaktionen durch die Verfeinerung von Workflows, die durch Dienstbeschreibungsnetze dargestellt werden, zu modellieren. Die Verschmelzung eines Objektnetzes mit einem Dienstbeschreibungsnetz ist in Definition 6.23 gegeben. Die sich hieraus ergebenden Eigenschaften sind in Satz 6.24 wiedergegeben. Wie verschiedene Fehlerquellen zu interpretieren sind, diskutiert dann Unterabschnitt 6.4.2.

6.2 Sequentialisierung von Free-Choice-Workflownetzen

Die in Modellen notwendige Nebenläufigkeit in Abläufen führt bei der Analyse der Netze häufig zu einem erhöhten Aufwand, der für die Analyse des funktionalen Verhaltens eines Dienstbeschreibungsnetzes jedoch nicht notwendig ist. Um diesen

erhöhten Aufwand zu vermeiden, werden Dienstbeschreibungsnetze sequentialisiert, indem die zugrundeliegenden Workflownetze in eine erweiterte Zustandsmaschine überführt werden. Dies wird als Sequentialisierung der Workflownetze bezeichnet.

Dieser Abschnitt betrachtet zunächst die Sequentialisierung von Workflownetze, so dass die Ergebnisse dieses Abschnitts im nächsten Abschnitt auf elementare Dienstbeschreibungsnetze übertragen werden können. Hierzu wird zunächst dargestellt, was Sequentialisierbarkeit bedeutet und wie eine Sequentialisierung erstellt werden kann. Hieran schließt sich eine Betrachtung der Sequentialisierung durch die im letzten Kapitel eingeführten Reduktionsregeln an.

6.2.1 Sequentialisierbarkeit

Bei Netzen mit algebraischen Strukturen führt die durch das nebenläufige Schalten von Transitionen hervorgerufene mögliche Zustandsraumsexplosion sehr schnell zu Netzen, deren Verhalten praktisch nicht mehr analysiert werden kann. Aus diesem Grund stellt die Sequentialisierbarkeit eine wünschenswerte Eigenschaft elementarer Dienstbeschreibungsnetze dar. Sie erlaubt es, die Analyse an einer Zustandsmaschine durchzuführen.

In dem hier verfolgten Ansatz soll die Reduktion möglicher Transitionsübergänge nicht durch Betrachtung des Zustandsraums erfolgen, sondern vielmehr direkt im Netz durchgeführt werden. Dabei wird in Kauf genommen, dass sich die Sprache eines Netzes durch die Sequentialisierung reduziert. Wichtig ist lediglich, dass die Sprache sämtlicher minimaler T-Invarianten in der Reduktion erhalten bleibt. Wie üblich ist hierzu zu einem Petrinetz eine Sprache definiert, indem jeder Transition eine Beschriftung zugewiesen wird. Eine detailliertere Einführung hierzu findet sich beispielsweise bei Priese und Wimmel (2008) in Kapitel 6.

Definition 6.1 (Beschriftetes Netz, Alphabet, Beschriftungsfunktion)
Ein Petrinetz $\mathcal{N} = (P, T, F)$ zusammen mit einem *Alphabet* $\mathbb{A}$ und einer *Beschriftungsfunktion* $\mathbf{l}_\mathcal{N} : T \to \mathbb{A} \cup \{\epsilon\}$ wird *beschriftetes Netz* $\mathcal{N}_\mathbf{l} = (\mathcal{N}, \mathbb{A}, \mathbf{l}_\mathcal{N})$ genannt, wobei $\epsilon \notin \mathbb{A}$ das *leere Wort* ist. ⧫

Wie üblich wird die Abbildung $\mathbf{l}_\mathcal{N}$ zu einem Homomorphismus $\mathbf{l}_\mathcal{N} : T^* \to \mathbb{A}^*$ erweitert, so dass gilt

- $\mathbf{l}_\mathcal{N}(\epsilon) = \epsilon$ und
- $\mathbf{l}_\mathcal{N}(\sigma t) = \mathbf{l}_\mathcal{N}(\sigma)\mathbf{l}_\mathcal{N}(t)$.

Die Folge des Alphabets $\mathbf{l}_\mathcal{N}(\sigma) \in \mathbb{A}^*$ wird als das zu der Schaltfolge σ korrespondierende Wort bezeichnet.

Ein korrektes Workflownetz wird nun durch eine Zustandsmaschine sequentialisiert, deren Transitionen die Transitionen des Workflownetzes dergestalt zugewiesen

werden, dass zu jeder Schaltfolge des Workflownetzes eine Schaltfolge der Zustandsmaschine existiert, die auf die Transitionen des Workflownetzes abbildet. Gleichzeitig muss jede Schaltfolge der Zustandsmaschine auf eine entsprechende Schaltfolge im Workflownetz abbildbar sein.

Dies lässt sich formalisieren, wenn man Permutationen von Schaltfolgen betrachtet. Zwei Schaltfolgen σ_1 und σ_2 eines Netzes $\mathcal{N}$ sind Permutationen voneinander, wenn durch Austausch nebenläufig aktivierter Transitionen die erste Schaltfolge in die zweite übergehen kann oder umgekehrt. Zu einer Schaltfolge in einem Workflownetz gibt es dann stets eine Permutation, so dass diese der Abbildung einer Schaltfolge der Sequentialisierung entspricht. Die folgende Definition formalisiert den Begriff der nebenläufigen Permutation.

Definition 6.2 (Nebenläufig permutierbare Schaltfolgen)
Gegeben ein korrektes Workflownetz $\mathcal{N}$ und zwei Schaltfolgen σ_1 und σ_2 von $[i]$ nach $[o]$. Die Schaltfolgen σ_1 und σ_2 sind *einfach nebenläufig permutierbar* gdw. $\sigma_1 = \sigma_a t_1 t_2 \sigma_b$ und $\sigma_2 = \sigma_a t_2 t_1 \sigma_b$ und für $[i] \xrightarrow{\sigma_a} \mathbf{m}$ gilt, dass $\mathbf{m} \xrightarrow{\{t_1,t_2\}}$ ist.
Sei R_σ die Relation der einfach nebenläufig permutierbaren Schaltfolgen, so dass $(\sigma_a, \sigma_b) \in R_\sigma$ für zwei Schaltfolgen σ_a und σ_b von $[i]$ nach $[o]$ gdw. σ_a und σ_b einfach nebenläufig permutierbar sind. Sei $\overset{\sigma}{\simeq} \overset{\text{def}}{=} R_\sigma^*$ die transitive und reflexive Hülle der Relation R_σ, die eine Äquivalenzrelation darstellt[1]. Zu jeder Schaltfolge von $[i]$ nach $[o]$ gibt es dann eine Äquivalenzklasse bzgl. $\overset{\sigma}{\simeq}$, die als $[\sigma]_{\overset{\sigma}{\simeq}}$ notiert wird.
Die Schaltfolgen σ_1 und σ_2 sind *nebenläufig permutierbar* gdw. sie in der gleichen Äquivalenzklasse bzgl. $\overset{\sigma}{\simeq}$ sind, also $\sigma_1 \in [\sigma_2]_{\overset{\sigma}{\simeq}}$ gilt. ♦

Es lässt sich leicht überprüfen, dass durch die Permutation der Parikh-Vektor einer Schaltfolge nicht verändert wird. Somit gilt für alle $\sigma' \in [\sigma]_{\overset{\sigma}{\simeq}}$, dass $\vec{\sigma'} = \vec{\sigma}$ ist. Eine Sequentialisierung ist eine Zustandsmaschine, die die gleichen Schaltfolgen der Äquivalenzklassen eliminiert, gleichzeitig aber zu jeder Schaltfolge einer Äquivalenzklasse des Ursprungsnetzes eine Schaltfolge besitzt, die auf diese Äquivalenzklasse abgebildet wird. Dies stellt sich formaler wie folgt dar.

Definition 6.3 (Sequentialisierung korrekter Workflownetze)
Gegeben ein korrektes und sicheres Workflownetz $\mathcal{N} = (P, T, F)$ und ein beschriftetes Netz $\mathcal{N}_\mathrm{l}^S = (\mathcal{N}^S, T, \mathbf{l}_{\mathcal{N}^S})$ mit disjunkten Mengen von Stellen und Transitionen, wobei T das Alphabet von $\mathcal{N}_\mathrm{l}^S$ ist. Es ist $\mathcal{N}_\mathrm{l}^S$ eine *Sequentialisierung* von $\mathcal{N}$ gdw.

- $\mathcal{N}^S$ ein korrektes Workflownetz und eine Zustandsmaschine ist,

[1] Offensichtlich ist die transitive reflexive Hülle transitiv und reflexiv. Die Symmetrie folgt aus der Permutationseigenschaft benachbarter Transitionen.

- für jede Schaltfolge σ mit $[i] \xrightarrow[\mathcal{N}]{\sigma} [o]$ in $\mathcal{N}$ eine Schaltfolge σ_S mit $[i^S] \xrightarrow[\mathcal{N}^S]{\sigma_S} [o^S]$ in $\mathcal{N}^S$ existiert und $\mathbf{l}_{\mathcal{N}^S}(\sigma_S) \in [\sigma]_{\underline{\sim}}$ ist,
- für jede Schaltfolge σ_S mit $[i^S] \xrightarrow[\mathcal{N}^S]{\sigma_S} [o^S]$ in $\mathcal{N}^S$ eine Schaltfolge σ mit $[i] \xrightarrow[\mathcal{N}]{\sigma} [o]$ in $\mathcal{N}$ existiert und $\mathbf{l}_{\mathcal{N}^S}(\sigma_S) = \sigma$ ist.

⧫

Vereinfacht lässt sich für die Sequentialisierung sagen, dass zu jeder in $(\mathcal{N}, [i])$ aktivierten Schaltfolge eine nebenläufige Permutation existiert, die in $(\mathcal{N}^S, [i^S])$ aktiviert ist, und dass jede in $(\mathcal{N}^S, [i^S])$ aktivierte Schaltfolge σ_S auch in $(\mathcal{N}, [i])$ aktiviert ist. Hierbei wurde die bijektive Abbildung vernachlässigt.

Die Definition der Sequentialisierung wird auf den Abschluss eines Workflownetzes übertragen, indem unter Einbeziehung der Transition t^* Schaltsequenzen von $[i]$ nach $[i]$ betrachtet werden.

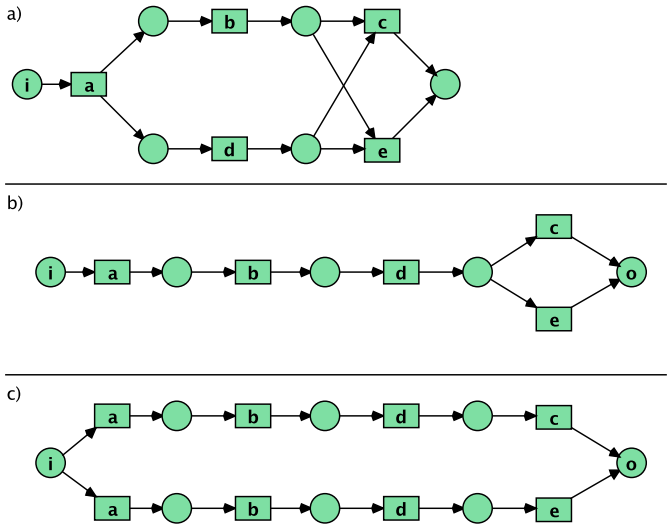


Abbildung 6.1: Ein Workflownetz und zwei mögliche Sequentialisierungen

Abbildung 6.1b) und Abbildung 6.1c) stellen zwei Sequentialisierungen des Workflownetzes in Abbildung 6.1a) dar, wobei $\mathbf{l}_{\mathcal{N}^S}$ für Abbildung 6.1b) eine Bijektion ist, was für die meisten praktischen Fälle sinnvoll ist, da dann die Transitionen, die zur Erreichung eines Wortes schalten müssen, aufgrund der Schaltfolge in $\mathcal{N}^S$ identifiziert werden können. Die Sequentialisierung in Abbildung 6.1c) verwendet hingegen mehr Transitionen als das Ursprungsnetz in Abbildung 6.1a).

Im Ursprungsnetz in Abbildung 6.1a) existieren vier Schaltfolgen von $[i]$ nach $[o]$, die entsprechend zu den vier Worten *abde*, *adbe*, *abdc* und *adbc* führen, wobei

die ersten beiden und die letzten beiden jeweils Permutationen voneinander sind. Entsprechend gibt es in den Sequentialisierungen in Abbildung 6.1b) und Abbildung 6.1c) Schaltfolgen mit den Worten *abdc* und *abde*. Umgekehrt gibt es zu jedem Wort einer Schaltfolge in Abbildung 6.1b) und Abbildung 6.1c) auch eine entsprechende Schaltfolge in Abbildung 6.1a). Das Netz in Abbildung 6.1c) ist hierbei jedoch offensichtlich nicht minimal und besitzt unnötige Transitionen.

Das Hinzufügen zusätzlicher Transitionen lässt sich bei der Sequentialisierung jedoch nicht immer vermeiden. Andernfalls könnte die Menge der Transitionen und somit auch die Abbildung $\mathbf{l}_{\mathcal{N}^S}$ stets eine Bijektion sein, so dass sich lediglich die Zahl der Stellen reduziert. Dass dies im Allgemeinen nicht immer möglich ist, zeigt das Beispiel des Netzes in Abbildung 6.2. Zur Sequentialisierung muss hier eine Transition verdoppelt werden, da es in dem dargestellten Netz eine Schaltfolge gibt, die nur t_4 umfasst, eine die nur t_6 umfasst und eine die beide Transitionen umfasst. Folglich muss eine der Transitionen in einer Sequentialisierung gedoppelt werden, oder es muss eine zusätzliche Transition eingeführt werden, die in der Zustandsmaschine auf das leere Wort ϵ abbildet. Hierdurch könnte die Transition t_4 oder die Transition t_6 „überbrückt“ werden, wodurch sich jedoch auch die Zahl möglicher Invarianten beziehungsweise deren Worte ändert. In obigem Beispiel wäre dann beispielsweise eine Schaltfolge ohne t_4 und ohne t_6 möglich, so dass das Netz hierdurch neue Invarianten erhält. Um dies zu verhindern, ist der Weg der Verdopplung vorzuziehen.

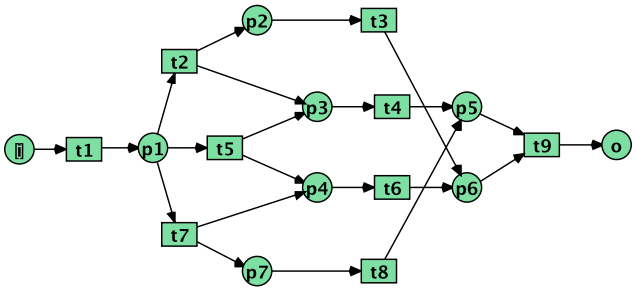


Abbildung 6.2: Das dargestellte Netz kann nicht ohne die Verdopplung der Transition t_4 oder der Transition t_6 sequentialisiert werden

Der Nachteil bei der Verdopplung von Transitionen ist, dass die Größe der Zustandsmaschine, die die Sequentialisierung darstellt, exponentiell wachsen kann, wenn das Netz besonders ungünstig aufgebaut ist. Würde man im Beispiel aus Abbildung 6.2 die Transitionen t_4 und t_6 wiederum durch Netze ersetzen, die wie das Netz in Abbildung 6.2 aufgebaut sind, so müssten bereits ganze Teilnetze verdoppelt werden und eine der Transitionen gar viermal im Netz vorhanden sein. Dies lässt sich weiter betreiben, so dass ein Netz mit exponentiell vielen Knoten entsteht.

Die folgende Definition differenziert Workflownetze anhand der Möglichkeit zur

Sequentialisierung. Ist eine Sequentialisierung eines Workflownetzes direkt durch eine Zustandsmaschine ohne die Verdopplung von Transitionen möglich, so wird ein Workflownetz als stark sequentialisierbar bezeichnet. Schwach sequentialisierbare Workflownetze sind hingegen lediglich durch Verdopplung von Transitionen durch eine Zustandsmaschine sequentialsierbar.

Definition 6.4 (Starke und schwache Sequentialisierbarkeit, Minimalität)
Ein korrektes Workflownetz $\mathcal{N}$ ist *stark sequentialisierbar* gdw. es eine Sequentialisierung von $\mathcal{N}$ durch ein beschriftetes Workflownetz $\mathcal{N}_{\mathrm{l}}^S = (\mathcal{N}^S, T, \mathbf{l}_{\mathcal{N}_{\mathrm{l}}^S})$ gibt, so dass $\mathbf{l}_{\mathcal{N}_{\mathrm{l}}^S}$ eine Bijektion ist.
Gibt es kein Netz $\mathcal{N}_{\mathrm{l}}^S$ mit einer solchen Bijektion, so ist $\mathcal{N}$ *schwach sequentialisierbar*.
Eine Sequentialisierung $\mathcal{N}_{\mathrm{l}}^S$ von $\mathcal{N}$ ist *minimal* gdw. es keine Sequentialisierung gibt, die weniger Transitionen als $\mathcal{N}_{\mathrm{l}}^S$ besitzt. ♦

Da jeder Transition der Sequentialisierung durch $\mathbf{l}_{\mathcal{N}_{\mathrm{l}}^S}$ maximal eine Transition zugewiesen wird, besitzt eine Sequentialisierung mindestens genauso viele Transitionen wie das ursprüngliche Workflownetz. Stark sequentialisierbare Netze besitzen somit stets eine minimale Sequentialisierung, deren Transitionsmenge die gleiche Kardinalität besitzt, wie die Transitionsmenge des Ursprungsnetzes, so dass $\mathbf{l}_{\mathcal{N}_{\mathrm{l}}^S}$ eine Bijektion ist.

Ist ein Workflownetz bereits eine Zustandsmaschine, so sollte sich die Netztopologie durch eine minimale Sequentialisierung nicht ändern. Dass dies tatsächlich der Fall ist, zeigt folgendes Lemma.

Lemma 6.5 (Sequentialisierung von Zustandsmaschinen)
Gegeben ein korrektes und sicheres Workflownetz $\mathcal{N} = (P, T, F)$, das eine Zustandsmaschine ist, und eine Sequentialisierung $\mathcal{N}_{\mathrm{l}}^S = (\mathcal{N}^S, T, \mathbf{l}_{\mathcal{N}^S})$ von $\mathcal{N}$. Ist $\mathcal{N}_{\mathrm{l}}^S$ minimal, so sind $\mathcal{N}$ und $\mathcal{N}^S$ isomorph.

Beweis:
Offensichtlich ist $\mathcal{N}$ stark sequentialisierbar, und es ist $\mathbf{l}_{\mathcal{N}^S}$ eine Bijektion. Da es in $\mathcal{N}$ keine einfach permutierbaren Schaltfolgen gibt, besteht die Äquivalenzklasse $[\sigma]_{\underset{\sim}{\sigma}}$ jeder Schaltfolge σ aus der einelementigen Menge $\{\sigma\}$. Vernachlässigt man der Einfachheit halber die bijektive Abbildung $\mathbf{l}_{\mathcal{N}^S}$ und setzt $T = T^S$ und $\mathbf{l}_{\mathcal{N}^S}(t) = t$, so muss für jede Schaltfolge σ gelten $[i] \xrightarrow[\mathcal{N}]{\sigma} [o]$ gdw. $[i^S] \xrightarrow[\mathcal{N}^S]{\sigma} [o^S]$. Würde eine Basis minimaler T-Invarianten in einem Netz nicht gleichzeitig Basis minimaler T-Invarianten im anderen Netz sein, so ließe sich eine minimale T-Invariante unendlich oft realisieren, die nicht im zweiten Netz vorhanden ist. (Das Subnetz jeder minimalen T-Invariante in einer Zustandsmaschine ist ein markierter Graph mit nur

einem Zyklus (vgl. Lemma 3.67), der in einem korrekten Workflownetz markierbar ist (vgl. Satz 3.66).) Somit müssen die möglichen Basen der T-Invarianten ebenfalls gleich sein und dann nach Lemma 5.13 auch die Kardinalitäten der Stellenmengen gleich sein. Wäre F verschieden, gäbe es in beiden Netzen eine Transition mit unterschiedlichem Vorbereich. Da jede Transition einer Zustandsmaschine nur eine Stelle im Vorbereich besitzt, könnte diese Transition dann in beiden Netzen nach unterschiedlichen Schaltfolgen feuern. Da beide Netze korrekt sind, widerspricht dies der Forderung $[i] \xrightarrow[\mathcal{N}]{\sigma} [o]$ gdw. $[i^S] \xrightarrow[\mathcal{N}^S]{\sigma} [o^S]$. □

Für korrekte Free-Choice-Workflownetze lassen sich die Kriterien der Sequentialisierung über die T-Invarianten formulieren, was den Nachweis, dass ein Netz eine Sequentialisierung ist, einfacher gestaltet. Dies wird durch die erste Bedingung in folgender Definition ausgedrückt. Die zweite Bedingung besagt, dass für jede Schaltfolge der Sequentialisierung auch eine Schaltfolge im Originalnetz existieren muss, auf die erstere Schaltfolge abbildbar ist. Dabei werden nur Permutationen von Schaltfolgen betrachtet, die in beiden Netzen in der Markierung $[i]$ aktiviert sind.

Lemma 6.6 (Sequentialisierung stark sequentialisierbarer FCWF-Netze) Gegeben ein korrektes stark sequentialisierbares Free-Choice-Workflownetz $\mathcal{N}$ mit Abschluss $\overline{\mathcal{N}}$ und ein beschriftetes Netz $\mathcal{N}_{\mathrm{l}}^S = (\mathcal{N}^S, \overline{T}, \mathbf{l}_{\mathcal{N}^S})$ mit disjunkten Mengen von Stellen und Transitionen. Es ist $\mathcal{N}_{\mathrm{l}}^S$ eine minimale Sequentialisierung von $\overline{\mathcal{N}}$ gdw.

- $\mathcal{N}^S$ der Abschluss eines korrekten Workflownetzes und eine Zustandsmaschine ist und $|T| = |T^S|$ gilt,
- zu einer Basis minimaler T-Invarianten $\mathbf{B}_T(\overline{\mathcal{N}})$ von $\overline{\mathcal{N}}$ eine Basis minimaler T-Invarianten $\mathbf{B}_T(\mathcal{N}^S)$ von $\mathcal{N}^S$ existiert, so dass $\vec{\iota} \in \mathbf{B}_T(\mathcal{N}^S)$ gdw. $\mathbf{l}_{\mathcal{N}^S}(\vec{\iota}) \in \mathbf{B}_T(\overline{\mathcal{N}})$, wobei $\mathbf{l}_{\mathcal{N}^S}(\vec{\iota})$ die komponentenweise Anwendung der Abbildung $\mathbf{l}_{\mathcal{N}^S}(\cdot)$ auf die Elemente des Vektors ist.
- für jede Schaltfolge σ_S mit $[i^S] \xrightarrow[\mathcal{N}^S]{\sigma_S}$ eine Schaltfolge σ in $\mathcal{N}$ mit $[i] \xrightarrow[\mathcal{N}]{\sigma}$ derart existiert, dass $\sigma = \mathbf{l}_{\mathcal{N}^S}(\sigma_S)$.

Beweis:
($\Rightarrow$) Dass $\mathcal{N}^S$ der Abschluss eines korrekten Workflownetzes und eine Zustandsmaschine ist und $|T| = |T^S|$ gilt, folgt direkt aus der Definition der Sequentialisierbarkeit und der Minimalität in Definition 6.3 und Definition 6.4. In einer Zustandsmaschine ist nach Lemma 3.67 jeder Zyklus das Subnetz einer T-Invariante. Gleichzeitig ist eine wohlgeformte Zustandsmaschine ein wohlgeformtes Free-Choice-Netz, so dass jede minimale T-Invariante eine T-Komponente ist (vgl. Lemma 5.14), die

wiederum lebendig ist, wenn alle Zyklen von ihr markiert sind, wie in Satz 3.66 gezeigt wurde. Da $(\mathcal{N}^S, [i^S])$ lebendig und sicher ist, kann jede Stelle markiert werden und damit auch jede minimale T-Invariante von $\mathcal{N}^S$ in einer erreichbaren Markierung realisiert werden. Folglich muss für jede T-Invariante $\vec{\iota} \in \mathbf{B}_T(\mathcal{N}^S)$ auch $\mathbf{l}_{\mathcal{N}^S}(\vec{\iota}) \in \mathbf{B}_T(\overline{\mathcal{N}})$ sein. Würde der umgekehrte Fall für ein $\vec{\iota} \in \mathbf{B}_T(\overline{\mathcal{N}})$ nicht gelten, so dürfte diese T-Invariante nicht realisierbar sein. Wählt man eine Transition $t \in \|\vec{\iota}\|$ und entfernt alle von $\vec{\iota}$ verschiedenen T-Invarianten aus $\mathbf{B}_T(\overline{\mathcal{N}})$, die t im Träger besitzen, so dass das Netz zusammenhängend bleibt, so wäre das Netz dann jedoch nicht mehr wohlgeformt, was Satz 5.19 widersprechen würde. Da $[i^S] \xrightarrow[\mathcal{N}^S]{\sigma_S\sigma'_S} [o^S]$ in einem korrekten Workflownetz für eine Schaltfolge σ'_S gilt, folgt die letzte Bedingung aus Definition 6.3.
($\Leftarrow$) Die erste Bedingung aus Definition 6.3 und die Bedingung der Minimalität aus Definition 6.4 folgt direkt aus der ersten Bedingung von Lemma 6.6. Die dritte Bedingung aus Definition 6.3 folgt direkt aus der dritten Bedingung von Lemma 6.6. Somit bleibt zu zeigen, dass für jede Schaltfolge σ mit $[i] \xrightarrow[\mathcal{N}]{\sigma} [o]$ in $\mathcal{N}$ eine Schaltfolge σ_S mit $[i^S] \xrightarrow[\mathcal{N}^S]{\sigma_S} [o^S]$ in $\mathcal{N}^S$ existiert und $\mathbf{l}_{\mathcal{N}^S}(\sigma_S) \in [\sigma]_{\widetilde{\sigma}}$ ist. Aufgrund der Überlegungen in ($\Rightarrow$) und aufgrund der zweiten Bedingung existiert eine Schaltfolge σ_S mit $[i^S] \xrightarrow[\mathcal{N}^S]{\sigma_S} [o^S]$ in $\mathcal{N}^S$, für die $\overrightarrow{\mathbf{l}_{\mathcal{N}^S}(\sigma_S)} = \vec{\sigma}$ ist. Wäre nicht $\mathbf{l}_{\mathcal{N}^S}(\sigma_S) = \sigma$, so gäbe es eine Teilschaltfolge σ'_S von σ_S, so dass in $\mathbf{m}$ mit $[i] \xrightarrow[\mathcal{N}]{\mathbf{l}_{\mathcal{N}^S}(\sigma'_S)} \mathbf{m}$ die Abbildung der in σ_S folgenden Transition nicht aktivierbar wäre. Dann wäre jedoch Bedingung 3 von Lemma 6.6 verletzt. □

Da die Verwendung von stark sequentialisierbaren Workflownetzen in der Regel vollkommen ausreichend ist, werden im Folgenden nur stark sequentialisierbare Workflownetze betrachtet. Zum Teil lassen sich jedoch auch Netze wie das in Abbildung 6.2 dargestellte sequentialisieren, indem sie durch die Reduktionsregeln aus Unterabschnitt 5.3.3 reduziert werden, was in Unterabschnitt 6.2.3 noch einmal aufgegriffen wird.

6.2.2 Vorgehen zur Sequentialisierung

Zur Sequentialisierung eines Free-Choice-Workflownetzes wird zunächst ein Verfahren angegeben, das anschließend analysiert wird. Das Verfahren ist korrekt, da es jedes stark sequentialisierbare korrekte Free-Choice-Workflownetz in eine Zustandsmaschine überführt, die die gleiche Anzahl von Transitionen besitzt, wie das Originalnetz. Dies wird im Anschluss an die beispielhafte Anwendung des Verfahrens gezeigt.

Verfahren zur Sequentialisierung

Für korrekte Workflownetze mit der Free-Choice-Eigenschaft lässt sich eine Sequentialisierung durch Betrachtung der minimalen T-Invarianten einer Basis konstruieren. Hierzu werden die T-Invarianten nacheinander realisiert und entsprechend in einer Zustandsmaschine angeordnet. Bei der Realisierung ist darauf zu achten, dass durch das zu frühe Feuern einer Transition unter Umständen die Möglichkeit verloren geht, dass eine Transition durch eine andere minimale T-Invariante „wiederverwendet“ wird. Aus diesem Grund wird das Konzept der verzögernden Stellen eingeführt. Transitionen im Nachbereich solcher Stellen können nur feuern, wenn keine Transition aktiviert ist, deren Stellen im Vorbereich nicht verzögernd sind.

Definition 6.7 (Verzögernde Stelle, verzögernde Realisierung)
Gegeben ein sicheres und korrektes Workflownetz $\mathcal{N}$. Eine Stelle $p \in P$ ist *verzögernd* gdw. $|{}^\bullet p| \geq 2$. Die Menge aller verzögernden Stellen wird als ${}^{\boxdot}P \stackrel{\text{def}}{=} \{p \in P \mid |{}^\bullet p| \geq 2\}$ notiert.
Eine Realisierung σ eines T-Vektors $\vec{\iota}$ ($\vec{\sigma} = \vec{\iota}$) heißt *verzögernd* in der Markierung $\mathbf{m}_0$ gdw. für beliebige $\sigma_1, \sigma_2 \in T^*$, $t \in T$ mit $\sigma = \sigma_1 t \sigma_2$ und $\mathbf{m}_0 \xrightarrow[\mathcal{N}]{\sigma_1} \mathbf{m}$ gilt:

$$ {}^\bullet t \cap {}^{\boxdot}P = \emptyset \text{ oder } \forall t' \in T : {}^\bullet t' \subseteq \|\mathbf{m}\| \text{ impliziert } {}^\bullet t' \cap {}^{\boxdot}P \neq \emptyset. $$

(Besitzt eine in $\mathbf{m}$ aktivierte Transition eine verzögernde Stelle im Vorbereich, so feuert diese Transition nur, wenn alle anderen aktivierten Transitionen ebenfalls eine verzögernde Stelle im Vorbereich aufweisen.) ⧫

In einer verzögernden Realisierung schaltet eine Transition mit einer verzögernden Stelle im Vorbereich also erst, wenn es keine Transition gibt, die in dieser Markierung schalten kann, ohne eine verzögernde Stelle im Vorbereich zu haben. Dies soll an einem Beispiel verdeutlicht werden. Abbildung 6.3 stellt ein einfaches Workflownetz dar, das korrekt ist und das die Free-Choice-Eigenschaft besitzt. Eine verzögernde Realisierung der T-Vektoren $[t_1, t_2, t_3, t_4, t_7]$ und $[t_1, t_4, t_5, t_6, t_7]$ würde im ersten Fall zunächst die Transitionen t_1, t_2 und t_3 schalten, bevor die Transition t_4 schaltet. Ebenso würde im zweiten Fall zunächst die Transition t_6 in einer entsprechenden Schaltfolge schalten, bevor die Transition t_4 schaltet.

Anhand des Beispiels in Abbildung 6.3 lässt sich auch der Vorteil der verzögernden Realisierung darstellen. Realisiert man zunächst $[t_1, t_2, t_3, t_4, t_7]$, so führt dies zu der Folge $t_1 t_2 t_3 t_4 t_7$. Diese Folge kann nun leicht um die Transitionen t_5 und t_6 ergänzt werden, so dass sich daraus die in Abbildung 6.4 dargestellte Sequentialisierung ergibt. Eine andere Reihenfolge wie $t_1 t_2 t_4 t_3 t_7$ hätte dies nicht erlaubt.

Zur Sequentialisierung eines Free-Choice-Workflownetzes kann nun ein Verfahren angegeben werden, welches im Folgenden vorgestellt wird. Hierbei wird sukzessive eine Zustandsmaschine aufgebaut, die den Anforderungen aus Definition 6.3 genügt,

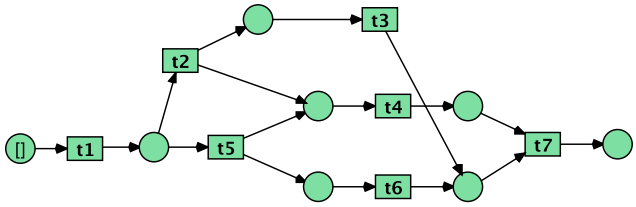


Abbildung 6.3: Ein einfaches Workflownetz

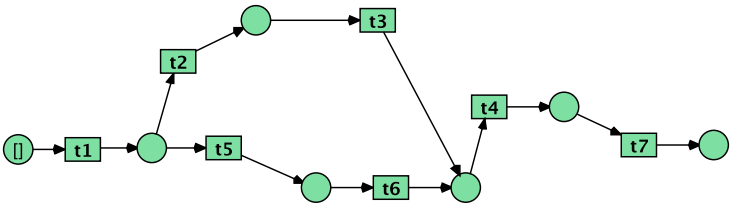


Abbildung 6.4: Eine Sequentialisierung des Netzes aus Abbildung 6.3

wenn das Workflownetz stark sequentialisierbar ist. Durch die Annahme der starken Sequentialisierbarkeit kann davon ausgegangen werden, dass $\mathbf{l}_{\mathcal{N}}$ eine bijektive Abbildung ist.

Für das Verfahren wird von der in Unterabschnitt 3.1 (S. 49) definierten Länge einer Folge N_w und dem j-ten Element $w[j]$ zu einer Folge w Gebrauch gemacht. Der Einfachheit halber sei durch t^S jeweils die Transition in T_S bezeichnet, die der Transition $t \in T$ entspricht, so dass $\mathbf{l}_{\mathcal{N}^S}(t^S) = t$ gilt. Des Weiteren gibt es eine Abbildung $\mathbf{p}_S : 2^P \to P^S$, die dem Vorbereich jeder Transition aus T ihren Vorbereich in P^S zuweist. Ist das erzeugte Netz eine Sequentialisierung, so ist sie minimal, da das erzeugte Netz stets die gleiche Kardinalität der Transitionsmenge wie das Ursprungsnetz aufweist.

Verfahren 6.8 (Sequentialisierung korrekter FCWF-Netzen)
Sei $\overline{\mathcal{N}}$ der Abschluss eines korrekten Free-Choice-Workflownetzes mit der Basis minimaler T-Invarianten $\mathbf{B}_T(\overline{\mathcal{N}})$. Ist $\overline{\mathcal{N}}$ stark sequentialisierbar, so lässt sich eine minimale Sequentialisierung $\mathcal{N}_{\mathrm{l}}^S$ von $\overline{\mathcal{N}}$ wie folgt konstruieren.

Initialisierung:

1. Initialisiere $\mathcal{N}_{\mathrm{l}}^S = (\mathcal{N}_S, T, \mathbf{l}_{\mathcal{N}_{\mathrm{l}}^S})$ mit $\mathcal{N}_S = (P^S, T^S, F^S)$, $P^S = \{i^S\}$, $T^S = \{t^S \mid t \in \overline{T}\}$ und $F^S = \emptyset$ sowie $\mathbf{l}_{\mathcal{N}^S}(t^S) = t$ für alle $t \in \overline{T}$.
2. Wähle ein $\vec{\iota} \in \mathbf{B}_T(\overline{\mathcal{N}})$ mit $\vec{\iota}(t^*) = 1$, färbe $\vec{\iota}$ und setze $T_M = \|\vec{\iota}\|$. Wähle eine verzögernde Realisierung σ von $\vec{\iota}$ in $[i]$ ($\vec{\sigma} = \vec{\iota}$ und $[i] \xrightarrow{\sigma}$).

Für jedes j mit $2 \leq j \leq N_\sigma$ sei $t = \sigma[j]$

- Füge ein neues p zu P^S hinzu.
- Füge Kante (p, t^S) und Kante $(\sigma[j-1]^S, p)$ zu F^S hinzu.
- Setze $\mathbf{p}_S({}^\bullet t) = p$.

Füge die beiden Kanten $(\sigma[N_\sigma]^S, i^S)$ und $(i^S, \sigma[1]^S)$ zu F^S hinzu und setze $\mathbf{p}_S({}^\bullet\sigma[1]) = i^S$.

Ausführung solange $T_M \neq T$:

1. Wähle ein $t^\# \in T \setminus T_M$ mit ${}^\bullet t^\# \cap {}^\bullet T_M \neq \emptyset$ und ein $\vec{\iota} \in \mathbf{B}_T(\overline{\mathcal{N}})$ mit $t^\# \in \|\vec{\iota}\|$, so dass für alle $t' \in \|\vec{\iota}\| \setminus T_M$ mit $t' \neq t^\#$ gilt ${}^\bullet t' \cap {}^\bullet T_M = \emptyset$.
2. Bilde T-Vektoren $\vec{\iota_1} = \vec{\iota} - \vec{\iota_2}$ und $\vec{\iota_2}$ mit $\vec{\iota_2}(t) = \begin{cases} \vec{\iota}(t) & \text{wenn } t \in T_M \\ 0 & \text{sonst} \end{cases}$
3. Wähle eine verzögernde Realisierung σ_1 von $\vec{\iota_1}$ in der Markierung $\mathbf{m}_1 \in [[i]\rangle_{\mathcal{N}_M}$, für die ${}^\bullet t^\# \subseteq \|\mathbf{m}_1\|$ ist, wobei $\mathcal{N}_M$ das Subnetz von $\overline{\mathcal{N}}$ mit den Knoten aus T_M und ${}^\bullet T_M \cup T_M^\bullet$ ist.
4. Für jedes j mit $2 \leq j \leq N_{\sigma_1}$ sei $t = \sigma_1[j]$
   - Füge ein neues p zu P^S hinzu.
   - Füge Kante (p, t^S) und Kante $(\sigma_1[j-1]^S, p)$ zu F^S hinzu.
   - Setze $\mathbf{p}_S({}^\bullet t) = p$.

   Füge Kante $(\mathbf{p}_S(\sigma_1[1]), \sigma_1[1]^S)$ zu F^S hinzu.
5. Setze $M_2^S = \{t^S \mid t \in \|\vec{\iota_2}\|\}$. Wähle eine Transition $t_a \in \|\vec{\iota_2}\|$, die in der Markierung $\mathbf{m}_2$ mit $\mathbf{m}_1 \xrightarrow{\sigma_1} \mathbf{m}_2$ aktiviert ist. Setze $t^x = t_a^S$.
   Solange es eine Transition $t' \in {}_{(\mathcal{N}_S)}{}^\bullet({}_{(\mathcal{N}_S)}{}^\bullet t^x) \cap M_2$ gibt, setze $t^x = t'$.
6. Füge Kante $(\sigma_1[N_{\sigma_1}], p_x)$ mit $\{p_x\} = {}^\bullet t^x$ zu F hinzu.
   Füge $\|\vec{\iota_1}\|$ zu T_M hinzu und färbe $\vec{\iota}$.
7. Setze $\mathbf{m}_x = [p_x]$.
   Solange $M_2^S \neq \emptyset$
   - Gibt es eine in $\mathbf{m}_x$ in $\mathcal{N}_S$ aktivierte Transition $t \in M_2^S$, so entferne t aus M_2^S und setze $\mathbf{m}_x = \chi[t^\bullet]$ (Feuere t).
   - Andernfalls breche ab, da $\overline{\mathcal{N}}$ nur schwach sequentialisierbar ist.

■

Beispielhafte Anwendung des Sequentialisierungsverfahrens

Im Folgenden wird das Verfahren 6.8 angewandt, wobei einmal ein stark sequentialisierbares Netz sequentialisiert wird und einmal ein nicht stark sequentialisierbares Netz überprüft wird.

Anwendung des Verfahrens 6.8 im Erfolgsfall Abbildung 6.5 zeigt ein Workflownetz, das durch Verfahren 6.8 sequentialisiert werden soll. Sei $\mathcal{N}$ in diesem Fall der Abschluss des Netzes, so muss zunächst eine Basis minimaler T-Invarianten $\mathbf{B}_T(\mathcal{N})$ berechnet werden. Diese besteht im vorliegenden Fall beispielsweise aus den vier minimalen T-Invarianten $\mathbf{B}_T(\mathcal{N}) = \{\vec{\iota_a}, \vec{\iota_b}, \vec{\iota_c}, \vec{\iota_d}\}$ die gegeben sind durch die Vektoren $\vec{\iota_a} = [t_1, t_2, t_3, t_4, t_8, t^*]$, $\vec{\iota_b} = [t_1, t_4, t_5, t_6, t_8, t^*]$, $\vec{\iota_c} = [t_1, t_4, t_6, t_7, t_8, t^*]$ und $\vec{\iota_d} = [t_4, t_9, t_{10}]$[2].

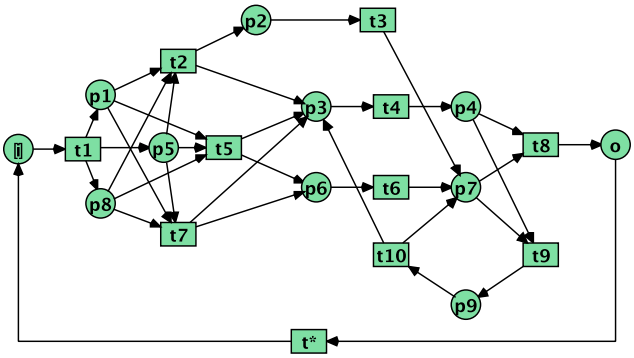


Abbildung 6.5: Ein zu sequentialisierendes korrektes Workflownetz

Die Sequentialisierung besteht zunächst nur aus der Stelle i und den Transitionen. Dies ist in Abbildung 6.6a) dargestellt. Als nächstes wird eine beliebige T-Invariante aus $\mathbf{B}_T(\overline{\mathcal{N}})$ gewählt, die dann verzögert realisiert wird. Sei dies in diesem Fall die T-Invariante $\vec{\iota_a}$, die durch die Schaltfolge $\sigma = t_1 t_2 t_3 t_4 t_8 t^*$ realisiert wird. Somit ist $T_M = \{t_1, t_2, t_3, t_4, t_8, t^*\}$ und die Transitionen der Schaltfolge werden über Stellen miteinander verbunden, wie dies in Abbildung 6.6b) dargestellt ist. Damit ist die Initialisierung beendet.

Die weitere Ausführung wählt nun in (1) ein $t^\# \in T \setminus T_M$ mit ${}^\bullet t^\# \cap {}^\bullet T_M \neq \emptyset$ und ein $\vec{\iota} \in \mathbf{B}_T(\mathcal{N})$ mit $t^\# \in \|\vec{\iota}\|$, so dass für alle $t' \in \|\vec{\iota}\| \setminus T_M$ mit $t' \neq t^\#$ gilt ${}^\bullet t' \cap {}^\bullet T_M = \emptyset$. Sei $t^\# = t_5$ und sei $\vec{\iota} = \vec{\iota_b}$. Dann ist in (2) $\vec{\iota_1} = [t_5, t_6]$, und es ist $\sigma_1 = t_5 t_6$. Die Markierung $\mathbf{m}_1$ in (3) entspricht der Markierung, die t_2 aktiviert hat, also ist $\mathbf{m}_1 = [p_1, p_5, p_8]$. In (4) wird dann ein p zu P^S hinzugefügt und es werden die entsprechenden Kanten hinzugefügt. Da $\mathbf{p}_S(\sigma_1[1]) = p_a$ ist, wird abschließend eine Kante von p_a zu t_5 hinzugefügt. Die Markierung $\mathbf{m}_2$ in (5) ist gegeben durch $\mathbf{m}_2 = [p_3, p_7]$. Da das Subnetz der T-Invariante $\vec{\iota}$ ein markierter Graph ist, ist $\mathcal{N}^S_{\vec{\iota_2}}$ ein Teilnetz dieses markierten Graphen. Die Transition t^x ist in diesem Fall durch t_4 als einzige in $\mathbf{m}_2$ aktivierte Transition gegeben. Somit wird in (6) eine Kante von t_6 nach p_c in $\mathcal{N}^S$ hinzugefügt, was in Abbildung 6.6c) dargestellt ist.

[2]Für Vektoren über $\{0, 1\}$ werden dabei nur die Werte angegeben, die verschieden von 0 sind. Dies geschieht in der Form $[a, b]$, so dass $[a, b](a) = [a, b](b) = 1$ (vgl. Abschnitt 3.1).

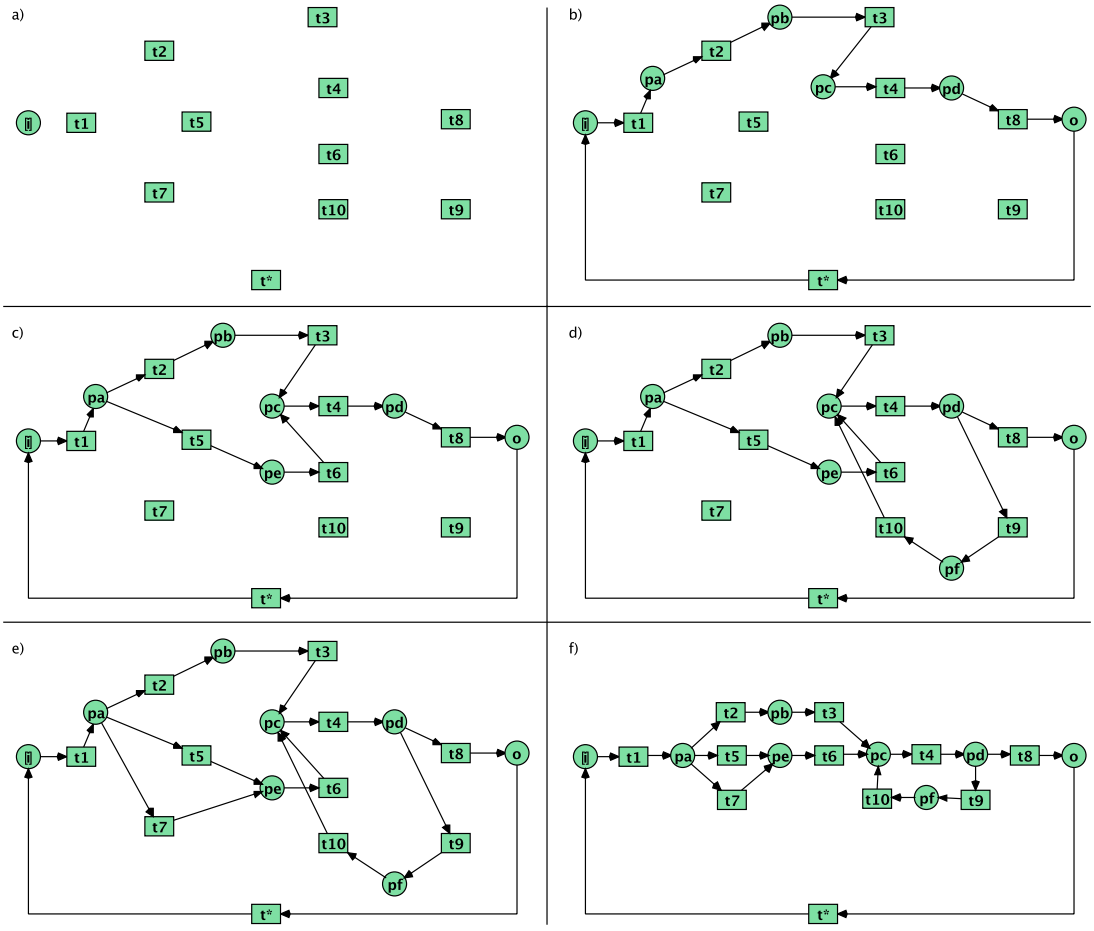


Abbildung 6.6: Die Sequentialisierung des Workflownetzes aus Abbildung 6.5 nach Verfahren 6.8

Anschließend ist $T_M = \{t_1, t_2, t_3, t_4, t_5, t_6, t_8, t^*\}$. In (7) wird überprüft, ob die T-Invariante tatsächlich vollständig realisierbar ist, so dass sichergestellt ist, dass das Netz stark sequentialisierbar war. Abschließend beginnt die Ausführung mit der nächsten minimalen T-Invariante.

Wiederum wird in (1) ein $t^{\#} \in T \setminus T_M$ ausgewählt. In Frage kommen hier t_7 und t_9. Sei $t^{\#} = t_9$ und sei $\vec{\iota} = \vec{\iota_d}$. Dann ist in (2) $\vec{\iota_1} = [t_9, t_{10}]$, und es ist $\sigma_1 = t_9 t_{10}$. Die Markierung $\mathbf{m}_1$ in (3) entspricht der Markierung, die t_8 aktiviert hat, also ist $\mathbf{m}_1 = [p_4, p_7]$. In (4) wird dann ein p zu P^S hinzugefügt und es werden die entsprechenden Kanten hinzugefügt. Da $\mathbf{p}_S(\sigma_1[1]) = p_d$ ist, wird abschließend eine Kante von p_d zu t_9 hinzugefügt. Die Markierung $\mathbf{m}_2$ in (5) ist gegeben durch $\mathbf{m}_2 = [p_3, p_7]$. Die Transition t^x ist durch t_4 gegeben. Somit wird in (6) eine Kante von t_{10} nach p_c in $\mathcal{N}^S$ hinzugefügt, was in Abbildung 6.6d) dargestellt ist. Anschließend ist $T_M = \{t_1, t_2, t_3, t_4, t_5, t_6, t_8, t_9, t_{10}, t^*\}$. In (7) wird überprüft, ob die T-Invariante

tatsächlich vollständig realisierbar ist, und es wird die Ausführung mit der nächsten minimalen T-Invariante wiederholt.

Die einzige in (1) verbleibende Transition ist t_7, so dass $t^\# = t_7$ und $\vec{\iota} = \vec{\iota_c}$ ist. Dann ist in (2) für $\vec{\iota_1}$ entsprechend $\vec{\iota_1} = [t_7]$, und es ist $\sigma_1 = t_7$. Für die Markierung $\mathbf{m}_1$ in (3) ist $\|\mathbf{m}_1\| = [p_1, p_5, p_8]$. In (4) wird dann mit $\mathbf{p}_S(\sigma_1[1]) = p_a$ eine Kante von p_a zu t_7 hinzugefügt. Für die Markierung $\mathbf{m}_2$ in (5) ist $\mathbf{m}_2 = [p_3, p_6]$. Die Transition t^x kann nun entweder t_4 oder t_6 sein. Sei $t^x = t_4$ gewählt. Da $t_6 \in M_2$ und da $t_6 \in {}^\bullet({}^\bullet t_4)$ ist, wird in (5) $t^x = t_6$ gesetzt. Somit wird in (6) eine Kante von t_7 nach p_e in $\mathcal{N}^S$ hinzugefügt, was in Abbildung 6.6e) dargestellt ist. Anschließend ist $T_M = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}, t^*\}$. In (7) wird überprüft, ob die T-Invariante tatsächlich vollständig realisierbar ist. Da dies der Fall ist, ist das Verfahren beendet.

Das graphisch aufbereitete Ergebnis des Verfahrens ist in Abbildung 6.6f) zu sehen. Das Netz hat als Basis die T-Invarianten $\vec{\iota_a} = [t_1, t_2, t_3, t_4, t_8, t^*]$, $\vec{\iota_b} = [t_1, t_4, t_5, t_6, t_8, t^*]$, $\vec{\iota_c} = [t_1, t_4, t_6, t_7, t_8, t^*]$ und $\vec{\iota_d} = [t_4, t_9, t_{10}]$, was natürlich genau den T-Invarianten der Basis des Ursprungsnetzes entspricht, da dies ja gerade von der starken Sequentialisierbarkeit gefordert wurde. Auch lässt sich überprüfen, dass die Schaltfolgen den Kriterien der Sequentialisierbarkeit entsprechen.

Anwendung des Verfahrens 6.8 im Misserfolgsfall Wie bereits auf Seite 232 erwähnt wurde, ist das Netz in Abbildung 6.2 nicht stark sequentialisierbar. Wendet man Verfahren 6.8 auf dieses Netz an, so muss es folglich zu einem Abbruch kommen.

Sei $\mathcal{N}$ wiederum der Abschluss des Netzes, und sei die Basis minimaler T-Invarianten gegeben durch $\mathbf{B}_T(\overline{\mathcal{N}}) = [\vec{\iota_a}, \vec{\iota_b}, \vec{\iota_c}]$ mit $\vec{\iota_a} = [t_1, t_2, t_3, t_4, t_9, t^*]$, $\vec{\iota_b} = [t_1, t_4, t_5, t_6, t_9, t^*]$ und $\vec{\iota_c} = [t_1, t_6, t_7, t_8, t_9, t^*]$.

Die Sequentialisierung besteht zunächst nur aus der Stelle i und den Transitionen. Dies ist in Abbildung 6.7a) dargestellt. Die T-Invariante $\vec{\iota_a}$ wird durch die Schaltfolge $\sigma = t_1 t_2 t_3 t_4 t_9 t^*$ verzögert realisiert. Somit ist $T_M = \{t_1, t_2, t_3, t_4, t_9, t^*\}$ und die Transitionen der Schaltfolge werden über Stellen miteinander verbunden, wie dies in Abbildung 6.7b) dargestellt ist. Damit ist die Initialisierung beendet.

Die weitere Ausführung wählt nun in (1) ein $t^\# \in T \setminus T_M$. Sei $t^\# = t_5$ und sei $\vec{\iota} = \vec{\iota_b}$. Dann ist in (2) $\vec{\iota_1} = [t_5, t_6]$, und es ist $\sigma_1 = t_5 t_6$. Die Markierung $\mathbf{m}_1$ in (3) entspricht der Markierung, die t_2 aktiviert hat, so dass $\mathbf{m}_1 = [p_1]$ ist. In (4) wird dann ein p zu P^S hinzugefügt und es werden die entsprechenden Kanten hinzugefügt. Da $\mathbf{p}_S(\sigma_1[1]) = p_a$ ist, wird abschließend eine Kante von p_a zu t_5 hinzugefügt. Für die Markierung $\mathbf{m}_2$ in (5) ist $\mathbf{m}_2 = [p_3, p_6]$. Die Transition t^x ist durch t_4 gegeben. Somit wird in (6) eine Kante von t_6 nach p_c in $\mathcal{N}^S$ hinzugefügt, was in Abbildung 6.7c) dargestellt ist. Anschließend ist $T_M = \{t_1, t_2, t_3, t_4, t_5, t_6, t_9, t^*\}$. In (7) wird überprüft, ob die T-Invariante tatsächlich vollständig realisierbar ist. Abschließend beginnt die Ausführung mit der nächsten minimalen T-Invariante.

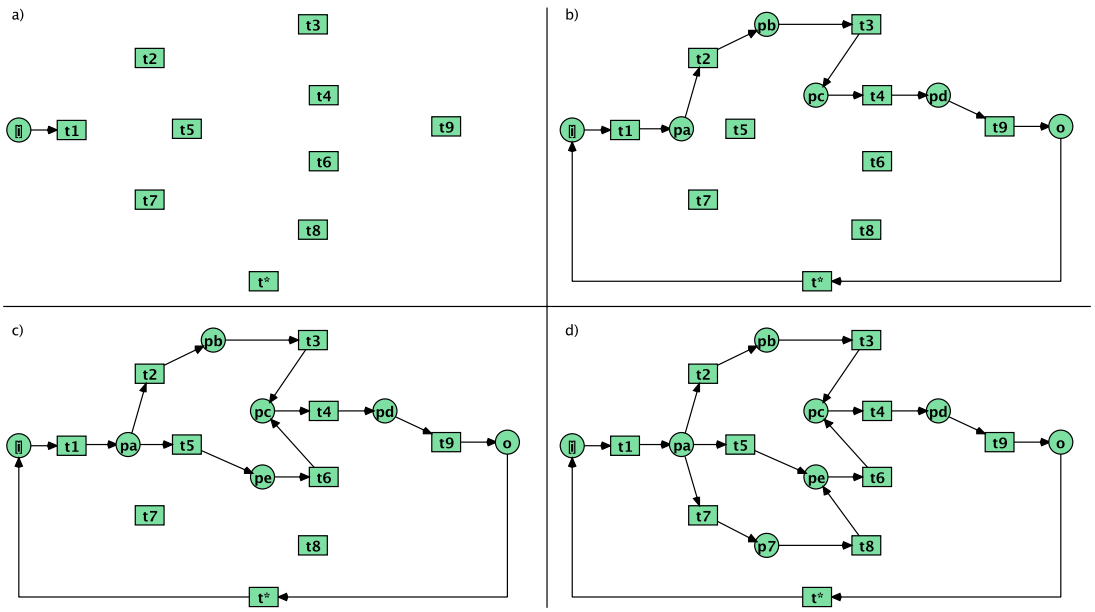


Abbildung 6.7: Die Sequentialisierung des Workflownetzes aus Abbildung 6.2 nach Verfahren 6.8

Wiederum wird in (1) t_7 als $t^{\#}$ ausgewählt, und somit ist $\vec{\iota} = \vec{\iota_c}$. Dann ist in (2) für $\vec{\iota_1}$ entsprechend $\vec{\iota_1} = [t_7, t_8]$, und es ist $\sigma_1 = t_7 t_8$. Es ist $\mathbf{m}_1 = [p_4, p_5]$ in (3) . In (4) wird dann ein p zu P^S hinzugefügt und es werden die entsprechenden Kanten hinzugefügt. Zudem wird mit $\mathbf{p}_S(\sigma_1[1]) = p_a$ eine Kante von p_a zu t_7 hinzugefügt. In (5) ist $\mathbf{m}_2 = [p_3, p_6]$. Die Transition t^x kann nur t_6 sein. Somit wird in (6) eine Kante von t_8 nach p_e in $\mathcal{N}^S$ hinzugefügt, was in Abbildung 6.6e) dargestellt ist. Anschließend ist $T_M = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t^*\}$. In (7) wird überprüft, ob die T-Invariante tatsächlich vollständig realisierbar ist. Es ist $M_2^S = \{t_1, t_6, t_9, t^*\}$ und $\mathbf{m}_x = [p_e]$. Somit wird zunächst t_6 aus M_2^S entfernt, und es ist $M_2^S = \{t_1, t_9, t^*\}$. Von diesen Transitionen ist jedoch keine in $\mathbf{m}_x = [p_c]$ aktivierbar, und es wird der Fehlerfall ausgegeben, da das Netz nicht stark sequentialisierbar ist.

Termination und Korrektheit des Verfahrens

Es soll nun der Nachweis erfolgen, dass das Verfahren 6.8 korrekt und vollständig ist und terminiert. Korrektheit und Vollständigkeit heißt hierbei, dass zu jedem stark sequentialisierbaren korrekten Free-Choice-Workflownetz eine entsprechende Zustandsmaschine ausgegeben wird, die das Workflownetz sequentialisiert.

Lemma 6.9 (Termination von Verfahren 6.8)
Das Verfahren 6.8 terminiert nach endlicher Zeit.

Beweis:
Es soll gezeigt werden, dass die Kardinalität von T_M in jedem Durchgang größer wird. Da $|T_M|$ maximal $|T|$ sein kann, muss nach endlicher Zahl von Durchläufen $T_M = T$ gelten, so dass die äußere Schleife abbricht. Die Schleife in 4 ist ohnehin endlich. Ebenso muss die Schleife in 7 abbrechen, da entweder eine aktivierte Transition aus M_2^S entfernt wird, oder die Schleife abbricht.

Ist $T \neq T_M$, so gibt es eine Transition $t \in T \setminus T_M$ mit ${}^\bullet t \cap {}^\bullet T_M \neq \emptyset$ und mit entsprechender minimaler T-Invariante $\vec{\iota} \in \mathbf{B}_T(\mathcal{N})$ mit $t \in \|\vec{\iota}\|$. Da $\mathcal{N}$ wohlgeformt ist, gibt es aufgrund von Korollar 5.18 auch eine minimale T-Invariante $\vec{\iota}$, so dass $t' \in \|\vec{\iota}\| \setminus T_M$ mit $t' \neq t$ impliziert ${}^\bullet t' \cap {}^\bullet T_M = \emptyset$. Da dann alle Transitionen aus $\|\vec{\iota}\|$ zu T_M hinzugefügt werden, die nicht bereits in T_M sind, ist die Kardinalität von T_M monoton wachsend, und es gilt die Aussage. □

Lemma 6.10 (Korrektheit von Verfahren 6.8)
Sei $\mathcal{N}$ ein korrektes Free-Choice-Workflownetz. Gibt das Verfahren 6.8 das Netz $\mathcal{N}_\mathrm{I}^S = (\mathcal{N}^S, T, \mathbf{l}_{\mathcal{N}^S})$ aus, so ist $\mathcal{N}_\mathrm{I}^S$ eine minimale Sequentialisierung von $\overline{\mathcal{N}}$.

Beweis:
Gebe das Verfahren 6.8 $\mathcal{N}_\mathrm{I}^S$ aus. Es ist zu zeigen, dass $\mathcal{N}_\mathrm{I}^S$ eine Sequentialisierung ist, da diese dann wegen $|T^S| = |T|$ minimal ist. Dass $\mathcal{N}_\mathrm{I}^S$ eine Zustandsmaschine ist, folgt direkt aus der Konstruktion, da zu jeder Transition nur eine Stelle im Vor- und eine Stelle im Nachbereich existieren. Zudem ist das resultierende Netz nach Satz 3.65 lebendig und sicher in der Markierung $[i^S]$. Es bleibt somit zu zeigen, dass die beiden weiteren Eigenschaften aus Lemma 6.6 ebenfalls gelten. Der Nachweis erfolgt, indem gezeigt wird, dass (*) jede minimale T-Invariante von $\overline{\mathcal{N}}$ eine Entsprechung in $\mathcal{N}_\mathrm{I}^S$ besitzt und dass (**) jede Schaltfolge in $\mathcal{N}_\mathrm{I}^S$ entsprechend in $\overline{\mathcal{N}}$ realisierbar ist. Sei $\vec{\iota_Z}$ die Summe der gefärbten T-Invarianten aus $\mathbf{B}_T(\overline{\mathcal{N}})$ und sei $T_Z = \mathbf{l}_{\mathcal{N}^S}(\|\vec{\iota_Z}\|)$, wobei die Abbildung $\mathbf{l}_{\mathcal{N}^S}(\cdot)$ auf Mengen erweitert wurde, indem alle Elemente der Menge abgebildet wurden. Dann wird für (*) gezeigt, dass die Abbildung der gefärbten T-Invarianten aus $\mathbf{B}_T(\overline{\mathcal{N}})$ auf $\mathcal{N}^S$ eine T-Invariantenbasis des Subnetzes von $\mathcal{N}^S$ aus T_Z und den Stellen im Vor- und Nachbereich darstellt.

Der Beweis erfolgt per Induktion über die Kardinalität der T-Invariantenbasis $\mathbf{B}_T(\overline{\mathcal{N}})$. Aufgrund von Korollar 5.18 lassen sich die T-Invarianten dieser Basis so anordnen, dass es stets eine Transition t mit ${}^\bullet t \cap {}^\bullet T_M \neq \emptyset$ und ein $\vec{\iota} \in \mathbf{B}_T(\mathcal{N})$ mit $t \in \|\vec{\iota}\|$ gibt, so dass für alle $t' \in \|\vec{\iota}\| \setminus T_M$ mit $t' \neq t$ gilt ${}^\bullet t' \cap {}^\bullet T_M = \emptyset$. Somit lässt sich das Netz über die minimalen T-Invarianten aus $\mathbf{B}_T(\overline{\mathcal{N}})$ aufbauen.

Gibt es nur eine minimale T-Invariante in $\mathbf{B}_T(\overline{\mathcal{N}})$ so wird durch die Initialisierung ein Zyklus erzeugt, der alle Transitionen dieser T-Invariante umfasst und der diese durch Stellen verbindet. Dieser Zyklus ist nach Lemma 3.67 eine minimale T-Invariante und es gilt (*). Da die minimale T-Invariante aus $\mathbf{B}_T(\overline{\mathcal{N}})$ in Schritt 2 der Initialisierung realisiert wurde, gilt auch (**).

Sei die Aussage also für $|\mathbf{B}_T(\overline{\mathcal{N}})| = n$ bewiesen und sei nun $|\mathbf{B}_T(\overline{\mathcal{N}})| = n+1$. Wie

bereits im Beweis zu Lemma 6.9 dargestellt wurde, gibt es aufgrund des Vorgehens in der Ausführung eine Transition t und eine T-Invariante $\vec{\iota}$ die den Kriterien in (1) genügt, sofern nicht $T = T_M$ ist. Somit lässt sich die Induktionsannahme auf das bislang konstruierte Netz $\mathcal{N}_M$ als das Subnetz von $\mathcal{N}$ mit den Knoten aus T_M und ${}^\bullet T_M \cup T_M^\bullet$ übertragen, da die gefärbten T-Invarianten aus $\mathbf{B}_T(\mathcal{N})$ eine Basis dieses Netzes darstellen, die die Kardinalität n besitzt.

Das in den Schritten (3) bis (6) erweiterte Netz $\mathcal{N}^S$ ist (ohne die Transitionen ohne Vor- oder Nachbereich aus der Initialisierung) eine Zustandsmaschine, da jede neu markierte Transition jeweils nur eine Stelle im Vor- und Nachbereich besitzt. Zudem ist das Netz zusammenhängend und somit der Abschluss eines Workflownetzes, das eine Zustandsmaschine ist. Aufgrund von (7) ist sichergestellt, dass jede minimale Invariante aus $\mathbf{B}_T(\mathcal{N})$ ihre Entsprechung in $\mathcal{N}^S$ besitzt, woraus (*) folgt.

Wäre die Entsprechung einer in $(\mathcal{N}^S, [i^S])$ aktivierten Schaltfolge σ_S nicht in $(\overline{\mathcal{N}}, [i])$ aktiviert, so gäbe es eine Teilschaltfolge σ'_S, deren Entsprechung aktiviert wäre. Aufgrund der verzögernden Realisierung in Schritt (3) ist jede Schaltfolge ausgehend von $[i]$ bis zur Markierung $\mathbf{m}_2$ in Schritt (5) aktiviert und besitzt eine Entsprechung in $\overline{\mathcal{N}}$. Somit muss σ'_S alle Transitionen aus $\|\vec{\iota_1}\|$ umfassen. Da die Stelle p_x bereits im Netz der ersten n T-Invarianten aus $\mathbf{B}_T(\mathcal{N})$ existiert, gibt es in $\mathcal{N}^S$ nun zwei Pfade von der Stelle $\mathbf{p}_S({}^\bullet\sigma[1])$ zur Stelle p_x und da $\mathcal{N}^S$ ein korrektes WF-Netz und eine Zustandsmaschine ist, stellt $(\vec{\iota_1} - \chi[\textsc{Alph}(\pi) \cap T])$ eine T-Invariante von $\mathcal{N}^S$ dar, wobei π der bereits existierende Pfad von $\mathbf{p}_S({}^\bullet\sigma[1])$ nach p_x ist. Folglich führen die Realisierungen der Entsprechungen von $\vec{\iota_1}$ und $\chi[\textsc{Alph}(\pi) \cap T]$ zur selben Markierung in $\overline{\mathcal{N}}$, und es muss in der resultierenden Markierung $\mathbf{m}_*$ der T-Vektor $\vec{\iota_2}$ in $\overline{\mathcal{N}}$ realisierbar sein, da andernfalls $\mathbf{m}_x$ eine Markierung wäre, von der die Markierung $[i]$ nicht mehr erreichbar wäre. Somit ist jede in $(\mathcal{N}^S, [i^S])$ aktivierte Schaltfolge σ_S in $(\overline{\mathcal{N}}, [i])$ aktiviert. Folglich ist $\mathcal{N}_\mathbf{l}^S$ eine minimale Sequentialisierung. □

Lemma 6.11 (Vollständigkeit von Verfahren 6.8)
Sei $\mathcal{N}$ ein korrektes Free-Choice-Workflownetz. Das Verfahren 6.8 terminiert mit der minimalen Sequentialisierung $\mathcal{N}_\mathbf{l}^S = (\mathcal{N}^S, \overline{T}, \mathbf{l}_{\mathcal{N}^S})$ von $\overline{\mathcal{N}}$, wenn $\overline{\mathcal{N}}$ stark sequentialisierbar ist.

Beweis:
Sei $\overline{\mathcal{N}}$ stark sequentialisierbar, ohne dass das Verfahren mit $\mathcal{N}_\mathbf{l}^S$ terminiert. Dann kann in Schritt (7) die Entsprechung einer minimalen T-Invariante aus $\vec{\iota} \in \mathbf{B}_T(\overline{\mathcal{N}})$ nicht in $\mathcal{N}^S$ realisiert werden. Somit gibt es eine T-Invariante $\vec{\iota_a} \in \mathbf{B}_T(\overline{\mathcal{N}})$ mit $\vec{\iota_a} \neq \vec{\iota}$, so dass das Hinzufügen von $\vec{\iota_a}$ dafür verantwortlich war, dass die Transitionen in dieser Reihenfolge zu $\mathcal{N}_S$ hinzugefügt wurden. Nach Lemma 5.16 und Satz 5.19 ist das Subnetz aus der Summe der beiden T-Invarianten $\vec{\iota_a}$ und $\vec{\iota}$ ebenfalls ein korrektes Free-Choice-Netz. Da in (3) eine verzögernde Realisierung gewählt wurde und jede Stelle außer die der Initialmarkierung in der Realisierung einer

minimalen T-Invariante nur einmal markiert wird (da es sich um T-Komponenten handelt), erstellt der Algorithmus für die beiden T-Invarianten eine korrekte Sequentialisierung. Folglich muss es eine weitere minimale T-Invariante $\vec{\iota_b} \in \mathbf{B}_T(\mathcal{N})$ verschieden von $\vec{\iota}$ und $\vec{\iota_a}$ geben, die dafür verantwortlich ist, dass die Transitionen von $\vec{\iota_b}$ so angeordnet werden mussten. Dann können die drei T-Invarianten $\vec{\iota}$, $\vec{\iota_a}$ und $\vec{\iota_b}$ nicht zusammen in einem stark sequentialisierbaren Netz vorhanden sein, und es ist bereits das Subnetz der Summe dieser drei Invarianten nicht stark sequentialisierbar. Somit gilt die Aussage. □

Fügt man Lemma 6.9, Lemma 6.10 und Lemma 6.11 zusammen, so ergibt sich daraus folgender Satz.

Satz 6.12 (Korrektheit von Verfahren 6.8)
Sei $\mathcal{N}$ ein korrektes Free-Choice-Workflownetz. Das Verfahren 6.8

- terminiert mit der minimalen Sequentialisierung $\mathcal{N}_{\mathrm{I}}^S = (\mathcal{N}^S, T, \mathbf{l}_{\mathcal{N}^S})$ von $\overline{\mathcal{N}}$ gdw. $\overline{\mathcal{N}}$ stark sequentialisierbar ist, und
- bricht ab gdw. $\overline{\mathcal{N}}$ nicht stark sequentialisierbar ist.

Für stark sequentialisierbare Free-Choice-Workflownetze erlaubt Verfahren 6.8 somit die Erstellung einer Zustandsmaschine, die bezüglich der bijektiven Abbildung der Transitionsmengen äquivalente T-Invarianten besitzt. Dies erlaubt es, die entsprechend beschrifteten Netze ebenfalls anhand der Zustandsmaschinen zu untersuchen.

Eine weiterer Ansatz zur Sequentialisierung ist durch die Verwendung von Reduktions- beziehungsweise Erweiterungsregeln gegeben. Diese werden im folgenden Unterabschnitt diskutiert.

6.2.3 Sequentialisierung durch Reduktionsregeln

Wie die Überprüfung der Korrektheit kann auch die Sequentialisierung eines Workflownetzes zum Teil mit Hilfe von Regeln erfolgen. Diese Regeln sind analog zu den Reduktionsregeln aus Unterabschnitt 5.3.3. Die Regel c) aus Tabelle 5.1 wird hierbei jedoch durch eine Regel ersetzt, die das Netz zunächst größer werden lässt, bevor es dann reduziert wird. Diese Regel ist in Tabelle 6.1 dargestellt. Der Vorteil dieses Vorgehens ist, dass bei der anschließenden Umkehrung der Reduktionsregeln mittels der in Unterabschnitt 5.3.3 eingeführten Reduktionsabbildungen eine Sequentialisierung entsteht.

Die entsprechende Regel in Tabelle 6.1 sorgt dabei für eine Hintereinanderausführung der Teilnetze, die aus p_1 und p_2 durch die Umkehrung der Reduktion entstehen können. Bei dieser Umkehrung der Reduktion muss dann in jeder Regel, die auf p_2

reduziert, die Transition t_1 durch die Transition t_1' ersetzt werden, so dass auch wirklich die korrekten Teilnetze ersetzt werden. Ebenso muss die Transition t_2 im Nachbereich von p_1 durch die Transition t_1' ersetzt werden. Folglich muss also bei der Umkehrung der Reduktionsregeln die Ersetzung der Kante (t_1, p_2) zur Ersetzung der Kante (t_1', p_2) werden. Ebenso muss eine Ersetzung der Kante (p_1, t_2) zu einer Ersetzung der Kante (p_1, t_1') werden. Dies kann entweder durch Ersetzung der Kanten in den Ersetzungsregeln geschehen oder durch eine spätere Abbildung. Durch diese Abwandlung von Regel c) ist gewährleistet, dass die aus den Stellen p_1 und p_2 durch die Reduktionsabbildungen zur Umkehrung der Reduktion hervorgehenden Subnetze sequentiell schalten.

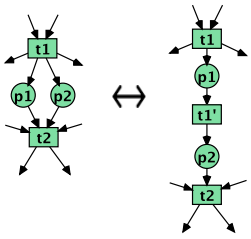
	Reduktionsregel	Ersetzungsregel $(P_1, T_1, F_1) \overset{\mathcal{N}}{\leftrightarrow} (P_2, T_2, F_2)$
c2		$P_1 = \emptyset,\ T_1 = \emptyset,$ $F_1 = \{(t_1, p_2), (p_1, t_2)\}$ $P_2 = \emptyset,\ T_2 = \{t_1'\},$ $F_2 = \{(p_1, t_1'), (t_1', p_2)\}$

Tabelle 6.1: Ersetzungsregeln zur Sequentialisierung.

Dass das Verhalten der resultierenden Netze bezüglich Korrektheit und Sicherheit äquivalent ist, lässt sich dadurch erkennen, dass die Regel c2) eine Anwendung der Regel c) und eine anschließende Anwendung der Regel a) aus Abbildung 5.7 ist. Auch ist leicht zu erkennen, dass das gesamte Netz eine Zustandsmaschine ist, wenn die Stellen p_1 und p_2 der rechten Seite der Regel durch Zustandsmaschinen mittels der umgekehrten Reduktionsregeln ersetzt werden, und das Netz vor der Ersetzung eine Zustandsmaschine ist. Die Hilfstransition t_1' kann später wieder durch die Regel a) entfernt werden. Dass es sich bei dieser Zustandsmaschine tatsächlich um eine Sequentialisierung handelt, lässt sich anhand der Überlegung erkennen, dass die Schaltfolgen der Verfeinerungen der Netze aus p_1 und p_2 so permutiert werden können, dass zunächst alle Transitionen aus der Verfeinerung von p_1 und dann alle Transitionen der Verfeinerung von p_2 schalten. Somit ist den Kriterien der Sequentialisierung aus Definition 6.3 Rechnung getragen.

Durch die in Unterabschnitt 5.3.3 in Abbildung 5.7 angegebenen Regeln lässt sich eine Subklasse der Free-Choice-Workflownetze reduzieren. Die Klasse reduzierbarer Netze ließe sich durch weitere Regeln ausweiten, die jedoch eine Verdopplung von Knoten bei der Sequentialisierung erfordern können und so die Größe des aus der

Sequentialisierung resultierenden Netzes negativ beeinflussen können. Dies kann jedoch manchmal notwendig sein, so dass ein solcher Ansatz hier kurz skizziert werden soll. Hierzu wird die Regel a) aus Abbildung 5.7 durch drei Regeln ersetzt, die in Tabelle 6.2 dargestellt sind. Dies ist notwendig, da andernfalls nicht stark sequentialisierbare Netze falsch sequentialisiert werden. Die Regeln a1) und a2) in Tabelle 6.2 sind dabei gleich der Regel a) aus Abbildung 5.7, nur dass die Vorbedingung zur Anwendung der Regel variiert. Die Regel a3) erzeugt zunächst ein größeres Netz, auf das sich dann die Regel a1) anwenden lässt.

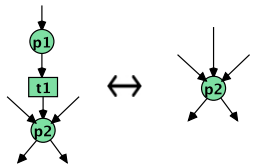
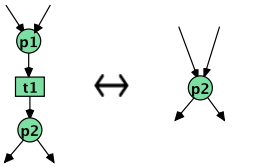
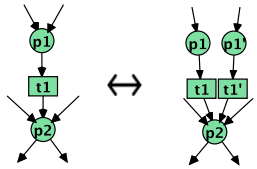
	Reduktionsregel	Ersetzungsregel $(P_1, T_1, F_1) \overset{\mathcal{N}}{\leftrightarrow} (P_2, T_2, F_2)$
a1		$P_1 = \{p_1\},\ T_1 = \{t_1\},$ $F_1 = {}^\bullet p_1 \times \{p_1\} \cup \{(p_1, t_1), (t_1, p_2)\}$ $P_2 = \emptyset,\ T_2 = \emptyset,\ F_2 = {}^\bullet p_1 \times \{p_2\}$
a2		$P_1 = \{p_1\},\ T_1 = \{t_1\},$ $F_1 = {}^\bullet p_1 \times \{p_1\} \cup \{(p_1, t_1), (t_1, p_2)\}$ $P_2 = \emptyset,\ T_2 = \emptyset,\ F_2 = {}^\bullet p_1 \times \{p_2\}$
a3		$P_1 = \emptyset,\ T_1 = \emptyset,$ $F_1 = \{(t_x, p_1)\}$ für ein $t_x \in {}^\bullet p_1$ $P_2 = \{p_1'\},\ T_2 = \{t_1'\},$ $F_2 = \{(p_1', t_1'), (t_1', p_2), (t, p_1')\}$ für obiges t_x

Tabelle 6.2: Ersetzungsregeln für komplexere Reduktion.

Die Reduzierung (a3) in Tabelle 6.2 ist lediglich eine doppelte Anwendung der Reduzierung aus Abbildung 5.7a), so dass die Erhaltung der Sicherheit und der Lebendigkeit und somit auch der Korrektheit gewährleistet ist. Besitzt die Stelle p_1 mehr als zwei Eingangskanten, so muss diese Regel auf alle diese Kanten angewandt werden.

Mittels dieser Regeln lässt sich beispielsweise das nicht stark sequentialisierbare Workflownetz in Abbildung 6.2 durch die Reduktionsregeln in Tabelle 5.1 mit der

Erweiterung aus Abbildung 5.8 reduzieren und im Anschluss dann sequentialisieren. Foglich lassen sich auch nicht stark sequentialisierbare Netze durch die oben angegebenen Regeln reduzieren und dann sequentialisieren, wenn über die Regeln aus Abbildung 5.7 hinaus auch die Regel aus Abbildung 5.8 zugelassen wird. Eine entsprechende Reduzierung findet sich in Abbildung 6.8.

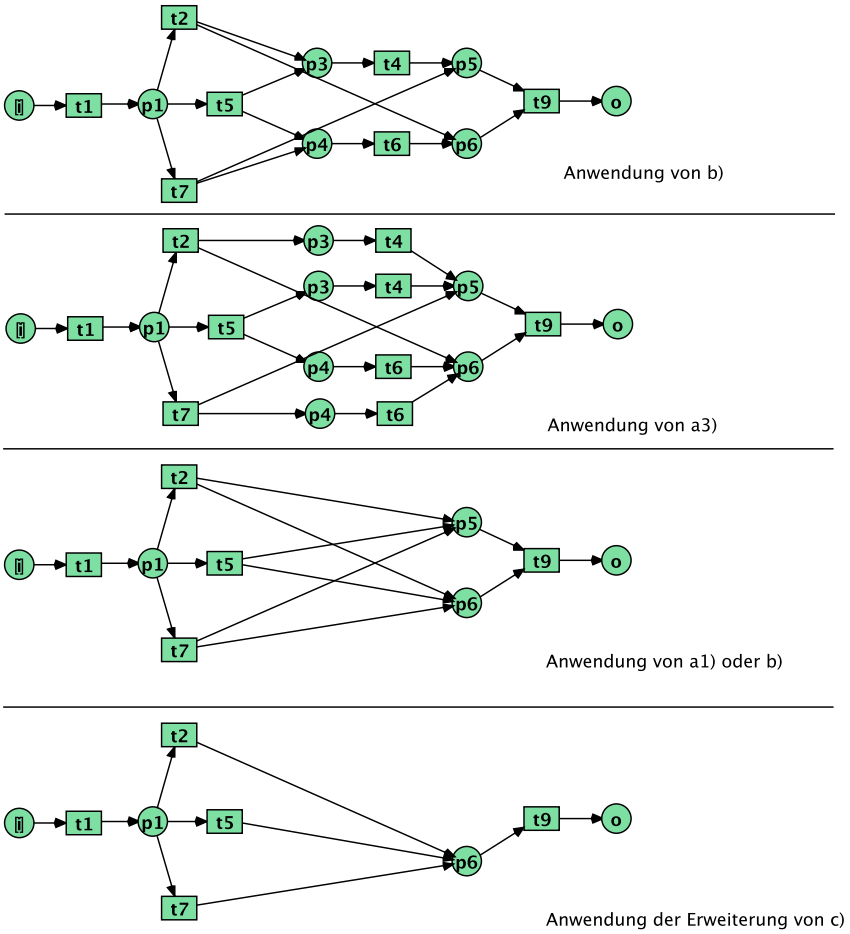


Abbildung 6.8: Reduktion des Netzes aus Abbildung 6.2

Bei der Reduktion des Netzes aus Abbildung 6.2 wird zunächst die Regel b) angewandt, wodurch die Knoten p_2 und t_3 auf die Transition t_2 sowie p_7 und t_8 auf die Transition t_7 abgebildet werden. Anschließend wird die Regel a3) angewandt, die das Netz größer werden lässt, die aber gleichzeitig garantiert, dass nicht stark sequentialisierbare Teile korrekt gedoppelt werden. Hierbei werden allerdings mehr Knoten gedoppelt als notwendig wäre.

Die Anwendung der Regel a1) oder der Regel b) erlaubt es dann das Netz weiter zu reduzieren. Abschließend muss dann noch die Erweiterung der Regel c) aus Ab-

bildung 5.8 angewandt werden, um eine zyklenfreie Zustandsmaschine zu erhalten, die dann durch die Regeln a1), a2) und d) auf eine Stelle reduziert werden kann.

Aus der Reduktion lässt sich dann wiederum ein Netz aufbauen, dass sequentialisiert ist. Dies ist in Abbildung 6.9 dargestellt. Hierzu muss jedoch bei der Anwendung der Regel aus Abbildung 5.8 jede Transition im Vorbereich einer der beiden Stellen p_5 oder p_6 verdoppelt werden, um zu gewährleisten, dass alle Kanten ersetzbar sind. Aus diesem Grund findet sich die Stelle p_5 entsprechend dreifach im Netz in Abbildung 6.9. Die Transitionen t'_2, t'_5 und t'_7 sind wiederum Hilfstransitionen, so dass sowohl die Knoten zwischen t_2 und p_5 als auch die zwischen t_2 und p_6 ersetzt werden können. Entsprechendes gilt für die anderen gedoppelten Transitionen.

Durch die Umkehrung der Reduktionsregel a1) beziehungsweise b) (je nachdem welche angewandt wurde) entsteht dann eine Zustandsmaschine mit den gleichen Invarianten wie das Originalnetz, wenn man die Transitionen mit gleichem Namen als gleich ansieht und die Hilfstransitionen t'_2, t'_5 und t'_7 ignoriert. Zuletzt werden daher diese Hilfstransitionen entfernt. Das resultierende Netz ist in Abbildung 6.9 unten dargestellt und besitzt die gleichen T-Invarianten (im Abschluss) wie das Netz in Abbildung 6.8 oben.

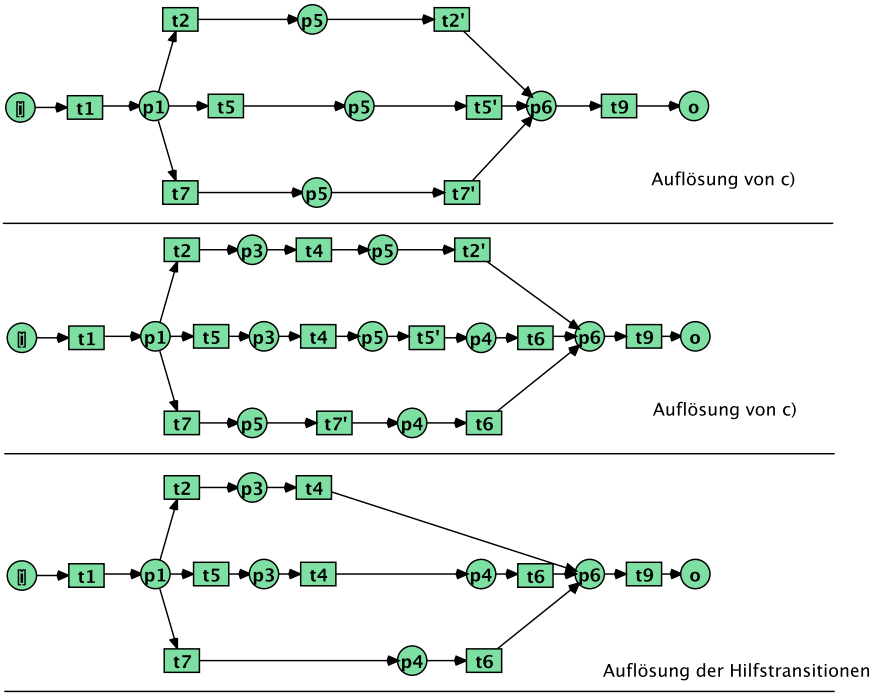


Abbildung 6.9: Zustandsmaschine aus der Reduktion von Abbildung 6.8

Ein Nachteil dieses Vorgehens ist das Entstehen doppelter Transitionen, weswegen die Regel aus Abbildung 5.8 auch bei der Sequentialisierung ausgeklammert

werden soll. Eine Möglichkeit, mit doppelten Transitionen umzugehen bietet der von Kiehn (1988) beschriebene Ansatz, in dem Teilnetze ähnlich einem Prozeduraufruf mehrfach verwendet werden können. Diese Ansätze sollen hier jedoch nicht weiter verfolgt werden, da die praktische Notwendigkeit von Netzen, die nicht stark sequentialisierbar sind, nicht in dem Maße gegeben ist.

6.3 Sequentialisierung von Free-Choice-Dienstbeschreibungsnetzen

Die Sequentialisierung von Free-Choice-Workflownetzen erlaubt nun auch die Sequentialisierung eines elementaren Free-Choice-Dienstbeschreibungsnetzes, sofern das zugrundeliegende Workflownetz sequentialisierbar ist. Hierbei ist jedoch zu beachten, dass Datenstellen nicht einfach entfernt werden können, da die entsprechenden Eingangsvariablen andernfalls bei der Ausführung der Anweisungsblöcke fehlen. Aus diesem Grund muss eine Erweiterung der Zustandsmaschine eingeführt werden, die es erlaubt, diese Datenstellen im Netz zu belassen. Ein Workflownetz, das zwar das Schaltverhalten einer Zustandsmaschine aufweist, das jedoch weitere Stellen zum Austausch von Daten besitzt wird als Quasi-Zustandsmaschine-Workflownetz (oder kurz QSMWF-Netz) bezeichnet und wird im folgenden Unterabschnitt eingeführt. Hierauf aufbauend wird die Sequentialisierung von Dienstbeschreibungsnetzen beschrieben. Abschließend wird dargestellt, wie die Sequentialisierung zur Analyse verwendet werden kann.

6.3.1 Erweiterte Sequentialisierung

Zur Repräsentation weiterer Stellen in einer Zustandsmaschine wird das Konzept der Quasi-Zustandsmaschine eingeführt. Bei diesem Netz sind sämtliche Transitionen Teil eines Subnetzes, das eine Zustandsmaschine darstellt, so dass es im Netz keine nebenläufigen Transitionen gibt.

Definition 6.13 (Quasi-Zustandsmaschine-Workflownetz)
Ein Netz $\mathcal{N}$ ist eine *Quasi-Zustandsmaschine* gdw. es eine S-Komponente $\mathcal{N}_S$ von $\mathcal{N}$ gibt, so dass $T_S = T$.
Ein Workflownetz ist ein *Quasi-Zustandsmaschinen-Workflownetz* (QSMWF-Netz) gdw. es eine Quasi-Zustandsmaschine ist. ⧫

Ein Quasi-Zustandsmaschinen-Workflownetz weist recht ähnliche Eigenschaften wie ein Workflownetz auf, das eine Zustandsmaschine ist, mit dem Unterschied, dass bei der Angabe von Eigenschaften alle S-Komponenten betrachtet werden müssen. Da ein Workflownetz mit einer S-Komponente, die i und o überdeckt auch

einen Abschluss mit einer S-Komponente besitzt, die i und o überdeckt, ist jeder Abschluss eines QSMWF-Netzes eine Quasi-Zustandsmaschine.

Satz 6.14 (Eigenschaften von QSMWF-Netzen)
Sei $\mathcal{N}$ ein QSMWF-Netz mit Abschluss $\overline{\mathcal{N}}$, so dass $T_S = \overline{T}$ für die S-Komponente $\mathcal{N}_S$ ist. Dann sind die folgenden drei Aussagen äquivalent:

1. $(\overline{\mathcal{N}}, [i])$ ist lebendig und sicher.
2. $\overline{\mathcal{N}}$ besitzt eine positive T-Invariante, ist S-überdeckbar und jede S-Komponente von $\overline{\mathcal{N}}$ überdeckt i.
3. Für alle $\mathbf{m} \in [[i]\rangle_{\overline{\mathcal{N}}}$ ist $\mathbf{m} \xrightarrow[\overline{\mathcal{N}}]{t}$ gdw. $\mathbf{m}|_{P_S} \xrightarrow[\mathcal{N}_S]{t}$.

Beweis:
$(1 \Rightarrow 2)$ Sei $(\overline{\mathcal{N}}, [i])$ lebendig und sicher. Nach Lemma 3.53 besitzt $\overline{\mathcal{N}}$ eine positive T-Invariante. Wäre $\overline{\mathcal{N}}$ nicht S-überdeckbar, so gäbe es eine Menge von Stellen P_X, nach deren Entfernung ein S-überdeckbares Netz entstehen würde, da ja $\mathcal{N}_S$ eine S-Komponente ist. Sei $\mathcal{N}_X$ dieses Netz. Durch Hinzufügen einer Stelle $p_x \in P_X$ zu $\mathcal{N}_X$ zusammen mit den entsprechenden Kanten entsteht wiederum ein stark zusammenhängendes Netz. Jede Schaltfolge, die eine T-Invariante in $[i]$ realisiert und die eine Marke auf p_x legt, muss diese auch wieder entfernen. Dann gibt es jedoch auch eine S-Invariante, die p_x den Wert -1 und den Stellen zwischen einer Transition im Vorbereich von p_x und dessen Nachbereich den Wert 1 zuweist. Hieraus lässt sich dann eine uniforme S-Invariante erzeugen, die p_x überdeckt. Wäre das Subnetz dieser uniformen S-Invariante keine S-Komponente, so gäbe es im Subnetz eine Transition mit zwei Eingangskanten. Da das Subnetz der uniformen S-Invariante jedoch nur mit einer Marke in $[i]$ markiert ist, kann es keine erreichbare Markierung geben, die diese Transition schaltet (vgl. Satz 3.50), was der Lebendigkeit widerspricht. Somit ist das Subnetz eine S-Komponente und es muss $P_X = \emptyset$ gelten.

Da jede S-Komponente ein Siphon darstellt und da jedes Siphon in einem korrekten Workflownetz initial markiert sein muss (vgl. Barkaoui u. a. (2007)) überdeckt jede S-Komponente die Stelle i.
$(2 \Rightarrow 3)$
$(\Rightarrow)$ $\mathbf{m} \xrightarrow[\overline{\mathcal{N}}]{t}$ impliziert $\mathbf{m}|_{P_S} \xrightarrow[\mathcal{N}_S]{t}$, da die Zahl der Stellen im Vorbereich von t gleich bleiben oder verringert werden.
$(\Leftarrow)$ Wäre $\mathbf{m}|_{P_S} \xrightarrow[\mathcal{N}_S]{t}$, ohne dass $\mathbf{m} \xrightarrow[\overline{\mathcal{N}}]{t}$ gilt, so gäbe es in $\mathbf{m}$ unmarkierte Stellen $P_x \subseteq ({}^\bullet t \setminus P_S)$ im Vorbereich von t. Es soll über $|P_X|$ gezeigt werden, dass dies nicht möglich ist. Ist $|P_X| = 0$, so besitzt t lediglich die Stelle aus P_S im Vorbereich, die dann markiert sein muss, da sie in $\mathbf{m}|_{P_S}$ markiert ist. Sei die Annahme also für $|P_X| = n$ bewiesen und sei nun $|P_X| = n+1$. Dann gibt es eine Stelle $p_x \in P_X$ nach

deren Entfernen aus $\overline{\mathcal{N}}$ die Transition t nach Induktionsannahme in $\mathbf{m}$ aktiviert ist.

Sei $\mathcal{N}^X$ das Netz aus der Vereinigung von $\mathcal{N}_S$ und einer S-Komponente $\mathcal{N}'_S$, die p_x überdeckt, so dass $\mathcal{N}^X$ nur genau zwei S-Komponenten besitzt. Es existiert stets die Vereinigung von $\mathcal{N}_S$ mit einer S-Komponente, die p_x überdeckt. Besitzt diese mehr als diese beiden S-Komponenten, so gibt es auch eine S-Komponente, die p_x überdeckt und die stattdessen gewählt werden kann, da jede der S-Komponenten das Subnetz einer minimalen S-Invariante einer Basis darstellt (aufgrund der positiven S-Invariante existiert eine solche) und die Aussage dann aus dem Basisergänzungssatz der linearen Algebra folgt. Sei also $\mathcal{N}^X$ das Netz aus der Vereinigung der zwei S-Komponenten $\mathcal{N}_S$ und $\mathcal{N}'_S$, das p_x und alle Stellen aus P_S überdeckt.

Wäre p_x die einzige Stelle, die in $\mathcal{N}^X$, nicht aber in $\mathcal{N}_S$ ist, so wäre p_x die einzige Stelle der S-Komponente und müsste in $\mathbf{m}$ markiert sein, da $\mathbf{m}$ aufgrund von Satz 3.49 auf allen S-Komponenten mit i übereinstimmt. Somit muss neben p_x eine weitere Stelle in $\mathcal{N}^X$ nicht aber in $\mathcal{N}_S$ existieren. Da nur die S-Komponente $\mathcal{N}'_S$ hinzukommt, nimmt die Zahl linear unabhängiger minimaler S-Invarianten in $\mathcal{N}^X$ um eine zu, so dass für $\mathcal{N}^X$ die Dimension des T-Invariantenvektorraums folglich um mindestens eins geringer sein muss, als für $\mathcal{N}_S$ (vgl. Lemma 5.13). Dann muss ein Pfad der S-Komponente $\mathcal{N}'_S$ zwei minimale T-Invarianten (also zwei Zyklen) von $\mathcal{N}_S$ verbinden, wie dies in Abbildung 6.10 dargestellt ist.

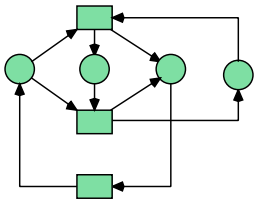


Abbildung 6.10: Ein Netz mit zwei S-Komponenten, dessen Dimension des T-Invariantenvektorraums durch die Vereinigung kleiner wird

Dann kann jedoch entweder nicht jede der beiden S-Komponenten die Stelle i überdecken oder es müssen Transitionen in $\mathcal{N}'_S$ sein, die nicht in $\mathcal{N}_S$ sind. Beides widerspricht den Anforderungen, so dass die Aussage gilt.
(3 $\Rightarrow$ 1) Aufgrund von Satz 3.65 ist $(\mathcal{N}_S, [i])$ lebendig und sicher. Da jede Schaltfolge von $\mathcal{N}_S$ aufgrund von (3) eine Schaltfolge von $\overline{\mathcal{N}}$ ist und umgekehrt, ist auch $\overline{\mathcal{N}}$ sicher und lebendig. □

Durch die erweiterte Sequentialisierung wird der Abschluss jedes stark sequentialisierbaren korrekten Free-Choice-Workflownetzes in eine Quasi-Zustandsmaschine überführt, die sicher und lebendig ist. Die erweiterte Sequentialisierung ergibt sich dabei anhand folgender Definition.

Definition 6.15 (Erweiterte Sequentialisierung)
Sei $\mathcal{N}$ ein stark sequentialisierbares korrektes FCWF-Netz, sei $\mathcal{N}^S$ eine minimale Sequentialisierung von $\overline{\mathcal{N}}$ und sei $\mathbf{l}_{\mathcal{N}^S} : \overline{T} \to T_S$ die zugehörige Bijektion zwischen den Transitionsmengen.
Eine *erweiterte Sequentialisierung* $\mathcal{N}^{S+}$ ist der in $[i]$ lebendige und sichere Abschluss eines QSMWF-Netzes, das $\mathcal{N}^S$ als Subnetz besitzt und bei dem für alle Transitionen $t_1, t_2 \in T$ gilt, dass $t_1 \in (t_2{}^{\bullet_{(\mathcal{N})}})^{\bullet_{(\mathcal{N})}}$ impliziert $\mathbf{l}_{\mathcal{N}^S}(t_1) \in (\mathbf{l}_{\mathcal{N}^S}(t_2)^{\bullet_{(\mathcal{N}^{S+})}})^{\bullet_{(\mathcal{N}^{S+})}}$. ♦

Da nach Satz 6.14 jede in $(\mathcal{N}_S, [i])$ aktivierte Schaltfolge auch in der erweiterten Sequentialisierung aktiviert ist, gilt somit auch für die erweiterte Sequentialisierung von $\mathcal{N}$, dass

- $\mathcal{N}^{S+}$ der Abschluss eines korrekten Workflownetzes und eine Quasi-Zustandsmaschine ist,
- für jede Schaltfolge σ mit $[i] \xrightarrow[\mathcal{N}]{\sigma} [o]$ in $\mathcal{N}$ eine Schaltfolge σ_S mit $[i^S] \xrightarrow[\mathcal{N}^{S+}]{\sigma_S} [o^S]$ in $\mathcal{N}^{S+}$ existiert und $\mathbf{l}_{\mathcal{N}^{S+}}(\sigma_S) \in [\sigma]_{\underset{\sim}{\sigma}}$ ist,
- für jede Schaltfolge σ_S mit $[i^S] \xrightarrow[\mathcal{N}^{S+}]{\sigma_S} [o^S]$ in $\mathcal{N}^{S+}$ eine Schaltfolge σ mit $[i] \xrightarrow[\mathcal{N}]{\sigma} [o]$ in $\mathcal{N}$ existiert und $\mathbf{l}_{\mathcal{N}^S}(\sigma_S) = \sigma$ ist.

Die wesentlichen Eigenschaften aus Definition 6.3 lassen sich also übertragen.

Nachdem durch das Verfahren 6.8 die Sequentialisierung eines Netzes erreicht ist, lässt sich die erweiterte Sequentialisierung dann durch die Ergänzung um die entsprechenden Stellen zwischen den Transitionen erreichen.

Verfahren 6.16 (Konstruktion der erweiterten Sequentialisierung)
Sei $\overline{\mathcal{N}}$ der Abschluss eines korrekten stark sequentialisierbaren FCWF-Netzes und sei $\mathcal{N}^S$ dessen minimale Sequentialisierung sowie $\mathbf{l}_{\mathcal{N}^S} : \overline{T} \to T_S$ die zugehörige Bijektion zwischen den Transitionen. Dann ergibt sich eine erweiterte Sequentialisierung $\mathcal{N}^{S+}$ durch folgendes Verfahren.

Initialisierung:
Initialisiere $\mathcal{N}^{S+} = \mathcal{N}^S$.

Ausführung:
Für alle $p \in P$ füge p^S zu P^{S+} hinzu.
Füge $\{(\mathbf{l}_{\mathcal{N}^S}(t), p^S) \mid (t, p) \in F\}$ und $\{(p^S, \mathbf{l}_{\mathcal{N}^S}(t)) \mid (p, t) \in F\}$ zu F^{S+} hinzu.

■

Für Dienstbeschreibungsnetze bietet die erweiterte Sequentialisierung den Vorteil, dass die Weitergabe der Daten sehr leicht übertragen werden kann, indem die

Stellen den gleichen Typ und die gleichen Ein- und Ausgangsvariablen erhalten wie im Ursprungsnetz. Eine Alternative wäre es gewesen, die Kantenanschriften zu erweitern, was jedoch im Allgemeinen etwas umständlicher ist. Zudem lässt sich so leichter erkennen, wann welche Ergebnisse einer Berechnung weiterverwendet werden. Es bleibt nun zu zeigen, dass das Verfahren 6.16 korrekt ist, da die Termination des Verfahrens offensichtlich ist.

Satz 6.17 (Verfahren 6.16 ist korrekt)
Sei $\overline{\mathcal{N}}$ der Abschluss eines korrekten stark sequentialisierbaren FCWF-Netzes und sei $\mathcal{N}^S$ dessen Sequentialisierung. Dann ergibt sich eine erweiterte Sequentialisierung $\mathcal{N}^{S+}$ durch das Verfahren 6.16.

Beweis:
Offensichtlich bleibt $\mathcal{N}^{S+}$ der Abschluss eines WF-Netzes. Zudem überdeckt $\mathcal{N}^S$ sämtliche Transitionen von $\mathcal{N}^{S+}$, so dass $\mathcal{N}^{S+}$ der Abschluss eines QSMWF-Netzes ist. Es bleibt somit zu zeigen, dass $\mathcal{N}^{S+}$ lebendig und sicher ist.

Sei $P^X = \{p \in P^{S+} \mid \neg\exists p' \in P^S : {}^\bullet p = {}^\bullet p' \wedge p^\bullet = p'^\bullet\}$ die Menge der Stellen, die nicht in P^S sind und die nicht den gleichen Vor- und Nachbereich wie eine Stelle aus P^S besitzen. Der weitere Beweis erfolgt per Induktion über die Kardinalität $|P^X|$. Ist $|P^X| = 0$, so ist jede Stelle des Netzes entweder auch in P^S oder sie besitzt den gleichen Vor- und Nachbereich wie eine Stelle aus P^S, in welchem Fall das Netz aufgrund von Satz 3.65 und aufgrund der Reduktionsregeln aus Abschnitt 5.3.3 sicher und lebendig ist.

Sei die Annahme also für $|P^X| = n$ bewiesen und sei nun $|P^X| = n + 1$. Dann lässt sich durch Entfernen einer Stelle p^x, die bei der Ausführung hinzugefügt wurde und die nicht in P_S ist, ein Netz erstellen, für das $|P^X| = n$ ist und für das nach Induktionsannahme somit die Behauptung gilt. Da für jede Schaltfolge σ von $[i]$ nach $[i]$ in $\mathcal{N}^S$ aufgrund der starken Sequentialisierbarkeit auch eine Schaltfolge σ' in $\overline{\mathcal{N}}$ von $[i]$ nach $[i]$ existiert, muss dies auch für jede Schaltfolge gelten, die eine Transition besitzt, die die Stelle p^x im Vorbereich besitzt. Da $\overline{\mathcal{N}}$ korrekt ist, muss in einer Schaltfolge σ', die eine Transition t_2 mit $p^x \in {}^\bullet t_2$ besitzt, auch eine Transition t_1 mit $p^x \in t_1^\bullet$ existieren, die in σ' vor t_2 schaltet. Da es eine Bijektion zwischen den Transitionsmengen $\overline{T}$ und T^S gibt, muss dies dann auch für $\mathcal{N}^{S+}$ gelten, und es gilt die Aussage. □

6.3.2 Vorgehen zur Sequentialisierung

Die Ergebnisse des letzten Unterabschnitts erlauben es, die Sequentialisierung elementarer Dienstbeschreibungsnetze zu definieren. Die Sequentialisierung betont die Möglichkeit, ein elementares Dienstbeschreibungsnetz in ein Dienstbeschreibungsnetz zu überführen, dessen zugrundeliegendes Netz eine Quasi-Zustandsmaschine ist, die korrekt ist. Dieses Netz kann dann weiter analysiert werden.

Die Sequentialisierung erfolgt durch die Verfahren aus dem letzten Abschnitt. Hierbei muss jedoch berücksichtigt werden, dass unter Umständen Daten zwischen Transitionen ausgetauscht werden, die in der einfachen Form der Sequentialisierung, wie sie in Abschnitt 6.2 dargestellt wurde, nicht berücksichtigt sind. Um zu gewährleisten, dass eine Transition die richtigen Stellen im Vor- und Nachbereich hat, wird der Sequentialisierung ein weiterer Schritt nachgeschaltet, der dies garantiert. Hierbei ist zu gewährleisten, dass jede Schaltfolge der ursprünglichen Sequentialisierung auch eine Schaltfolge des neuen Netzes ist und umgekehrt.

Definition 6.18 (Sequentialisierung elementarer SD-Netze)
Die Sequentialisierung eines elementaren Dienstbeschreibungsnetzes $\mathcal{SDN}$ ist ein Netz $\mathcal{SDN}^S = (\Sigma^{\mathbf{Q}}, X, \mathcal{N}^S, d^S, \mathbf{cmd}^S, W^S, G^S)$, so dass

- das zugrundeliegende Netz $\mathcal{N}^S$ eine erweiterte Sequentialisierung von $\overline{\mathcal{N}}$ ist und eine S-Komponente $\mathcal{N}_{S*}$ mit $p \in P_{S*} \implies d^S(p) = s_{\mathbf{token}}$ besitzt,
- für alle $t \in T$ und für $t^S = \mathbf{l}_{\mathcal{N}^S}(t)$ gilt $\mathbf{cmd}^S(t^S) = \mathbf{cmd}(t)$ und $G^S(t^S) = G(t)$,
- für alle $(p,t) \in F$ ein $(p^S, \mathbf{l}_{\mathcal{N}^S}(t)) \in F^S$ mit $d^S(p^S) = d(p)$ existiert, so dass $W^S(p^S, \mathbf{l}_{\mathcal{N}^S}(t)) = W(p,t)$,
- für alle $(t,p) \in F$ ein $(\mathbf{l}_{\mathcal{N}^S}(t), p^S) \in F^S$ mit $d^S(p^S) = d(p)$ existiert, so dass $W^S(\mathbf{l}_{\mathcal{N}^S}(t), p^S) = W(t,p)$, und
- für alle $t \in T$ und für $t^S = \mathbf{l}_{\mathcal{N}^S}(t)$ gilt $|{}^\bullet t^S \setminus \{p \in P^S \mid d^S(p) = s_{\mathbf{token}}\}| = |{}^\bullet t \setminus \{p \in P \mid d(p) = s_{\mathbf{token}}\}|$ und $|t^{S\bullet} \setminus \{p \in P^S \mid d^S(p) = s_{\mathbf{token}}\}| = |t^\bullet \setminus \{p \in P \mid d(p) = s_{\mathbf{token}}\}|$.

⧫

Die Sequentialisierung stellt also eine Quasi-Zustandsmaschine dar, die eine S-Komponente als Subnetz besitzt, die nur anonyme Marken verwendet. Diese S-Komponente stellt den Kontrollfluss dar. Zudem besitzen die entsprechenden Transitionen der Sequentialisierung den selben Anweisungsblock und dieselben Ein- und Ausgangsvariablen. Auch sind die Stellen in ihrem jeweiligen Vor- und Nachbereich genauso getypt wie die Stellen des Ursprungsnetzes. Die Konstruktion eines solchen Netzes kann über das Verfahren 6.16 erfolgen.

Es soll nun gezeigt werden, dass eine solche Sequentialisierung das gleiche Schaltverhalten wie das Originalnetz besitzt, wenn man Schaltfolgen von einem initialen zu einem finalen Zustand betrachtet. Hierzu muss zunächst eine Folgerung der funktionalen Ausführbarkeit elementarer Dienstbeschreibungsnetze gezeigt werden, bevor dann auf die Verhaltensäquivalenz eines elementaren Dienstbeschreibungsnetzes und seiner Sequentialisierung eingegangen wird.

Lemma 6.19 (Schaltfolgen elementarer Dienstbeschreibungsnetze)
Gegeben ein elementares Free-Choice-Dienstbeschreibungsnetz $\mathcal{SDN}$. Sei Q_1 ein Netzzustand und seien σ_α und σ'_α zwei Schaltfolgen mit $Q_1 \xrightarrow{\sigma_\alpha} Q$ und $Q_1 \xrightarrow{\sigma'_\alpha} Q'$, so dass σ_α und σ'_α jeweils einen Teil einer minimalen T-Invariante des Abschlusses des zugrundeliegenden Netzes $\overline{\mathcal{N}}$ realisieren.
Dann ist $Q = Q'$, wenn alle Transitionen von $\mathcal{SDN}$ gleich häufig in σ_α und in σ'_α vorkommen (also σ_α und σ'_α dieselben T-Vektoren von $\mathcal{N}$ realisieren).

Beweis:
Da jede minimale T-Invariante des zugrundeliegenden Netzes $\mathcal{N}$ nach Lemma 5.14 eine T-Komponente ist, wird bei jeder Realisierung dieser T-Komponente jede Stelle nur maximal einmal markiert. Dies gilt auch für die Schaltfolgen σ_α und σ'_α, da sie Teilschaltfolgen einer solchen T-Invariante sind. Es soll gezeigt werden, dass dann jede Stelle im Nachbereich einer Transition, die in der Schaltfolge σ_α schaltet, mit dem gleichen Wert markiert wird.

Solange die beiden Schaltfolgen die gleichen Transitionen schalten, führen sie auch zum gleichen Zustand. Gibt es eine Transition, die in σ_α schaltet, bevor sie in σ'_α schaltet, so muss es zwei nebenläufig aktivierte Transitionen geben, so dass eine in σ_α und eine in σ'_α schaltet, da das Netz sicher ist und jede T-Komponente ein markierter Graph ist. Da die Vor- und Nachbereiche der beiden Transitionen dann disjunkt sind, ändern sich die Marken im Vorbereich der Transitionen nicht. Aufgrund der funktionalen Ausführbarkeit der Schaltmodi (vgl. Def. 4.36) ist die Schaltreihenfolge der Transition unerheblich für den Ergebniszustand. Somit lassen sich die Transitionen vertauschen, ohne dass sich der Finalzustand ändert. Dies kann solange durchgeführt werden, bis $\sigma_\alpha = \sigma'_\alpha$ ist. Da der Finalzustand jedesmal unverändert ist, gilt die Aussage. □

Satz 6.20 (Äquivalentes Verhalten der Sequentialisierung)
Gegeben ein elementares Free-Choice-Dienstbeschreibungsnetz $\mathcal{SDN}$ und und dessen Sequentialisierung $\mathcal{SDN}^S$. Es gilt für einen initialen Netzzustand $(\mathbf{m}_0, \mathfrak{Q}_0^G)$ und einen finalen Netzzustand $(\mathbf{m}_f, \mathfrak{Q}_f^G)$ von $\mathcal{SDN}$, dass

$$(\mathbf{m}_f, \mathfrak{Q}_f^G) \in [(\mathbf{m}_0, \mathfrak{Q}_0^G)\rangle_{\mathcal{SDN}} \text{ gdw. } (\mathbf{m}_f^S, \mathfrak{Q}_f^G) \in [(\mathbf{m}_0^S, \mathfrak{Q}_0^G)\rangle_{\mathcal{SDN}^S},$$

wobei $\mathbf{m}_0^S$ die Markierung ist, die i^S den Wert von $\mathbf{m}_0(i)$ zuweist und allen anderen Stellen den Wert $\mathbf{0}$ und $\mathbf{m}_f^S$ die Markierung ist, die o^S den Wert von $\mathbf{m}_f(o)$ zuweist und allen anderen Stellen den Wert $\mathbf{0}$.

Beweis:
Im Rahmen des Beweises werden die Transitionsmengen von $\mathcal{SDN}$ und von $\mathcal{SDN}^S$ als gleich angenommen, so dass eine Abbildung über $\mathbf{l}_{\mathcal{N}^S}$ entfällt. Dies macht die folgende Darstellung schlanker und leichter verständlich.
($\Rightarrow$) Sei $(\mathbf{m}_f, \mathfrak{Q}_f^G) \in [(\mathbf{m}_0, \mathfrak{Q}_0^G)\rangle_{\mathcal{SDN}}$. Dann gibt es eine Schaltfolge σ_α in $\mathcal{SDN}$ mit

$(\mathbf{m}_0, \mathfrak{Q}_0^G) \xrightarrow[\mathcal{SDN}]{\sigma_\alpha} (\mathbf{m}_f, \mathfrak{Q}_f^G)$. Sei σ die Schaltfolge im zugrundeliegenden Netz $\mathcal{N}$, die dieselben Transitionen in derselben Reihenfolge verwendet. Der Beweis erfolgt über die Zahl der minimalen T-Invarianten, die σ vollständig realisiert. Realisiert σ keine minimale T-Invariante von $\mathcal{N}$, so ist σ entweder eine Schaltfolge von $\mathcal{N}^S$ oder es lässt sich nach Lemma 6.19 durch Vertauschen von nebenläufigen Transitionen eine Schaltfolge erzeugen, die auch eine Schaltfolge von $\mathcal{N}^S$ ist, da es nach Definition 6.3 eine solche Schaltfolge geben muss. Da beide Schaltfolgen die gleichen Transitionen schalten und diese jeweils die gleichen Anschriften und die gleichen Ein- und Ausgangsvariablen besitzen, gilt die Aussage.

Sei die Annahme also für eine Schaltfolge bewiesen, die n minimale T-Invarianten vollständig realisiert und sei σ eine Schaltfolge, die $n+1$ minimale T-Invarianten vollständig realisiert. Dann lässt sich eine minimale T-Invariante wählen, die vollständig realisiert wurde. Diese T-Invariante lässt sich abwickeln, indem sämtliche Knoten der T-Invariante zusammen mit den Transition in σ, die der nächsten minimalen T-Invariante angehören bzw. die zum finalen Zustand führen, noch einmal dem Netz $\mathcal{SDN}$ hinzugefügt werden. Dabei erhalten sämtliche Knoten die gleichen Anschriften. Dies ist schematisch in Abbildung 6.11 dargestellt. Das Resultat dieses Vorgehens sind zwei Schaltfolgen, deren Ausführung zu einem identischem Resultat führt, wobei eine der Schaltfolgen die minimale T-Invariante realisiert und die andere dies nicht tut. Somit gilt die Aussage für das letztere Netz, indem die letzte T-Invariante nicht realisiert wird.

Dies bedeutet, dass es eine Schaltfolge in der zugehörigen Sequentialisierung gibt, die ebenfalls die minimale T-Invariante realisieren kann. Folglich kann auch die Schaltreihenfolge der Schaltmodi von σ_α so zu einer Schaltfolge σ'_α umgruppiert werden, dass der erreichte Finalzustand derselbe ist, σ'_α jedoch auch in $\mathcal{SDN}^S$ eine aktivierte Schaltfolge darstellt.
($\Leftarrow$) Sei $(\mathbf{m}_f^S, \mathfrak{Q}_f^G) \in [(\mathbf{m}_0^S, \mathfrak{Q}_0^G)\rangle_{\mathcal{SDN}^S}$. Dann gibt es eine Schaltfolge σ_α in $\mathcal{SDN}^S$ mit $(\mathbf{m}_0^S, \mathfrak{Q}_0^G) \xrightarrow[\mathcal{SDN}^S]{\sigma_\alpha} (\mathbf{m}_f^S, \mathfrak{Q}_f^G)$. Diese Schaltfolge ist nach der Definition der Sequentialisierbarkeit in Definition 6.3 auch im zugrundeliegenden Netz von $\mathcal{SDN}$ realisierbar. Da alle Transitionen die gleichen Anschriften besitzen und da sämtliche Variablen an die gleichen Werte gebunden werden (vgl. Def. 6.18), gilt die Aussage. □

Satz 6.20 zeigt die funktionale Äquivalenz eines elementaren Dienstbeschreibungsnetzes und seiner Sequentialisierung. Dies ermöglicht es, bei der Analyse elementarer Dienstbeschreibungsnetze die Sequentialisierung eines Dienstbeschreibungsnetzes zu analysieren, wenn entschieden werden soll, wie sich eine Schaltfolge funktional verhält. Auch kann der symbolische Erreichbarkeitsgraph auf der Grundlage der Sequentialisierung gebildet werden, was den Zustandsraum deutlich reduziert.

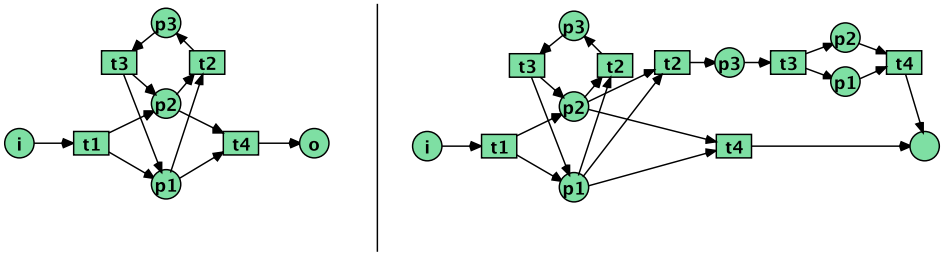


Abbildung 6.11: Abwicklung einer minimalen T-Invariante

Beispiel 6.1: Die Sequentialisierung soll anhand eines Beispiels verdeutlicht werden. Als zugrundeliegendes Netz dient das Netz aus Abbildung 6.3, dessen Sequentialisierung in Abbildung 6.4 zu sehen ist. Abbildung 6.12 stellt ein Dienstbeschreibungsnetz dar, das eine Erweiterung des Netzes aus Abbildung 6.3 um Stellen als zugrundeliegendes Netz besitzt.

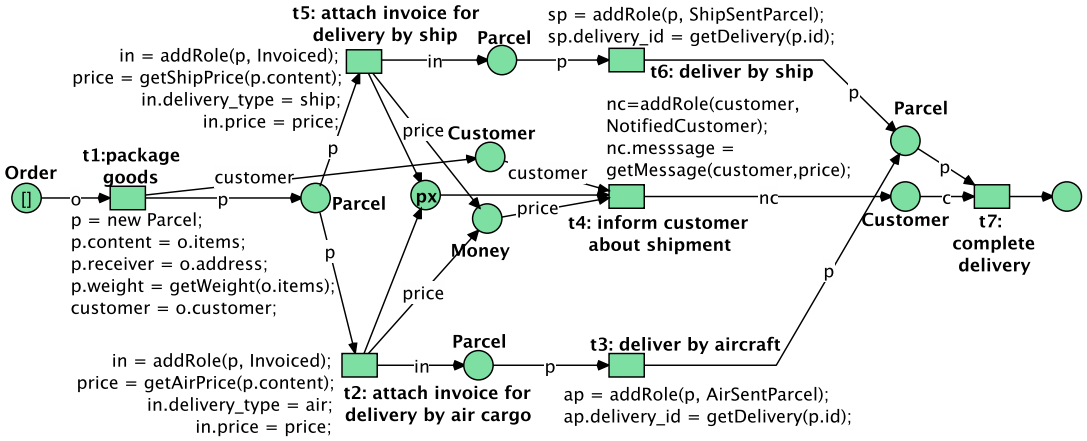


Abbildung 6.12: Ein zu sequentialisierendes Dienstbeschreibungsnetz

Das Netz in Abbildung 6.12 stellt die Abfertigung und Auslieferung eines Pakets dar. Zunächst wird ein neues Paket erzeugt und Inhalt und Empfänger des Pakets gesetzt. Anschließend erfolgt entweder ein Versand per Schiff oder per Luftfracht, wobei zuvor die entsprechenden Rechnungen berechnet werden. Der Kunden wird dann nebenläufig zur Versendung über das Verschiffen und den Preis informiert. Abschließend wird die Auslieferung abgeschlossen und der Ablauf beendet.

Transition t_1 kopiert die Bestelldaten des Orders und setzt die entsprechenden Parameter des Pakets. Dieses erhält dann je nachdem, ob es per Schiff oder per Luftfracht verschickt wird, das dynamische Konzept *Invoiced* mit dem entsprechenden

Preis und der Zustellungsart. Der Kunde erhält daraufhin das dynamische Konzept *NotifiedCustomer* und wird über die Zustellung informiert. Nebenläufig hierzu erhält das Paket eine Zustellungsnummer und wird ausgeliefert.

Das zugrundeliegende Netz besitzt zwei minimale T-Invarianten, wenn man den Abschluss bildet. Diese sind durch $[t_1, t_4, t_5, t_6, t_7, t^*]$ und durch $[t_1, t_2, t_3, t_4, t_7, t^*]$ gegeben. Eine Sequentialisierung des zugrundeliegenden Netzes lässt sich durch das Verfahren 6.8 in Unterabschnitt 6.2.2 erreichen. Diese muss dann durch das Verfahren 6.16 erweitert werden, so dass nicht nur dieselben Transitionen in der Sequentialisierung existieren, sondern auch die Datenstellen vorhanden sind. Dies ist in Abbildung 6.13 dargestellt. In diesem Fall sind zur Sequentialisierung lediglich die Kanten (t_3, p_x) und (t_6, p_x) hinzugekommen, während die Kanten (t_2, p_x) und (t_5, p_x) entfernt wurden. Dass das Netz eine Quasi-Zustandsmaschine ist, ist dabei nicht sofort zu erkennen. Durch das Entfernen der Datenstellen im zugrundeliegenden Netz erkennt man jedoch leicht, dass es sich bei dem Netz um eine Quasi-Zustandsmaschine handelt. Dies ist in Abbildung 6.14 zu sehen.

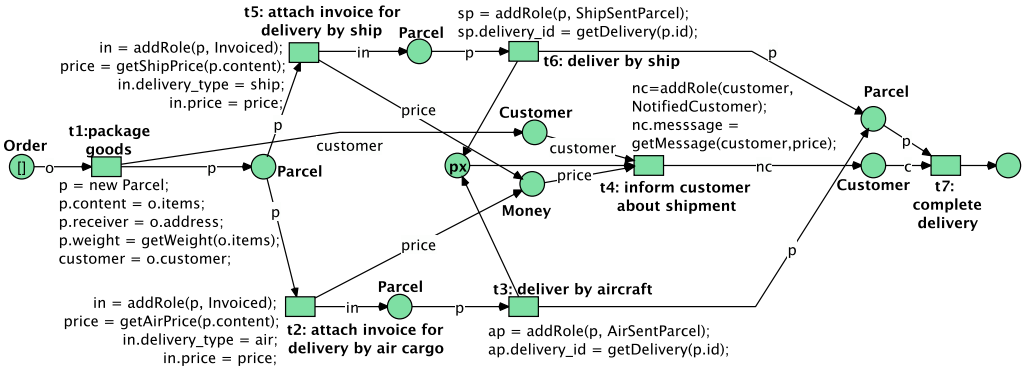


Abbildung 6.13: Die Sequentialisierung des SD-Netzes aus Abbildung 6.12

Bei der Sequentialisierung bleiben die Datenstellen und die Transitionen erhalten und nach wie vor miteinander verbunden. Es wird folglich nur eine weitere Abhängigkeit zwischen den Transitionen eingeführt, die dafür sorgt, dass jede minimale T-Invariante nur noch auf genau eine Weise realisierbar ist. Dies ist besonders gut an der S-Komponente (ohne t^*) des QSWF-Netzes zu sehen, die in Abbildung 6.14 dargestellt ist.

Da zu jeder vorher möglichen Schaltfolge nach wie vor eine Permutation dieser Schaltfolge möglich ist, ist es auch möglich, von einem gegebenen Initialzustand die gleichen finalen Endzustände zu erreichen, wie im Ursprungsnetz. Hierbei reduziert sich bereits in diesem einfachen Fall die Zahl der zu untersuchenden Schaltfolgen von vier auf zwei, was dieses Vorgehen für die Analyse und das Testen interessant macht. ◇

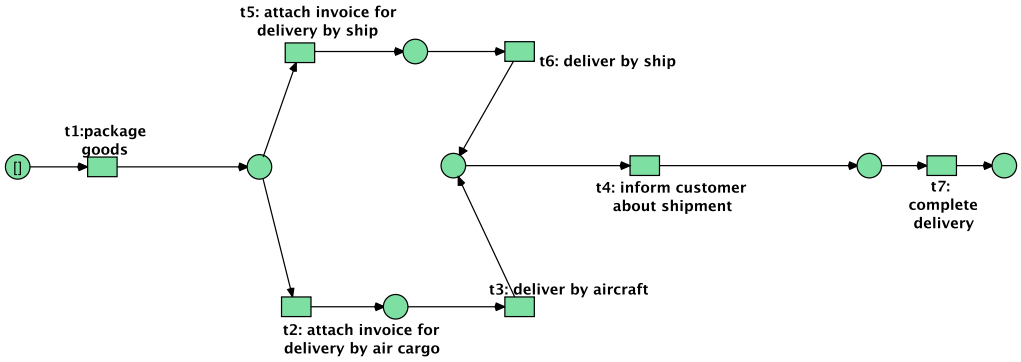


Abbildung 6.14: Die Zustandsmaschine des Abbildung 6.13 zugrundeliegenden Netzes

6.3.3 Sequentialisierung und Analyse

Die Anwendung der Sequentialisierung umfasst in erster Linie eine schnellere Analyse und beschleunigte Ausführung von Testszenarien, da lediglich ein Automatenmodell (eine Zustandsmaschine) überprüft werden muss. Da keinerlei Lokationen reserviert werden müssen, ist es für elementare Dienstbeschreibungsnetze auch nicht mehr notwendig, dass ein Anweisungsblock belegungskonsistent ist. Somit kann jede Anweisung eines Anweisungsblocks zu einer eigenen Transition werden. Auf diese Weise lassen sich elementare Dienstbeschreibungsnetze in algebraische Netze überführen. Dies soll hier kurz skizziert werden.

Vergegenwärtigt man sich Abbildung 4.7 noch einmal, so lässt sich durch diese nun das Verhalten elementarer Dienstbeschreibungsnetze auf algebraische Netze übertragen. Da klassische algebraische Netze keine Ordnung auf den Sorten zulassen, muss diese ebenso wie die dynamischen Sorten zuvor entfernt werden. Da die Typsicherheit durch das Dienstbeschreibungsnetz gegeben ist, lässt sich dieser Schritt ohne weiteres später wieder rückgängig machen. Aufbauend auf Goguen und Meseguer (1992) lässt sich hierfür ein Vergissfunktor auf den Zuständen verwenden, wie im Anschluss an Definition 3.21 skizziert wurde.

Jede Sorte der Zustandssignatur $\Sigma^{\mathfrak{Q}} = (S, \preceq, \Omega)$, die nicht der statischen Signatur angehört, wird zunächst auf eine spezielle Sorte $\mathfrak{C}$ abgebildet. Ebenso wird für Operatoren, die ein entsprechendes Sortensymbol in der Domäne besitzen, dieses durch $\mathfrak{C}$ ersetzt. Auf diese Weise entfällt die Hierarchie auf den Sorten der Systemsignatur. Allen Objekten des Zustandes, die nicht der statischen Algebra angehören, wird dann jeweils eine Konstante der Sorte $\mathfrak{C}$ zugewiesen, wodurch sich statt einer Algebra zur Interpretation des Zustandes eine Spezifikation im Netz verwenden lässt, wie dies für algebraische Netze klassischerweise der Fall ist. Der Systemzustand ist eine Konstante Q einer speziellen Sorte $s_{\mathfrak{Q}}$ der Spezifikation des algebraischen Netzes. Zur Veränderung des Zustandes sind die Operatoren `get` und `set` definiert,

die den Wert einer Lokation verändern. Um die Sorten der statischen Signatur zu respektieren, ist hier eine Nutzung spezifischer Operatoren für die entsprechenden Sorten denkbar, wie `getInt` oder `setString`. Die `set`-Operatoren bilden von einem Zustand (aus $s_{\mathfrak{Q}}$) auf einen Zustand (aus $s_{\mathfrak{Q}}$) ab, die `get`-Operatoren liefern hingegen den Wert einer Lokation zurück. Abbildung 6.15 veranschaulicht das Vorgehen anhand einer Update- und einer Zuweisungsanweisung.

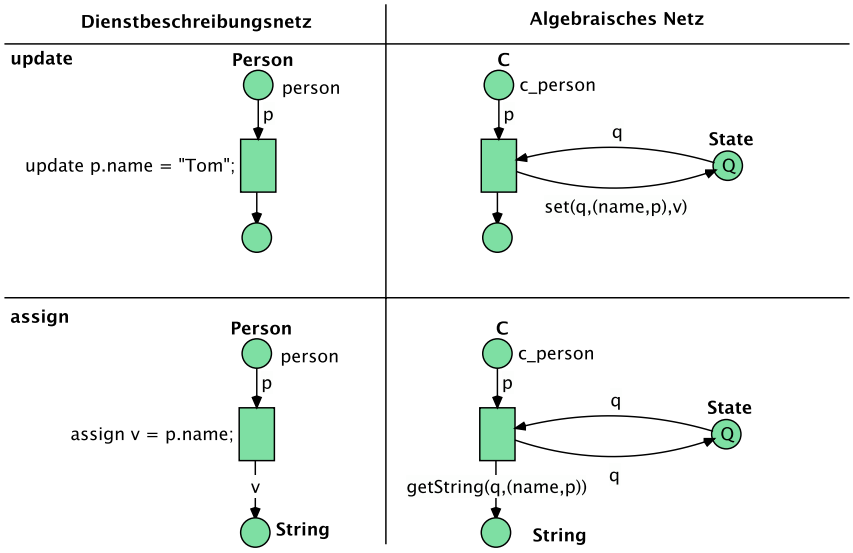


Abbildung 6.15: Veränderung des Zustandes

Die Sorte *Person* ist im algebraischen Netz der Sorte $\mathfrak{C}$ gewichen, die hier durch ein C dargestellt wurde. Bei der Update-Anweisung wird der Zustand verändert, weswegen ein durch die `set`-Anweisung veränderter Zustand an der Ausgangskante der Transition zu finden ist. Bei der Zuweisungsanweisung wird der Zustand lediglich genutzt, um einen Wert abzufragen, dies geschieht über den `getString`-Operator, der dann eine Zeichenkette zurückliefert. Hierbei ist sichergestellt, dass der Operator *name* tatsächlich auf eine Zeichenkette abbildet, da dieser Operator in der Signatur des Dienstbeschreibungsnetzes entsprechend definiert wurde.

Die Operatoren der Systemsignatur werden bei diesem Vorgehen zu Symbolen, die angeben, an welcher Stelle (Lokation) der Wert eines Zustandes abgefragt oder geändert werden soll. Eine geeignete Spezifikation für das algebraische Netz sollte dabei in ihren Gleichungen angeben, dass $\texttt{get}(\texttt{set}(\mathfrak{Q}, l, v), l) = v$ ist. Sorten-Updates müssen in diesem Fall ebenfalls auf herkömmliche Updates reduziert werden. So ließe sich eine Sortenlokation durch eine Abbildung auf einen booleschen Wert angeben, so dass $\texttt{get}(Q, (Person, x)) = true$ ist, wenn x im Zustand Q von der Sorte Person ist.

An dieser Stelle soll die Idee, Dienstbeschreibungsnetze durch eine Überführung

in algebraische Netze zu analysieren, nicht weiter vertieft werden, da das prinzipielle Vorgehen anhand obiger Skizzierung erkennbar ist, die genaue Überführung dann jedoch wiederum abhängig vom für die Analyse verwendeten Werkzeug ist. Insbesondere die Möglichkeit der Termersetzung ist zur Modellierung der Algebra notwendig, was auch den Einsatz von Termersetzern wie Maude (vgl. Maude, 2009) interessant macht. Hierbei lässt sich die monoidale Struktur von Petrinetzen ausnutzen, wie von Meseguer und Montanari (1990) und Stehr (2002) ausgeführt. Für algebraische Netze existieren eine Reihe von Model Checkern wie Ina (INA, 2003), Maria (Mäkelä, 2000) oder Lola (Schmidt, 2003). Aus diesem Grund ist es interessant, Dienstbeschreibungsnetze in algebraische Netze überführen zu können.

Entscheidend bei der Überführung ist jedoch die Verwendung einer Spezifikation, die zwei Terme gleich setzt, wenn diese auch in einem Systemzustand gleich sind. Dies bedeutet auch, dass die statische Algebra auf den statischen Sorten initial für diese Spezifikation sein muss. In diesem Fall lässt sich dann ein Zustand eines Dienstbeschreibungsnetzes anhand einer Markierung kodieren und in einem algebraischen Netz verwenden. Schaltet eine Transition des algebraischen Netzes, so lässt sich das Ergebnis wiederum als Zustand interpretieren. Dies entspricht somit einer Veränderung durch Kantenanschriften im Dienstbeschreibungsnetz. Abbildung 6.16 stellt diesen Zusammenhang dar.

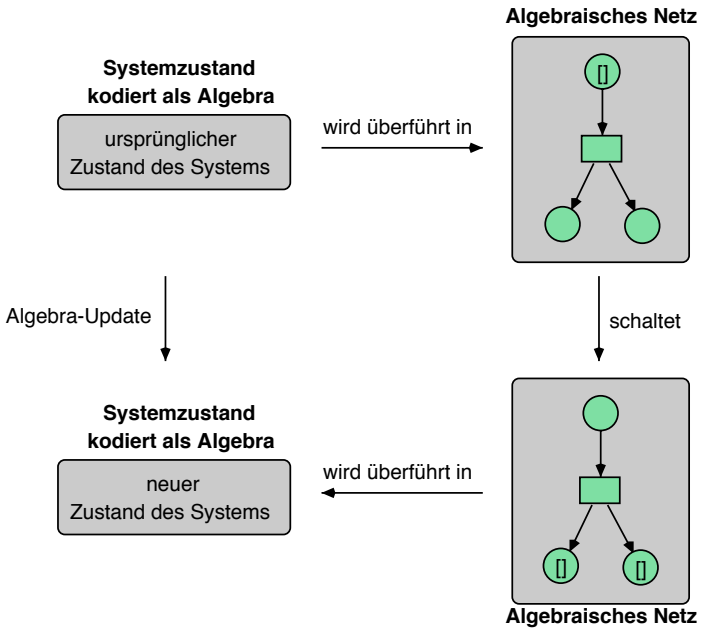


Abbildung 6.16: Zustand als Algebra und als Netz

Insbesondere im Zusammenhang mit initialbeschrifteten Netzen ist es so beispielsweise interessant, ob für ein Dienstbeschreibungsnetz gewährleistet ist, dass

ein Term niemals einen bestimmten Wert (wie etwa $\perp\!\!\!\perp$) einnimmt. Dies lässt sich jedoch im Allgemeinen nur für Netze mit endlichem Zustandsraum analysieren. Durch die Eigenschaft, in eine Quasi-Zustandsmaschine überführbar zu sein, ist die Analyse für sequentialisierbare Free-Choice-Dienstbeschreibungsnetze deutlich effizienter möglich.

Zur symbolischen Analyse lassen sich verschiedene Techniken wie binäre Entscheidungsdiagramme (BDDs) (vgl. Yavuz-Kahveci u. a., 2001) oder Bitvektoren (vgl. Shankar, 2000) zur Repräsentation der möglichen symbolischen Zustände einsetzen. Eine detailliertere Darstellung dieser Möglichkeiten findet sich bei Jhala und Majumdar (2009).

6.4 Objektzustände und modulare Netzstrukturen

Die bisherigen Betrachtungen zur Analyse von Dienstbeschreibungsnetzen haben in erster Linie die funktionalen Aspekte und den Ablauf betrachtet. Neben diesen Aspekten ist auch die Verhaltensanalyse von Diensten im Zusammenspiel mit den veränderten Objekten ein wichtiger Aspekt. Interessant ist hierbei einmal die Analyse von Objekten, die in einem Dienst verwendet werden, und den Zuständen, die sie einnehmen können, und zum anderen die Koordination verschiedener Dienste, die an einer Konversation beteiligt sind. In beiden Fällen ist eine Überprüfung, ob ein Dienst dem vorgegebenen Verhalten entspricht, sinnvoll. Dies soll im Folgenden näher betrachtet werden.

6.4.1 Zustandsübergänge bei Objekten

Wie in Abschnitt 2.4.2 angeschnitten, können Objekte ihren Typ ändern, was als eine Änderung des Objektzustandes aufgefasst werden kann. Hierbei wird zwischen Attributänderungen und relevanten Zustandsänderungen unterschieden. Obwohl der Übergang im Allgemeinen fließend ist, wird hier angenommen, dass Zustandsänderungen stets eine Rollenänderung mit sich bringen. Bei einer solchen Änderung des Objektzustandes ist es häufig sinnvoll, nicht alle Zustandsänderungen zuzulassen, sondern die Menge der möglichen Zustandsänderungen einzuschränken. Zur Darstellung des Objektzustands eignen sich wiederum höhere Petrinetze, da sie es erlauben, Netze in Netzen darzustellen. Dieser auf Valk (1998) zurückgehende Ansatz ermöglicht es, den Objektzustand zu modellieren. Ansätze, objektorientierte Systeme mit Petrinetzen umzusetzen, finden sich beispielsweise bei Moldt (1996).

Objektzustände in Dienstbeschreibungsnetzen

Ein einfaches Beispiel möglicher Zustandsübergänge eines Auftrags ist in Abbildung 6.17 gegeben. Ein Auftrag kann in diesem Beispiel die Zustände **ordered**,

payed und **shipped** einnehmen. Dabei kann ein Auftrag nur verschickt werden, wenn er zuvor bereits bestellt und bezahlt wurde. Hieraus ergeben sich dann Konsequenzen für mögliche Workflows, so dass die Zustände eines Objektes häufig eng mit der Modellierung der Workflows verwandt sind, die die Zustände des Objektes verändern.

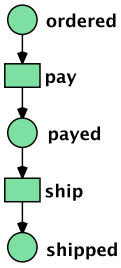


Abbildung 6.17: Zustände eines Auftrags

Da Zustandsänderungen eines Objektes nicht spontan durch das Objekt selber erfolgen können, muss jede Zustandsänderung durch das Workflownetz initiiert werden. Es darf also keinerlei Transitionen im Netz des Objektes zwischen zwei vom Workflownetz geschalteten Transitionen in diesem Netz geben, die nicht vom Workflownetz geschaltet werden. Im Beispiel darf das Objekt nicht selbstständig in den Zustand **payed** wechseln, ohne dass dies durch einen Workflow veranlasst wurde. Dies stellt einen wesentlichen Unterschied zu den klassischen Objektnetzen von Valk (1998) dar. Ein Workflow, der das Objekt verändert, ist vereinfacht in Abbildung 6.18 gegeben. Ein Auftrag wird gleichzeitig verschickt, und es wird der Bezahlvorgang bearbeitet.

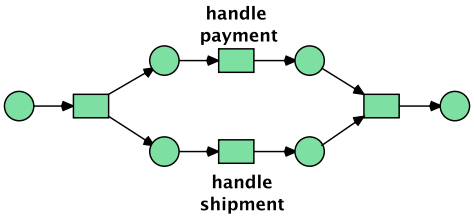


Abbildung 6.18: Bearbeitung eines Auftrags

An diesem Workflow ist zunächst nichts ungewöhnlich. Kombiniert man jedoch die möglichen Objektzustände mit der Abarbeitungssequenz im Workflow so ergibt sich, dass die modellierte Nebenläufigkeit im Workflow eine falsche gleichzeitige Abarbeitung des Verschickens und des Bezahlens suggeriert. Aus diesem Grund

ist der Workflow schlecht modelliert, da aufgrund der Objektzustände stets eine sequentielle Abarbeitung stattfindet.

Durch die Verwendung von Rollen als dynamischen Konzepten können Rollenübergänge auch für die Konzepte von Dienstbeschreibungsnetzen eingeschränkt werden. Dies führt dann zu einem Diagramm, wie es in Abbildung 6.17 dargestellt ist. Auch wenn diese Rollenmigrationsdiagramme prinzipiell beliebig komplex sein können, so wird hier davon ausgegangen, dass sie lediglich einer Subklasse wohlstrukturierter Workflownetze entsprechen, da dies in vielen praktischen Fällen der Fall ist.

Indem man die Transitionen des Rollenmigrationsdiagramms mit den Transitionen verschmelzen lässt, die diese Rollenübergänge im Netz bewirken, erhält man ein neues Netz. Für das aktuelle Beispiel ist dieses Netz in Abbildung 6.19 dargestellt. Hier wird offensichtlich, dass das Netz keinerlei nebenläufig aktivierte Transitionen besitzt.

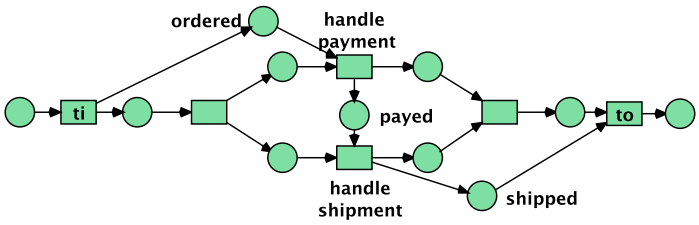


Abbildung 6.19: Zusammenführung von Ablauf und Objektzuständen

Bei der Zusammenführung der Netze wird das Objektnetz durch eine initiale Transition t_i zusammen mit dem Workflownetz initialisiert, und durch eine finale Transition t_o wird erreicht, dass lediglich die finale Stelle markiert ist.

In einem Dienstbeschreibungsnetz werden die Transitionen anhand der Anschriften verschmolzen. Gibt es mehrere *addRole*-Anschriften oder *removeRole*-Anschriften, die dasselbe Objekt verändern, so stellen diese Anschriften jeweils den entsprechenden Zustandsübergang des Objektes dar. Teilweise ist es auch sinnvoll für Attributänderungen eigene Objektzustände zu definieren, die dann bei der Änderung des entsprechenden Attributs überprüft werden. Dies geschieht dann bei einer entsprechenden *update*-Anweisung. Hierbei ist ohne eine aufwendige symbolische Analyse nicht immer ermittelbar, ob zwei Terme zum gleichen Objekt auswerten. Vielfach wird ein Objekt in einem Ablauf jedoch stets gleich referenziert, so dass in diesen Fällen eine Überprüfung möglich ist.

Neben der unterdrückten Nebenläufigkeit stellt die Verklemmung eine weitere unerwünschte Situation dar. Wählt man etwa die Bearbeitung des Auftrags in Abbildung 6.20a) so führt die gemeinsame Betrachtung von Objektzustand und Ablauf zu einem Netz, dass ein in $[i]$ unmarkiertes Siphon besitzt. Dies ist in in

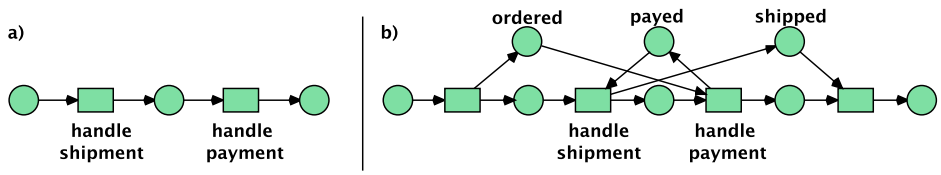


Abbildung 6.20: Auftragsbearbeitung führt zu Deadlock

Abbildung 6.20b) dargestellt.

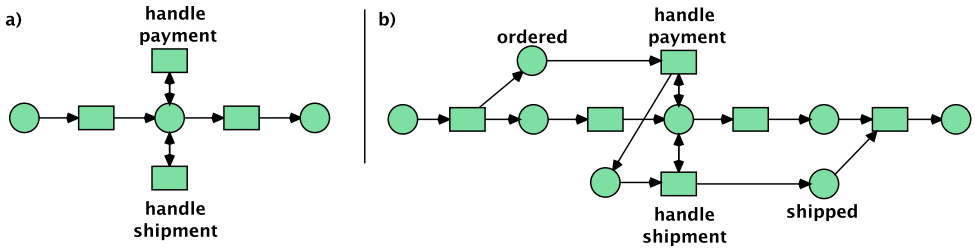


Abbildung 6.21: Auftragsbearbeitung führt zu neuem Verhalten

Darüber hinaus gibt es Situationen, in denen das resultierende Netz der gemeinsamen Betrachtung nicht mehr die T-Invarianten besitzt, die das ursprüngliche Workflownetz besaß. Dies ist etwa bei einer Bearbeitung der Aufgabe durch den Workflow in Abbildung 6.21a) der Fall. Dies wird ebenfalls als fehlerhafte Modellierung angesehen, da hierdurch wiederum der tatsächliche Ablauf stets vom modellierten Ablauf abweicht. Abbildung 6.21b) zeigt den erweiterten Ablauf.

Objektzustandsnetze

Objektzustandsnetze formalisieren die Zustände der Objekte, die in einem Dienstbeschreibungsnetz verwendet werden. Dazu wird jede Transition mit einem dynamischen Konzept annotiert, das beim Schalten dieser Transition hinzugefügt beziehungsweise abgezogen wird. Dies wird als Vektor über der Menge S^{dyn} notiert. Dabei stellt der Wert -1 ein abgezogenes Konzept und der Wert $+1$ ein hinzugefügtes Konzept dar.

Als Netze zur Modellierung der Zustände dienen *elementar-wohlstrukturierte Workflownetze*, die eine Subklasse der wohlstrukturierten Workflownetze darstellen. Sie lassen sich nur durch die Regeln a), b), c), d) und f) aus Abbildung 5.7 auf die Stelle i reduzieren, wobei p_2 in Regel a) nur eine Eingangskante und t_1 in der Regel b) lediglich eine Ausgangskante besitzen darf. Es geht direkt aus den

Reduktionsregeln hervor, dass diese Netzklasse stets sicher und korrekt sowie S- und T-überdeckbar ist.

Definition 6.21 (Objektzustandsnetz)
Zu einem Konzept $\mathfrak{c} \in S^{stat}$ ist ein *Objektzustandsnetz* $\mathcal{O}_\mathfrak{c} = (\mathcal{N}_O, \rho_\mathfrak{c})$ gegeben durch ein elementar-wohlstrukturiertes Workflownetz $\mathcal{N}_O$ und durch eine Abbildung $\rho_\mathfrak{c} : T_O \setminus (\{t_i\} \cup T_F) \to \{-1, 0, 1\}^{|S^{dyn}|}$, die jeder Transition aus T_O einen Vektor über die dynamischen Sorten zuweist, wobei für alle $t \in T_O \setminus (\{t_i\} \cup T_F)$ gilt

- $|||\rho_\mathfrak{c}(t)||| = 1$ und (Nur ein Element wird verändert.)
- $\rho_\mathfrak{c}(t)(\mathfrak{c}_1) \neq 0$ impliziert, dass $\mathfrak{c}_1 \preceq \mathfrak{c}$, dass $\mathfrak{c}_1 \neq \mathfrak{c}$ und dass es kein $t' \in T_O$ gibt mit $\rho_\mathfrak{c}(t)(\mathfrak{c}_1) = \rho_\mathfrak{c}(t')(\mathfrak{c}_1)$ und $t \neq t'$. (Jedes veränderte Konzept muss ein echtes Subkonzept von $\mathfrak{c}$ sein kann nur einmal so verändert werden.)

Die Transition t_i stellt dabei die Initialisierungstransition dar und die Menge $T_F \subseteq T_O$ ist die Menge der Finaltransitionen, so dass $i^\bullet = t_i$ und ${}^\bullet o = T_F$ gilt. ⧫

Für jedes Konzept bietet ein Objektzustandsnetz die Möglichkeit, die möglichen Zustandsübergänge einzuschränken. Jeder Term in einem Dienstbeschreibungsnetz, der ein solches Objekt repräsentiert, stellt somit gleichzeitig ein markiertes Objektzustandsnetz dar, auf das durch den Term referenziert wird. Dieses Modell ist dem Prinzip der Netze in Netzen sehr nahe, bietet hier jedoch den Unterschied, dass Transitionen nur vom Systemnetz, also dem übergeordneten Workflownetz, geschaltet werden können[3].

Da Transitionen nur gemeinsam feuern können, lässt sich diese Verschachtelung zur Analyse aufbrechen, indem die Transitionen des Objektzustandsnetzes mit den Transitionen des Dienstbeschreibungsnetzes verschmolzen werden, sofern sie die gleichen Rollen ändern. Aufgrund der Bedingung $|||\rho_\mathfrak{c}(t)||| = 1$ ist jeder Transition des Objektnetzes eine eindeutige Änderung zugewiesen, so dass die entsprechenden Änderungen im Dienstbeschreibungsnetz $\mathcal{SDN}$ eindeutig einer Transition aus T_O zugewiesen werden können. Der Einfachheit halber wird dabei davon ausgegangen, dass stets nur eine Sorte eines Terms verändert wird. Sollten sich mehrere Sorten ändern, so ist es möglich, Sequenzen durch die Regel a) oder b) in Abbildung 5.7 im Objektzustandsnetz zusammenzufassen, wobei die entsprechenden Vektoren addiert werden müssen. Andererseits können durch die Regeln in Unterabschnitt 5.4.3 auch mehrere Konzeptänderungsanweisungen in einem Dienstbeschreibungsnetz auf verschiedene Transitionen verteilt werden.

[3] Folgt man der Definition der *Unary Elementary Object Systems* von Valk (1998), so ist die dortige Relation ρ eine Abbildung der Transitionen des Objektnetzes auf die Transitionen des Systemnetzes. Aufgrund der Ähnlichkeit wurde hier das Symbol $\rho_\mathfrak{c}$ gewählt.

Ein Objektzustandsnetz ist korrekt, wenn es keine Schaltfolge gibt, die zu einem negativen Wert führt, da dann ein Konzept entfernt werden würde, bevor es hinzugefügt wurde, und wenn es keine Schaltfolge gibt, die die Sortenhierarchie verletzt. Zudem wird verlangt, dass eine markierte Stelle stets eine bestimmte Sortenhierarchie repräsentiert, so dass zwei unterschiedliche Schaltfolge, die zur selben Markierung führen, dieselben Konzeptveränderungen repräsentieren müssen. Um dies zu formalisieren wird zu einer Schaltfolge σ der Vektor der Konzeptveränderungen $\rho_{\mathfrak{C}}(\sigma)$ angeben durch den Vektor

$$\rho_{\mathfrak{C}}(\sigma) = \sum_{j=1}^{N_\sigma} \rho_{\mathfrak{C}}(\sigma[j])$$

Dieser Vektor stellt für korrekte Objektzustandsnetze einen positiven Vektor dar, der jedem Konzept einen Wert aus $\{0, 1\}$ zuweist. Folglich kann dieser Vektor für korrekte Objektzustandsnetze wiederum als charakteristische Funktion über die Konzepte interpretiert werden.

Definition 6.22 (Korrektheit von Objektzustandsnetzen)
Ein zu einem Konzept $\mathfrak{c} \in S^{stat}$ gegebenes Objektzustandsnetz $\mathcal{O}_{\mathfrak{c}}$ ist *korrekt* gdw. für alle Schaltfolgen σ gilt, dass

- $\rho_{\mathfrak{C}}(\sigma) \in \{0,1\}^{|S^{dyn}|}$, (Abgezogene Sorten müssen zuvor hinzugefügt werden.)
- $\rho_{\mathfrak{C}}(t)(\mathfrak{c}_1) \neq 0$ und $\mathbf{m} \xrightarrow{t}$ für $[i] \xrightarrow{\sigma} \mathbf{m}$ impliziert $\rho_{\mathfrak{C}}(\sigma)(\mathfrak{c}_2) = 1$ für alle $\mathfrak{c}_1 \preceq \mathfrak{c}_2 \preceq \mathfrak{c}$ mit $\mathfrak{c}_2 \neq \mathfrak{c}$, (Wird ein Konzept verändert, so muss das Objekt auch sämtliche Superkonzepte bzgl. der Ordnung $\preceq$ besitzen.)
- $\rho_{\mathfrak{C}}(t)(\mathfrak{c}_2) = -1$ und $\mathbf{m} \xrightarrow{t}$ für $[i] \xrightarrow{\sigma} \mathbf{m}$ impliziert $\rho_{\mathfrak{C}}(\sigma)(\mathfrak{c}_1) = 0$ für alle $\mathfrak{c}_1 \preceq \mathfrak{c}_2 \preceq \mathfrak{c}$ mit $\mathfrak{c}_1 \neq \mathfrak{c}_2$, (Wird ein Konzept entfernt, so muss es das bzgl. der Ordnung $\preceq$ kleinste Konzept eines Objektes sein.)
- $[i] \xrightarrow{\sigma} \mathbf{m}$ und $[i] \xrightarrow{\sigma'} \mathbf{m}$ impliziert $\rho_{\mathfrak{C}}(\sigma) = \rho_{\mathfrak{C}}(\sigma')$ oder $\mathbf{m} = [o]$. (Führen zwei Schaltfolge zu einer Markierung, so ist auch die Veränderung durch die Schaltfolgen gleich, oder die Markierung ist die finale Markierung.)

♦

Die Korrektheit besagt, dass das Objektzustandsnetz nur Rollenveränderungen modellieren darf, die tatsächlich auftreten können, und dass die Superrollen einer hinzugefügten Rolle ebenfalls hinzugefügt werden müssen beziehungsweise die Subrollen einer entfernten Rolle ebenfalls entfernt werden müssen. Ebenso müssen entfernte Rollen vorher hinzugefügt worden sein. Auf diese Weise bleibt die Rollenhierarchie stets intakt.

Dadurch, dass jede Markierung einen eindeutigen Objektzustand darstellt, ist die Netzstruktur weiter eingeschränkt, da jedes Hinzufügen und jedes Entfernen eines Konzepts nur einmal vorkommen darf. Dies führt dazu, dass die finale Stelle o in vielen Fällen die einzige Stelle des Objektzustandsnetzes ist, die einen Vorbereich mit mehr als einer Transition besitzt.

Ein Objektzustandsnetz wird an ein Objekt der Algebra gebunden. Jede Transition, die den Zustand dieses Objektes verändert, verändert auch den Zustand des Netzes, sofern der entsprechende Zustandsübergang im Netz modelliert ist. Durch die Verschmelzung von Objektzustandsnetz und Dienstbeschreibungsnetz kann überprüft werden, ob ein Dienstbeschreibungsnetz objektkonform mit einem Objekt ist.

Definition 6.23 (Verschmelzung von Objektzustandsnetzen)
Gegeben ein sicheres elementares Dienstbeschreibungsnetz $\mathcal{SDN}$ und eine Menge M_X von Term-Transitions-Paaren der Art (τ, t), so dass in jedem erreichbaren Netzzustand Q gilt, dass die Aktiviertheit von t in Q impliziert, dass der Term τ unter jeder Variablenbelegung, die t aktiviert, zu q auswertet für ein $q \in S_Q$. Sei zudem an jeder Transition lediglich eine Konzeptänderungsanweisung über diesem Term definiert und sei $(\mathcal{N}_O, \rho_{\mathfrak{C}})$ das Objektnetz zur Sorte $\mathfrak{c}$ des Terms τ.
Sei $\mathfrak{tr}_\rho : T_O \to 2^T$ eine Abbildung mit

$$\begin{aligned}\mathfrak{tr}_\rho(t) = \quad & \{t' \in T \mid (\mathit{addRole}(\tau, \mathfrak{c}) \in \mathbf{cmd}(t') \wedge (\tau, t) \in M_X \wedge \rho_{\mathfrak{C}}(t)(\mathfrak{c}) = 1\} \cup \\ & \{t' \in T \mid (\mathit{removeRole}(\tau, \mathfrak{c}) \in \mathbf{cmd}(t') \wedge (\tau, t) \in M_X \wedge \rho_{\mathfrak{C}}(t)(\mathfrak{c}) = -1\}\end{aligned}$$

Dann ist die *Verschmelzung* eines Objektzustandsnetzes $\mathcal{O}_{\mathfrak{c}}$ mit einem Dienstbeschreibungsnetz $\mathcal{SDN}$ gegeben durch ein Workflownetz $\mathcal{N}_\rho = (P_\rho, T_\rho, F_\rho)$ mit

$$\begin{aligned}P_\rho &\stackrel{\text{def}}{=} P \cup P_O \cup \{i_\rho, o_\rho\} \\ T_\rho &\stackrel{\text{def}}{=} T \cup \{t_{i,\rho}, t_{o,\rho}\} \\ F_\rho &\stackrel{\text{def}}{=} F \cup \{(i_\rho, t_{i,\rho}), (t_{i,\rho}, i), (o, t_{o,\rho}), (t_{o,\rho}, o_\rho)\} \cup \\ &\quad \{t_{i,\rho}\} \times t_i^{\bullet_{(\mathcal{N}_O)}} \cup \{(o_O, t_{o,\rho})\} \cup \\ &\quad \{(p, t) \in P_O \times T \mid (p, t') \in F_O \wedge t \in \mathfrak{tr}_\rho(t')\} \cup \\ &\quad \{(t, p) \in T \times P_O \mid (t', p) \in F_O \wedge t \in \mathfrak{tr}_\rho(t')\}\end{aligned}$$

♦

Betrachtet man das entstehende Workflownetz wiederum als einem Dienstbeschreibungsnetz zugrundeliegendes Netz, so lassen sich sämtliche Anschriften des Dienstbeschreibungsnetzes auf dieses Workflownetz übertragen, da die Knoten- und Kantenmengen lediglich erweitert werden. Ergänzt man zudem die entsprechenden Anschriften für die Transitionen t_i und t_o sowie für deren Stellen im Vor- und Nachbereich, so ergibt sich wiederum ein elementares Dienstbeschreibungsnetz, was im folgenden Beispiel dargestellt ist.

Beispiel 6.2: Die Verschmelzung von Dienstbeschreibungsnetz und Objektzustandsnetz soll an einem Beispiel illustriert werden. Das Dienstbeschreibungsnetz in Abbildung 6.22 stellt eine detailliertere Umsetzung des Workflownetzes in Abbildung 6.18 dar, wobei lediglich die Konzeptänderungsanweisungen dargestellt wurden und auf die Update-Anweisungen verzichtet wurde.

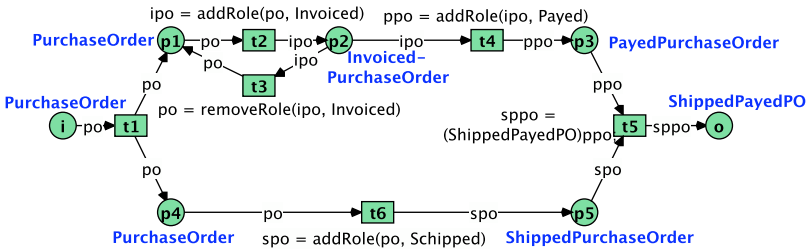


Abbildung 6.22: Ein Dienstbeschreibungsnetz mit Konzeptänderungsanweisungen, das das Workflownetz aus Abbildung 6.18 detaillierter darstellt

Verschmilzt man das Netz in Abbildung 6.22 mit dem in Abbildung 6.17 dargstellten Objektzustandsnetz, so erhält man das Netz in Abbildung 6.23. Das in Abbildung 6.17 dargestellte Objektzustandsnetz besitzt dabei die Abbildung $\rho_{\mathfrak{C}}(t)$ mit den Werten $\rho_{\mathfrak{C}}(pay)(Payed) = +1$ und $\rho_{\mathfrak{C}}(ship)(Shipped) = +1$. Da die Rollen im Dienstbeschreibungsnetz aus Abbildung 6.22 an den Transitionen t_3 und t_4 hinzugefügt werden, ergibt sich die in Abbildung 6.23 dargestellte Verschmelzung.

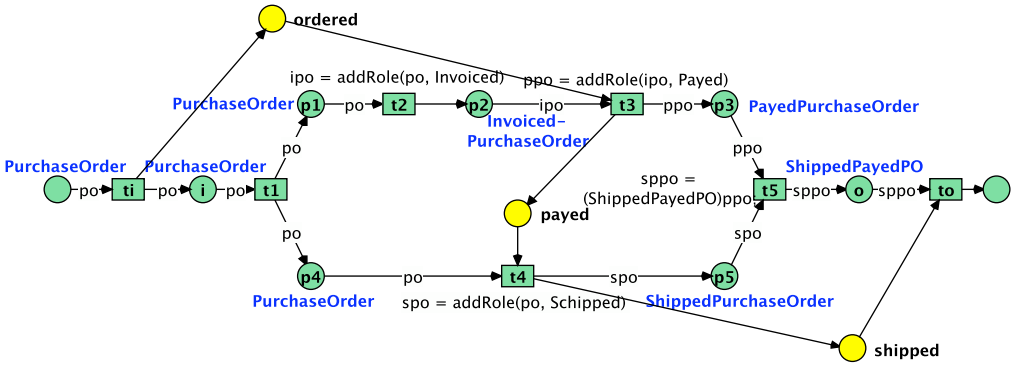


Abbildung 6.23: Das Dienstbeschreibungsnetz aus Abbildung 6.22 verschmolzen mit dem Objektzustandsnetz aus Abbildung 6.17

Darüber hinaus gibt es eine Kante von der Initialisierungstransition t_i zum Startzustand *ordered* und vom finalen Zustand *shipped* zur Endtransition t_o, so dass das gesamte Netz ein Workflownetz darstellt. ◇

Bei der Verschmelzung bleibt vielfach ein Rumpfnetz von $\mathcal{O}_\mathfrak{c}$ über, das durch die Reduktionsregeln aus Abbildung 5.7 reduziert werden kann. Dieses Rumpfnetz repräsentiert nicht erreichte Zustände, da ein Objekt nicht notwendigerweise durch ein Dienstbeschreibungsnetz in einen Endzustand überführt werden muss. So würde ein Bezahldienst das Objekt im Zustand *payed* belassen, was keinen Fehler darstellt. Zur Reduktion wird die Menge T_R gebildet, die sämtliche Transitionen aus T_O enthält, die nicht verschmolzen wurden. Durch die Reduktionsregeln werden dann sämtliche Transitionen aus $T_R \setminus T_F$ und sämtliche Stellen im Vor- und Nachbereich dieser Transitionen reduziert. Dabei darf jedoch niemals eine Transition aus $T_O \setminus T_O$ an der Reduktion beteiligt sein, so dass fehlende Zustandsübergänge erhalten bleiben.

Im verbleibenden Netz ist eine Schaltfolge von $[i]$ nach $[o]$, die in $\mathcal{N}$ möglich ist, ebenfalls möglich, sofern sie nicht die Zustandsübergänge des Objektes verletzt. In folgendem Satz werden zum Beweis dieser Aussage Schaltfolgen auch für sichere elementare Dienstbeschreibungsnetze verwendet anstatt Schaltmodi zu verwenden, da die Variablenbindung aufgrund der Sicherheit der Netze durch die Angabe der Transitionen bestimmt wird.

Satz 6.24 (Schaltfolgen der Verschmelzung)
Gegeben die Verschmelzung $\mathcal{N}_\rho$ des Objektzustandsnetzes $\mathcal{O}_\mathfrak{c}$ mit einem sicheren elementaren Dienstbeschreibungsnetz $\mathcal{SDN}$. Eine Folge von Schaltmodi σ_α von einem initialen Netzzustand zu einem finalen Netzzustand in $\mathcal{SDN}$ erzeugt nur Zustandsübergänge des Objektes c, die auch in $\mathcal{O}_\mathfrak{c}$ modelliert sind, gdw. $t_{i,\rho}\sigma t_f t_{o,\rho}$ für ein $t_f \in T_F$ eine Schaltfolge von $[i_\rho]$ nach $[o_\rho]$ in $\mathcal{N}_\rho$ ist, nachdem das Rumpfnetz reduziert wurde. (Die Schaltfolge σ stellt wiederum die Reduktion der Schaltmodi aus σ_α auf eine Folge von Transitionen dar.)

Beweis:
($\Rightarrow$) Sei σ_α eine Folge von Schaltmodi von einem initialen Netzzustand zu einem finalen Netzzustand in $\mathcal{SDN}$. Aufgrund der Sicherheit des Dienstbeschreibungsnetzes ist σ eine Schaltfolge in $\mathcal{N}$, wenn man die Folge auf Transitionen reduziert. Wäre $t_{i,\rho}\sigma t_{o,\rho}$ keine Schaltfolge in $\mathcal{N}_\rho$, so gäbe es eine Transition $t \in T$, die entweder nicht in $\textsc{Alph}(\sigma)$ ist, oder die durch eine Stelle aus P_O nicht in der Schaltfolge schalten kann. In ersterem Fall konnte eine Transition des Objektnetzes nicht reduziert werden, obwohl sie nicht verschmolzen wurde. Dann ist ein notwendiger Zustandsübergang entgegen der Voraussetzung nicht in $\mathcal{SDN}$ modelliert.

Da im anderen Fall keinerlei weitere Transitionen zu $\mathcal{O}_\mathfrak{c}$ hinzugekommen sind außer $t_{i,\rho}$ und $t_{o,\rho}$, von denen die erste stets in $[i_\rho]$ schalten kann, gibt es eine Teilschaltfolge σ' von σ, so dass $t_{i,\rho}\sigma'$ in $[i_\rho]$ aktiviert ist und zu einer Markierung $\mathbf{m}$ führt. In dieser Markierung würde in σ dann eine Transition t schalten, die jedoch eine Stelle aus P_O im Vorbereich besitzt, die nicht markiert ist. Dies ist jedoch nur möglich, wenn ein Zustandsübergang in $\mathcal{O}_\mathfrak{c}$ notwendig war, der in $\mathcal{SDN}$

nicht modelliert wurde.
($\Leftarrow$) Sei $t_{i,\rho}\sigma t_{o,\rho}$ eine Schaltfolge von $[i_\rho]$ nach $[o_\rho]$ in $\mathcal{N}_\rho$, nachdem das Rumpfnetz reduziert wurde. Dann ist σ auch eine Schaltfolge in $\mathcal{N}$. Zudem ist $t_i\sigma t_f$ reduziert auf die verschmolzenen Transitionen eine Schaltfolge in $\mathcal{O}_\mathfrak{c}$ von $[i_O]$ nach $[o_O]$. Da beide Schaltfolgen zur finalen Markierung führen, stellte σ eine Schaltfolge von $\mathcal{SDN}$ dar, die die Objektzustandsübergänge respektiert. □

Es sei darauf hingewiesen, dass das Ergebnis der Verschmelzung kein korrektes Workflownetz sein muss, da es sein kann, dass in diesem Netz durch das Schalten einer finalen Transition des Objektzustandsnetzes Marken im Vorbereich von Transitionen des Dienstbeschreibungsnetzes entfernt werden, die dort noch für das Schalten einer Transition notwendig sind. Korrektheit ist daher als Eigenschaft nicht gefordert. Interessant sind nun die Fälle, in denen es im Workflownetz Zustandsübergänge gibt, die so nicht im Objektzustandsnetz vorgesehen sind. Dies wird im folgenden Unterabschnitt diskutiert.

6.4.2 Analyse von Objektzustandsnetzen

Bei der Verschmelzung des Objektzustandsnetzes mit einem Dienstbeschreibungsnetz entsteht ein neues Workflownetz, das auf verschiedene Kriterien untersucht werden kann. Von besonderem Interesse sind dabei

nicht reduzierbare Transitionen, die nicht verschmolzen wurden, was ein Hinweis darauf ist, dass ein Zustandsübergang im Modell fehlt,

nicht i-überdeckende Siphons, die anzeigen, dass die Zustandsübergänge im Dienstbeschreibungsnetz die falsche Reihenfolge aufweisen,

veränderte T-Invarianten, die anzeigen, dass die modellierten Abläufe nicht mit den tatsächlichen Abläufen übereinstimmen oder dass die notwendigen Abläufe gar nicht möglich sind, und

verschwundene Nebenläufigkeit, die dann auftritt, wenn zwei nebenläufige Transitionen durch die Objektzustände serialisiert werden.

Alle vier dieser Unstimmigkeiten sind Indikatoren dafür, dass das Netz nicht so modelliert wurde, wie es bei den gegebenen Objektzuständen sinnvoll wäre. Die verschiedenen Unstimmigkeiten lassen sich durch die bereits angegebenen Verfahren erkennen. Nicht reduzierbare Transitionen können dabei offensichtlich durch die Anwendung der Reduktionsregeln erkannt werden. Die anderen Fälle sollen im Folgenden diskutiert werden.

Nicht i-überdeckende Siphons Durch das Entstehen neuer Siphons, die nicht die Stelle i überdecken, entstehen von $[i]$ in der Verschmelzung erreichbare Markierungen, von denen $[o]$ nicht mehr erreichbar ist, da in einem korrekten Workflownetz jedes Siphon die Stelle i beinhalten muss (vgl. Barkaoui u. a., 2007). Diese Siphons lassen sich durch den Algorithmus 5.1 berechnen, da jedes Siphon eine Falle des inversen Netzes ist. Da jedes Siphon im lebendigen Abschluss eines Workflownetzes die Stelle i überdecken muss, darf das maximale Siphon des Netzes nach dem Entfernen der Stelle i nur die leere Menge sein. Dies gilt auch für die verschmolzenen Netze, da es andernfalls keine unendliche Schaltfolge im Abschluss gibt, die es auch im nicht-korrekten Fall geben muss.

Veränderte T-Invarianten Die Veränderung der T-Invarianten lässt sich überprüfen, indem alle minimalen T-Invarianten einer Basis überprüft werden. Hierzu müssen die minimalen T-Invarianten um Transitionen aus T_O erweitert werden. Während der Wert der Transitionen aus T somit festgelegt ist, ist der Wert der Transitionen aus T_O dann zu berechnen. Gibt es keine entsprechenden Invarianten, so stellt dies einen Fehler dar, da die Abläufe im Dienstbeschreibungsnetz von den tatsächlichen Abläufen abweichen. Da sich jede minimale T-Invariante in einem korrekten Free-Choice-Workflownetz realisieren lässt, ein Teil dieser T-Invarianten jedoch bei der Verschmelzung verloren geht, stimmt das tatsächliche Verhalten der Abläufe nicht mit dem modellierten überein.

Fehlende Nebenläufigkeit Bei der fehlenden Nebenläufigkeit sind zwei Transitionen, die in $\mathcal{N}$ nebenläufig aktivierbar sind, durch die Betrachtung der Objektzustandsübergänge in $\mathcal{N}_\rho$ stets sequentiell. Dies lässt sich für Free-Choice-Dienstbeschreibungsnetze feststellen, indem für zwei verschmolzene Transitionen zunächst mittels des Verfahrens aus Satz 5.34 festgestellt wird, ob diese in $\mathcal{N}$ nebenläufig aktivierbar sind. Ist dies der Fall, so wird dies auch für die entsprechenden Transitionen in $\mathcal{O}_\mathfrak{c}$ überprüft. (Da sich elementar-wohlstrukturierte Netze in Free-Choice-Netze überführen lassen, kann dies ebenfalls mit dem Verfahren aus Satz 5.34 erfolgen.).

Weitergehende Analyse Eine Erweiterung besteht darin, die Rollen nicht immer explizit anzugeben, wie dies in den bisher vorgestellten Fällen der Fall war. So wäre auch eine Rolle *LargeAmount* denkbar, die automatisiert hinzugefügt wird, wenn etwa die Summe der Preise eine Grenze übersteigt. Um nicht jedesmal explizit im Workflow die Überprüfung der Preise modellieren zu müssen, kann die Rolle dann einfach hinzugefügt werden, wenn ein Attribut den Wert x übersteigt. Gegebenenfalls müssen in diesem Fall weitere Maßnahmen, wie etwa eine Überprüfung einer Bestellung durch eine weitere Person, erfolgen, die dann separat im Objektlebenszyklus modelliert und so überprüft werden können.

In der fortgeschritteneren Analyse sollte auch überprüft werden, ob eine Schaltfolge überhaupt ausführbar ist. Zudem ist es häufig sinnvoll, Schleifen abzuwickeln, die durch minimale T-Invarianten repräsentiert werden. So ist es in der Regel ausreichend, den ersten Durchlauf einer Schleife zu analysieren, um zu erkennen, ob die Zustandsübergänge eines Objektes respektiert werden. Aufwendiger wird die Analyse hingegen, wenn beispielsweise verschiedene Objekte einer Liste verschiedene Zustandsänderungen erfahren. Dies sind jedoch Analysen, die häufig nur mit Hilfe einer aufwendigeren symbolischen Analyse des Zustandsraums einhergehen können, so dass hier davon Abstand genommen werden soll.

6.4.3 Dienstinteraktionen

Da Dienstbeschreibungsnetze nicht nur einzelne Dienste, sondern auch zusammengesetzte Dienste darstellen sollen, ist es notwendig, dass sich Dienstbeschreibungsnetze kombinieren lassen. Dies wird durch die Verwendung kommunizierender Workflownetze ermöglicht, die spezielle Transitionen ausweisen, die dann durch weitere Netze ersetzt werden können.

Kommunizierende Workflownetze sind herkömmliche Workflownetze mit speziell ausgewiesenen Platzhaltertransitionen, die durch Workflownetze ersetzt werden können. Für Dienstbeschreibungsnetze werden Platzhaltertransitionen entsprechend durch Dienstbeschreibungsnetze als Teilnetze ersetzt, bis das Gesamtnetz ein Dienstbeschreibungsnetz ohne Platzhaltertransitionen ist. Für komplexere Interaktionen können Interaktionsmuster festgelegt werden, die angeben, wie mehrere Dienstbeschreibungsnetze miteinander interagieren. Diese Interaktionsmuster stellen die Beschreibung einer Verhaltensschnittstelle dar, die dann durch Verfeinerungsmechanismen adaptiert werden kann.

Kommunizierende Workflownetze haben dabei einen ähnlichen Aufbau wie die von Massuthe u. a. (2005) vorgestellten offenen Workflownetze. Diese sind wiederum eng verwandt mit den von Köhler-Bußmeier (2009c) vorgestellten Ablaufnetzen, die dort als Grundlage der dort definierten Dienstnetze dienen. Beide Konzepte werden hier noch weiter dadurch spezialisiert, dass die Randstellen nicht beliebig sein dürfen, sondern dass sie der Verfeinerung von Transitionen eines Workflownetzes entsprechen müssen. Ein solches Netz wird in Abgrenzung zu obigen Konzepten kommunizierendes Workflownetz genannt. Dabei sind die zu verfeinernden Transitionen Bestandteil der Netze, die später ersetzt werden, so dass das Gesamtnetz ein Workflownetz bleibt. (Siehe hierzu auch Lehmann (2003).)

Definition 6.25 (Kommunizierendes Workflownetz, Platzhaltertransition)
Ein Netz $\mathcal{N} = (P, T, F)$ heißt *kommunizierendes Workflownetz* (kWF-Netz) gdw.

- $\mathcal{N}$ ein Worklflownetz ist und

- es eine Menge von Transitionen $T_c \subseteq T$ gibt, so dass für alle $t \in T_c$ gilt $|{}^\bullet t| = |t^\bullet| = 1$.

Die Menge T_c wird als Menge der *Platzhaltertransitionen* bezeichnet. ⧫

Die Menge der Platzhaltertransitionen T_c wird später durch Dienstbeschreibungsnetze ersetzt. Dies geschieht anhand des in Definition 3.75 angegebenen Vorgehens, wobei Platzhaltertransitionen in Dienstbeschreibungsnetzen durch weitere Dienstbeschreibungsnetze ersetzt werden, in denen wiederum die Platzhaltertransitionen ersetzt werden. Auf diese Weise entstehen Interaktionsmuster, die der Zusammenfassung aller beteiligten Netze entsprechen, und die insgesamt ein Workflownetz darstellen. Dies soll im Folgenden detaillierter dargestellt werden.

Vom zugrundeliegenden Workflownetz übernimmt ein Dienstbeschreibungsnetz die beiden ausgezeichneten Stellen i und o. Die Typisierung dieser Stellen ergibt die Schnittstelle eines Dienstbeschreibungsnetzes, die als $\mathbf{I}(\mathcal{SDN})$ notiert wird und als $\mathbf{I}(\mathcal{SDN}) \stackrel{\text{def}}{=} (d(i), d(o))$ definiert ist. Neben dieser Schnittstelle ist zu jedem Dienstbeschreibungsnetz mit einer nicht-leeren Menge von Platzhaltertransitionen eine Menge von Schnittstellen definiert, die sich durch den Vor- und durch den Nachbereich einer Platzhaltertransition ergeben. Dabei ist zu jeder Platzhaltertransition $t \in T_c$ die Schnittstelle $\mathbf{I}(t) \stackrel{\text{def}}{=} (s_\text{i}, s_\text{o})$ gegeben durch $s_\text{i} = d(p_i)$ mit $\{p_i\} = {}^\bullet t$ und $s_\text{o} = d(p_o)$ mit $\{p_o\} = t^\bullet$.

Definition 6.26 (Kompatibilität und Ersetzung)
Seien $\mathcal{SDN}_i = (\Sigma^\mathbf{Q}, X_i, \mathcal{N}_i, d_i, \mathbf{cmd}_i, W_i, G_i)$ für $i \in \{1, 2\}$ zwei Dienstbeschreibungsnetze mit gleicher Zustandssignatur $\Sigma^\mathbf{Q}$ und disjunkten Knotenmengen. Die Schnittstelle $\mathbf{I}(t) = (s_i^t, s_o^t)$ einer Platzhaltertransition t des Dienstbeschreibungsnetzes $\mathcal{SDN}_1$ ist *kompatibel* mit der Schnittstelle des Dienstbeschreibungsnetzes $\mathbf{I}(\mathcal{SDN}_2) = (s_i^\mathcal{N}, s_o^\mathcal{N})$ gdw. $s_i^t \preceq s_i^\mathcal{N}$ und $s_o^\mathcal{N} \preceq s_o^t$ gilt.
Sind $\mathbf{I}(t)$ und $\mathbf{I}(\mathcal{SDN}_2)$ kompatibel, so kann t durch $\mathcal{SDN}_2$ *ersetzt* werden, was notiert wird als $\mathcal{SDN}_{1[t \to \mathcal{SDN}_2]} = (\Sigma^\mathbf{Q}, X_1 \cup X_2, (P', T', F'), d', \mathbf{cmd}', W', G')$, wobei

- $P' = P_1 \cup P_2 \setminus (\{i_2\} \cup t^\bullet)$, $T' = T_1 \cup T_2 \setminus \{t\}$ und
  $F' = F \cap (P' \times T' \cup T' \times P') \cup ({}^\bullet t \times i_2^\bullet) \cup (\{o_2\} \times t^{\bullet\bullet})$
- $d'(p) = \begin{cases} d_1(p) & \text{wenn } p \in P_1 \\ d_2(p) & \text{wenn } p \in P_2 \end{cases}$
- $\mathbf{cmd}'(t) = \begin{cases} \mathbf{cmd}_1(t) & \text{wenn } t \in T_1 \\ \mathbf{cmd}_2(t) & \text{wenn } t \in T_2 \end{cases}$
- $W'(p,t) = \begin{cases} W_1(p,t) & \text{wenn } t \in T_1 \\ W_2(p,t) & \text{wenn } t \in T_2 \end{cases}$ und $W'(t,p) = \begin{cases} W_1(t,p) & \text{wenn } t \in T_1 \\ W_2(t,p) & \text{wenn } t \in T_2 \end{cases}$

- $G'(t) = \begin{cases} G_1(t) & \text{wenn } t \in T_1 \\ G_2(t) & \text{wenn } t \in T_2 \end{cases}$

Hierbei ist i_2 die Eingangsstelle des Workflownetzes $\mathcal{N}_2$. ⧫

Bei der Ersetzung werden also die Eingangsstelle des zweiten Netzes, t und die Stellen im Nachbereich von t entfernt. Dabei bleiben alle Kanten erhalten, die nicht mit einem der entfernten Knoten verbunden sind. Die Kanten werden so angepasst, dass das resultierende Netz wiederum ein Workflownetz ist, wie dies auch in Definition 3.75 der Fall ist. Die Anschriften an den Kanten werden jeweils für die mit der Kante verbundenen Transition übernommen, so dass die Typisierung korrekt bleibt. Es lässt sich somit leicht überprüfen, dass folgendes Lemma gilt.

Lemma 6.27 (Schnittstellenersetzung erzeugt Dienstbeschreibungsnetz) Seien $\mathcal{SDN}_1$ und $\mathcal{SDN}_2$ Dienstbeschreibungsnetze mit gleicher Zustandssignatur $\Sigma^{\mathbf{Q}}$ und disjunkten Variablen- und Knotenmengen, und seien $\mathbf{I}(t)$ und $\mathbf{I}(\mathcal{SDN}_2)$ kompatibel für eine Platzhaltertransition von $\mathcal{SDN}_1$. Dann ist $\mathcal{SDN}_{1[t \to \mathcal{SDN}_2]}$ ein Dienstbeschreibungsnetz.

Die Ersetzung ist geringfügig anders als die Ersetzung in Definition 3.75, da auf diese Weise die Stellentypisierung beibehalten werden kann. Dennoch lässt sich leicht erkennen, dass sich die Ergebnisse aus Satz 3.76 direkt auf die Ersetzung von Platzhaltertransitionen übertragen lassen.

Bei der Ersetzung ist prinzipiell zwar eine rekursive Ersetzung durch das aufrufende Netz denkbar, jedoch wird hier davon ausgegangen, dass der Ersetzungsgraph keine Zyklen enthält. Somit ist jede Ersetzung endlich.

Ein wesentlicher Unterschied der Modellierung von Dienstbeschreibungsnetzen gegenüber Ansätzen, wie sie von Massuthe u. a. (2005) und von Köhler-Bußmeier (2009c) diskutiert werden, ist, dass Dienstbeschreibungsnetze nur Transitionen verfeinern. Ist eine Kommunikation über mehrere Dienstbeschreibungsnetze hinweg zustandsbehaftet, so muss dies anhand des speziellen Session-Tokens modelliert werden, es ändert sich also ein Attribut des Zustandes. Dies ist zwar weniger offensichtlich als die Modellierung durch Stellen, auf denen ein Token verbleibt. Sie ist jedoch näher an der tatsächlichen technischen Realisierung vieler dienstbasierter Systeme, in denen Dienste vielfach zustandslos sind. Hinzu kommt, dass verschiedene Dienstaufrufe meist unterschiedliche Protokolle des Dienstes ansteuern, die häufig im Dienst wiederverwendet werden können. In einer Abfolge wie Login, Informationsanfrage, Logout wird der Status, ob ein Nutzer eingeloggt ist, über den Dienst hinweg als Information vorgehalten. Die Teilabläufe Login und Logout lassen sich jedoch für andere Operationen des Dienstes (beispielsweise weitere Anfragen) wiederverwenden. Da ein solcher Session-Kontext wiederum ein Objekt des Systems darstellt, können obige Überprüfungen mittels der Objektzustandsnetze auch

für dieses Objekt erfolgen, so dass sichergestellt ist, dass jeder Aktion ein Login vorausgegangen ist.

Dienstbeschreibungsnetze mit gleicher Schnittstelle können nun in Klassen zusammengefasst werden. Eine solche Klassifizierung findet sich beispielsweise auch bei Köhler-Bußmeier (2009c), wo die Klassifikation anhand der Verhaltensgleichheit erfolgt. Da die einzelnen Operationen eines Dienstes nach dem Verständnis des hier vorgestellten Ansatzes stets zustandslos sind, müssen Reihenfolgen, die bei einem Dienstaufruf eingehalten werden müssen, durch Netze kodiert werden.

Ein solches Netz mit mehreren Platzhaltertransitionen, die dann durch verschiedene Aufrufe desselben Dienstes ersetzt werden, wird Interaktionsmuster genannt. Auf diese Weise lassen sich Dienste mit einer gleichen Abfolge ähnlicher Subdienste zu Dienstklassen zusammenfassen, wobei sich wiederum eine Nähe zu dem von Köhler-Bußmeier (2009c) vertretenen Ansatz zeigt. Dies soll anhand des obigen Beispiels eines Dienstes erläutert werden, der zunächst einen Login, dann eine Abfrage und dann einen Logout erfordert. Ein entsprechendes Interaktionsmuster ist in Abbildung 6.24 dargestellt.

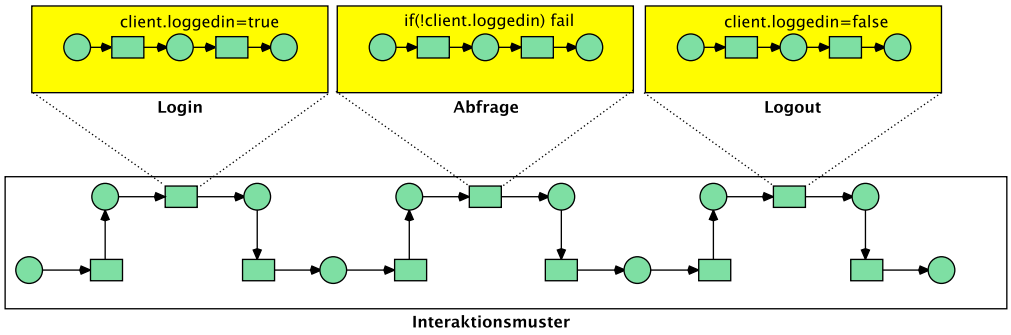


Abbildung 6.24: Ein einfaches Interaktionsmuster

Möglich wäre etwa, dass alle Informationsdienste eines Bereiches diesem Interaktionsmuster folgen. Später können die Dienste um ein weiteres Interaktionsmuster erweitert werden, dass es erlaubt, sich einmal anzumelden und mehrere Dienste abzufragen. Verwendet ein Dienstnutzer dann dieses alte Interaktionsmuster, so kann der Login-Teil des Dienstes durch ein Protokoll ersetzt werden, das zunächst überprüft, ob der Nutzer sich bereits anderweitig authorisiert hat (beispielsweise über ein Single Sign On-Verfahren).

Aufgrund des Interkationsmusters ist festgelegt, dass es nicht möglich ist, zuerst eine Informationsabfrage zu starten und sich dann einzuloggen. Statt ein Interaktionsmuster festzulegen, wäre es ebenso möglich, die Interaktion direkt zwischen den Teilen des Dienstes zu kodieren, indem zwischen je zwei Protokollen eine Stelle existiert, die belegt sein muss, damit das nächste Protokoll starten kann. Dies bietet

jedoch den Nachteil, dass die entstehenden Netze häufig keine Workflownetze mehr sind. Ist es etwa möglich, den Logout zu vergessen oder gibt es einen automatischen Timeout, so ist eine Transition notwendig, die einen Token entfernt, ohne dass sie mit dem übrigen Netz verbunden ist, da ein Timeout des Logouts unabhängig vom Aufrufer ist. Dies würde den Workflowcharakter des Netzes verletzen und ist daher nicht gewollt.

Da ein Interaktionsmuster bereits ein Dienstbeschreibungsnetz und somit ein Workflownetz ist, lässt es sich mit den Regeln verfeinern, die den Workflowcharakter beibehalten und die in Abschnitt 5.3.3 vorgestellt werden. Diese Regeln belassen wichtige Eigenschaften des Workflownetzes wie die Korrektheit und die Sicherheit intakt.

6.5 Zusammenfassung

Dieses Kapitel widmete sich der weitergehenden Analyse elementarer Dienstbeschreibungsnetze. Dabei werden zwei Arten der Analyse betrachtet. Zum einen wird dargestellt, wie sich ein elementares Dienstbeschreibungsnetz durch Sequentialisierung in ein ebenfalls elementares Dienstbeschreibungsnetz überführen lässt, in dem es keine nebenläufig aktivierten Transitionen gibt, zum anderen wird dargestellt, wie durch die Modellierung von Zustandsübergängen anhand der im Netz verwendeten Objekte weitere Korrektheitskriterien ermittelt werden können. Dieser Abschnitt fasst zunächst die Ergebnisse zusammen und ordnet dann die bisher dargestellten Netzklassen ein.

Ergebnisse

Die Sequentialisierung wird für elementare Free-Choice-Dienstbeschreibungsnetze dargestellt und erfolgt in Abschnitt 6.2 zunächst über die Betrachtung der zugrundeliegenden Workflownetze. Die Sequentialisierung eines korrekten Free-Choice-Workflownetzes ist in Definition 6.3 in Abschnitt 6.2.1 wiedergegeben. Eine solche Sequentialisierung besitzt zu jeder Schaltfolge des ursprünglichen Netzes eine Schaltfolge, die bis auf die Permutation nebenläufiger Transitionen der ursprünglichen Schaltfolge entspricht. Gleichzeitig gibt es in der Sequentialisierung keinerlei nebenläufig aktivierbare Transitionen, so dass sich die Menge der möglichen Schaltfolgen unter Umständen drastisch reduzieren kann. Dabei ist eine Sequentialisierung so definiert, dass die Sequentialisierung einer Zustandsmaschine isomorph zu sich selbst ist, da eine Zustandsmaschine ja bereits keine Nebenläufigkeit mehr aufweist.

Unterabschnitt 6.2.2 betrachtet ein Verfahren zur Sequentialisierung eines korrekten Free-Choice-Workflownetzes. Dieses Verfahren erlaubt es, ein stark sequentialisierbares korrektes Free-Choice-Workflownetz in eine Zustandsmaschine zu überführen, die die Eigenschaften der Sequentialisierung erfüllt und gleichzeitig eine

bijektive Abbildung zwischen den Transitionen besitzt, so dass jeder Schaltfolge leicht eine Entsprechung im Ursprungsnetz zugewiesen werden kann. Dieses Verfahren ist mit Verfahren 6.8 gegeben. Die Korrektheit und die Vollständigkeit des Verfahrens wird dann in Satz 6.12 gezeigt. Unterabschnitt 6.2.3 widmet sich im Anschluss der Sequentialisierung anhand von Reduktionsregeln, die dadurch erfolgen kann, dass zwei nebenläufige Teilnetze nacheinander in einer Sequentialisierung durch eine Transition verbunden werden.

Abschnitt 6.3 überträgt die Ergebnisse aus Abschnitt 6.2 auf Free-Choice-Dienstbeschreibungsnetze und erläutert die Sequentialisierung dieser Netzklasse. Hierzu wird die erweiterte Sequentialisierung anhand eines Quasi-Zustandsmaschinen-Workflownetzes eingeführt. Ein solches Netz ist ein S-überdeckbares Netz, bei dem es eine S-Komponente gibt, die jede Transition umfasst. Hierdurch ist gewährleistet, dass niemals zwei Transitionen gleichzeitig schalten können, da in dieser S-Komponente stets nur eine Transition aktiviert ist. Die erweiterte Sequentialisierung aus Definition 6.15 erlaubt dann ein solches Netz in der Sequentialisierung, wodurch die Stellen im Vor- und Nachbereich jeder Transition erhalten bleiben können. Die erweiterte Sequentialisierung kann mit dem Verfahren 6.16 erstellt werden und wird dann in Abschnitt 6.3.2 zur Definition der Sequentialisierung elementarer Free-Choice-Dienstbeschreibungsnetzen in Definition 6.18 verwendet. Satz 6.20 stellt im Anschluss hieran dar, dass das Schaltverhalten der Sequentialisierung dem des ursprünglichen Netzes entspricht, wenn man durch Permutation nebenläufiger Transitionen entstehende Schaltfolgen als gleich ansieht.

Unterabschnitt 6.3.3 stellt abschließend dar, wie sich die Sequentialisierung dann zur Analyse heranziehen lässt, indem auf bekannte Verfahren zur Analyse algebraischer Netze zurückgegriffen wird. Hierzu stellt der Systemzustand ein Element der Algebra dar, auf dem Operationen zum Verändern von Lokationen definiert sind. Die Ordnung der Sorten kann hierbei vernachlässigt werden, da durch die strenge Typisierung der Dienstbeschreibungsnetze bereits gewährleistet ist, dass hierbei keine Fehler entstehen können. Mögliche hierbei auftretende Unstimmigkeiten werden in Unterabschnitt 6.4.2 erläutert, bevor Unterabschnitt 6.4.3 dann auf die Interaktion von Diensten eingeht. Hier steht vor allem die Möglichkeit im Vordergrund, Dienstbeschreibungsnetze durch Dienstbeschreibungsnetze zu verfeinern, wie dies auch für Workflownetze möglich ist. Hierbei muss für Dienstbeschreibungsnetze jedoch die Typisierung der Stellen überprüft werden.

Einordnung der Netzklassen

In dieser Arbeit wurden verschiedene Subklassen der Workflownetze und der Dienstbeschreibungsnetze eingeführt, die an dieser Stelle noch einmal zusammengefasst werden sollen. Die Betrachtung der Workflownetze wurden auf die Betrachtung von Free-Choice-Workflownetzen beschränkt, die in Kapitel 5 näher analysiert worden sind. Eine Teilmenge dieser Workflownetze sind die stark sequentialisierbaren Free-

Choice-Workflownetze, die in diesem Kapitel betrachtet worden sind. Dies ist in Abbildung 6.25 dargestellt. Gleichzeitig ist jedes korrekte Free-Choice-Workflownetz sicher, so dass sich die in Abbildung 6.25 dargestellte Beziehung ergibt. Hierbei wurden die stark sequentialisierbaren Free-Choice-Workflownetze als zentrales Element der Betrachtung in dieser Arbeit hervorgehoben.

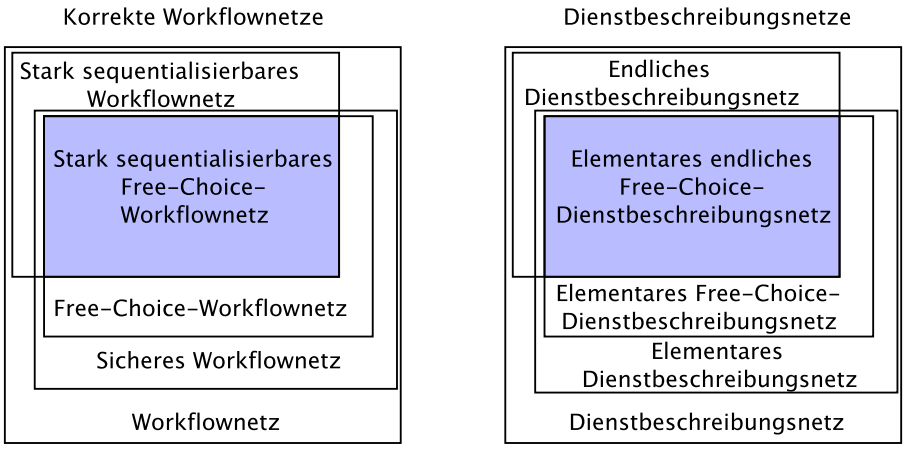


Abbildung 6.25: Die in dieser Arbeit betrachteten Netzklassen

Dienstbeschreibungsnetzen liegen Workflownetze zugrunde, so dass sich diese ebenfalls anhand des zugrundeliegenden Workflownetzes klassifizieren lassen. Darüber hinaus lässt sich die Elementarität und die Endlichkeit eines Netzes analysieren. Die Analyse der schwachen Korrektheit erfolgte im vorangegangenen Kapitel für elementare endliche Free-Choice-Dienstbeschreibungsnetze, wie sie in Definition 5.41 angegeben wurden. Diese Netzklasse ist in Abbildung 6.25 ebenfalls hervorgehoben. Die Sequentialisierung in diesem Kapitel erfolgt dann wiederum für die Netzklasse der elementaren endlichen Free-Choice-Dienstbeschreibungsnetze, deren zugrundeliegendes Workflownetz ein stark sequentialisierbares Free-Choice-Workflownetz ist. Dies entspricht den Hervorhebungen auf beiden Seiten. Die Subsumptionsbeziehungen lassen sich leicht anhand der in den jeweiligen Kapiteln gegebenen Eigenschaften überprüfen, die an dieser Stelle nicht noch einmal referenziert werden sollen.

7 Praktische Verwendung von Dienstbeschreibungsnetzen

Dieses Kapitel schildert die praktische Nutzung von Dienstbeschreibungsnetzen bei der Modellierung dienstbasierter Systeme. Aufbauend auf den Ergebnissen aus Kapitel 2 werden zunächst die Charakteristika dienstbasierter Systeme und entsprechender Modelle noch einmal erläutert, bevor dann eine eigene Architektur präsentiert wird, in der Dienstbeschreibungsnetze als wesentlicher Teil der Modellierung genutzt werden.

7.1 Dienstbeschreibungsnetze zur Modellierung

Dieses Kapitel beschreibt die Nutzung von Dienstbeschreibungsnetzen zur Modellierung dienstbasierter Systeme, weswegen mit der Vorstellung möglicher Einsatzszenarien der Modellierung dienstbasierter Geschäftsprozesse begonnen wird. Hierbei wird die Modellierung inter- und intraorganisationaler Workflows mit Petrinetzen dargestellt, und es wird diskutiert, welche Vorgehensmodelle sich zur Modellierung dienstbasierter Systeme anbieten. Hieran schließt sich eine Einbettung des Modellierungsansatzes in den Kontext bestehender petrinetzbasierter Architekturen an, die dann zu einer allgemeinen Architektur dienstbasierter Systeme angepasst werden. Anhand eines Beispiels wird dann der Einsatz von Dienstbeschreibungsnetzen näher erläutert.

Ansätze der Modellierung

Die Modellierung inter- oder intraorganisationaler Abläufe erfolgt in der Regel top-down, indem Prozesse zunächst abstrakt definiert werden und dann nach und nach verfeinert werden. Dieses Vorgehen wird hier noch einmal erläutert und dann auch für die hier vorgestellte Vorgehensweise übernommen, wobei einzelne Teilprozesse auch bottom-up durch Dienstbeschreibungsnetze gekapselt werden können. Dieses Vorgehen entspricht dem Ansatz bestehender Vorgehensmodelle in diesem Bereich, die in Abschnitt 7.2 dargestellt werden.

Die Einbettung des Ansatzes in bestehende Überlegungen zu petrinetzbasierten Systemen erlaubt es, bestehende Betrachtungsweisen zu übernehmen. Insbesondere

orientiert sich der Ansatz dabei an der petrinetzbasierten agentenorientierten Softwareentwicklung (Paose). Diese wird daher noch einmal zusammen mit der dienstbasierten Anpassung Pas erläutert, bevor dann im darauffolgenden Abschnitt die Dienstarchitektur detaillierter dargestellt wird.

Eine Dienstarchitektur

Die Dienstarchitektur zur Modellierung dienstbasierter Systeme übernimmt im Wesentlichen die Ebenen der agentenorientierten Modellierung aus Mulan. Die Darstellung der Ebenen wird jedoch der dienstbasierten Umgebung angepasst, so dass sich hier zum Teil recht deutliche Änderungen ergeben, da Dienste im Allgemeinen nicht migrieren und somit nicht als Netze in Netzen dargestellt werden müssen. Vielmehr lauschen Dienste dem Nachrichtenverkehr an einem Bus und führen ein Verhalten aus, wenn eine Nachricht für sie eintrifft. Dies wird in der angepassten Version der Architektur in Abschnitt 7.3 reflektiert. Neben dem Aufbau der Dienste werden auch die Sicht der Daten über eine Ontologie und die Sicht der Abläufe betrachtet. Alle drei Sichten zusammen ergeben das Modell, wobei die Organisationssicht die Elemente des Modells darstellt, die Datensicht die Daten des Modells spezifiziert und die Prozesssicht den Ablauf darstellt. Dies wird am Anfang von Abschnitt 7.4 beispielhaft umgesetzt.

Verwendung von Dienstbeschreibungsnetzen

Abschnitt 7.4 beginnt mit der beispielhaften Umsetzung der Konzepte der vorherigen Abschnitte mittels Dienstbeschreibungsnetzen. Auf jeder der Ebenen der Architektur werden die verschiedenen Sichten kurz skizziert, und es wird ein Objekt der nächsten Ebene genauer spezifiziert. Zur Modellierung der Daten wird ein Teil einer Ontologie angegeben, deren Konzepte dann in den Netzen verwendet werden. Da die Darstellung eines gesamten Systems den Rahmen dieser Arbeit sprengen würde, werden jeweils nur Teilaspekte einer Ebene näher erläutert.

Die am Beispiel vorgestellte Methodik wird dann näher diskutiert. Hierbei wird darauf eingegangen, wie sich Simulation und Analyse und tatsächliche Ausführung der Dienste voneinander trennen lassen. Abschließend erfolgt eine kurze Diskussion über mögliche Vereinfachungen in der graphischen Darstellung der Dienstbeschreibungsnetze, die zudem eine leichtere Überprüfung der Elementarität der Netze erlauben.

7.2 Modellierung dienstbasierter Systeme

Dieser Abschnitt verfolgt zunächst das Ziel darzustellen, wie dienstbasierte Systeme modelliert werden und wie sich Petrinetze nutzen lassen, um dienstbasierte

Systeme zu modellieren. Naheliegend ist hierbei die Modellierung der Abläufe durch Workflownetze, die zunächst dargestellt wird. Abläufe können dabei sowohl interals auch intraorganisationaler Natur sein. Im Anschluss hieran wird diskutiert, welche Eigenschaften überhaupt zu modellieren sind und welche Vorgehensmodelle es zum Entwurf dienstbasierter Systeme gibt. Abschließend erfolgt eine Einbettung in einen bestehenden Ansatz petrinetzbasierter Agentenmodellierung.

7.2.1 Anwendung dienstbasierter Modellierung

Der Einsatz dienstbasierter Systeme ist vor allem in komplexen Szenarien sinnvoll, in denen Flexibilität und die lose Kopplung der einzelnen Systemeinheiten erforderlich ist. Unternehmen sind häufig in funktionale Einheiten (Abteilungen) aufgeteilt, die einen gewissen Zweck verfolgen, wie etwa die Buchhaltung, der Vertrieb oder das Lager. Geschäftsprozesse erstrecken sich vielfach über eine Vielzahl solcher funktionaler Einheiten. Durch die Verwendung von Diensten kann die prozessspezifische Funktionalität einer organisatorischen Einheit gekapselt werden und dann verschiedenen Prozessen zur Verfügung gestellt werden. Speziell durch die Verwendung von Schnittstellendefinitionen, die auch eine Verhaltensdefinition beinhalten, können Prozesse so zunächst allgemein definiert werden und dann von den einzelnen funktionalen Einheiten spezialisiert und ihren Bedürfnissen entsprechend angepasst werden. Dieses Vorgehen hat van der Aalst (2002, 2003) zur Modellierung interorganisationaler Workflows mit Petrinetzen dargestellt, was hier noch einmal skizziert werden soll. Im Anschluss daran wird auf die Verwendung dieser Vorgehensweise für die Entwicklung dienstbasierter Systeme in Organisationen eingegangen.

Modellierung interorganisationaler Workflows

Die präzise Definition von Schnittstellen ist gerade bei interorganisational verteilten Workflows von besonderer Bedeutung, da häufig nicht alle Beteiligten bereit sind, ihren jeweiligen Teil eines Ablaufs vollständig gegenüber den anderen Teilnehmern zu veröffentlichen. Einen Ansatz hierzu bietet die Unterscheidung zwischen öffentlichem und privatem Workflow. Ein öffentlicher Workflow enthält die Teile des Gesamtworkflows, die allen Beteiligten bekannt sind, während der private Workflow das beschreibt, was ein Teilnehmer zu leisten hat. Diese Unterscheidung, die etwa in van der Aalst (2002) und Bussler (2002) verwendet wird, ist jedoch nicht immer glücklich, da der „öffentliche“ Hauptworkflow ebenfalls geheim sein mag. Unternehmen, die auf eine Reihe von Zulieferern angewiesen sind, besitzen einen Hauptworkflow, den sie kontrollieren und der die Abläufe mit den Zulieferern koordiniert. Ein solches Unternehmen hat in der Regel kein Interesse daran, diesen Hauptworkflow publik zu machen.

Die Realisierung eines interorganisationalen Workflows einerseits durch einen öffentlichen Workflow und andererseits durch Schnittstellen, die durch private Work-

flows realisiert werden, wird von van der Aalst (2002, 2003) beschrieben. Hierzu wird zunächst ein interorganisationaler Workflow erstellt, dessen Aktivitäten und Subworkflows auf die verschiedenen Beteiligten aufgeteilt werden. Der öffentliche Hauptworkflow koordiniert die Teilworkflows der Beteiligten. Die privaten Workflows der Beteiligten können hierbei bis auf ihre Schnittstelle zum öffentlichen Workflow vollständig als Black Box betrachtet werden. Der grundsätzliche Aufbau dieses Vorgehens umfasst die folgenden Schritte:

1. Spezifizierung des öffentlichen Ablaufs und der Schnittstellen
2. Aufteilung des öffentlichen Ablaufs unter den Beteiligten
3. Anpassung der privaten Workflows durch jeden der Beteiligten
4. Verfeinerung der privaten Abläufe durch jeden der Beteiligten anhand von Vererbungsregeln.

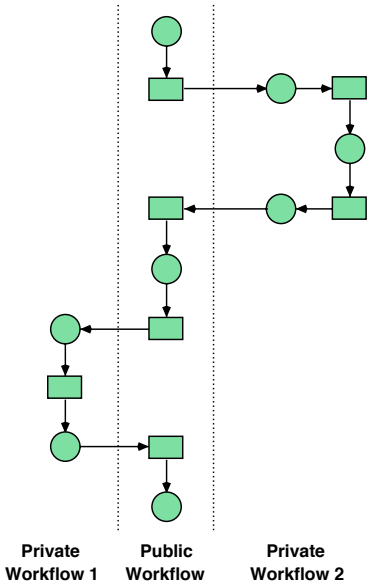


Abbildung 7.1: Ein öffentlicher Workflow

(1) Zunächst wird von den Beteiligten ein öffentlicher Workflow definiert. Hierbei gilt es für die Beteiligten, Übereinstimmung über die allgemeine Struktur zu erlangen und über Schlüsselaktivitäten und ihre Schnittstellen. Die Schnittstellen werden darüber spezifiziert, welche Abhängigkeiten hinsichtlich des Kontrollflusses und der Daten vorhanden sind. Hier kann eine semantische Beschreibung zusätzlich noch funktionale Abhängigkeiten ausdrücken. Prozessverzeichnisse können verwendet werden, um den Designprozess an dieser Stelle zu erleichtern. Dieses Modell

enthält keinerlei organisationsspezifische Informationen wie Implementationsdetails oder beteiligte Ressourcen.

(2) Im zweiten Schritt zeichnen sich die Beteiligten verantwortlich, Teile des Prozesses zu erfüllen. Dies führt zu einer Partitionierung des Workflows entlang der Organisationsgrenzen wie dies in Abbildung 7.1 anhand eines Workflownetzes dargestellt ist. Der Gesamtworkflow ist aufgeteilt in den privaten Workflow 1 und den privaten Workflow 2 sowie den öffentlichen Hauptworkflow.

(3) Bei der Partitionierung kann es passieren, dass die entstehenden Netze nicht mehr länger Workflownetze darstellen, da es mehrere Ein- oder Ausgangsstellen geben kann. Aus diesem Grund wird in diesem Schritt die Struktur des Netzes angepasst, so dass das Verhalten nach außen gleich ist, sich jedoch ein Workflownetz ergibt. Abbildung 7.2 stellt eine mögliche Partitionierung dar. Hierbei wurden die Fragmente des öffentlichen Workflownetzes mit zusätzlichen Stellen versehen, um es zu einem Workflownetz zu machen.

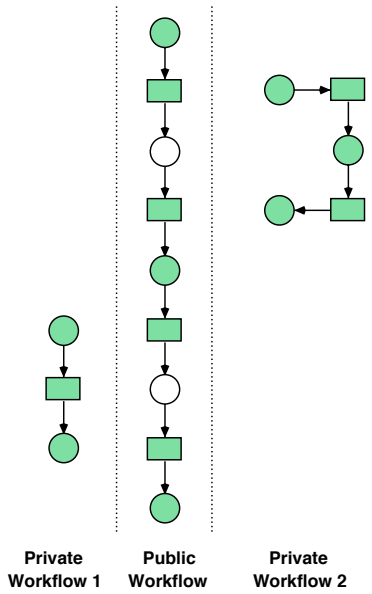


Abbildung 7.2: Aufteilung des öffentlichen Workflows

(4) Zuletzt passt jeder der Beteiligten seinen privaten Workflow dergestalt an, dass er seine internen Prozesse reflektiert. Hierbei wird die Bearbeitung von Aktivitäten spezifiziert, Rollen zugewiesen und der Datenfluss spezifiziert. Des Weiteren können noch Regeln, Deadlines oder andere Abhängigkeiten wie etwa Einschränkungen durch Policies angegeben werden. Um zu garantieren, dass der lokale Workflow mit dem öffentlichen konsistent ist, geben van der Aalst und Basten (1997, 2002) eine Reihe von Vererbungsregeln an. Werden nur diese Regeln zu Verfeinerung angewendet, so ist das resultierende Workflownetz immer noch konsistent zum

öffentlichen Workflow.

Dienstbeschreibungsnetze und die in Abschnitt 6.4.3 beschriebenen Interaktionsmuster erlauben es nun, diese Möglichkeiten der Workflowmodellierung um Daten und Funktionalität zu bereichern, so dass der öffentliche Workflow nicht mehr nur vorgibt, welches Verhalten gefordert wird, sondern auch angibt, wie sich die Daten bei Aufruf eines Dienstes, der nicht öffentlich ist, ändern sollen. Auf diese Weise kann sichergestellt werden, dass Schnittstellen über das Verhalten hinaus gleich interpretiert werden. Durch die Verwendung von Reduktionsregeln kann dann das Verhalten der privaten Teilnetze spezialisiert werden.

Modellierung intraorganisationaler Systeme

Die Vorgehensweise zur Modellierung interorganisationaler Workflows lässt sich auch innerhalb von Organisationen anwenden, in denen es häufig separate Fachabteilungen gibt, die Teile eines unternehmsweiten Prozesses realisieren, ohne sämtliche Details der Realisierung anderer Fachabteilungen zu kennen. Wiederum ist es hier wichtig, zunächst Schnittstellen bereitzustellen, die dann von den Fachabteilungen implementiert werden. In dienstbasierten Ansätzen wird die Funktionalität der einzelnen Abteilungen dabei durch Dienste gekapselt. Ein Dienst, der einen Geschäftsprozess realisiert, kann dann auf die von den einzelnen funktionalen Einheiten bereitgestellten Dienste zurückgreifen.

Bei einem dienstbasierten Ansatz besteht stets eine Abwägung zwischen der Wiederverwendbarkeit der Dienste und der Einfachheit des Gesamtsystems. Die Modellierung sehr feingranularer Dienste führt dazu, dass diese in der Regel leicht wiederverwendet werden können und so aus den sehr einfachen Diensten komplexere Dienste erstellt werden können, die eine spezifische Lösung darstellen. Änderungen lassen sich dann unter Verwendung anderer Dienste implementieren, was im günstigsten Fall zu einer hohen Wiederverwendbarkeit und Flexibilität führt. Andererseits birgt diese Herangehensweise die Gefahr, dass es eine Vielzahl von Diensten gibt, die nicht mehr adäquat verwaltet werden können. In diesem Fall werden Dienste nicht wiederverwendet, weil sie nicht aufgefunden werden, oder weil sich der Modellierer nicht über ihre genaue Funktionalität sicher ist. Auch leidet die Performanz des Gesamtsystems bei einer Vielzahl von Diensten, da dann auch eine Vielzahl von Nachrichten zwischen den Diensten ausgetauscht werden müssen. Letzteres ist vor allem dann problematisch, wenn aufwendig zu kodierende Nachrichtenstandards verwendet werden, wie etwa auf XML-basierende Nachrichtenformate.

Hier sind geeignete Werkzeuge notwendig, die Modellierer und Ersteller dienstbasierter Systeme dabei unterstützen, eine Vielzahl von Diensten zu verwalten und ihre Kommunikation zu optimieren. Letzteres lässt sich erreichen, indem versucht wird, automatisiert das optimale Nachrichtenformat für Dienste zu finden. So lassen sich Dienste auf dem selben Server über einen Prozeduraufruf aufrufen, während

Dienste in verschiedenen Organisationen über einen XML-basierten Standard wie den Webservice-Standard (Booth u. a., 2004) kommunizieren. Eine abstrakte Beschreibung der Dienstfunktionalität bietet hier den Vorteil, dass sie in verschiedene konkrete Implementationen übersetzt werden kann. Diese Implementationen können dann die zur Kommunikation notwendigen Nachrichtenformate beherrschen.

Ein schnittstellenbasierter Entwurf dienstbasierter Systeme bietet bei der Verwaltung der Dienste Vorteile, da zunächst die Funktionalität und das Verhalten abstrakt beschrieben wird und dann wiederum durch verschiedene Implementationen verfeinert wird. Somit können zunächst Schnittstellen ausgewählt werden, von denen ausgehend dann konkrete Dienste ausgewählt werden können. Diese Schnittstellen können zudem in verschiedenen Dimensionen kategorisiert werden. Beispiele hierfür wären etwa die verantwortliche organisationale Einheit, die benötigten Ressourcen oder die funktionale Beschreibung der Auswirkungen. Diese Dimensionen entsprechen den in Unterabschnitt 2.2.2 vorgestellten Sichten auf einen Prozess.

Sind die Schnittstellenbeschreibungen und ihre Komposition zudem ausführbar, so eröffnet sich die Möglichkeit, anhand der Schnittstellen definierte Systeme zu simulieren und ihr Verhalten zu analysieren. So lassen sich bereits vor der eigentlichen Implementation Schwächen eines Systems analysieren. Eine Simulation der Schnittstelle erlaubt zudem, die Funktionalität der Schnittstelle automatisiert zu erfassen, und so die Dienste anhand ihrer Funktionalität zu klassifizieren. Dies ist jedoch im Allgemeinen nicht mittels einer konkreten Ausführung der Dienste möglich. Vielmehr ist eine Ausführung auf symbolischer Ebene notwendig, die von konkreten Aufrufparametern abstrahiert.

Eine detaillierte Beschreibung der Funktionalität von Diensten ermöglicht somit eine Analyse komplexer Dienste zur Modellierungszeit. So nennen auch Wolff u. a. (2005) als wesentliche Motivationsgründe für eine detaillierte Beschreibung der Dienstfunktionalität und der Dienstschnittstelle die Auswahl von Dienstfunktionalität, das Erkennen inkompatibler Ein- und Ausgabeparamter, das Erkennung von Zyklen und Verklemmungen, die Bewertung von Diensten anhand ihrer Dienstgüte, die Überprüfung von Workfloweigenschaften und die Überprüfung von Policies. Insbesondere die Überprüfung von Policies stellt ein weiteres interessantes Anwendungsfeld einer detaillierten formalen Beschreibung dar, da durch sie überprüft werden kann, ob die Policies einer Organisation erfüllt werden. So kann beispielsweise sichergestellt werden, dass sensible Daten nicht an Unberechtigte weitergeleitet werden oder dass stets der hinsichtlich gegebener Kriterien beste Dienst ausgewählt wird. Wolff u. a. (2005) sehen als größtes Hindernis bei der Erstellung detaillierter Beschreibungen von Diensten den initialen Aufwand hierfür. Ließe sich aus einer Dienstbeschreibung jedoch die Implementation eines Dienstes erzeugen, so würde dieser Aufwand weitaus geringer sein. Vielmehr wäre gewährleistet, dass Dokumentation, formale Beschreibung und Implementation konsistent wären.

7.2.2 Kontext der Modellierung dienstbasierter Systeme

Bevor im nächsten Abschnitt eine petrinetzbasierte Architektur für dienstbasierte Systeme dargestellt werden soll, die Petrinetze und zur Verfeinerung auch Dienstbeschreibungsnetze als zentralen Bestandteil verwendet, widmet sich dieser Unterabschnitt zunächst der Darstellung der Annahmen, die über dienstbasierte Systeme getroffen werden. Zudem werden Vorgehensmodelle zur Umsetzung dienstbasierter Systeme beleuchtet, an denen sich die Nutzung von Dienstbeschreibungsnetzen und ihre Vorteile später erläutern lassen.

Annahmen über dienstorientierte Systeme

Dienste sind in Abschnitt 2.3 als reaktive Elemente charakterisiert, die über Nachrichten kommunizieren und die eine eindeutige Schnittstelle und ein weitestgehend vorhersagbares Verhalten besitzen. Sie werden also anhand einer Nachricht aufgerufen und bearbeiten eine Anfrage. Der Aufruf eines Dienstes wird entweder vom Dienst oder im Fall eines Kommunikationsfehlers vom zugrundeliegenden Transportsystem durch eine Nachricht quittiert.

Die Umwelt von Diensten ist nicht deterministisch, was zur Folge hat, dass viele Eigenschaften von Diensten erst zur Laufzeit entschieden werden können. Dies gilt in besonderem Maße für interorganisationale Dienste, wo die Umwelt zudem nur partiell beobachtet werden kann. Dienste besitzen zudem nur eine lose Kopplung mit ihren Dienstnutzern. Aus diesem Grund besitzen Dienste auch keinen aufruferspezifischen Zustand, da sie nur in einer Instanz existieren. Vielmehr müssen zusammengehörige Aufrufe eines oder mehrerer Dienste anhand eines speziellen Tokens identifiziert werden, welcher den Kontext der Konversation identifiziert.

Das in Abschnitt 2.3.2 dargestellte Grundmodell dienstorientierter Architekturen ermöglicht es, von diversen Implementationsspezifika zu abstrahieren. So erlaubt die Verwendung eines Enterprise Service Bus die Abstraktion von spezifischen Nachrichtenformaten. Mit Hilfe der in Abschnitt 2.4.1 diskutierten Verwendung von Ontologien im Dienstbereich ist es dann möglich, Dienstarchitekturen auf einer abstrakten Ebene zu beschreiben. Die Modellierung von Diensten kann dadurch von konkreten Daten- und Nachrichtenformaten abstrahieren und ihr Augenmerk stattdessen auf eine Beschreibung mittels der Konzepte einer Ontologie richten. In dieser Ontologie finden sich sämtliche für die Modellierung des Unternehmens relevanten Datentypen. Sie stellen die Typen der sogenannten Fachobjekte dar.

Auf dieser Ebene der Abstraktion tauschen Dienste dann lediglich Fachobjekte (also in einer Ontologie definierte Konzepte) aus, die auf entsprechende Nachrichten abgebildet werden. Zusammen mit dem Konzept der Vermittlungsdienste und des Enterprise Service Bus (vgl. Abschnitt 2.3.2) ermöglicht dies eine relativ starke Abstraktion von Implementationsdetails der Kommunikation und erlaubt es, ein

leichter verständliches Modell zu erstellen[1].

Die Fachobjekte dienen jedoch nicht nur zum Nachrichtenaustausch, sondern auch zur Darstellung des Systemzustands. Diese Idee wird mittels der Verwendung von Dienstbeschreibungsnetzen und ihrem zentralen Zustand konsequent umgesetzt. Hierbei wird davon ausgegangen, dass dienstbasierte Systeme in einem Universum von Objekten eingebettet sind und dass sich die Objekte dieses Universums durch Konzepte beschreiben lassen. Ein aktueller Zustand des Systems kann so mit Hilfe des Vokabulars einer Ontologie kommuniziert und dargestellt werden. Dies bedeutet insbesondere auch, dass sämtliche Objekte des Systems einer endlichen Menge von Konzepten zugeordnet werden können. Der Zustand des Systems stellt somit eine Momentaufnahme dar, die anhand einer Menge von Konzepten kommuniziert wird[2].

Das Universum eines dienstbasierten Systems ist der Ausschnitt der Welt, der für dieses System relevant ist. Ein Dienst nimmt eine Änderung am Zustand des Universums vor oder liefert Informationen über den derzeitigen Zustand des Universums (oder beides). Es mag zwar Dienste geben, die weder den Zustand des Gesamtsystem ändern noch Informationen über ihn bereitstellen, jedoch werden diese Dienste nicht in Planungs- oder Kompositionsszenarien berücksichtigt, da sich nicht entscheiden lässt, was für eine Aufgabe ein solcher Dienst besitzt[3].

Bestehende Vorgehensmodelle

Zur Modellierung von dienstbasierten Geschäftsprozessen gibt es eine Reihe von Vorgehensmodellen. Diese unterscheiden sich im Allgemeinen von Vorgehensmodellen zur Entwicklung traditioneller (bspw. objektorientierter) Software, da bei dienstorientierten Architekturen ein weit stärkerer Fokus auf der Modularisierung und der Prozessbezogenheit liegt. Thomas u. a. (2009) nennen hier als wesentliche Differenzierungsmerkmale dienstorientierter Systeme die starke Modularisierung durch die Kapselung von Funktionalität in Diensten, die starke Abstraktion durch die Trennung zwischen dem Dienst und seiner eigentlichen Implementation, die starke Prozessorientierung und die Beachtung einer ausgewogenen Granularität der Dienste bei der Umsetzung. Diese Faktoren müssen in einem Vorgehensmodell entsprechend berücksichtigt sein.

[1] Viele kommerzielle Hersteller unterscheiden ebenfalls zwischen einem Plattform Independent Model (PIM) und einem Plattform Specific Model (PSM).

[2] Es mag für einen Teile des Systems sinnvoll sein, intern ein nicht-zustandsbasiertes Modell zur Repräsentation des Systems zu verwenden, um Entscheidungen zu treffen. Dieses ist dann jedoch Teil eines separaten Modells, das nicht nach außen sichtbar ist.

[3] Denkbar wären hier etwa Dienste, die Nachrichten protokollieren, so dass im Fehlerfall die Kommunikation, die zum Fehler führte, nachvollzogen werden kann. Diese Dienste ändern jedoch das Verhalten konkreter Workflows nicht und sind somit nicht unbedingt Gegenstand eines Modells.

Eine detaillierte Übersicht über den aktuellen Stand zum Thema Vorgehensmodelle zur Entwicklung serviceorientierter Softwaresysteme liefern Thomas u. a. (2009). Den meisten dieser Vorgehensmodelle ist eine Mischung aus Top-Down- und Bottom-Up-Ansatz gemeinsam. Zunächst werden die Geschäftsprozesse allgemein modelliert und durch ihre Ziele und die notwendigen Schritte zum Erreichen dieser Ziele charakterisiert. Anschließend werden bestehende Softwareeinheiten durch Dienste gekapselt, sofern sie sinnvoll in einer SOA verwendet werden können. Diese Mischform wird meist als *Meet-in-the-Middle-Ansatz* oder kürzer als *Middle-Out-Ansatz* bezeichnet. Thomas u. a. (2009) weisen darauf hin, dass die wenigsten Ansätze eine durchgängige Unterstützung durch Werkzeuge mit sich bringen.

Die auf Mitra (2005) zurückgehende geschäftsgetriebene Entwicklung (engl. Business-Driven Development (BDD)) von dienstbasierten Architekturen verwendet ebenfalls die oben erwähnte Mischung aus einem Top-Down- und einem Bottom-Up-Vorgehen. Hierbei geht der Ansatz von den strategischen Zielen einer Organisation aus und entwickelt hieraus die notwendigen Prozesse. Dies geschieht entweder über Use Cases oder direkt aus der Zielbeschreibung. Die dargestellten Prozesse werden dann so detailliert modelliert, dass sich hieraus Workflows ableiten lassen, sofern die Prozesse eine entsprechende Automatisierung zulassen. Dann werden bestehende funktionale Einheiten des Unternehmens, die den Prozess unterstützen können, durch Dienste gekapselt, woraus ein initiales Modell entsteht. Aufgrund der strategischen Kriterien werden dabei die optimalen Dienste ausgewählt. Abschließend wird das Gesamtsystem implementiert.

In diesem Kontext des BDD stellen auch Stein und Ivanov (2007) ein Vorgehensmodell zur Entwicklung von Geschäftsservicen vor. Dieses beschreibt detailliert die einzelnen Schritte von den Geschäftsprozessen zu ihrer technischen Umsetzung als Dienst. Dabei wird davon ausgegangen, dass die Prozesse bereits in adäquaten Modellen vorliegen. Den dabei fehlenden Schritt der Identifikation der Geschäftsprozesse aufgrund der strategischen Ziele eines Unternehmens beschreiben beispielsweise Jones und Morris (2005).

Als wesentliche Schwachstellen eines Top-Down-Ansatzes identifizieren Koehler u. a. (2008) die Tatsache, dass sich ein Top-Down-Ansatz, der von den Geschäftsprozessen ausgeht, nicht nahtlos auf eine technische Plattform übertragen lässt. Hierbei können einerseits technische Probleme wie mangelnde Performanz auftreten, und andererseits kann die bestehende Infrastruktur unter Umständen nur schwer an die Modelle angepasst werden. Koehler u. a. (2008) identifizieren als Lösung dieses Problems einen Ansatz, der es erlaubt, aus einem Modell von Geschäftsprozessen ein Model von Workflows bzw. von deren technischen Umsetzung zu erzeugen. Um dies zu erreichen sollte es möglich sein, ein Modell sukzessive um weitere Informationen anzureichern. Hierzu gehören die verwendeten Daten, die notwendigen Ressourcen und die Entscheidungslogik, aufgrund derer bestimmte Subprozesse ausgewählt werden. Weiterhin ist es notwendig zu beschreiben, was einen Ablauf überhaupt initiiert, was häufig anhand von Regeln erfolgt.

Auch im Kontext der Multiagentensysteme hat sich ein Top-Down-Ansatz etabliert, wie etwa der GAIA-Ansatz zum Entwurf agentenorientierter Softwaresysteme von Wooldridge u. a. (2000). Interessant ist hierbei, dass die modellierten Agenten sehr starke Charakteristika von Diensten aufweisen, da sie in ihrer Autonomie und Proaktivität eingeschränkt werden. Im Vordergrund bei diesem Ansatz stehen Agenten, ihre Rollen und die Interaktionen zwischen Agenten, die in einem Top-Down-Ansatz verfeinert werden. Eine Diskussion des Ansatzes findet sich bei Jakob und Weiß (2004) (S. 48ff).

Regeln, Ereignisse und Ontologien

Regeln werden in verschiedenen Vorgehensmodellen wie etwa bei Stein und Ivanov (2007) auf verschiedenen Ebenen des Modells eingesetzt. Sie dienen einmal als Rahmen für strategische Entscheidungen und werden in diesem Fall eher informell beschrieben. Ein Beispiel wäre, dass Zulieferer bevorzugt aus Osteuropa stammen sollen. Sie dienen des Weiteren der Einhaltung von Policies in Unternehmen. In diesem Fall lassen sie sich meist formalisieren und automatisiert überprüfen. Ein Beispiel wäre etwa das Vier-Augen-Prinzip bei größeren Geschäftstransaktionen. Zudem dienen Regeln der konkreten Ablaufsteuerung in Geschäftsprozessen. Sie initialisieren in diesem Fall Workflows und weisen ihnen Ressourcen zu. So kann beispielsweise beim Eingang einer Kundenbestellung diese gleich durch einen entsprechenden Workflow bearbeitet werden. Die Regel weist diesem Workflow gleichzeitig den dem Kunden entsprechenden Kundenbetreuer zu.

Kharbili und Stein (2008) und Ly u. a. (2008) weisen darauf hin, dass eine deklarative Angabe von Regeln auf den konkreten Fachobjekten bislang am erfolgversprechendsten sei und sich vielfach bewährt habe. Dieser Ansatz entspricht dem hier vertretenen Ansatz, Fachobjekte durch Ontologien darzustellen. Auf den Konzepten dieser Ontologien können dann Regeln definiert werden, die zu entsprechenden Aktionen führen. Hierdurch lassen sich die verschiedenen Ebenen der Policy-Überwachung und der Geschäftsregeln zu einer Darstellung vereinfachen, auch wenn sie im praktischen Einsatz der Übersichtlichkeit halber separiert werden sollten. Regeln können dabei entweder Workflows aktivieren oder sie können Zustandsübergänge unterdrücken. Während letztere Möglichkeit vermehrt im Rahmen der Policy-Implementation Verwendung finden dürfte, sind jedoch beide Regelarten in beiden Anwendungsgebieten möglich. So kann etwa eine in der Policy deklarierte Ausnahme eine entsprechende Behandlung initiieren, sobald ein bestimmter Systemzustand erreicht ist.

Die Entscheidung, wann ein Aufeinanderfolgen von Bearbeitungsschritten eines Prozesses durch Regeln beschrieben werden sollte und wann dies durch einen Workflow dargestellt wird, ist keinesfalls immer eindeutig zu treffen. So gibt es Systeme, die vollständig auf Regeln setzen und in denen jeder Workflow implizit durch Regeln beschrieben wird (vgl. Orriëns u. a., 2005). Dies bietet zwar den Vorteil einer

weit höheren Flexibilität, die jedoch durch den Nachteil erkauft wird, dass es wesentlich schwieriger ist, die Prozesse zu analysieren, und dass die Modelle weitaus weniger intuitiv zu erstellen sind. Aus diesem Grund hat sich vielfach ein hybrider Ansatz durchgesetzt, der auch in dieser Arbeit verfolgt werden soll. Einen solchen Ansatz verfolgt beispielsweise Fresco (vgl. Piccinelli u. a., 2003; Zirpins und Piccinelli, 2004; Zirpins, 2007). Regeln steuern dabei die Einhaltung von Policies und übernehmen die Koordination größerer Workflowfragmente. Solche Regeln werden beispielsweise von Orriëns u. a. (2005) als *Management Rules* bezeichnet und gegenüber *Development Rules* zur Erstellung der Workflowfunktionalität abgegrenzt.

7.2.3 Einbettung in den PAOSE-Ansatz

Der im Rahmen dieser Arbeit verwendete Ansatz baut auf dem Ansatz der petrinetzbasierten agenten-orientierten Softwareentwicklung auf, modifiziert diesen jedoch teilweise. Dieser Ansatz wird daher zunächst kurz erläutert. Anschließend erfolgt eine detailliertere Schilderung der hier verwendeten Architektur im nächsten Abschnitt.

Der PAOSE- und der PAS-Ansatz

Als Kontext der Umsetzung dient der Ansatz der petrinetzbasierten agenten-orientierten Softwareentwicklung (Paose-Ansatz) von Moldt (2006). (Eine detailliertere Betrachtung des Ansatzes findet sich bei Cabac (2010).) Dieser Ansatz dient der Umsetzung agenten-orientierter Systeme durch petrinetzbasierte Modelle. Der Paose-Ansatz unterscheidet zwischen den Rollen, die Agenten implementieren, und den Interaktionen zwischen diesen Rollen. Rollen charakterisieren das Verhalten eines Agenten und sind somit mit einer Verallgemeinerung des Dienstkonzeptes zu vergleichen. Wie Dienste können auch Rollen von mehreren Agenten erfüllt werden und ein Agent kann aus einer Menge verschiedener Rollenträger etwa nach Qualitätskriterien auswählen, mit welchem er interagieren möchte.

Da Dienste jedoch weniger kollaboratives Verhalten aufweisen als Agenten, wird der Paose-Ansatz hier der dienstbasierten Entwicklung von Softwaresystemen angepasst. Somit stehen nicht mehr in dem Maße die Interaktionen zwischen Agenten im Vordergrund sondern stattdessen die Delegation der Ausführung funktional abgegrenzter Teilprozessen an Dienste. Aus dem kollaborativen Ansatz wird somit eine eher hierarchische Herangehensweise der Modellierung. Mit einem weniger starken Fokus auf der Interaktion ist auch die Verwendung der von Odell u. a. (2000) eingeführten Agenteninteraktionsdiagramme nicht in jedem Fall sinnvoll. Zwar ist die Überführung der Agenteninteraktionsdiagramme in Petrinetze, wie sie von Cabac u. a. (2003), Cabac und Moldt (2004) und Cabac (2010) beschrieben wird, auch im Dienstbereich denkbar, jedoch orientieren sich Ansätze wie die BPMN (vgl.

Abschnitt 2.2.3) stärker an den eigentlichen Unternehmensprozessen und dürften folglich intuitiver in diesem Kontext sein.

Moldt und Ortmann (2009) führen den PAS-Ansatz ein, was für Petrinetze, Agenten und Dienste (Services) steht. Zur Vereinheitlichung der Konzepte von Diensten und Agenten unterscheiden Moldt und Ortmann (2009) im PAS-Ansatz die Modellbestandteile

- **Fachobjekt**, um die relevanten Datentypen des Modells zu repräsentieren,
- **Geschäftsrolle**, zur Beschreibung des Verhaltens von Agenten und Diensten, so dass Dienste eine Spezialisierung dieser Art von Rollen sind, und
- **Geschäftsprozess** zur Koordination der Bearbeitung der durch Fachobjekte repräsentierten Objekte im Unternehmen und zur Koordination der Aufrufe der entsprechenden Dienste und Agenten.

Durch die Wahl des Begriffs Geschäftsprozess wird stärker der mögliche kollaborative Charakter der Interaktion in den Vordergrund gestellt, als der Begriff Workflow dies erlauben würde. Agenten übernehmen dabei die nicht automatisierbaren Prozesse und die Entscheidung für die Verwendung bestimmter Workflows anhand von Regeln. Der Übergang zwischen der Rolle eines Agenten und einem Dienst ist dabei fließend, wovon durch die Verwendung des Begriffs Geschäftsrolle abstrahiert wird. Workflows werden jedoch im Allgemeinen durch Dienste gekapselt, die ihre Ausführung übernehmen. So kann beispielsweise beim Eintreffen einer Anfrage eines Kunden zunächst eine Verhandlung zwischen Agenten (etwa den Vertriebsmitarbeitern) notwendig sein, bevor dann ein Workflow durch einen Dienst ausgeführt wird. Agenten bezeichnen dabei nicht nur technische Systeme, sondern umfassen auch menschliche Akteure.

Mulan

Eng mit dem PAOSE-Ansatz verwandt ist die petrinetzbasierte Agentenplattform MULAN, deren Konzeption auf Rölke (1999, 2004) zurückgeht. MULAN teilt ein Multiagentensystem in vier Ebenen ein, wobei jede Ebene eine Verfeinerung der darüber liegenden Ebene ist. Beginnend mit der Ebene des Multiagentensystems folgt die Ebene der Agentenplattform, die Ebene des Agenten und die Ebene der Protokolle. Dies erlaubt es, das Verhalten von Agenten immer detaillierter zu untersuchen. Ein entsprechender Ansatz wird auch hier für Dienste in Abschnitt 7.3.1 vorgestellt.

Durch die Ergänzung CAPA von Duvigneau (2002) ist MULAN ein vollständiges Multiagentensystem, das zudem die Möglichkeit besitzt, mit anderen Agentenplattformen zu kommunizieren. Gleichzeitig können die Abläufe in MULAN in Form von Protokollen als Petrinetze modelliert werden. Diese Möglichkeit der Modellierung

von Abläufen als Netz einerseits und deren Ausführung in einem konkreten Kontext andererseits soll im hier verfolgten Ansatz ebenfalls übernommen werden.

Mulan wurde zudem um die Möglichkeit erweitert, Konzepte über Ontologien festzulegen, die dann von Agenten zur Kommunikation verwendet werden. Die Interaktion zwischen Agenten wird anhand der eintreffenden Nachrichten anhand von Regeln bestimmt. Somit ist auch in Mulan/Capa die Auswahl der Abläufe durch Regeln bestimmt, die von Agenten ausgewählt werden. Hier ergeben sich deutliche Parallelen zwischen dem Mulan zugrundeliegenden Modell und der Modellierung von Diensten, wie sie im letzten Unterabschnitt vorgestellt wurde. Die hieraus entstehende Modellierungstechnik ist Gegenstand des nächsten Unterabschnitts.

Vergegenwärtigt man sich, dass Agenten auch menschliche Akteure sein können, so ist der Schritt vom Multiagentensystem zur Organisation relativ naheliegend. Entsprechende Überlegungen finden sich beispielsweise in Wester-Ebbinghaus und Moldt (2008a,b). Diese Überlegung ist auch für die Modellierung von Geschäftsprozessen sinnvoll, da hierdurch der Top-Down- respektive der Middle-Out-Ansatz vieler Vorgehensmodelle unterstützt wird. Darüber hinausgehend finden sich bei von Lüde u. a. (2003, 2009) Ansätze zur Modellierung von soziologischer Organisationstheorien mit Petrinetzen.

7.3 Eine petrinetzbasierte Dienstarchitektur

Aus dem vorausgegangenen Abschnitt lassen sich wesentliche Charakteristika der Vorgehensmodelle zum Entwurf dienstbasierter Systeme erkennen. Fast alle Vorgehensmodelle verfolgen einen Top-Down-Ansatz, in den bestehende Funktionalität durch Dienste eingebettet wird, was häufig als Middle-Out-Ansatz bezeichnet wird. Hierzu werden ausgehend von den strategischen Geschäftszielen die Geschäftsprozesse ermittelt und diese dann durch Workflows modelliert, sofern ihr Grad an Automatisierung dies zulässt. Bestehende Funktionalität wird durch Dienste gekapselt und in das Modell mit einbezogen. Modernere Ansätze verwenden zudem meist eine semantische Beschreibung der Daten anhand von Ontologien, die dann auch zur Definition der Regeln genutzt werden können. Diese Regeln setzen einerseits die Geschäftslogik um und ermöglichen andererseits die Einhaltung von Policies.

Die in Abschnitt 2.2.2 betrachteten fünf verschiedenen Sichten auf Geschäftsprozesse werden hier wieder aufgegriffen. Sie werden jedoch auf lediglich drei Sichten reduziert: Die Organisationssicht, die Datensicht und die Prozesssicht. Letztere fasst dabei die Steuerungs- und die Funktionssicht aus Abschnitt 2.2.2 zusammen.

Motiviert durch das Top-Down-Vorgehen vieler Vorgehensmodelle werden, dem Agentenmodell von Rölke (2004) folgend, vier Ebenen der Modellierung betrachtet, die jeweils einen unterschiedlichen Grad an Detaillierung der Modellierung wiedergeben. Dies erlaubt es, Daten und Teilprozesse unterschiedlichen organisationalen Einheiten zuzuweisen, was die Modellierung komplexer Systeme unterstützt.

Die Einteilung der Ebenen orientiert sich dabei an der Organisation des Gesamtsystems und unterscheidet die Ebenen Organisation, Abteilung, Rollenträger und Dienstimplementation. Sowohl organisatorische Elemente als auch Prozesse und Daten können diesen Ebenen zugeordnet werden, was in den folgenden Unterabschnitten detaillierter betrachtet wird. Da die Organisation dabei die Grundlage für die Ebenen bildet, folgt zunächst das Ebenenmodell der Organisationssicht.

7.3.1 Ebenen der Organisationssicht

Der hier verfolgte Ansatz, verschiedene Detaillierungsebenen zur Verfügung zu stellen, orientiert sich an der petrinetzbasierten agentenorientierten Softwareentwicklung und verwendet das in Köhler u. a. (2001) vorgestellte Modell der vier Ebenen. Dieses wurde ursprünglich für Multiagentensysteme entworfen und wurde in Moldt u. a. (2004b, 2005) an die Erfordernisse von dienstorientierten Architekturen angepasst. An dieser Stelle ist der Fokus der Modellierung etwas weniger technisch als in Moldt u. a. (2005), so dass sich die Ebenen leicht unterscheiden. Die Ebenen lassen sich weitestgehend direkt von den in Rölke (2004) vorgestellten Ebenen übernehmen. Vergegenwärtigt man sich, dass ein Diensttyp eine genau definierbare Teilrolle eines Agenten ist, so wird die Nähe eines Multiagentensystems zur dienstorientierten Architektur deutlich. Bei letzterer sind lediglich Agenten im System zugelassen, die keinerlei Autonomie und keinerlei proaktives Verhalten besitzen.

Dieser Unterabschnitt stellt zunächst die Ebenen allgemein vor, um dann jede Ebene im Kontext der Ressourcen- und Organisationssicht detaillierter zu betrachten.

Die Ebenen

Die Ebenen des Modells sind an die entsprechenden Ebenen von Köhler u. a. (2001) angelehnt. Um die organisationale Einbettung zu betonen, wurde für die erste Ebene statt des Begriffs des Multiagentensystems die Bezeichnung Organisation gewählt. Die zweite Ebene wird entsprechend nicht als Agentenplattform, sondern als Abteilung bezeichnet. Auf der dritten Ebene – der Ebene des Agenten in Mulan – wird dann entsprechend ein Dienst angeboten, so dass diese Ebene als Dienstebene bezeichnet wird. Ein Dienst ist dabei eine konkrete Realisierung einer Dienstschnittstelle beziehungsweise eines Diensttyps. Die konkrete Realisierung bezieht sich dabei weniger auf die tatsächliche Implementierung als vielmehr auf eine konkrete Auswahl von Ressourcen und weiteren Diensten zur Realisierung des Dienstes. Diese Realisierung ist durch das Dienstprotokoll gegeben, das abstrakt das Verhalten des Dienstes beschreibt. Auch hier existiert wiederum eine starke Analogie zum Verhaltensprotokoll des Agenten aus Mulan. Die Ähnlichkeit der Ebenen ist auch in Abbildung 7.3 erkennbar, wo die folgenden vier Ebenen unterschieden werden:

- die Ebene des Organisationsnetzwerks,

- die Ebene der Abteilung,
- die Ebene des Dienstes und
- die Ebene des Dienstprotokolls.

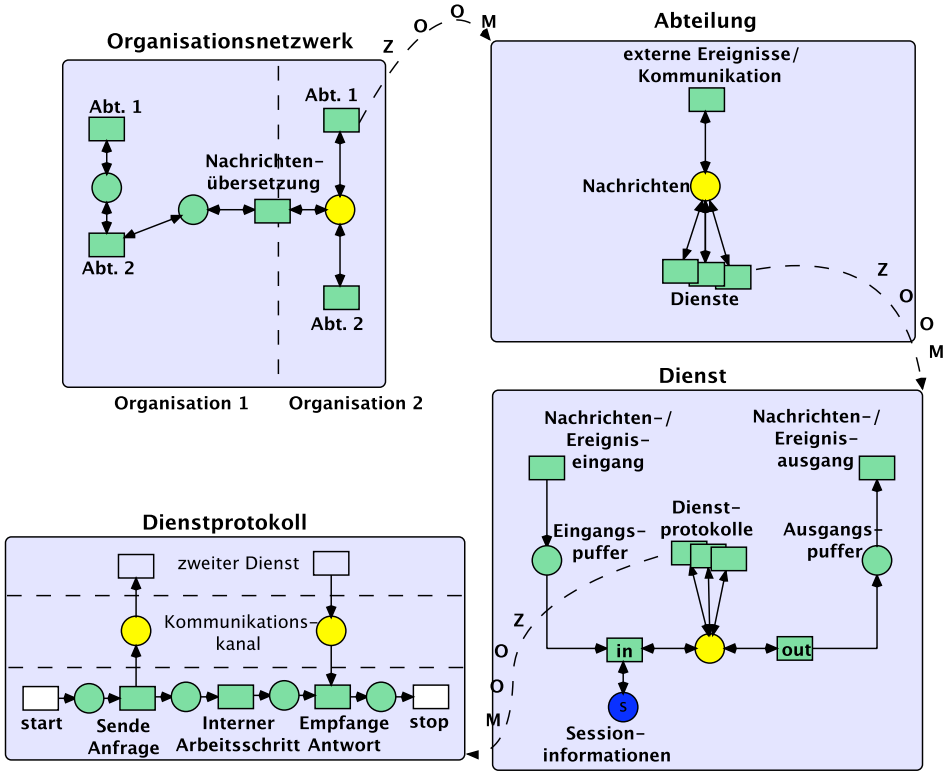


Abbildung 7.3: Die vier Ebenen einer petrinetzbasierten Dienstarchitektur

Abbildung 7.3 ist eine Abwandlung einer Abbildung aus Moldt u. a. (2005). Die Abbildung gibt einen Überblick über die vier Ebenen und ihre Beziehungen zueinander. Jede der vier Ebenen repräsentiert dabei einen speziellen Aspekt des Dienstmodells, der einen gewissen Detaillierungsgrad widerspiegelt. Die Ebenen erweitern dabei jeweils einen Aspekt des durch eine Transition oder eine Stelle repräsentierten Elements des Systems.

Im Gegensatz zu Moldt u. a. (2005) und zu Rölke (2004) ist die Sichtweise hier weniger technisch getrieben. Sie nimmt stattdessen stärker Rücksicht auf den Aufbau der Organisation. Dies reflektiert die im vorigen Abschnitt diskutierte Top-Down-Vorgehensweise vieler Vorgehensmodelle zur Erstellung dienstbasierter Systeme.

Die obere linke Teilabbildung zeigt die Organisationen des Gesamtsystems. Hier lassen sich verschiedene Abteilungen identifizieren, die an einem Prozess beteiligt

sind. Auf dieser Ebene lässt sich das Zusammenspiel von Teilen verschiedener Organisationen oder Unternehmen erkennen. Eine detailliertere Darstellung der Abteilung ist in der oberen rechten Ecke gegeben. Das dargestellte Netz repräsentiert eine Verfeinerung der Abteilungstransition.

In jeder der Abteilungen einer Organisation gibt es verschiedene von dieser Abteilung angebotene Dienste, die über einen speziellen Dienst – den Verzeichnisdienst – aufgefunden und angesprochen werden können. Jeder dieser Dienste stellt eine Detaillierung der `Dienst`-Transition dar. Somit müsste hier für jeden Dienst eine eigene Transition existieren. Diese wurden jedoch schematisch zu einer Transition zusammengefasst. Ein Dienst ist somit ein transitionsberandetes Subnetz, das über Kanten an die gleichen Stellen wie die `Dienst`-Transition angebunden ist. Abbildung 7.4 stellt die Ersetzung der Transitionen durch entsprechende Subnetze auf der jeweiligen Ebene dar.

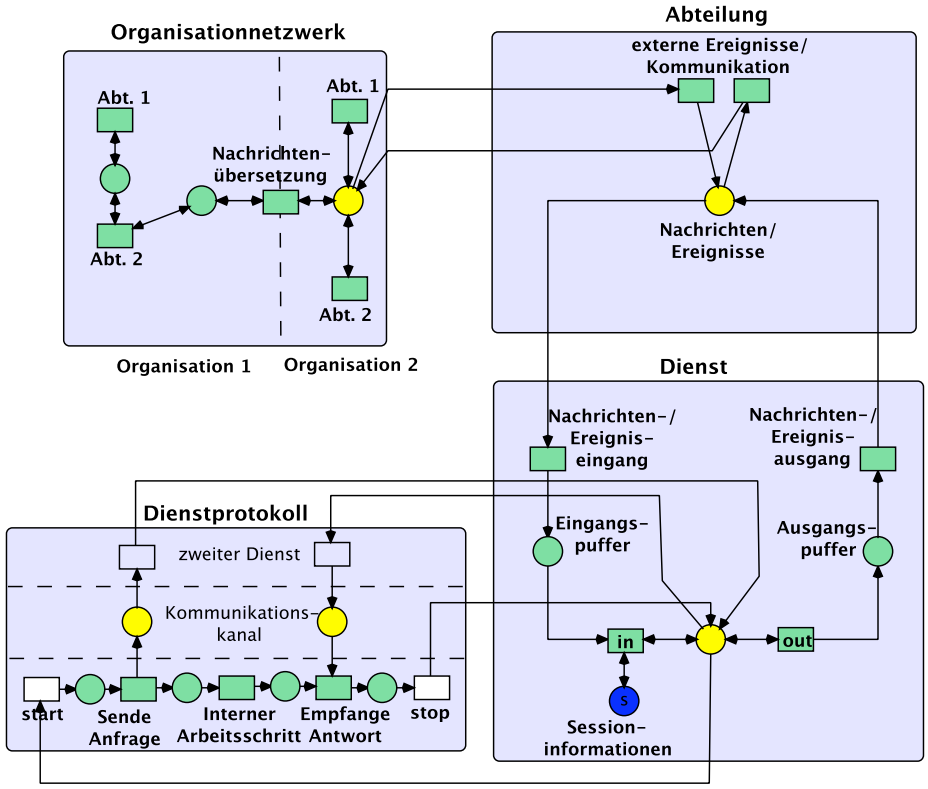


Abbildung 7.4: Explizite Darstellung der Kanten in der Dienstarchitektur

Die Erweiterung der Netzstruktur stellt einen wesentlichen Unterschied gegenüber der Verfeinerung von Marken in der MULAN-Architektur nach Rölke (2004) dar. Dies ist darin begründet, dass die Migration im Allgemeinen nicht zu den charakteristischen Fähigkeiten der Dienste gehört, und es daher nicht notwendig

ist, Dienste als Marken in einem Netz darzustellen. In einem dienstbasierten System werden vielmehr ausschließlich Nachrichten ausgetauscht. Zudem geschieht dies asynchron, so dass eine Kommunikation über synchrone Kanäle (vgl. Christensen und Hansen, 1993), wie dies in MULAN modelliert ist, nicht sinnvoll erscheint. Die jeweils durch den Zoom-Pfeil angegebene nächste Ebene in Abbildung 7.3 stellt daher auch keine Verfeinerungen der Marken dar, sondern repräsentiert vielmehr eine Erweiterung der jeweiligen Nachrichtenstellen durch ein transitionsberandetes Subnetz.

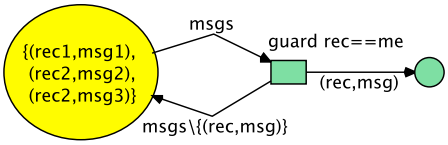


Abbildung 7.5: Nachrichtenkommunikation im System

Die Darstellung des Systems in Abbildung 7.4 ist kein ausführbares Petrinetz. Einerseits fehlen die Marken auf den Stellen und andererseits ist die Kommunikation nur schematisch dargestellt. In der tatsächlichen Umsetzung erfolgt die Kommunikation stets über Mengen von Nachrichten, so dass alle Kanten zu den gelb eingefärbten Kommunikationsstellen Doppelkanten sind. Dies führt jedoch dazu, dass der Nachrichtenfluss nicht mehr erkennbar ist, weswegen die Darstellung in Abbildung 7.4 keine Doppelkanten verwendet. Die Umsetzung der Kommunikation wird in Abbildung 7.5 dargestellt. Auf jeder Nachrichtenstelle liegt eine Menge von Nachrichten, die vereinfacht durch den Empfänger `rec` und den Inhalt `msg` beschrieben wird. Jede Abteilung beziehungsweise jeder Dienst extrahiert aus dieser Menge der Nachrichten die für ihn relevanten Nachrichten. In Abbildung 7.5 geschieht dies allein anhand des Empfängers einer Nachricht. Ebenso wäre hier jedoch eine Priorisierung anhand verschiedener Kriterien denkbar. Die Nachrichten können dann im entsprechenden Subsystem weiterverarbeitet werden.

Ebene des Organisationsnetzwerks

Die Ebene des Organisationsnetzwerks stellt die erste Ebene des Modells dar. Die Transitionen modellieren auf dieser Ebene die Abteilungen als elementare Organisationseinheiten mit den von ihnen angebotenen Diensten. Die Stellen repräsentieren die Kommunikationskanäle, über die asynchron kommuniziert werden kann. Sie stellen also den logischen Bus dar, über den organisationsintern und organisationsübergreifend kommuniziert wird. Die Marken auf den Stellen stellen die Nachrichten und Ereignisse dar, die zwischen den Abteilungen ausgetauscht werden. Durch Typisierung der Kanäle kann festgelegt werden, welche Ereignisse und welche Nachrichten zwischen den Abteilungen ausgetauscht werden können. So wird

die Interaktion verschiedener interagierende Organisationen und Abteilungen auf dieser Ebene dargestellt.

Organisationsnetzwerke repräsentieren einen Ausschnitt verschiedener Organisationen und deren Interaktion. Die an diesem Netzwerk beteiligten Organisationseinheiten dienen dabei der Erfüllung eines gemeinsamen Zwecks. Dies kann etwa die Abarbeitung eines Auftrags sein. Wie dieser Zweck dann durch einen Geschäftsprozess erreicht wird und wie er durch einen Workflow automatisiert wird ist dann Gegenstand weiterer Detaillierung in den folgenden Ebenen respektive Gegenstand der Prozesssicht in Unterabschnitt 7.3.3.

In Abbildung 7.3 ist ein Gesamtsystem bestehend aus zwei Organisationen zu sehen, die jeweils zwei Abteilungen besitzen. Die Abteilung 2 von Organisation 1 kann dabei mit beiden Abteilungen von Organisation 2 direkt kommunizieren. Abteilung 1 von Unternehmen 1 kann hingegen nur über die Abteilung 2 mit Organisation 2 kommunizieren. Zwischen den Organisationen ist in dem dargestellten Fall keine direkte Kommunikation über einen verteilten Bus möglich. Vielmehr müssen Nachrichten der einen Organisationen in Nachrichten der anderen Organisation übersetzt werden, was durch einen speziellen Dienst geschieht, der zudem Sicherheitsrestriktionen implementieren kann. Durch Kantenanschriften und durch eine Typisierung der Stellen kann die Art der ausgetauschten Nachrichten respektive der übermittelten Ereignisse detaillierter beschrieben werden. In dieser Ebene lässt sich erkennen, welche Abteilungen miteinander kommunizieren können, und es kann der Grad der Vernetztheit eines Organisationsteils dargestellt werden.

Ebene der Abteilung

Der recht grobe Überblick auf Organisationsebene wird auf Abteilungsebene verfeinert. Hier werden die Nachrichten, die eine Abteilung betreffen, entgegengenommen und durch entsprechende Dienste weiterverarbeitet. Die in Abbildung 7.3 durch die `Dienste`-Transition repräsentierten Dienste umfassen dabei eine Vielzahl verschiedener Dienste, die wiederum über einen gemeinsamen Bus kommunizieren. Da Dienste im Allgemeinen nicht ihren Ausführungsort wechseln, werden sie als Kopie der Transition `Dienste` modelliert, weswegen diese Transition mehrfach existiert. Sie sollte dabei streng genommen für jeden Dienst einmal in Abbildung 7.3 existieren. Die Zuordnung der Nachrichten zu einem bestimmten Dienst lässt sich in gefärbten Netzen über verschiedene Farben der Nachrichten lösen. Die Ereignisverarbeitung wird von (einem oder mehreren) regelbasierten Diensten übernommen, die ebenfalls eine Verfeinerung der Transition `Dienste` darstellen. Allgemein lassen sich ereignisverarbeitende Dienste als speziell implementierte Dienste auffassen, die aufgrund einer Eingangsnachricht in Form eines Ereignisses eine Ausgangsnachricht erzeugen, die einen verarbeitenden Dienst aktiviert.

Die Darstellung von Ereignissen und Nachrichten ist schematisch zusammengefasst, da ihre Weiterverarbeitung in beiden Fällen durch Dienste erfolgt. Der Ein-

gang eines Ereignisses oder der Eingang einer Nachricht führt dazu, dass ein Dienst ausgeführt wird, indem die Nachricht an ihn weitergeleitet wird. Die Nutzung einer gemeinsamen Nachrichtenstelle stellt wiederum den asynchronen Charakter der Nachrichtenverarbeitung über einen Bus dar. Eine Nachricht wird dann einer der Randtransitionen zur Kommunikation oder zur Ereignisverarbeitung übermittelt und entsprechend bearbeitet.

Abteilungen stellen organisatorische Einheiten einer Organisation dar, die über Nachrichten angesprochen werden oder die Ereignisse verarbeiten. Die von einer Abteilung bereitgestellten Dienste können allgemeiner Natur sein, wie etwa die Möglichkeit des Versandes von Emails oder des Zugriffs auf eine Datenbank, oder sie können spezielle fachliche Anforderungen bedienen, wie etwa das Bearbeiten eines Kundenwunsches. Die interne Kommunikation zwischen Diensten erfolgt dabei über die `Nachrichten`-Stelle, während die externe Kommunikation über spezielle Nachrichtenkanäle erfolgt. Jeder Dienst filtert dabei die ankommenden Nachrichten und Ereignisse danach, ob sie für ihn relevant sind. Aus Sicherheitsgründen wird hier in der Regel zudem dafür gesorgt, dass nur bestimmte Dienste auf Nachrichten zugreifen können.

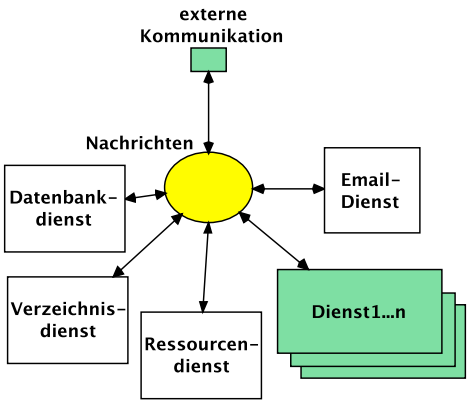


Abbildung 7.6: Detailliertere Ansicht der Abteilungsebene

Eine detailliertere Ansicht der im oberen rechten Quadranten von Abbildung 7.3 dargestellten Abteilungsebene erlaubt es darzustellen, welche Dienste in einer Abteilung zur Verfügung stehen. Eine mögliche Verfeinerung ist in Abbildung 7.6 dargestellt. Die Abteilung stellt mögliche interne Dienste bereit, die ein Dienst nutzen kann, und bietet dem Dienst eine Laufzeitumgebung, über die er Anfragen erhält und über die komplexe Dienste auch selbst Anfragen an andere Dienste stellen können.

In Abbildung 7.6 sind neben den nach außen sichtbaren Diensten noch verschiedene lokale Dienste modelliert, wie zum Beispiel der Datenbankzugriff und die Möglichkeit, Emails zu verschicken. Aufgabe der Abteilung ist also einerseits der

Versand und der Empfang von Nachrichten und andererseits die Bereitstellung gekapselter funktionaler Einheiten als Dienst. Abteilungsintern können hierbei verschiedene Kommunikationsprotokolle zugelassen sein, die nicht unbedingt nach außen zur Verfügung stehen.

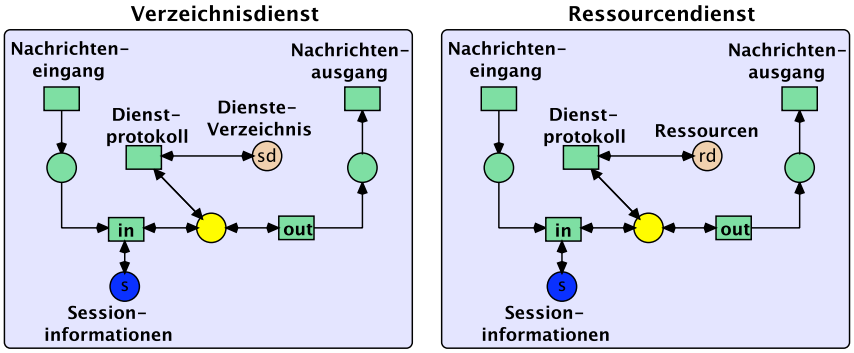


Abbildung 7.7: Verzeichnisdienst und Ressourcendienst

Neben den abteilungsspezifischen Diensten gibt es in jeder Abteilung einen Verzeichnisdienst und einen Ressourcendienst, die für das Auffinden anderer Dienste beziehungsweise für das Zuordnen von Ressourcen notwendig sind. Diese sind in Abbildung 7.7 dargestellt. Sie modellieren die im System existierenden Dienste beziehungsweise die vorhandenen Ressourcen auf eigenen Stellen. Zudem implementieren sie Zugriffsbeschränkungen auf Ressourcen und Dienste. Spezielle Dienste sind zudem ereignisauslösende Dienste, die aufgrund von Umweltbedingungen ein Ereignis erzeugen und dieses dem System über die Nachrichtenstelle mitteilen.

Ebene des Dienstes

Wie ein Agent in MULAN verarbeitet ein Dienst Nachrichten und erzeugt als Ergebnis der Verarbeitung neue Nachrichten. Um verschiedene Dienstaufrufe einer Kommunikation mit einem Dienstnutzer zuordnen zu können, werden über Session-Tokens zusammengehörige Aufrufe identifiziert. Auf diese Weise kann etwa überprüft werden, ob ein Dienstnutzer sich bereits authentifiziert hat. Das Verhalten eines Dienstes wird wie das Verhalten eines Agenten in MULAN durch Protokolle dargestellt. Diese sind Verfeinerungen der `Dienstprotokolle`-Transition, was den zentralen Charakter der Kommunikation über Nachrichten unterstreicht. Auch hier müsste es streng genommen eine eigene Transition für jedes Dienstprotokoll geben.

Jeder Dienst besitzt einen individuellen Pool von eingehenden und ausgehenden Nachrichten. Die den Dienst betreffenden Nachrichten werden dabei von der Transition `Nachrichten-/Ereigniseingang` vom Bus übernommen und dem Pool zugeführt. Die Dienstprotokolle filtern ebenfalls die für sie relevanten Nachrichten. Dies

kann in gefärbten Netzen leicht über Aktivierungsbedingungen und entsprechende Farben der Nachrichtenmarken erfolgen. Die `in`-Transition greift auf Session-Informationen zu, um die Kommunikationshistorie verfolgen zu können. Darüber hinaus kann hier auch festgehalten werden, ob sich der Aufrufer bereits autorisiert hat. Um über mehrere Aufrufe hinweg Informationen austauschen zu können, wird eine Session mit einem Zustand verwendet. Der Aufrufer muss sich in diesem Fall über einen Token identifizieren und der Dienst ist aufgrund dessen in der Lage, auf die Kommunikationshistorie zurückzugreifen.

Um seine Funktionalität bereitstellen zu können, muss ein Dienst mit anderen Diensten kommunizieren können. Diese Kommunikation ist modelliert durch die Transitionen `Nachrichten-/Ereigniseingang` und `Nachrichten-/Ereignisausgang`. Ereignisauslösende Dienste können über die Ausgangstransition beispielsweise Ereignisse kommunizieren, deren Beobachtung ihnen obliegt.

Zur Realisierung der Dienstprotokolle stellt ein Dienstanbieter im Allgemeinen den Protokollen noch eine Vielzahl von Rechenoperationen bereit, die dann in die Bearbeitung der Protokolle einfließen. Dies können einfache Operationen wie Summen sein, aber auch komplexere Berechnungen, die nicht durch einen Dienst speziell gekapselt werden.

Das dienstbasierte System lässt sich um neue Dienste ergänzen, indem die Nachrichtentransitionen eines neuen Dienstes mit der Nachrichtenstelle der Abteilung über Kanten verbunden werden. Dieser neue Dienst kann separat analysiert werden und kann über das Diensteverzeichnis des Verzeichnisdienstes aufgefunden und genutzt werden. Der Einfachheit halber soll das Gesamtsystem hier zunächst als statisch angenommen werden, so dass zur Entwurfszeit bereits bekannt ist, welche Dienste im System existieren.

Benutzt ein Dienst eine Ressource, so wird diese ebenfalls auf dieser Ebene angegeben. Ressourcen können dabei einerseits Mitarbeiter, andererseits aber auch technische Geräte oder Maschinen sein. Diese werden hier in Abgrenzung zu Stelle und Transition als Raute notiert.

Ebene des Dienstprotokolls

Das Verhalten eines Dienstes wird durch ein Dienstprotokoll modelliert. Dies sind Petrinetze, die eng mit Workflows verwandt sind, was sich in ihrer Darstellung widerspiegelt. Sie besitzen jedoch im Allgemeinen Randstellen, über die sie mit anderen Diensten kommunizieren, so dass das Gesamtsystem die Struktur eines Workflownetzes besitzt. Die Kommunikation erfolgt dabei durch Verschmelzung der Stellen.

Das Eintreffen einer Nachricht oder eines Ereignisses startet ein solches Netz, das dann die entsprechende Funktionalität des Dienstes beschreibt. Die Dienstprotokolle spezifizieren hierbei die Reihenfolge, in der andere Dienste kontaktiert werden müssen und in der die Daten verarbeitet werden müssen. Dies ist eng verwandt mit

dem was Beschreibungen in BPEL4WS Andrews u. a. (2005) für zusammengesetzte Dienste leisten. Petrinetze erlauben hier eine allgemeine Modellierung. Zudem sind Beschreibungen in BPEL4WS oder OWL-S in Petrinetze überführbar (vgl. Stahl, 2005; Moldt und Ortmann, 2004b).

Zur Vereinheitlichung der Darstellung werden nicht nur die Abläufe durch Protokollnetze dargestellt, sondern auch die Regeln zur Ereignisverarbeitung so repräsentiert. Dies erlaubt eine einheitliche Sichtweise auf die verschiedenen Arten von Diensten.

Im unteren linken Quadranten von Abbildung 7.3 ist beispielhaft ein Protokollnetz dargestellt. Dieses Netz ist hier ohne Daten und ohne Funktionalität dargestellt. Durch die Verwendung der in Kapitel 4 beschriebenen Dienstbeschreibungsnetze kann ein solches Protokollnetz beliebige andere Netze aufrufen und Berechnungen durchführen. Die Kommunikation erfolgt dabei asynchron über gemeinsame Stellen mit anderen Diensten, die verschmolzen werden. Der Kommunikationskanal ist in diesem Fall durch gelbe Stellen dargestellt. Diese gelben Stellen repräsentieren den gesamten Kommunikationskanal, der durchaus mehr als eine Stelle umfassen kann, der sich jedoch auf eine Stelle reduzieren lässt.

Zur strukturierten Beschreibung der Dienstprotokolle können die auf Cabac u. a. (2003) zurückgehenden Netzkomponenten verwendet werden, was von Moldt u. a. (2005) erläutert wurde. Netzkomponenten sind Netz-Templates, die häufig wiederholt auftretende typische Strukturen in einem Verhaltensprotokoll beschreiben. Sie ermöglichen eine leichtere Lesbarkeit des Netzes und ermöglichen eine schnellere Modellierung. Ihre Entstehung ist an die Workflow-Muster von van der Aalst u. a. (2000) und von Moldt und Rölke (2003) angelehnt, so dass sie die wichtigsten Workflow-Konstrukte reflektieren.

7.3.2 Repräsentation der Datensicht durch Ontologien

Die Diagramme der Ebenen des vergangenen Unterabschnittes dienen in erster Linie der Darstellung der Organisationssicht, auch wenn zum Teil Elemente der Kontrollfluss- und der Anwendungssicht dargestellt werden. Bislang fehlt jedoch eine explizite Darstellung der Datensicht. Dieser Unterabschnitt nimmt sich der Darstellung der Daten an. Im darauffolgenden Unterabschnitt wird dann noch einmal die Möglichkeit zur Repräsentation der Funktionalität in Protokollen thematisiert.

Verwendung von Ontologien

Der Austausch von Nachrichten in dienstbasierten Systemen macht eine eindeutige Definition der in diesen Nachrichten verwendeten Datentypen notwendig. Die Verwendung von Ontologien ermöglicht es in diesem Zusammenhang, von konkret ausgetauschten Datenformaten zu abstrahieren und auf einer höheren Abstraktionsebene die entsprechenden Konzepte zu modellieren. Das Modell kann somit stärker

von technischen Details abstrahieren und besitzt eine größere Nähe zum Anwendungskontext.

Durch die Verwendung einer Ontologie kann für konkrete Nachrichtenformate angegeben werden, wie die Konzepte der Ontologie auf diese Nachrichtenformate abgebildet werden sollen. Die Sprache OWL-S (vgl. Martin u. a., 2004a) nutzt hier etwa die Möglichkeit, über Groundings Konzepte auf die Web Service Description Language (WSDL, 2001) abzubilden, so dass bei Aufrufen von Diensten Konzepte einer Ontologie verwendet werden können. Gerade bei XML-basierten Datenformaten wird diese Möglichkeit bereits von einer Reihe von Umsetzungen des ESB-Konzeptes unterstützt. Somit ist es möglich, ein Modell auf Ebene der verwendeten Fachobjekte zu erstellen, die durch eine Ontologie modelliert sind.

Neben der Darstellung der Datentypen zur Verwendung im technischen System bietet eine Ontologie zudem eine zentrale Instanz, in der die Datentypen eines Systems dokumentiert und erläutert werden können. Dies löst das in vielen klassischen Vorgehensmodellen verwendete Glossar zur Beschreibung der wichtigsten Artefakte eines Softwaresystems ab und präzisiert gleichzeitig die verwendeten Daten. In einem Top-Down-Ansatz können zunächst die wichtigsten Fachobjekte definiert werden, die dann bei einer späteren detaillierteren Darstellung des Systems um weitere Attribute und weitere Fachobjekte ergänzt werden.

Der Zustand eines Systems wird einerseits durch die Prozesse bestimmt und andererseits durch die aktuellen Werte der modellierten Fachobjekte. Somit ist es notwendig, dass entsprechende Veränderungen von Fachobjekten auch im System ausgedrückt werden können. Ein Ausdrucksmittel, welches die Modellierung deutlich intuitiver gestaltet, welches jedoch aufgrund der objektorientierten Wurzeln vieler Softwaresysteme auch in dienstbasierten Systemen eher selten umgesetzt wird, ist die Verwendung von dynamischen Konzepten oder Rollen. Mit ihrer Hilfe ist es möglich, dass Objekte zur Laufzeit einen weiteren Typ annehmen. So können beispielsweise Personen zu Kunden werden und folglich weitere Attribute aufweisen, die sie zuvor nicht hatten. Die Verwendung dynamischer Konzepte ist Gegenstand des folgenden Unterabschnitts.

Dynamische Konzepte

Die Verwendung dynamischer Konzepte in Form von Rollen wurde bereits in Abschnitt 2.4.2 thematisiert. Sie erlaubt häufig eine wesentlich intuitivere Herangehensweise an die Modellierung. So ist es in der Regel leichter zu vermitteln, dass eine Person weitere Attribute bekommt und nun Kunde ist, als dass entweder ein zweites Objekt existiert, dass den Kunden repräsentiert oder die Person ein Attribut Rollen besitzt, über das dann die Kundeneigenschaft modelliert ist. Aus diesem Grund wird in dem Ansatz dieser Arbeit von der Möglichkeit dynamischer Konzepte Gebrauch gemacht. Im Unterschied zu vielen modernen objektorientierten Programmiersprachen werden also Rollen nicht als Adjunkte aufgefasst, sondern

als tatsächliche Typänderung wie in Abschnitt 2.4.2 erläutert wurde.

Dies bietet zwar einerseits den Nachteil, dass ein Konzept jede Rolle nur einmal erfüllen kann, erlaubt es jedoch, dass das Objekt und seine Rollen stets die gleiche Identität besitzen und dass nicht indirekt auf die Rollen zugegriffen werden muss. Anders als in der objektorientierten Programmierung könnte dies andernfalls bei Konzepten einer Ontologie nicht durch den Zugriff über Methoden gekapselt werden, da Konzepte keine solchen Methoden besitzen, und würde daher zu komplizierten Modellen führen. Dies weist Ähnlichkeit zu Überlegungen aus dem Bereich der objektorientierten Datenbanken auf wie sie beispielsweise Wieringa u. a. (1995) beschreibt.

Dass ein Konzept jede Rolle lediglich einmal erfüllen kann, mag zunächst als Einschränkung gesehen werden. So ist es beispielsweise möglich, dass ein Angestellter bei mehreren Firmen arbeitet und somit die Rolle *Angestellter* mehrfach erfüllt. Eine solche Sicht ist jedoch mit Blick auf die Modellierung von Prozessen schwierig, da einem Prozess meistens eine konkrete Rolle übergeben werden soll und nicht durch Zufall entschieden werden soll, welche Rolle gewählt wird. So ist es bei einem Prozess einer Firma A im Allgemeinen nicht sinnvoll eine Rolle *Angestellter* zu übergeben, die einen Angestellten der Firma B repräsentiert. Aus diesem Grunde wird von der Möglichkeit abgesehen, dass ein Objekt dieselbe Rolle mehrfach innehaben kann. Wenn ein Objekt verschiedene Subrollen R_1 und R_2 einer Rolle R innehaben soll, so kann dies durch eine weitere Subrolle $R_{1\cap 2}$ modelliert werden, die die Rollen R_1 und R_2 zusammenfasst und somit die Schnittmenge der Elemente der jeweiligen Rollen repräsentiert.

Im Falle des Angestellten könnte es etwa eine Rolle *Angestellter* sowie die beiden Subrollen *Angestellter bei Firma A* und *Angestellter bei Firma B* geben. Diese beiden Rollen könnten wiederum eine gemeinsame Subrolle *Angestellter bei Firma A und B besitzen*. Jede dieser Rollen besitzt ein Objekt jedoch nur einmal[4].

Eine Rolle als dynamisches Konzept kann von einem Objekt angenommen und abgelegt werden, ohne dass andere Rollen, die sich nicht in einer Sub- oder Supertypbeziehung zu der betroffenen Rolle befinden, beeinträchtigt werden. Insbesondere können auch mehrere Rollen von einem Objekt wahrgenommen werden. So kann ein Mensch gleichzeitig Freund, Student und Angestellter sein. Beendet er das Angestelltenverhältnis, so bleiben die beiden übrigen Rollen davon unbetroffen.

Dies unterscheidet Rollen von nicht-dynamischen Konzepten, die hier als natürliche Typen bezeichnet werden sollen. Der natürliche Typ eines Objektes ändert

[4] Denkbar wäre etwa, dass die Rolle *Angestellter* ein Attribut *arbeitet_bei* besitzt, welches die beiden Subrollen dann auf die entsprechenden Firmen einschränken. Ein Objekt, das die Subrolle der beiden Rollen innehat, hätte somit als Einschränkung von *arbeitet_bei* die Vereinigung der beiden Firmen und genügt somit beiden Rollen, besitzt jedoch alle Rollen nur einmal. Dies stimmt mit dem Gebrauch von Rollen in der natürlichen Sprache überein. Hier würde man auch nicht sagen, dass jemand „Studenten“ geworden ist, weil er Student an zwei Einrichtungen ist.

sich nicht, solange das Objekt existiert. So kann ein Objekt vom Typ Person nicht aufhören, diesem Typ anzugehören, ohne selbst zu verschwinden. Im hier verfolgten Ansatz besitzt jedes Objekt genau einen natürlichen Typ, was im Allgemeinen keine Einschränkung darstellt, da sich durch Vererbungshierarchien stets ein gemeinsamer Subtyp mehrerer Konzepte bilden lässt.

Die Hierarchie der Rollen ist in der Regel orthogonal zur Hierarchie der natürlichen Typen. Somit ergeben sich zwei Dimensionen, nämlich einmal die Hierarchie der Rollen und einmal die Hierarchie der natürlichen Typen. Dies ist in Abbildung 7.8 exemplarisch dargestellt. Dynamische Konzepte wurden bereits in Abschnitt 2.4.2 eingeführt. Für die Rollenhierarchie wird zudem davon ausgegangen, dass es keine Mehrfachvererbung gibt, so dass mit dem Entfernen einer Rolle auch alle Subrollen verloren gehen. Dies ist im Allgemeinen der Intuition näher als die Verwendung von Mehfachvererbungsmechanismen.

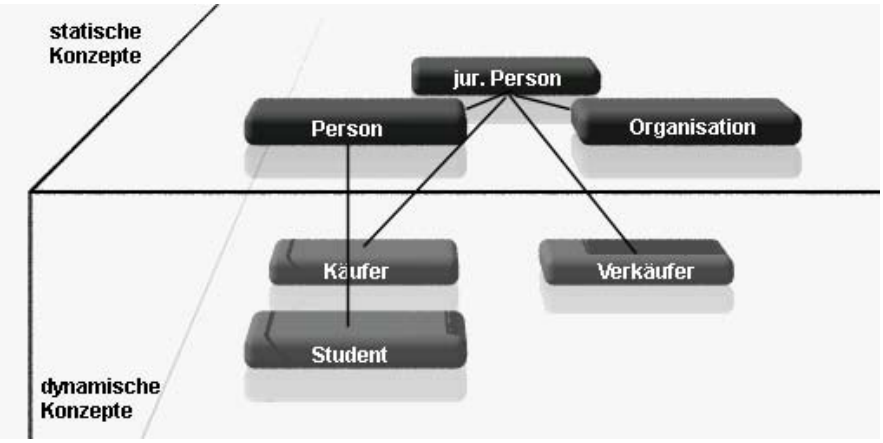


Abbildung 7.8: Rollen und natürliche Typen als orthogonale Konzepte

Zur Modellierung von Rollenänderungen in Prozessen sind zwei Herangehensweisen denkbar. Zum einen lässt sich eine Rolle explizit durch einen Befehl ändern oder sie wird implizit durch die Angabe weitere Attribute geändert. So kann eine Person zu einem Studenten werden, weil dies explizit deklariert wurde und er nun den Typ Student besitzt, oder es kann implizit durch die Vergabe eines Studentenausweises geschehen. Im letzteren Fall würde ein Schlussfolgerungsmechanismus folgern, dass das Besitzen eines Studentenausweises diese Person zu einem Studenten macht.

Im Rahmen dieser Arbeit wird die in Dienstbeschreibungsnetzen bereits eingeführte explizite Zuordnung von Rollen durch spezielle Anweisungen verwendet. Sie bietet zwar den Nachteil, dass jeweils explizit angegeben werden muss, wie sich Rollen ändern, bietet jedoch andererseits den Vorteil der größeren Flexibilität. Die implizite Rollenzuordnung könnte dazu führen, dass Rollen fälschlicherweise zugeordnet würden, weil ein Objekt gerade Attribute besitzt, die eine bestimmte Rolle

nahelegen, obwohl ein Objekt diese Rolle nicht besitzt. Auf diese Weise entstehen leichter schwer nachvollziehbare Modellierungsfehler. Zudem ließen sich viele Fehler erst zur Laufzeit beziehungsweise Simulationszeit erkennen, da eine statische Typisierung eingeschränkt würde. So ließe sich prinzipiell auch einem Unternehmen ein Studentenausweis zuordnen, wobei dann erst zur Simulationszeit festgestellt würde, dass dies dem Modell widerspräche.

Die Möglichkeiten, Rollen wie in Abbildung 7.8 dargestellt zu modellieren, werden derzeit noch sehr eingeschränkt durch Werkzeuge unterstützt. Die Notwendigkeit, zwischen dynamischen und statischen Konzepten zu unterscheiden entsteht erst, wenn man Abläufe betrachtet, die Konzepte und Rollen verändern können. Da Ontologien im Agenten- und Dienstkontext jedoch bislang in erster Linie dazu verwendet werden, die übertragenen Nachrichten zu interpretieren, ist eine Unterscheidung zwischen dynamischen und statischen Konzepten nicht notwendig, da ohnehin nur ein Ausschnitt der Welt dargestellt wird, ohne anzugeben, wie sich dieser ändert. Aus diesem Grund werden Rollen bislang von keinem gängigen Modellierungswerkzeug für Ontologien unterstützt.

Die in Unterabschnitt 6.4 diskutierte Möglichkeit, den Übergang zwischen Rollen ebenfalls zu modellieren, erlaubt es zu überprüfen, ob die Veränderungen der Rollen eines Objektes sinnvoll sind. So kann man beispielsweise anhand von Rollenmigrationsdiagrammen festlegen, welche Rollen in welche anderen Rollen übergehen dürfen. Auf diese Weise kann etwa sichergestellt werden, dass die an Rollen gebundenen Rechte nicht an eine falsche Person übergeben werden[5].

Detailebenen der Datensicht

Die Verwendung von dynamischen Konzepten erlaubt es, gerade auf den weniger detaillierten Ebenen der Organisation und der Abteilung zunächst abstrakt festzulegen, wie sich die Attribute oder der Typ eines Objektes ändern. So kann auf Ebene der Organisation beispielsweise angegeben werden, dass ein *Auftrag* in einer Abteilung *Vertrieb* zu einem Ereignis führt, das dann wiederum seine weitere Bearbeitung initiiert. Dabei wechselt der Auftrag vom Zustand unbearbeitet zum Zustand geprüft. Gleichzeitig wird der Kunde als Premiumkunde identifiziert, so dass der Auftrag die Rolle *Premiumauftrag* erhält.

Welche Attribute dem Auftrag zuzurechnen sind und welche einem neuen dynamischen Konzept zuzurechnen sind, ist dabei stark von der Anwendungsdomäne abhängig. Generell sollten Attribute, die ein Objekt eines Konzeptes stets zu einem Zeitpunkt seines Lebenszyklus inne hat, nicht als dynamisches Konzept modelliert werden.

Dynamische Konzepte erlauben bei der Modellierung von Geschäftsprozessen eine recht intuitive Darstellung der Abläufe in Organisationen und unterstützen eine

[5] So sollte es normalerweise nicht möglich sein, dass etwa ein Träger der Rolle *Praktikant* die Rolle *Geschäftsführer* annimmt.

Top-Down-Vorgehensweise. Aus diesem Grund sind dynamische Konzepte ein zentraler Bestandteil von Dienstbeschreibungsnetzen. Die Modellierung der Abläufe ist Gegenstand des nächsten Unterabschnitts.

7.3.3 Repräsentation der Prozesssicht durch Petrinetze

Neben der Organisationssicht und der im letzten Unterabschnitt diskutierten Datensicht fehlt nun noch die Darstellung der Kontrollflusssicht und der funktionalen Sicht durch Petrinetze, die gemeinsam als Prozesssicht bezeichnet werden. Zur Darstellung der funktionalen Sicht wird der Zustand des Gesamtsystems anhand des Zustands der einzelnen Fachobjekte repräsentiert. Die funktionale Sicht stellt dann mögliche Änderungen dieses Gesamtzustandes dar. Hierbei werden höhere Petrinetze zur Darstellung des Verhaltens von Diensten genutzt, wie dies in Köhler und Ortmann (2005) dargestellt wurde.

Ablaufmodellierung durch Petrinetze

Die Modellierung des Kontrollflusses geschieht anhand von Petrinetzen. Der Idee der offenen Workflownetze von Martens (2005) und Schmidt (2005) folgend kommunizieren die Dienstprotokolle verschiedener Dienste durch Verschmelzung von Randstellen. Dass sich daraus ergebende Netz ist dann ein Workflownetz, dass durch ein Ereignis von einem Dienst markiert und somit initiiert wird. Hierbei ist jedoch im Gegensatz zu Schmidt (2005) eine der verschmolzenen Stellen stets die Initial- oder die Finalstelle eines offenen Workflownetzes, wodurch eine Hierarchie der Netze entsteht. Durch Ersetzen der Subnetze durch eine Transition erhält man stets ein Workflownetz, das auf Korrektheit überprüft werden kann. Analoges ist auch für Dienstbeschreibungsnetze möglich, wie dies in Kapitel 6 im Rahmen der Analyse der entsprechenden Netze detaillierter erläutert wurde.

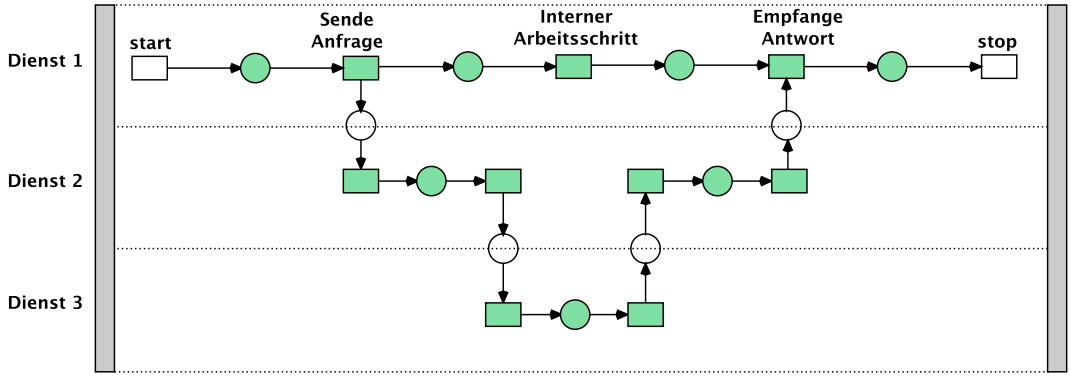


Abbildung 7.9: Schematische Darstellung der Dienstprotokolle

Jedes Dienstprotokoll hat somit einen Aufbau wie das in Abbildung 7.9 schematisch dargestellte Dienstprotokoll. Durch eine Nachricht wird ein solches Protokoll initialisiert und arbeitet dann eine Reihe von Anweisungen ab oder delegiert die Ausführung an ein Protokoll eines anderen Dienstes. Die Ausführung in verschiedenen Diensten wird durch gestrichelte Linien repräsentiert, so dass jeder Dienst seine eigene Swimlane besitzt. Diese Swimlanes können wiederum zu Pools zusammengefasst werden, die dann ganze Abteilungen repräsentieren. Die Verwendung von Swimlanes ist in der Modellierung häufig anzutreffen. Sie sind beispielsweise auch in der in Unterabschnitt 2.2.3 beschriebenen BPM-Notation und in Agenteninteraktionsdiagrammen von Odell u. a. (2000) vorzufinden.

Die Kanalstellen zwischen den Swimlanes repräsentieren den Nachrichtenkanal, der durch den Enterprise Service Bus bereitgestellt wird. Dies ermöglicht eine Modellierung der Kontrollstruktur bei gleichzeitiger Abstraktion von Details der Nachrichtenkommunikation. Durch Verwendung der Konzepte der Ontologie aus der Datensicht im Protokoll können zudem statische Schnittstellen angegeben werden, indem die Nachrichtenkanäle getypt werden.

Ebenfalls unter Verwendung der Datentypen der Ontologie kann die Funktionalität von Abläufen und Teilabläufen dargestellt werden. Hierzu wird dargestellt, wie sich die Ausführung eines Ablaufs oder einer Aktion auf den aktuellen Zustand des Systems auswirkt. Dieser Zustand wird durch die Werte der Fachobjekte beschrieben, die sich durch entsprechende Aktionen ändern. Je nach Detaillierung kann ein solcher Übergang sehr abstrakt formuliert werden, wie beispielsweise, dass eine Person zum Kunden wird, oder sie kann sehr konkret angeben, wie sich ein Attribut ändert, etwa indem die Änderung des Kontostandes angegeben wird.

Die Ereignisverarbeitung wird durch eine besondere Art von Diensten bereitgestellt. Dies kann ebenfalls durch Protokolle modelliert werden, die jedoch lediglich aus einem eintreffenden Ereignis eine ausgehende Nachricht an einen verarbeitenden Dienst erzeugen. Durch die Verwendung höherer Petrinetze kann die Bearbeitung von Regeln durch Aktivierungsbedingungen repräsentiert werden. Für große Systeme dürfte hierbei jedoch die Performanz deutlich schlechter sein als bei alternativen Herangehensweisen zur Regelverarbeitung wie etwa durch den RETE-Algorithmus von Forgy (1982). Aus diesem Grund ist für entsprechend große Systeme dann eine andere Art der Regelverarbeitung notwendig, wie sie etwa durch das Werkzeug Jess (2008) geboten werden.

Ebenen der Ablaufmodellierung

Die verschiedenen Ebenen bieten bei der Betrachtung des Prozessmodells die Möglichkeit, das System sukzessive zu verfeinern. Hierbei können unter Verwendung der Konzepte der Ontologie zunächst statische und dynamische Schnittstellen abstrakt beschrieben werden und die Kontrollstruktur allgemein definiert werden. Auf Ebene der Organisation kann hierzu angegeben werden, welche Abteilungen welche

Aufgaben zu erledigen haben. Auf Abteilungsebene werden diese Aufgaben dann konkreten Diensten zugewiesen, die auf der Ebene der Dienstprotokolle implementiert werden. Ähnlich den Rollen von Agenten lassen sich für Dienste durch ihre Diensttypen abstrakte Schnittstellenbeschreibungen angeben, die dann jeweils im Modell der entsprechenden Ebene verwendet werden.

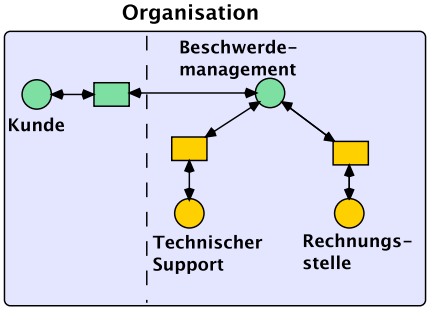


Abbildung 7.10: Beispielhafte Darstellung eines Organisationsausschnitts

Abbildung 7.10 stellt beispielhaft einen Ausschnitt einer Organisation dar. In dieser Organisation gibt es eine Abteilung für das Beschwerdemanagement, eine Abteilung für den technischen Support und eine Abteilung, die für die Bearbeitung von Rechnungen verantwortlich ist. Bei Eingang einer Beschwerde soll zunächst festgestellt werden, ob es sich um ein technisches Problem, ein Problem mit einer Rechnung oder um ein allgemeines Problem handelt. Je nachdem werden spezielle Dienste der verschiedenen Abteilungen aufgerufen.

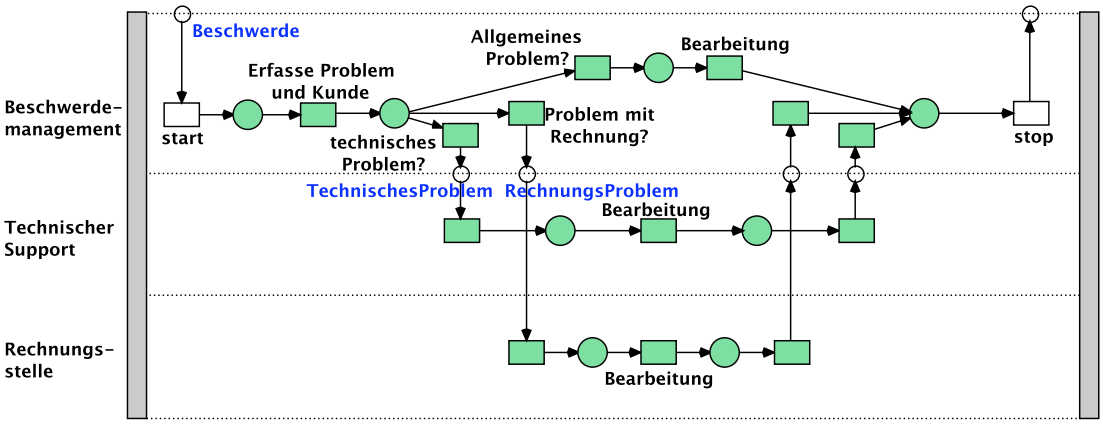


Abbildung 7.11: Kontrollflusssicht auf Organisationsebene

Dieses Verhalten wird in Abbildung 7.11 modelliert. Das dargestellte Netz mit

den entsprechenden Swimlanes modelliert den Kontrollfluss auf der Ebene der Organisation. Die Eingangsstellen der Dienste sind durch Konzepte der Ontologie getypt, die durch blauen Text beschriftet sind. Auf dieser Ebene wird somit lediglich das Verhalten der einzelnen Abteilungen zueinander spezifiziert. Dennoch kann auch auf dieser Ebene bereits wiederholt auftretende Funktionalität identifiziert werden, die dann durch Dienste gekapselt wird.

Auf der Ebene der Abteilung wird jeder der in einer Swimlane dargestellten Kontrollflüsse dann weiter verfeinert. Hierzu werden Farben in den Netzen eingesetzt, die angeben wie der Kontrollfluss verzweigt und welche Daten den Kontrollfluss kontrollieren. Zudem kann dargestellt werden, wie sich das Dienstverhalten auf das Gesamtsystem auswirkt. Eine detaillierte Diskussion des Vorgehens ist Gegenstand des nächsten Abschnitts, wo zunächst die verwendeten Modellbestandteile und die Art der verwendeten Netze näher erläutert werden. Hieran schließt sich ein Beispiel an, dass die Sichten auf den unterschiedlichen Ebenen wiedergibt.

Zwischen der Abteilung und dem Dienst lassen sich prinzipiell weitere Organisationselemente wie Position und Rolle einfügen, wie dies etwa bei Köhler-Bußmeier (2009a) geschieht. Dies bietet bei großen Modellen eine sinnvolle Strukturierung durch das Einziehen zusätzlicher Abstraktionsschichten. Da sich diese Elemente jedoch lediglich in ihrem Umfang von einer Abteilung unterscheiden, werden sie hier unter der Ebene der Abteilung subsumiert.

Jede der Ebenen gibt über zusätzliche Details des Gesamtsystems Auskunft. Analog zu den Rollen in Multiagentensystemen werden auch bei Diensten die Schnittstellen über ihre Nachrichten und ihr Verhalten definiert. Dies führt zu der oben dargestellten allgemeinen Darstellung des Verhaltens auf Ebene der Organisation. Die Schnittstellen der Abteilungen werden dann bereits weiter konkretisiert, wobei auf jeder Ebene die Kommunikationspartner – die Abteilungen beziehungsweise ihre Dienste – durch ihre Schnittstellen repräsentiert werden.

7.4 Umsetzung der Architektur mit Dienstbeschreibungsnetzen

Die in den vorausgegangenen beiden Abschnitten eingeführte Dienstarchitektur lässt bereits deutlich erkennen, dass eine Vielzahl von Modellbestandteilen durch Dienstbeschreibungsnetze umgesetzt werden können. So lassen sich Ontologien direkt auf die Konzepte von Dienstbeschreibungsnetzen abbilden. Zudem besteht die Möglichkeit dynamische Konzepte direkt im Netz zu modellieren. Ebenso lassen sich Schnittstellen und Interaktionsmuster darstellen. Die folgenden Unterabschnitte diskutieren die Umsetzung detaillierter.

7.4.1 Verwendung von Dienstbeschreibungsnetzen am Beispiel

Wie bereits am Anfang von Kapitel 4 zur Motivation von Dienstbeschreibungsnetzen vorgestellt wurde, lassen sich komplexere Szenarien mit Dienstbeschreibungsnetzen aus einfacheren Szenarien durch Verfeinerung entwickeln. So lassen sich die in Unterabschnitt 7.2.1 eingeführten interorganisationalen Workflow zunächst als einfache Netze modellieren, die dann sukzessive um Datentypen und Verhalten ergänzt werden, bis sie zu Dienstbeschreibungsnetzen werden.

Neben der Verhaltensbeschreibung durch Dienstbeschreibungsnetze umfasst das Modell aus Abschnitt 7.3 die Modellierung der verschiedenen Ebenen, die dann wiederum durch Swimlanes auch in das Modell des Prozesses als Dienstbeschreibungsnetz einfließen. Gleichzeitig werden zur Annotation der Daten die Konzepte und ihre Attribute benötigt, die sich als Ontologie beschreiben lassen. Aus diesem Grund werden in folgendem Beispiel die Ebenen der Modellierung beschrieben, wobei zu jeder Ebene auch die beteiligten Daten und die Prozesse angegeben werden.

Als Beispiel dient der Auftragseingang in einem Unternehmen, was im Folgenden beschrieben werden soll. Das Beispiel orientiert sich dabei zum Teil an Beispielen von Allweyer (2005). Zur Beschreibung des Beispiels werden die einzelnen Ebenen der Architektur aus Abschnitt 7.3 nacheinander beschrieben, wobei jeweils die Ressourcen- oder Organisationssicht, die Datensicht durch die Angabe einer Ontologie und die Prozesssicht wiedergegeben werden. Es sei dabei darauf hingewiesen, dass die Ebene des Organisationsnetzwerks als oberste Betrachtungsebene nicht mit der Ressourcen- oder Organisationssicht zu verwechseln ist, die die Anordnung der Ressourcen und Dienste jeder Ebene darstellt. Abgeschlossen wird dieser Unterabschnitt durch eine Diskussion der Dienstprotokolle, die sich aus den verschiedenen Sichten der darüber liegenden Ebenen ergeben. Als abkürzende Schreibweise wird dabei auf die explizite Angabe von *update*- und *assign*-Anweisungen verzichtet, da diese aus dem Kontext hervorgehen. Stattdessen wird ein Attribut lediglich durch ein Gleichheitszeichen gesetzt.

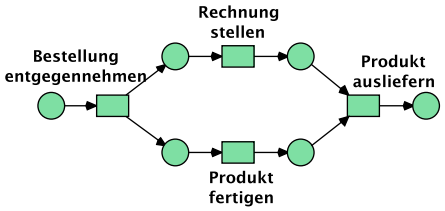


Abbildung 7.12: Vereinfachte Darstellung des Beispielablaufs

Abbildung 7.12 stellt den generellen Prozess stark vereinfacht dar. Dieser wird in den folgenden Unterabschnitten detaillierter betrachtet, wobei sich stets auf einen Aspekt der darüber liegenden Ebene konzentriert wird.

Die Ebene des Organisationsnetzwerks

Die Ebene des Organisationsnetzwerks stellt die verschiedenen an einem Prozess oder an einer Gruppe von Prozessen beteiligten Organisationen sowie die Prozesse zwischen diesen Organisationen dar. Auf dieser Ebene ist in der Regel noch keine detaillierte Modellierung der Daten sinnvoll. Vielmehr werden lediglich die Typen der Schnittstellen angegeben, die dann später weiter konkretisiert werden.

Die Ressourcen- oder Organisationssicht Auf der Ebene des Organisationsnetzwerks sind die am Prozess beteiligten Organisationen ein Unternehmen U und ein Kunde K. Die beteiligten Abteilungen im Unternehmen sind der Einkauf, die Buchhaltung und die Logistik beim Kunden K sowie der Verkauf, die Buchhaltung, die Fertigung und die Logistik beim Unternehmen U. Diese Abteilungen sind in Abbildung 7.13 dargestellt, wobei die Abteilungen des Kunden zu einer abstrakten Abteilung Kunde zusammengefasst worden sind. Dies reduziert die Schnittstellen nach außen auf die relevanten Änderungen.

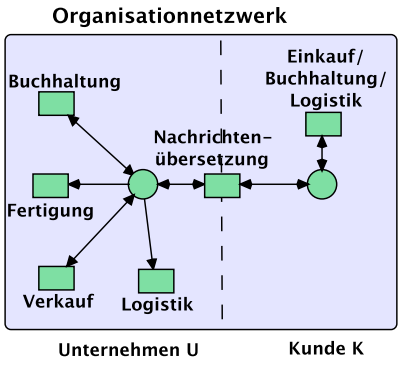


Abbildung 7.13: Das Organisationsnetzwerk des Beispiels

Das Datenmodell Die Daten können auf dieser Abstraktionsebene recht abstrakt durch die Angabe von Typen beschrieben werden. Welche Attribute diese Typen später besitzen kann dann in einer weiteren Verfeinerung entschieden werden. Hierbei kann es auch notwendig sein, das Typsystem noch einmal anzupassen. Im Beispiel sind die Typen

- *PurchaseOrder* zur Beschreibung der Bestellaufträge,
- *ManufacturingOrder* zur Beschreibung von Fertigungsaufträgen,
- *Order* zur Beschreibung von Aufträgen allgemein,

- *Invoice* zur Beschreibung einer Rechnung,
- *Product* zur Beschreibung eines Produktes,
- *Parcel* zur Beschreibung eines Pakets und
- *Receipt* zur Beschreibung einer Quittung

notwendig, wobei Typen hier stets englische Namen erhalten.

Das Prozessmodell Das Prozessmodell stellt auf dieser Ebene die Interaktion zwischen den verschiedenen Abteilungen der Organisationen dar. Die verschiedenen Abteilungen werden durch Swimlanes und die beteiligten Organisationen durch Pools dargestellt. Für eine detailliertere Einführung in die Verwendung von Swimlanes und Pools sei auf Allweyer (2005) oder auf die Spezifikation der BPMN (2008) verwiesen.

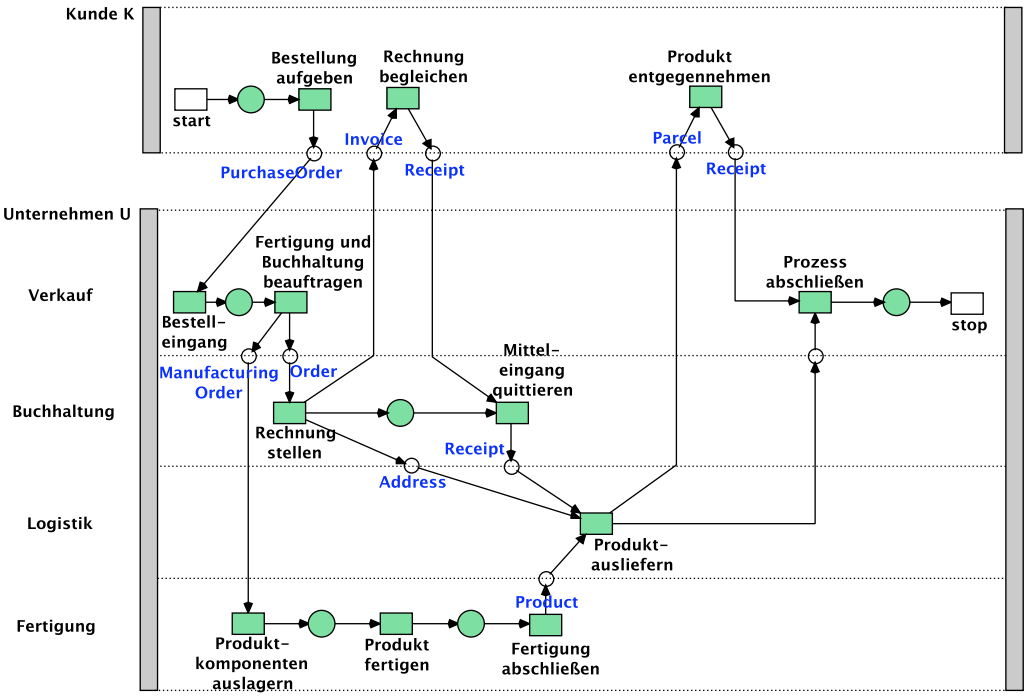


Abbildung 7.14: Der übergeordnete Prozess des Beispiels

Abbildung 7.14 stellt den Prozess auf oberster Ebene dar. Die Typen des Datenmodells an den Schnittstellen sind dabei in blauer Schrift dargestellt. Der Kunde K initiiert in seinem Einkauf die Bestellung der Ware durch Versand eines Auftrags (*PurchaseOrder*). Dieser geht beim Verkauf von Unternehmen B ein und wird dort

weiterversandt an die Produktion (als *MarnufacturingOrder*) und an die Buchhaltung (als *Order*). Nach Abschluss der Rechnungslegung und nach Abschluss der Produktion werden Quittung (*Receipt*) und Produkt (*Product*) an die Logistik überantwortet, wo sie dann als Pakete (*Parcel*) verschickt werden. Nach Quittierung des Empfangs (durch einen *Receipt*) durch den Kunden wird der Prozess beendet.

Es lässt sich erkennen, dass der Gesamtprozess durch den Kunden initiiert werden muss und dann vom Verkauf des Unternehmens weiterverarbeitet wird. Der Eingang eines Auftrags stellt somit ein Ereignis dar, auf das reagiert wird, indem der weitere Prozess bearbeitet wird.

Auf dieser Modellierungsebene ist das dargestellte Netz lediglich ein Workflownetz, das mittels der herkömmlichen Techniken zur Überprüfung von Workflownetzen überprüft werden kann. Auf der nächsten Ebene wird dieses Netz verfeinert und als Dienstbeschreibungsnetz dargestellt.

Die Ebene der Abteilung

Auf der Ebene der Abteilung soll die Logistik exemplarisch weiter verfeinert werden. Diese besteht auf der Ebene des Organisationsnetzwerks aus der Transition *Produkt ausliefern*. Da diese Transition zwei Eingangsstellen und zwei Ausgangsstellen aufweist, stellt das Subnetz kein Workflownetz dar. Dies soll im praktischen Rahmen jedoch zugelassen werden, wenn es durch Hinzufügen zweier Stellen und zweier Transitionen im Vor- und Nachbereich der Randstellen möglich ist, aus dem Netz ein Workflownetz zu erstellen.

Die Ressourcen- oder Organisationssicht In der Organisationssicht sind nun die Dienste erkennbar, die von der Logistik-Abteilung bereitgestellt werden und die in diesem Kontext relevant sind. Diese Dienste umfassen den Dienst zur Vorbereitung des Auslieferung, den Dienst zur Produktverpackung, den Dienst zum Drucken des zum Paket hinzugefügten Dokumentes und den Dienst zum Versenden des Pakets. Diese Dienste werden später im Prozessmodell genutzt und sind in Abbildung 7.15 dargestellt.

Das Datenmodell Das Datenmodell muss an dieser Stelle weiter verfeinert werden. Dies geschieht anhand einer Ontologie, deren Konzepte und Attribute in einer einem Klassendiagramm ähnlichen Struktur dargestellt werden. Zunächst werden die Konzepte der eingehenden Objekte *Product* und *Receipt* sowie des ausgehenden Konzepts *Parcel* benötigt. Des Weiteren werden interne Konzepte benötigt, die die verschiedenen Zwischenzustände des Systems abbilden. Dies sind *CheckedParcel*, um anzugeben, dass der Inhalt eines Pakets überprüft wurde, und *ScheduledParcel*, um anzugeben, dass ein Paket auf seine Auslieferung wartet, sowie *DeliveredParcel*, um anzugeben, dass ein Paket ausgeliefert wurde. Dies ist in Abbildung 7.16 dargestellt. Die Abbildung ist dabei auf die wesentlichen Konzepte reduziert und ist an

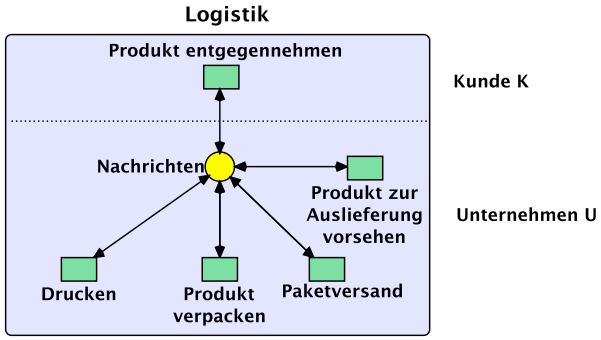


Abbildung 7.15: Die im Beispiel relevanten Dienste der Logistik

UML-Klassendiagramme angelehnt. Die Vererbung ist hier jedoch durch einfache Pfeile dargestellt.

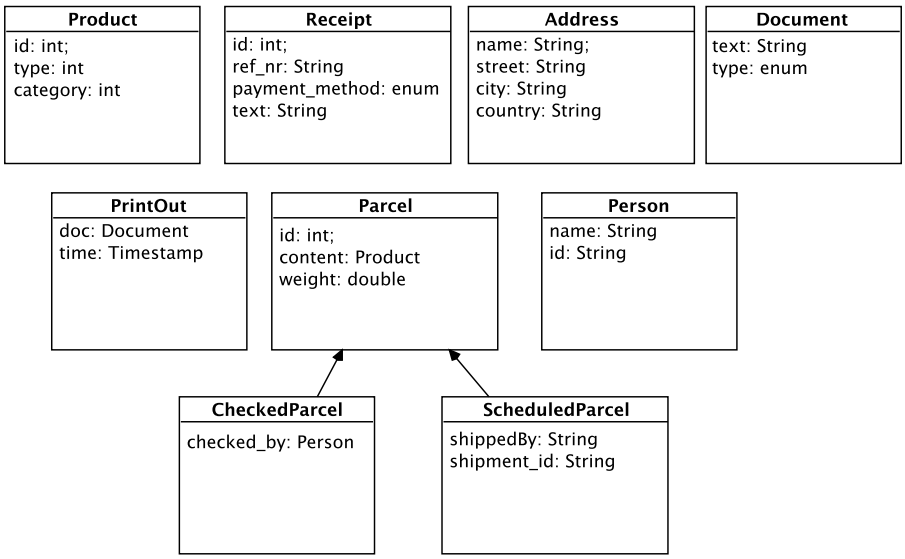


Abbildung 7.16: Ein Ausschnitt des Datenmodells der Logistik im Beispiel

Als Konzepte existieren das Produkt (*Product*), das versandt werden soll, der Beleg (*Receipt*) vom Kunden, der zusammen mit der Adresse (*Address*) als Dokument (*Document*) ausgedruckt und dem Paket (*Parcel*) als Ausdruck (*PrintOut*) hinzugefügt wird. Das Paket wird dann gepackt und überprüft (*CheckedParcel*) durch eine Person (*Person*). Abschließend wird es zur Versendung vorgesehen (*ScheduledParcel*).

Das Prozessmodell Das Prozessmodell stellt den Prozess der Logistik unter Einbeziehung der Dienste dar. Die Dienste sind wiederum den Swimlanes zugeordnet. In diesem Fall wird jeder Dienst nur einmal aufgerufen. Der entsprechende Ablauf ist in Abbildung 7.17 dargestellt.

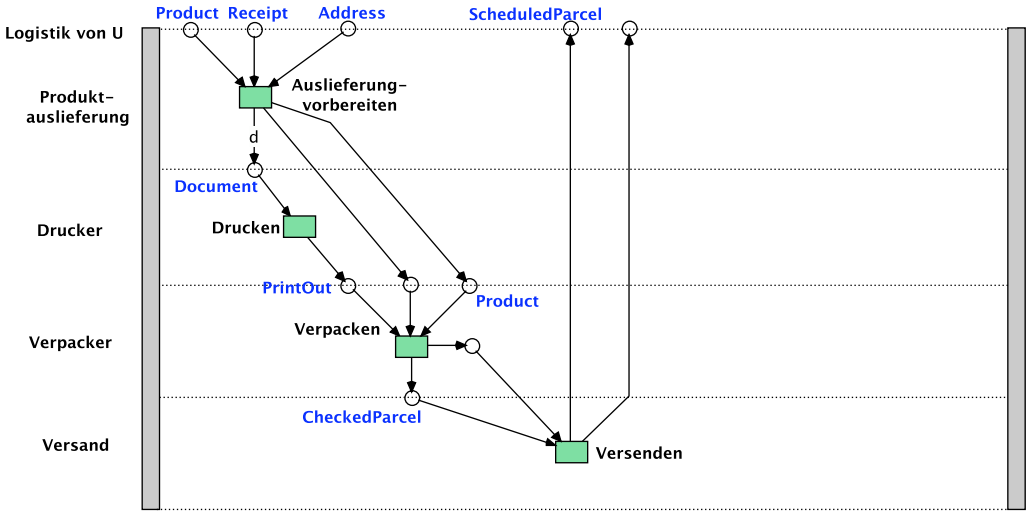


Abbildung 7.17: Der Logistik-Teilprozess des Beispiels

Im dargestellten Vorgang wird zunächst ein allgemeiner Dienst der Produktauslieferung aufgerufen, der aus den übergebenen Informationen ein Dokument erzeugt, das später ausgedruckt und dem Paket mitgegeben werden muss. Der Verpacker nimmt dieses Dokument aus dem Drucker, überprüft den Inhalt des Pakets und reicht dieses dann zur Versendung weiter. Abschließend wird das Paket versendet. Auf dieser Ebene der Beschreibung werden wiederum lediglich Datentypen spezifiziert, es wird jedoch noch nicht das Verhalten der Dienste festgelegt. Dies erfolgt auf der nächsten Ebene.

Die Ebene der Dienste

Die Ebene der Dienste konkretisiert die Ausführung weiter, indem logische Abläufe in ihre tatsächliche Struktur überführt werden. Das Prozessmodell der darüberliegenden Ebene zeigt ein sequentielles Modell, bei dem die verschiedenen Dienste nacheinander ausgeführt werden. In einem realen Szenario wird dies meist durch einen Dienst gesteuert. In diesem Fall ist dies der Dienst der Produktauslieferung. Dieser Dienst ist somit ein prozesszentrierter Dienst (vgl. Abschnitt 2.3.2), der die anderen Dienste koordiniert.

Neben der Darstellung der tatsächlichen Realisierung der Dienste werden auf dieser Ebene auch Ressourcen mit einbezogen. So ist zum Drucken ein Drucker

notwendig und es ist zum Verpacken ein Verpacker notwendig. Beide Ressourcen müssen dem Dienst zur Erfüllung seiner Aufgabe zur Verfügung stehen. Dies wird einmal in der Ressourcensicht dargestellt und einmal in der Prozesssicht, die den Dienst spezifiziert.

Betrachtet man den Produktauslieferungsdienst, so überprüft dieser zunächst, ob die Ressource Drucker und die Ressource Packer verfügbar sind, da andernfalls kein Paket versandt werden kann. Ist dies nicht der Fall, so ruht der Prozess, bis die Workflow-Engine durch ein entsprechendes Ereignis dazu aufgefordert wird, erneut zu überprüfen, ob die Ressourcen verfügbar sind. Im Folgenden sollen sowohl der Auslieferungsdienst als auch der Verpackungsdienst näher betrachtet werden. Während ersterer ein prozessorientierter Dienst ist, der weitere Dienste aufruft, ist letzterer lediglich ein einfacher Basisdienst.

Die Ressourcen- und Organisationssicht Die Organisationssicht des Auslieferungsdienstes ist relativ einfach, da es nur einen Dienst gibt, der die Anfrage entgegen nimmt und dann die anderen Dienste aufruft. Aus diesem Grund wird hier lediglich die Organisationssicht des Verpackungsdienstes dargestellt. Diese ist in Abbildung 7.18 wiedergegeben.

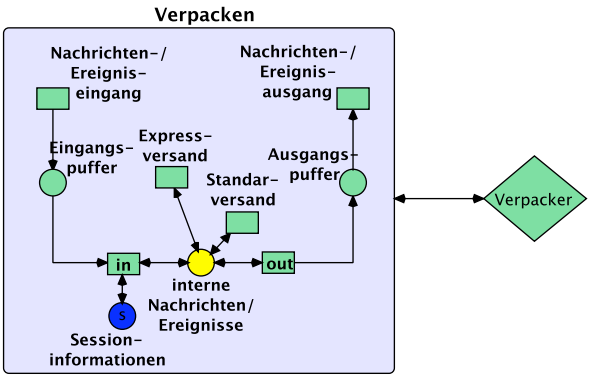


Abbildung 7.18: Der Verpackendienst des Beispiels

Der Verpackdienst besitzt zwei Dienstprotokolle, je nachdem ob eine Expresslieferung oder eine Standardlieferung gewünscht wird. Aufgrund der Eingangsdaten kann der Dienst dann die Nachricht an das entsprechende Protokoll weiterreichen. Zudem benötigt der Dienst eine Ressource vom Typ *Verpacker*. Jede Ressource besitzt ein Attribut *available*, das anzeigt, ob diese Ressource verfügbar ist. Damit der Prozess des Versendens nur startet, wenn alle Ressourcen verfügbar sind, fragt der Auslieferungsdienst zunächst ab, ob alle Ressourcen verfügbar sind. Diese können dann genutzt werden. Wird ein Dienst mehrfach aufgerufen, so besitzt er die Möglichkeit, Kontextinformationen in einer Session abzulegen, wovon hier jedoch

kein Gebrauch gemacht wird. Denkbar wäre es jedoch, mehrere Produkte einer Bestellung zu sammeln und gemeinsam zu verschicken, so dass der Kontext die zu erwartenden Produkte vorhält.

Die Prozesssicht Da sich die Datensicht nicht von der Datensicht der Abteilung unterscheidet, soll hier direkt auf die Prozesse eingegangen werden. Auf dieser Ebene werden alle Änderungen beschrieben, die ein Dienst vornimmt und welche Ressourcen er hierzu benötigt. Die konkrete Implementierung erfolgt dann durch die Dienstprotokolle auf der nächsten Ebene.

Der Dienst der Produktauslieferung soll hier näher betrachtet werden, da er der zentrale Dienst ist, der die anderen Dienste koordiniert. Er stellt zunächst das Versanddokument bereit, welches durch den Drucker ausgedruckt wird. Dieses Dokument und das Produkt wird an die Verpackung weitergeleitet, wie dies in Abbildung 7.19 dargestellt ist. Anstatt mehrere Stellen zu verwenden, wurden bei der Darstellung mehrere Variablen durch eckige Klammern zusammengefasst. Abschließend wird das überprüfte Paket zum Versenden weitergeleitet. Subdienste, die aufgerufen werden, sind in der Darstellung durch Transitionen mit Dreiecken dargestellt. Dies sind die in Unterabschnitt 6.4.3 eingeführten Platzhaltertransitionen, die dann durch weitere Dienstbeschreibungsnetze verfeinert werden können, wobei diese Dienste den Dienstaufrufen entsprechen.

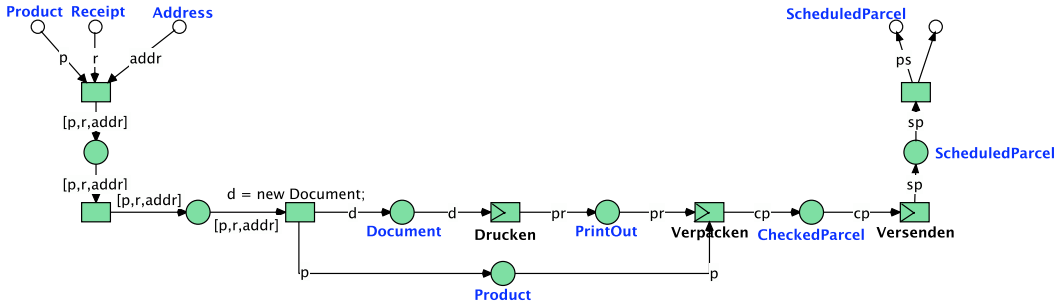


Abbildung 7.19: Das Prozessmodell des Produktauslieferungsdienstes des Beispiels

Das Netz in Abbildung 7.19 stellt ein Dienstbeschreibungsnetz dar, in dem jedoch noch keine Berechnungen durchgeführt werden und in dem die Ausgangsvariablen der Platzhaltertransitionen noch nicht gebunden sind. Dies erfolgt dann durch die Bindung der Platzhaltertransitionen an einen Dienst, was einer Ersetzung der Transition durch das entsprechende Dienstbeschreibungsnetz des Dienstes entspricht.

Wie mögliche Dienstbeschreibungsnetze, die die Platzhaltertransitionen ersetzen, aussehen, ist in Abbildung 7.20 wiedergegeben. Auf dieser Ebene werden ebenfalls noch keine konkreten Veränderungen der Attribute vorgenommen. Vielmehr wird

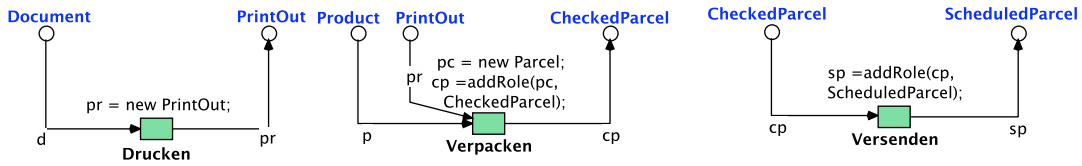


Abbildung 7.20: Das Prozessmodell der Basisdienste des Beispiels

hier dargestellt, wie der Fluss der Daten ist und wo neue Konzepte erzeugt werden oder weitere Attribute hinzugefügt werden. Der Dienst zum Drucken erzeugt lediglich ein Objekt vom Typ *PrintOut*. der Dienst zum Verpacken erwartet ein Objekt vom Typ *PrintOut* und ein Objekt vom Typ *Product* und erzeugt ein Paket (*Parcel*), das dann das dynamische Konzept *CheckedParcel* annimmt, wenn es überprüft wurde. Das Versenden eines solchen Pakets erfolgt dann durch den Dienst Versenden, der dem Paket das zusätzliche dynamische Konzept *ScheduledParcel* zukommen lässt.

Auf dieser Ebene werden lediglich die Hauptabläufe modelliert. Eine detaillierte Fehlerbehandlung ist hingegen Gegenstand der nächsten Ebene.

Die Ebene der Dienstprotokolle

Dienstprotokolle verfeinern wiederum die Ebene der Dienste und stellen die eigentliche Implementation des Verhaltens dar. Sie rufen andere Dienste auf und repräsentieren so das in der Prozesssicht der Dienste modellierte Verhalten. Vielfach wird dies genau durch die in der Prozesssicht für einen Dienst modellierte Funktionalität erreicht, die um Implementationsdetails ergänzt werden muss. Während in den anderen Ebenen eine Unterscheidung der Sichten zum besseren Verständnis sinnvoll ist, fallen die verschiedenen Sichten für Dienstprotokolle zusammen, da sie genau die Implementierung repräsentieren.

Eine Möglichkeit des Druckdienstes ist in Abbildung 7.21 dargestellt. Hierbei ist jedoch lediglich die Veränderung der Daten, also des Zustands dargestellt. Dienste wie das Drucken eines Dokuments oder das Verpacken eines Produkts benötigen jedoch realweltliche Ressourcen wie einen Drucker oder einen Verpacker, die angesprochen werden müssen. Diese Ressourcen wurden oben bereits auf Ebene der Dienste dargestellt. Sie können nun im Dienstprotokoll als spezielle Konstanten modelliert werden.

Im Fall des Druckens gibt es somit eine Konstante *printer* im Modell, auf der ein Operator *print* aufgerufen werden kann. Dieser Operator könnte beispielsweise entweder eine Fehlermeldung oder einen Erfolgsstatus zurückgeben, was abhängig von der Implementation ist. Eine entsprechende Erweiterung des Druckdienstes ist in Abbildung 7.22 dargestellt. Dabei wurde die Anwendung des Operators

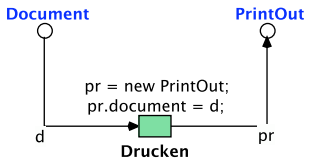


Abbildung 7.21: Das Dienstprotokoll des Druckdienstes ohne Ressourcen

print wie für objektorientierte Systeme üblich durch *printer.print*() notiert, anstatt *print*(*printer*) zu schreiben. In diesem Fall wird zudem davon ausgegangen, dass ein PrintOut ein weiteres Attribut *status* besitzt.

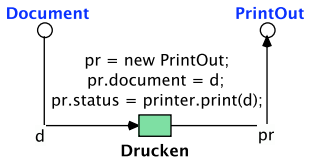


Abbildung 7.22: Das Dienstprotokoll des Druckdienstes

Vernachlässigt wurde in Abbildung 7.22 die Möglichkeit der Ausnahmebehandlung. So wäre es innerhalb des Protokolls denkbar auf einen anderen Drucker zu wechseln, wenn der erste Drucker beispielsweise defekt ist oder kein Papier mehr besitzt. Auch wäre es denkbar, in diesem Fall eine weitere Ressource (etwa über einen Wartungsdienst) zu informieren, die dieses Problem löst. Um das Beispiel jedoch nicht übermäßig komplex werden zu lassen, wurde auf diese Möglichkeiten hier verzichtet.

Auch für die anderen Dienstprotokolle sind Ressourcen unter Umständen wichtig, ohne dass sie direkt verwendet werden. So muss das Dienstprotokoll des Produktauslieferungsdienst sicherstellen, dass alle Ressourcen zur Verfügung stehen, und es muss die notwendigen Ressourcen reservieren. In diesem Fall wird davon ausgegangen, dass nur ein Verpacker und ein Drucker existieren, so dass lediglich überprüft werden muss, ob eine Ressource verfügbar ist. Dies geschieht durch das Attribut *available*, das alle Ressourcen besitzen, was in Abbildung 7.23 für den prozessorientierten Produktauslieferungsdienst dargestellt ist.

Der Dienst überprüft zunächst, ob alle Ressourcen zur Verfügung stehen, und stellt dann das Versanddokument bereit, welches durch den Drucker ausgedruckt wird. Dieses Dokument und das Produkt wird an die Verpackung weitergeleitet. Der Produktauslieferungsdienst aus Abbildung 7.23 nimmt einen Auftrag an und ist dann solange blockiert, bis alle benötigten Ressourcen zur Verfügung stehen. Dies sind in diesem Fall ein Drucker und ein Packer. Ist dies der Fall, so werden die

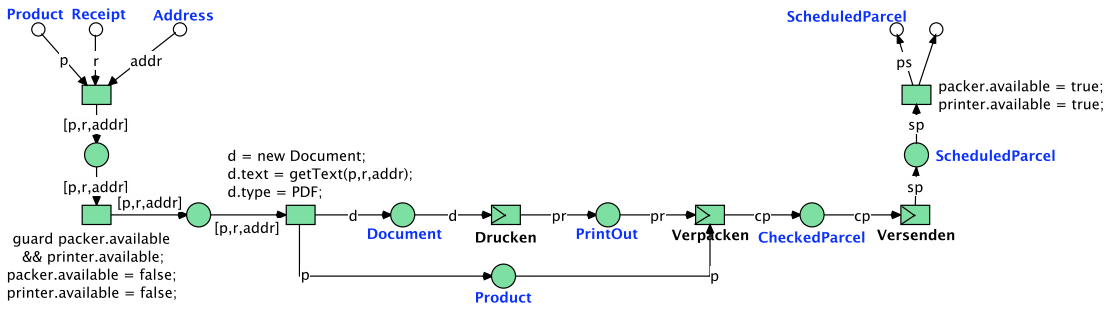


Abbildung 7.23: Das Dienstprotokoll des Produktauslieferungsdienstes

entsprechenden Daten an die Folgedienste weitergereicht.

Der Aufruf von Operatoren auf Ressourcen bietet die Möglichkeit, diese Operatoren entsprechend zu implementieren, so dass ihr Aufruf eine realweltliche Konsequenz (etwa das Drucken eines Dokuments) besitzt. Gleichzeitig kann es eine zweite Implementation geben, die die Simulation oder die Analyse erlaubt. Dieses Vorgehen wird im nächsten Unterabschnitt als Grounding bezeichnet, das es erlaubt, durch Dienstbeschreibungsnetze modellierte Dienste sowohl direkt aufzurufen als auch zu analysieren.

7.4.2 Ausführbare Modelle mit Dienstbeschreibungsnetzen

Wie am Beispiel im letzten Unterabschnitt sichtbar wurde, bieten Dienstbeschreibungsnetze eine gute Möglichkeit, die in Organisationen vorhandenen Prozesse abzubilden. Der zentrale Zustand von Dienstbeschreibungsnetzen reflektiert die in vielen Organisationen vorherrschende Vorstellung eines eindeutigen logischen Zustands der Daten, der dann durch mehrere physikalische Datenträger repräsentiert wird. Dass dabei die Datenkonsistenz vielfach nicht gewährleistet ist, stellt in erster Linie ein Problem der praktischen Realisierung und weniger des Modells dar.

Dieser Unterabschnitt diskutiert die im letzten Unterabschnitt beispielhaft eingeführte Verwendung von Dienstbeschreibungsnetzen und das im Zusammenhang damit dargestellte Vorgehen. Dabei wird auf das Beispiel Bezug genommen, um die vorgestellten Konzepte zu erläutern. Wie in Abschnitt 7.2.2 dargestellt, erfolgt die Modellierung dienstbasierter Systeme in der Regel durch ein Top-Down-Vorgehen, das um ein Bottom-Up-Vorgehen zur Einbettung bestehender Systeme ergänzt wird. Dienstbeschreibungsnetze bieten hier die Möglichkeit, bestehende Systeme durch Teilnetze oder durch einzelne Transitionen zu kapseln, deren Verhalten dann durch den Aufruf eines Dienstes implementiert wird, so dass die Ausführung an das Teilsystem delegiert wird.

Ausführung und Modellierung

Da der Datenfluss in Dienstbeschreibungsnetzen ebenfalls modelliert ist, lassen sich die Modelle der Dienstbeschreibungsnetze durch eine Implementierung der Transitionen oder der Teilnetze in ausführbare Systeme überführen. Die Berechnung der durch Operatorsymbole gegebenen Funktionen in den Netzen kann durch eine reale Berechnung implementiert werden, wobei die Algebra dann genau die reale Anwendung einer Funktion ist. Dabei können Funktionen sowohl als Dienste implementiert werden als auch statisch im System vorhanden sein, da sie zur statischen Algebra gehören, wie beispielsweise die Summenfunktion. Die Veränderungen am Zustand kann im realen System durch die Veränderung der Datenspeicher erfolgen. In der Regel werden Daten in Datenbanken oder Dokumentenmanagementsystemen gespeichert, deren Veränderung durch einen Aufruf entsprechender Dienste möglich ist.

Nutzt man Ressourcen als Konstanten, wie dies in Abbildung 7.23 zu sehen ist, so lassen sich auch Bedingungen an Ressourcen oder die Veränderung von Ressourcen beschreiben. Hierbei ist es wichtig, dass eine Veränderung einer Ressource sich im Modell widerspiegelt, sofern der Zustand der Ressource für den Ablauf wichtig ist. Andernfalls kann es zu Inkonsistenten zwischen dem Modell und der realen Entsprechung kommen.

Auch wenn die Verwaltung von Ressourcen nicht Gegenstand dieser Arbeit ist, lässt sich auf diese Weise zwischen der *Beschreibungsebene* und der *Ausführungsebene* eines Systems unterscheiden. Erstere gibt das Modell des Systems wieder, während letztere die konkrete Ausführung dieses Modells erlaubt. Idealerweise lässt sich für die Ausführungsebene kontrollieren, ob sie mit dem Modell konsistent ist. Dies erlauben Dienstbeschreibungsnetze, indem die Implementation einer Ressource ausgetauscht wird. Hierbei kann entweder eine symbolische Analyse sinnvoll sein, oder es werden die Ressourcen bei der Simulation schlichtweg ignoriert. Wenn Beschreibung und Implementation übereinstimmen gilt hiernach bei erfolgreicher Ausführung des Dienstes, dass die Veränderungen des Systemzustands den Veränderungen in der Realität entsprechen. Der Aufruf des Dienstes geschieht hierbei über ein *Grounding*, wie es etwa auch bei der Beschreibungssprache OWL-S (2006) existiert. Dieses Grounding bindet Ressourcen und Dienstaufrufe, wie sie ebenfalls in Abbildung 7.23 zu sehen sind, an konkrete Dienste und Funktionsimplementationen. Die Übersetzung zwischen Konzepten und Nachrichtenformat kann hierbei vom Service-Container oder ESB oder durch den Dienst selbst geschehen.

Prinzipiell sieht der Aufruf eines komplexen Dienstes wie in Abbildung 7.24 beschrieben aus. Hierbei steuert der zusammengesetzte Dienst auf der linken Seite den Aufruf von Subdiensten die ihrerseits wiederum auf Datenbanken oder andere Systeme im Backend zugreifen, um die Zustandsänderung vorzunehmen. Dabei entsprechen Update-Anweisungen der Dienstbeschreibungsnetze UPDATE-Befehle auf der Datenbank. Die Konzeptänderungsanweisungen werden durch INSERT- bzw.

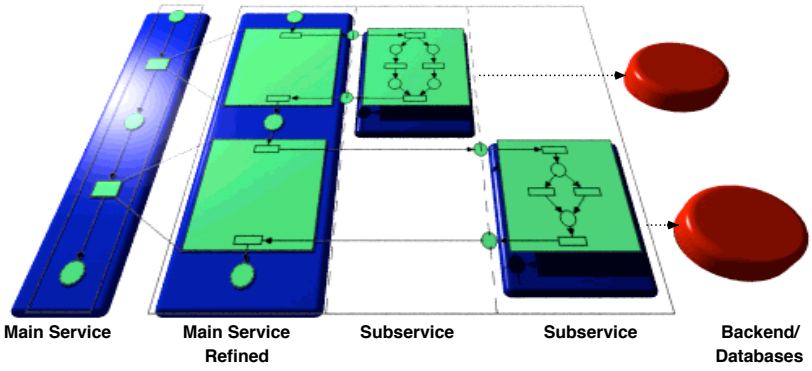


Abbildung 7.24: Aufruf eines Dienstes

DELETE-Befehle auf der Datenbank umgesetzt.

Die aufgerufenen Dienste sind dabei durch die Dienstbeschreibungsnetze der Dienste klassifiziert. So kann es verschiedene Druckdienste im Beispiel des letzten Unterabschnitts geben. Diese gehorchen jedoch alle dem in der Abbildung 7.22 vorgestellten Dienstprotokoll eines Druckdienstes. Variieren kann hierbei lediglich die angesprochene Ressource.

Je nachdem, ob die Operatoren im Dienstbeschreibungsnetz direkt ausgewertet werden, oder ob stattdessen Terme im Netz schalten, kann in Abhängigkeit der Implementation der Algebra zwischen Ausführungsebene und Beschreibungsebene gewechselt werden. In der Ausführungsebene kann ähnlich wie in der Umsetzung CAPA (Duvigneau, 2002) der Multiagenten-Architektur MULAN (Rölke, 2004) ein Dienst tatsächlich ausgeführt werden. In der Beschreibungsebene kann hingegen eine symbolische Analyse der Terme oder eine Simulation des Verhaltens erfolgen.

Ontologien zur Datenrepräsentation

Die im Netz verwendeten Datentypen können durch eine Ontologie modelliert werden, die auf eine Algebra abgebildet wird. Wie in Definition 2.2 dargestellt, besteht die terminologische Struktur einer Ontologie aus einer Menge von Konzeptsymbolen $\mathfrak{C}$ zusammen mit einer Menge von Relationssymbolen $\mathfrak{R}$, wobei auf den Konzeptsymbolen zudem eine Ordnung $\preceq_{\mathfrak{C}} \subseteq \mathfrak{C} \times \mathfrak{C}$ existiert. Sind für die Relationssymbole lediglich funktionale Relationen zugelassen, so lassen sich sämtliche Relationssymbole als Attribute interpretieren, was eine direkte Abbildung der Konzeptsymbole auf die Sorten einer Zustandssignatur und der Relationssymbole auf die Operatoren erlaubt. Die Sortenhierarchie ergibt sich dann durch die Übertragung der Ordnung $\preceq_{\mathfrak{C}}$. Die Domäne und die Codomäne der Operatoren ist durch die Arität der Relationen vorgegeben. Eventuell existierende Axiome einer terminologischen Struktur

können zwar nicht direkt in der algebraischen Signatur beschrieben werden, ihre Verletzung kann jedoch während der Analyse der Netze überprüft werden. Die Abbildung der Datenstruktur von der Ontologie auf die Signatur erlaubt während der Modellierung durch Dienstbeschreibungsnetze das Typsystem anzupassen.

Die Interpretation der Zustandssignatur erfolgt durch die assertorische Struktur (vgl. Def. 2.3) einer Ontologie, die dann jedes Konzept und jedes Attribut der Zustandssignatur interpretiert. Hierdurch können die Strukturen der Signatur zu realweltlichen Dingen in Bezug gesetzt werden. Verwendet man über die Möglichkeiten klassischer Ontologien hinausgehend noch Rollenmigrationsdiagramme, so ergibt sich zudem die Möglichkeit, die möglichen Konzeptveränderungen einer Ontologie einzuschränken und ihre Verwendung zu analysieren.

Eine Einschränkung gegenüber Ontologien im Allgemeinen ist, dass Dienstbeschreibungsnetze ähnlich den meisten objektorientierten Systemen nur Attribute nicht aber Relationen zulassen. Diese Einschränkung bietet jedoch andererseits den Vorteil, dass sich die Konzepte der Ontologie direkt auf Klassen einer objektorientierten Programmiersprache übertragen lassen, was insbesondere für die Implementation der Operatoren eine Erleichterung ist. Allgemeine Relationen müssen dann durch entsprechende Attribute mit Mengen oder Listen als Wertebereich kodiert werden.

Ressourcen in der Ausführung

Auch wenn Ressourcen bei der Analyse vernachlässigt werden, sind sie während der Ausführung der Netze notwendig, da andernfalls keine realweltlichen Dinge angesprochen werden können. Eng verwandt mit der Nutzung von Ressourcen sind Ereignisse, die eine Veränderung im Zustand einer Ressource mitteilen. So ist es im Beispielnetz aus Abbildung 7.23 notwendig, dass sich der Zustand *available* der Ressource *packer* ändert, da andernfalls das Netz niemals ausgeführt werden kann. Dies kann durch ein Ereignis erfolgen, das ausgelöst wird, wenn sich dieses Attribut ändert, wie dies beispielsweise bei der letzten Transition des dargestellten Ablaufs der Fall ist. Die Bindungssuche im Netz erfährt in diesem Fall durch das ausgelöste Ereignis, dass die entsprechenden Transitionen erneut auf ihre Aktiviertheit überprüft werden müssen.

Die Implementation von Operatoren auf den Ressourcen erfolgt durch eine Programmiersprache wie Java. Hierdurch ist es möglich, dass beim Aufruf des *print*-Operators das übergebene Dokument direkt ausgedruckt wird. Auch hier kann es wiederum notwendig sein, über Ereignisse zu steuern, dass ein Vorgang abgeschlossen ist. Das Drucken eines Dokuments etwa blockiert solange, bis der Druckvorgang beendet ist und das Dokument für die weitere Verarbeitung bereit ist. Auf diese Weise kann sichergestellt werden, dass der reale Prozess und das Modell synchronisiert sind.

Mitarbeiter können ebenfalls als Ressourcen verstanden werden. Wie im Beispiel

des letzten Unterabschnitts kann die Ressource *packer* ein Mitarbeiter sein, der die Rolle *Packer* besitzt. Der Mitarbeiter bekommt dann entweder durch das System das nächste zu verarbeitende Objekt zugewiesen, oder er wählt ein Objekt aus einer Maske aus. Im ersten Fall wird der Ressource durch das System zugewiesen, was zu tun ist, im letzteren Fall wählt die Ressource dies selbstständig. Für das Modell ist dies nicht unbedingt relevant, da sich hierdurch nur die Art unterscheidet, wie das Attribut *packer.available* zu wahr auswertet, und somit nach welcher Reihenfolge die Prozesse fortgesetzt werden.

Die Analyse von Ressourcen soll an dieser Stelle ausgeklammert werden, da sie ein eigenes Forschungsfeld darstellt. Zwar wäre es denkbar, die Analyse der Objektzustände auf Ressourcen zu übertragen, jedoch ist hierbei zu beachten, dass der Zustand von Ressourcen in der Regel über viele Dienste hinweg bestimmt wird, so dass die Verfahren aus Kapitel 6 so nicht direkt umsetzbar sind. Aus diesem Grund wird die Analyse von Ressourcen hier vernachlässigt.

Testen der Komponenten

Neben dem Übergang vom Modell zur Implementation liegt ein weiterer Vorteil von Dienstbeschreibungsnetzen in der leichten Testbarkeit einzelner Komponenten. So ist es möglich, sämtliche Operatorimplementationen separat zu testen. Ebenso lassen sich Dienstbeschreibungsnetze mit entsprechenden Objekten initialisieren und ausführen, so dass sich jede Komponente des Systems individuell testen lässt. Eine Entwicklung entsprechender Testumgebungen stellt hier einen interessanten weiteren Forschungsaspekt dar, der hier jedoch nicht näher erläutert werden soll.

Dienstbeschreibungsnetze zur Dienstbeschreibung

Die Beschreibung von Diensten mit Dienstbeschreibungsnetzen ermöglicht Modellierung, Analyse und Ausführung dienstbasierter Systeme, ohne dass hierzu verschiedene Modelle oder Implementationen notwendig sind. Vielmehr kann durch die unterschiedliche Implementation der Operatoren zwischen der tatsächlichen Ausführung und der Simulation unterschieden werden. Durch die Sequentialisierung elementarer Dienstbeschreibungsnetze kann eine Analyse auf funktionaler Ebene erfolgen, die es erlaubt, mögliche Szenarien einer Ausführung zu analysieren.

Der Entwurf eines dienstbasierten Systems erfolgt dabei wie im Beispiel dargestellt durch eine grobe Analyse der vorhanden Systemkomponenten, die dann jeweils weiter verfeinert wird. Dabei wird einerseits die Prozesssicht und andererseits die Datensicht verfeinert. Bei der Analyse hilft die in Abschnitt 7.3 vorgestellte Architektur. Auch wenn dem Vorgehen ein Top-Down-Ansatz zugrundeliegt, lassen sich bestehende Systeme integrieren, indem ihr Verhalten jeweils durch eigene Dienstbeschreibungsnetze gekapselt wird, die dann eingebunden werden.

7.4.3 Erweiterungen und Vereinfachungen

Dienstbeschreibungsnetze bilden als Formalismus eine gute Grundlage für die Ausführung und die Analyse. Häufig ist es jedoch aufwendiger, als unbedingt notwendig, Konstrukte mit Petrinetzen zu modellieren. Aus diesem Grund lassen sich eine Reihe von Vereinfachungen auf graphischer Ebene vornehmen, die die Darstellung schlanker und leichter lesbar gestalten. Diese Vereinfachungen umfassen einerseits den Aufruf von Subdiensten und andererseits die Nutzung von Schleifen und Verzweigungen.

Darstellung von Schleifen

Gerade die Umsetzung der Endlichkeitsbedingung, wie sie in Definition 5.38 angegeben wurde, resultiert häufig in einem recht aufwendigen Netz, das eigentlich nur eine Liste oder eine Menge bearbeitet. Aus diesem Grund soll ein FOR-Konstrukt eingeführt werden, dass diese Notation verschlankt. Ein Beispiel eines Netzes, das eine Liste von Elementen bearbeitet ist in Abbildung 7.25 dargestellt.

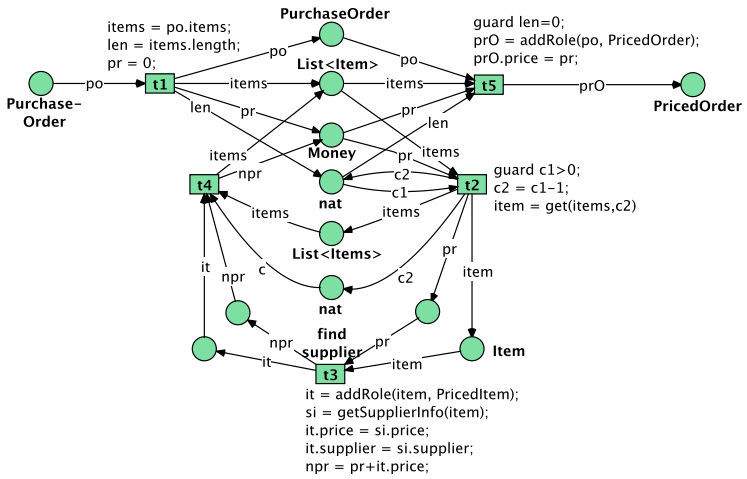


Abbildung 7.25: Ein Dienstbeschreibungsnetz, das das Auffinden von Zulieferern modelliert

Das Beispiel zeigt einen Ablauf, in dem für eine eingehende Bestellung (*PurchaseOrder*) die Zulieferer aufgefunden werden sollen und dann für jeden Zulieferer der Preis ermittelt werden soll. Dieser Preis wird summiert und dann dem Auftrag zugewiesen. Dieser erhält dazu das dynamische Konzept *PricedOrder*. Von der genauen Darstellung der Konzepte soll an dieser Stelle abstrahiert werden, sie dienen lediglich dazu, dem Beispiel einen Kontext zu geben.

Die Liste der bestellten Posten und die Länge dieser Liste werden nach dem Schalten der ersten Transition t_1 auf verschiedene Stellen abgelegt. Durch die Schleife mit

den Transitionen t_2, t_3 und t_4 wird jedes Element dieser Liste einmal aus der Liste extrahiert, und es werden seine Supplier-Informationen ermittelt. Dies geschieht über den Operator *getSupplierInfo*, der extern implementiert werden muss.

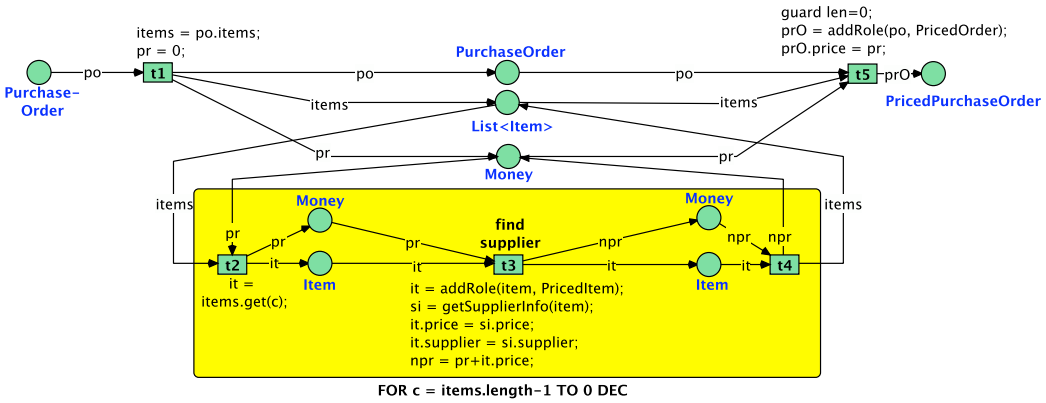


Abbildung 7.26: Das Dienstbeschreibungsnetz aus Abbildung 7.25 mit Schleifenkonstrukt

Dieses Netz lässt sich unter Verwendung eines Schleifenkonstruktes für einfache Schleifen deutlich schlanker darstellen, wie dies in Abbildung 7.26 zu sehen ist. Ein solches Schleifenkonstrukt bringt zudem den Vorteil mit sich, dass eine Überprüfung der Endlichkeitsbedingung entfallen kann, da diese durch die Syntax des Konstrukts vorgegeben ist. Beim Schleifenkonstrukt wird eine in der Anschrift angegebene Variable (c) entsprechend einer FOR-Schleife inkrementiert oder dekrementiert und in jedem Schleifendurchlauf der ersten Transition zur Verfügung gestellt.

Alternative Verzweigungen

Die Einbindung von Alternativtransitionen, wie sie in Definition 5.7 eingeführt worden sind, lässt sich ebenfalls durch geeignete Konstrukte erreichen. Bei Alternativen kann stets bei einer Transition das Schlüsselwort `default` verwendet werden. Dies ist beispielsweise in Abbildung 7.27 dargestellt. Ebenso wäre eine Bearbeitung von Alternativen von oben nach unten möglich, wobei `if-else`-Konstrukte erlaubt sind. Ein entsprechendes Beispiel ist in Abbildung 7.27 auf der rechten Seite dargestellt. Bei beiden Beispielen wurde die Bedingung in eckigen Klammern notiert, anstatt die *guard*-Anweisung zu nutzen, was ebenfalls einer kompakteren Schreibweise dient, die an die BPMN angelehnt ist.

Während `if-else`-Konstrukte bei zwei Alternativen sinnvoll sind, entspricht die `default`-Anweisung dem gleichen Konstrukt bei `switch`-Anweisungen in vielen Programmiersprachen.

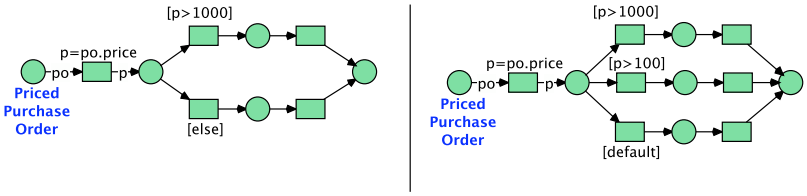


Abbildung 7.27: `default`- und `if-else`-Konstrukte im Dienstbeschreibungsnetz

Aufruf von Subdiensten

Der Aufruf von Subdiensten kann ebenfalls kompakter dargestellt werden. Bereits in den Netzen in Abbildung 7.19 wurde der Aufruf von Subdiensten durch eine Transition mit einem Dreieck gekennzeichnet. Dieses Vorgehen gibt explizit die Platzhaltertransition in einem Dienstbeschreibungsnetz an, die dann ersetzt werden kann. Hierdurch lässt sich der Aufruf von Subdiensten kompakt notieren, und es kann ermittelt werden, an welchen Positionen im Netz bei der Implementierung ein tatsächlicher Dienst aufgerufen werden muss. Wie dieser Dienst aufzurufen ist, ist dann Gegenstand des Dienstprotokolls. Hier ist aber auch eine Auswahl eines Dienstes zur Laufzeit denkbar, sofern er das Dienstprotokoll des an dieser Stelle vorgesehenen Dienstes erfüllt. Somit stellen Dienstprotokolle immer auch eine Klasse möglicher Dienste mit ähnlichem Verhalten dar. Unterschiede kann es hierbei dann in den genutzten Ressourcen geben.

Beim Aufruf gleichartiger Dienste kann es sinnvoll sein, ein Konstrukt zu erlauben, dass n gleichartige Dienste aufruft, die dann in der Analyse sequentialisiert werden können. Müssen etwa stets n Produkte bei verschiedenen Zulieferern beschafft werden, und ist sichergestellt, dass das resultierende Dienstbeschreibungsnetz elementar ist, so können diese Teilnetze des Dienstbeschreibungsnetzes ohne weiteres sequentialisiert werden. Dies erlaubt insbesondere auch dann eine Analyse der Abläufe, wenn erst zur Laufzeit bekannt ist, wieviele Teildienste nebenläufig ausgeführt werden sollen.

Vereinfachungen der Anweisungen

Die in Kapitel 4 in Abschnitt 4.2 eingeführten Anweisungen sind für die formale Definition der Netze von Vorteil, da eindeutig zwischen den verschiedenen Arten von Anweisungen unterschieden werden kann. Im praktischen Gebrauch ist es jedoch häufig unvorteilhaft, jede Update-Anweisung mit einem *update* einzuleiten und jede Zuweisungsanweisung mit einem *assign* einzuleiten. Aus diesem Grund ist es sinnvoll, diese Anweisungen wegzulassen, da sich aufgrund des Ausdrucks auf der linken Seite des Gleichheitszeichens stets ergibt, ob es sich um eine Update oder um eine Zuweisung handelt. Steht auf der linken Seite eine Variable, so stellt die

Anweisung eine Zuweisung dar, ist der Term hingegen keine Variable, und nutzt der Term einen Operator der Systemsignatur, so handelt es sich um ein Update.

Neben den Zuweisungs- und Update-Anweisungen lässt sich auch das erzeugen eines Objektes kompakter darstellen. Statt einer *new*-Anweisung gefolgt von verschiedenen Zuweisungsanweisungen zur Initialisierung des neu erzeugten Objektes lässt sich hier auch eine Notation der folgenden Art verwenden.

```
x = new Person[name="Tom", age=23, sex=male];
```

Hierdurch fallen die nachfolgenden Zuweisungsanweisungen weg, was eine deutlich kompaktere Notation erlaubt.

Nutzen der syntaktischen Erweiterungen

Zusammen mit den Schleifen-Konstrukten erlauben die `default` und `if-else`-Anweisung bereits wesentliche Elemente der elementaren Dienstbeschreibungsnetze direkt durch die Netzsyntax zu erzwingen. Die Überprüfung auf die Free-Choice-Eigenschaft ist ebenfalls leicht möglich und ist in den meisten Fällen genau wie die Sequentialisierbarkeit des Dienstbeschreibungsnetzes gegeben, da andere Modelle eher unintuitiv sind. Auch die Typisierungskonformität dürfte in den meisten praktischen Fällen gewährleistet sein, so dass lediglich die funktionale Ausführbarkeit der Elementarität der Dienstbeschreibungsnetze des Modells im Weg stehen kann. Hier muss eine Untersuchung an einer Vielzahl praktischer Fälle zeigen, in wie weit dieses Kriterium der Elementarität aus Definition 5.10 zu streng für praktische Zwecke ist. Andernfalls lassen sich sämtliche Analysen direkt auf das modellierte Dienstbeschreibungsnetz der Dienste übertragen.

Die hier vorgestellten Ergänzungen der Notation führen dann zu den in der Einleitung dargestellten wünschenswerten Eigenschaften zur Beschreibung von Diensten und erlauben die im Beispielnetz in Abbildung 1.1 verwendete Notation zur Beschreibung von Diensten. Die dort dargestellten Netze sind elementare Free-Choice-Dienstbeschreibungsnetze und verwenden die in Unterabschnitt 6.4.3 eingeführten Dienstinteraktionsmuster zur Kommunikation untereinander. Zudem wird dort auch das Konzept der Swimlanes aus diesem Kapitel bereits verwendet.

7.5 Zusammenfassung

Dieses Kapitel zeigte die Möglichkeiten der praktischen Nutzung von Dienstbeschreibungsnetzen auf. Hierzu wurde zunächst in Abschnitt 7.2 erläutert, welche Vorgehensweisen es zur Modellierung dienstbasierter Systeme gibt und zu welchen Zwecken eine solche Modellierung eingesetzt wird. Zudem wurde untersucht, wie sich der hier diskutierte Ansatz in bestehende petrinetzbasierte Modelle von Agenten und Diensten einbettet.

Durch die Möglichkeit der sukzessiven Verfeinerung unterstützen Dienstbeschreibungsnetze die in Unterabschnitt 7.2.1 diskutierten inter- und intraorganisationalen Szenarien. Als Vorgehensmodell bietet sich hierbei ein Top-Down-Vorgehen an, bei dem bestehende Teilsysteme durch Dienste bottom-up gekapselt werden, wie in Unterabschnitt 7.2.2 dargestellt wurde. Beides ist durch Dienstbeschreibungsnetze möglich und wird in Anlehung an den in Unterabschnitt 7.2.3 dargestellten PAOSE-Ansatz umgesetzt. Dabei wird die Modellierung von Systemen über die Möglichkeiten von Workflows hinaus unterstützt, da auch die Typisierung von Schnittstellen überprüft werden kann. Die Daten werden dabei anhand einer Ontologie modelliert, die systemweit gleich interpretiert werden muss. Hierbei kann jedoch eine systemweite Ontologie in den Abteilungen weiter spezifiziert werden. Durch Regeln kann bestimmt werden, welcher Dienst aufgrund eines eingehenden Ereignisses gewählt wird. Diese Ereignisse sind ebenfalls Teil der Ontologie.

Die verschiedenen Sichten werden in Abschnitt 7.3 anhand einer petrinetzbasierte Dienstarchitektur vorgestellt, die auf der vorhandenen petrinetzbasierten Agentenarchitektur MULAN aufbaut, diese jedoch teilweise recht stark ändert, um sie an den Dienstkontext anzupassen. So gibt es keine eigenen Wissenbasen der Dienste, und es werden zur Modellierung keine Netze-in-Netzen verwendet, da Dienste konzeptionell nicht migrieren. Vielmehr lauschen Dienste an einem logischen Bus nach Nachrichten, aufgrund derer sie dann einen Ablauf abarbeiten. Dieses Verhalten wird durch die entsprechende Dienstarchitektur reflektiert. Die Dienstarchitektur betrachtet hierzu jeweils die Sicht der Organisation in Unterabschnitt 7.3.1, der Daten in Unterabschnitt 7.3.2 und der Abläufe in Unterabschnitt 7.3.3. Das Zusammenspiel der Sichten wird dann in Abschnitt 7.4 anhand eines Beispiels dargestellt.

Das Vorgehen zur Erstellung dienstbasierter Systeme wird in Abschnitt 7.4 zunächst an einem Beispiel in Unterabschnitt 7.4.1 dargestellt, bevor das dort angewandte Vorgehensmodell anschließend in Unterabschnitt 7.4.2 näher erläutert wird. Das verwendete Vorgehensmodell ergibt sich dabei aus der Analyse verschiedener Vorgehensmodelle in Abschnitt 7.2 und aus den Überlegungen zur petrinetzbasierten Modellierung aus Abschnitt 7.3. Abschließend stellt Unterabschnitt 7.4.3 eine Reihe möglicher Erweiterungen der Notation des Formalismus vor, die zu einer einfacheren Darstellung und einer schnelleren Erstellung von Modellen führen.

8 Zusammenfassung und Ausblick

Dieses Kapitel fasst die wesentlichen Ergebnisse der Arbeit zusammen und erörtert sich hieran anschließende Fragestellungen in einem Ausblick. Ziel der Arbeit war es, einen benutzbaren Formalismus als Modellierungstechnik für dienstbasierte Geschäftsprozesse vorzustellen, was durch die Dienstbeschreibungsnetze gelungen ist. Hieran anschließen können sich nun Untersuchungen zum praktischen Einsatz des Formalismus und zu seiner weitergehenden Analyse sowie Überlegungen, diesen Formalismus in seiner Ausdrucksmächtigkeit zu erweitern und in Software-Entwicklungsansätze einzubetten.

8.1 Zusammenfassung

Die vorliegende Arbeit beschreibt eine Modellierungstechnik für dienstbasierte Geschäftsprozesse. Die Modellierung erfolgt durch den vom Autor entwickelten Formalismus der Dienstbeschreibungsnetze, der einen benutzbaren Formalismus mit einer theoretischen Grundlage bereitstellt und somit über die Modellierung hinaus eine weitergehende Analyse ermöglicht. Dienstbeschreibungsnetze erlauben es, die Operationalisierung eines Geschäftsprozesses durch einen Workflow anhand eines Dienstes zu repräsentieren und zu modellieren. Durch die Verwendung von Dienstbeschreibungsnetzen zur Modellierung, zur Analyse und zur Ausführung des Modells ergibt sich darüber hinaus die Möglichkeit, die Korrektheit von Modellen zu überprüfen und die Modelle gleichzeitig auszuführen oder zu simulieren. Durch die Trennung der Datenrepräsentation, der Funktionalitätsrepräsentation und der Ausführung können diese Teile des Systems zudem gezielt getestet werden.

Zur Darstellung der Modellierungstechnik und zur Darstellung des ihnen zugrundeliegenden theoretischen Fundaments wurden zunächst die allgemeinen Begrifflichkeiten erläutert, bevor dann Dienstbeschreibungsnetze und ihre Eigenschaften definiert und erläutert worden sind. Hierzu wurde in **Kapitel 2** zunächst eine Definition der grundlegenden Begriffe Dienst, Geschäftsprozess und Workflow gegeben und der Aufbau eines dienstbasierten Systems skizziert. Hierdurch wird ein allgemeines Verständnis der wesentlichen Begriffe der Arbeit gegeben, indem dargestellt wird, was ein Geschäftsprozess ist und wie sich seine Operationalisierung anhand eines Workflows durch Dienste abbilden lässt. Zur Repräsentation der Daten wurde zudem erläutert, wie sich Konzepte in einer Organisation durch Ontologien modellieren lassen.

Zur formalen Definition der Dienstbeschreibungsnetze in Kapitel 4 und zum Beweis von Eigenschaften von Dienstbeschreibungsnetzen in Kapitel 5 werden zudem Grundlagen des Formalismus und vorhandene Ergebnisse anderer Autoren zusammengefasst und in **Kapitel 3** dargestellt. Die wesentlichen Gebiete, die hier dargestellt werden, sind Algebren und ordnungssortierte Algebren auf der einen Seite und Petrinetze und ihre Eigenschaften auf der anderen Seite. Zudem wird anhand der algebraischen Petrinetze das Zusammenspiel der beiden Formalismen vorgestellt.

Kapitel 4 führt den vom Autor entwickelten Formalismus der Dienstbeschreibungsnetze ein, was zunächst umgangssprachlich und im Anschluss daran formal erfolgt. Hierzu werden zunächst die Syntax und die Semantik der Anweisungen des Dienstbeschreibungsnetzes eingeführt, die dazu dienen, einen Systemzustand zu manipulieren. Dies erfolgt in Abschnitt 4.2, wo zunächst in Unterabschnitt 4.2.1 der Begriff des Zustands präzisiert wird, bevor die Anweisungen auf einem solchen Zustand dann in Unterabschnitt 4.2.2 auf syntaktischer Ebene dargestellt werden und in Unterabschnitt 4.2.3 ihre Semantik erläutert wird.

Unter Verwendung der Anweisungen und der sich hieraus zusammensetzenden Anweisungsblöcke werden in Abschnitt 4.3 Dienstbeschreibungsnetze eingeführt. Dies erfolgt über die statischen Netzeigenschaften in Unterabschnitt 4.3.1, die den Aufbau eines Dienstbeschreibungsnetzes widerspiegeln, woran sich die Betrachtung der dynamischen Netzeigenschaften in Unterabschnitt 4.3.2 anschließt. Neben der zentralen Definition eines Dienstbeschreibungsnetzes in Definition 4.16 sowie der Markierung eines Dienstbeschreibungsnetzes in Definition 4.17 findet sich hier auch die Definition der Schaltregel und der Folgemarkierung in Definition 4.21 beziehungsweise in Definition 4.24. Während erstere Definition für Dienstbeschreibungsnetze ohne die Möglichkeit, dynamische Konzept zu entfernen, gilt, repräsentiert letztere Definition die Schaltregel für Dienstbeschreibungsnetze mit dieser Möglichkeit. Diese Schaltregeln müssen unterschieden werden, da andernfalls das Entfernen einer dynamischen Sorte die Stellentypisierung verletzen könnte. Unterabschnitt 4.3.3 setzt das Verhalten der Dienstbeschreibungsnetze zu anderen Netzformalismen in Beziehung. Hier stehen vor allem die zugrundeliegenden Workflownetze und die algebraischen Netze im Vordergrund. Es stellt sich in diesem Zusammenhang heraus, dass alle wichtigen Fragen der Erreichbarkeit und der Lebendigkeit des Abschlusses für Dienstbeschreibungsnetze im Allgemeinen unentscheidbar sind.

Analog zu Workflownetzen ist auch für Dienstbeschreibungsnetze die Überprüfung von Eigenschaften von Interesse, die auf eine korrekte Modellierung des Netzes schließen lassen und die vom Autor analog zum Korrektheitsbegriff für Workflownetze definiert worden sind. Aus diesem Grund schließt Kapitel 4 mit der Angabe verschiedener Eigenschaften von Dienstbeschreibungsnetzen, zu deren wichtigsten die funktionale Ausführbarkeit aus Definition 4.36 und die schwache Korrektheit aus Definition 4.46 gehören. Die funktionale Ausführbarkeit bezeichnet dabei den Umstand, dass die Zustandsänderungen zweier nebenläufig aktivierter Transitionen stets unabhängig voneinander sind, so dass die Schaltreihenfolge der Transitio-

nen auf den resultierenden Folgezustand keinerlei Auswirkung hat. Die schwache Korrektheit ist eine Abschwächung des Korrektheitsbegriffs für Workflownetze, der dann auf Dienstbeschreibungsnetze mit Daten übertragen wird.

Die Unentscheidbarkeit der schwachen Korrektheit legt es nahe, die Ausdruckskraft einzuschränken und dafür eine bessere Analysierbarkeit von Dienstbeschreibungsnetzen zu erreichen. Die vom Autor dieser Arbeit entwickelten und in **Kapitel 5** eingeführten elementaren Dienstbeschreibungsnetze stellen einen solchen Kompromiss zwischen Ausdruckskraft und Analysierbarkeit dar. Unter Verwendung der in Kapitel 3 dargestellten formalen Grundlagen wird in Kapitel 5 für diese Subklasse der Dienstbeschreibungsnetze dargestellt, wie sich die Eigenschaften der funktionalen Ausführbarkeit und der schwachen Korrektheit überprüfen lassen, wozu zunächst elementare Dienstbeschreibungsnetze in Abschnitt 5.2 dargestellt werden. Definition 5.10 formalisiert elementare Dienstbeschreibungsnetze. Aufgrund der einfacheren Struktur dieser Netze fallen für diese Netzklasse die Schaltregel mit und diejenige ohne Entfernen dynamischer Konzepte zusammen. Gleichzeitig gilt für diese Netzklasse stets die schwache Korrektheit, wie in Satz 5.11 gezeigt wird.

Ob ein gegebenes Dienstbeschreibungsnetz elementar ist, lässt sich in Polynomialzeit zeigen, wenn das zugrundeliegende Workflownetz die Free-Choice-Eigenschaft besitzt und das Dienstbeschreibungsnetz endlich ist. Da eine Vielzahl von praktisch relevanten Abläufen durch Free-Choice-Workflownetze darstellbar sind, werden im weiteren Verlauf des Kapitels Dienstbeschreibungsnetze auf Free-Choice-Dienstbeschreibungsnetze eingeschränkt. Ihre Analyse erfolgt dann über die Analyse des zugrundeliegenden Free-Choice-Workflownetzes. Aus diesem Grund widmet sich Abschnitt 5.3 der Analyse der Korrektheit und der Analyse erreichbarer Markierungen in Free-Choice-Workflownetzen. Hierzu werden in Unterabschnitt 5.3.1 zunächst allgemeine Eigenschaften von Free-Choice-Netzen dargestellt. Zu den wichtigsten Ergebnissen gehört hierbei das vom Autor gezeigte Ergebnis aus Satz 5.19, nach dem das Subnetz einer T-Invariante eines wohlgeformten Free-Choice-Netzes wiederum ein wohlgeformtes Free-Choice-Netz ist. Diese Ergebnisse ermöglichen einerseits, ein Verfahren zur Überprüfung der Wohlgeformtheit eines Free-Choice-Workflownetzes in Verfahren 5.28 anzugeben, und erlauben es andererseits, die gleichzeitige Markierbarkeit von Stellen in einem solchen Netz festzustellen, was mittels der Ergebnisse aus Satz 5.33 möglich ist. Beide Ergebnisse wurden vom Autor gezeigt und werden in Abschnitt 5.4 zur Überprüfung der Elementarität von Free-Choice-Dienstbeschreibungsnetzen benötigt. Um die Analyse beschleunigen zu können, wurden in Unterabschnitt 5.3.3 noch verschiedene Reduktionsregeln dargestellt, die es erlauben, ein Workflownetz vor der weitergehenden Analyse zu reduzieren.

Die Elementarität kann nun in Abschnitt 5.4 über die Eigenschaften des zugrundeliegenden Netzes gezeigt werden, wobei Free-Choice-Dienstbeschreibungsnetze zunächst in Unterabschnitt 5.4.1 auf endliche Free-Choice-Dienstbeschreibungsnetze eingeschränkt werden. Diese erlauben für jede T-Invariante des zugrundeliegenden

Workflownetzes lediglich eine endliche Zahl von Realisierungen der Transitionen dieser T-Invariante im Dienstbeschreibungsnetz, so dass jede Schaltfolge im Dienstbeschreibungsnetz endlich wird. Dies wird durch einen Zähler im Dienstbeschreibungsnetz erreicht. In den meisten praktischen Fällen lässt sich ein hinreichend großer Initialwert finden, so dass ein Dienstbeschreibungsnetz als endlich angenommen werden kann. Unterabschnitt 5.4.2 analysiert dann die Elementarität dieser Netze, was anhand der in diesem Kapitel dargestellten Ergebnisse in Polynomialzeit möglich ist. Abschließend werden in Unterabschnitt 5.4.3 noch einmal die Reduktionsregeln auf Dienstbeschreibungsnetze ausgeweitet, und es wird skizziert, wie sich Teile eines Dienstbeschreibungsnetzes durch diese Reduktionsregeln vereinfachen lassen.

Um zu entscheiden, ob ein Dienst sinnvoll modelliert wurde, ist die schwache Korrektheit häufig notwendig, meist jedoch nicht hinreichend. Eine weiterführende Analyse von Dienstbeschreibungsnetzen ist jedoch in der Regel nur über eine Betrachtung des (symbolischen) Zustandsraums möglich. Eine solche Analyse birgt jedoch die Gefahr, dass der Zustandsraum zu groß ist, um noch beherrschbar zu sein. Ein Weg, die Explosion des Zustandsraums zu verhindern, ist durch die Entfernung der Nebenläufigkeit aus dem Netz gegeben. Dies ist durch ein vom Autor dieser Arbeit entwickeltes Verfahren möglich, das in **Kapitel 6** eingeführt wird. Dieses Verfahren basiert wiederum auf der Analyse des zugrundeliegenden Workflownetzes, so dass zunächst die Sequentialisierung eines Free-Choice-Workflownetzes in Abschnitt 6.2 betrachtet wird. Unterabschnitt 6.2.1 beschreibt zunächst, was ein Sequentialisierung ist und wann ein Netz stark sequentialisierbar ist. Zu einem stark sequentialisierbaren Workflownetz existiert eine Zustandsmaschine mit gleicher Transitionsmenge, so dass zu jeder Schaltfolge im Ursprungsnetz bis auf Vertauschung nebenläufiger Transitionen eine Schaltfolge in der Sequentialisierung existiert. Dies ist in Definition 6.3 und in Definition 6.4 festgehalten. Ist ein Free-Choice-Workflownetz stark sequentialisierbar, so gibt Unterabschnitt 6.2.2 mit dem vom Autor entwickelten Verfahren 6.8 eine Vorgehensweise zur Sequentialisierung dieses Netzes an. Die Korrektheit dieses Verfahrens wird dann in Satz 6.12 dargestellt. Wiederum lässt sich eine Sequentialisierung dadurch beschleunigen, dass durch Reduktionsregeln das zu sequentialisierende Netz verkleinert wird. Hierzu werden in Unterabschnitt 6.2.3 verschiedene vom Autor für die Sequentialisierung angepasste Regeln skizziert, die aufbauend auf den Regeln aus Unterabschnitt 5.3.3 eine Sequentialisierung eines Netzes erlauben.

Abschnitt 6.3 widmet sich der Sequentialisierung elementarer Free-Choice-Dienstbeschreibungsnetze. Diese werden dadurch sequentialisiert, dass zunächst das zugrundeliegende Workflownetz sequentialisiert wird, und dann die Anschriften des ursprünglichen Netzes auf diese Sequentialisierung übertragen werden. Dabei gehen jedoch Stellen verloren, die zum Austausch von Daten notwendig sind. Aus diesem Grund werden Quasi-Zustandsmaschinen-Workflownetze eingeführt, bei denen eine S-Komponente des Netzes alle Transitionen überdeckt, so dass es keine nebenläufig

aktivierten Transitionen im Netz geben kann. Die in Unterabschnitt 6.3.1 eingeführte erweiterte Sequentialisierung bildet ein Free-Choice-Workflownetz auf eine Quasi-Zustandsmaschine ab, wobei der Vor- und Nachbereich einer Transition in Takt bleibt, jedoch gegebenenfalls durch weitere Stellen ergänzt wird, so dass es keine nebenläufig aktivierten Transitionen mehr gibt. Die erweiterte Sequentialisierung kann dann direkt auf elementare Dienstbeschreibungsnetze übertragen werden, was in Unterabschnitt 6.3.2 anhand von Definition 6.18 geschieht. Abschließend erfolgt in Satz 6.20 der Nachweis, dass die erweiterte Sequentialisierung zu den gleichen finalen Netzzuständen führt wie das Ursprungsnetz. Wie anhand der Sequentialisierung dann eine Analyse erfolgen kann, wird in Unterabschnitt 6.3.3 skizziert, wobei dargestellt wird, wie sich ein Dienstbeschreibungsnetz in ein algebraisches Netz mit einem zentralen Zustand überführen lässt.

Den Abschluss von Kapitel 6 bildet die Analyse der Zustände eines Objektes im Zusammenhang mit der Analyse des elementaren Dienstbeschreibungsnetzes in Abschnitt 6.4. Objekte können hierbei verschiedene Zustände annehmen, die über Rollen dargestellt werden. Diese Darstellung der Objektrollen und ihrer möglichen Übergänge wird meist als Rollenmigrationsdiagramm bezeichnet. Hierbei sind nicht alle Zustandsübergänge sinnvoll, so dass analysiert werden kann, ob das in einem Ablauf modellierte Verhalten dem Verhalten der Objektzustände entspricht. Ist dies nicht der Fall, so ist meist ein Modellierungsfehler gegeben.

Die Arbeit endet mit einer Analyse des praktischen Einsatzes von Dienstbeschreibungsnetzen in **Kapitel 7**. In diesem Kapitel wird zunächst in Abschnitt 7.2 die Modellierung von dienstbasierten Systemen und von Geschäftsprozessen dargestellt und analysiert. Unterabschnitt 7.2.1 stellt zunächst die Verwendungsmöglichkeiten petrinetzbasierter Ansätze dar, bevor Unterabschnitt 7.2.2 den praktischen Einsatz und Vorgehensmodelle zum Entwurf dienstbasierter Systeme skizziert. Dabei stellt sich ein Top-Down-Vorgehen als sinnvoll heraus, das durch die Einbettung bestehender Systeme ergänzt wird. Die Vorgehensweise bestehender Ansätze der petrinetzbasierten Modellierung wird in Unterabschnitt 7.2.3 anhand des Paose-Ansatzes erläutert, der dann auf dienstbasierte Systeme übertragen wird. Dieses Vorgehen zur Modellierung wird anhand einer petrinetzbasierten Dienstarchitektur in Abschnitt 7.3 genauer ausgeführt. Diese Architektur besitzt vier Ebenen, anhand derer die ein Modell zunächst grob betrachtet werden kann und dann sukzessive verfeinert werden kann, so dass letztlich aus einer weiteren Verfeinerung des Modells die eigentliche Implementation erwächst. Dieses Vorgehen hat den Vorteil, dass Modell und Implementation direkt miteinander verknüpft sind, und es somit keine Lücke zwischen Modell und Implementation gibt. Unterabschnitt 7.3.1 führt das Modell anhand der verschiedenen Ebenen einer Organisation ein, während Unterabschnitt 7.3.2 dann detaillierter auf die Modellierung der Daten eingeht und Unterabschnitt 7.3.3 dann die Modellierung der Prozessen betrachtet.

Eine beispielhafte Umsetzung dieses Vorgehens findet sich in Abschnitt 7.4, woran sich eine Analyse des Vorgehens anschließt. Unterabschnitt 7.4.1 schildert zunächst,

wie sich Dienstbeschreibungsnetze zur Modellierung der Prozesse einsetzen lassen. Unterabschnitt 7.4.2 geht dann auf die Ausführbarkeit der Modelle ein. Abschließend betrachtet Unterabschnitt 7.4.3 mögliche Erweiterungen und Vereinfachungen in der Notation, so dass sich Dienstverhalten durch Dienstbeschreibungsnetze kompakter darstellen lässt. Abgeschlossen wird das Kapitel durch die Angabe möglicher Vereinfachungen in der Notation, so dass Modelle schneller erstellt werden können.

Insgesamt stellt die Arbeit einen Formalismus bereit, mit dem sich Dienste modellieren und ausführen lassen und der darüber hinaus eine Analyse der Modelle erlaubt. Der praktische Einsatz dieses Formalismus muss nun zeigen, in wie weit noch Ergänzungen hinsichtlich der Ausdruckskraft und der Analyse notwendig sind. Ein Teil möglicher Ergänzungen findet sich im nun folgenden Ausblick.

8.2 Ausblick

Der in dieser Arbeit vorgestellte Ansatz zur Modellierung dienstbasierter Geschäftsprozesse eröffnet eine Vielzahl weiterer Möglichkeiten, die den vorgestellten Ansatz verwenden, ihn für den praktischen Einsatz ergänzen oder die Analyse des vorgestellten Netzformalismus verbessern. Die sich aus den Ergebnissen dieser Arbeit ergebende Folgefragen sollen daher in diesem Abschnitt skizziert werden. Dabei wird zunächst auf Ergänzungen des Formalismus eingegangen, um im Anschluss daran den praktischen Einsatz der Netze im Dienst- und Agentenkontext zu diskutieren.

Der Formalismus der Dienstbeschreibungsnetze ist relativ ausdrucksstark. Dennoch lassen sich Erweiterungen identifizieren, die zusätzliche, in praktischen Fällen relevante Möglichkeiten der Modellierung mit sich bringen. Eine dieser Ergänzung wäre die Verwendung von Netzen oder Netzsystemen als Datentypen in einem Dienstbeschreibungsnetz. Dies ist ähnlich dem Netze-in-Netzen Prinzip von Valk (1998), das auch in den Referenznetzen von Kummer (2002) umgesetzt wurde. Die Netze als Datentypen könnten über Operatoren ergänzt und verfeinert werden, wie dies etwa bei den Hornets von Köhler-Bußmeier (2009b) erfolgt. Weitere Operatoren auf den Netzsystemen können dann die Markierung im Netz verändern und so ein Schalten des Netzes repräsentieren. Diese Möglichkeit würde es erlauben, einem Dienst nicht nur Daten zu übergeben, sondern auch Funktionalität anhand eines Dienstbeschreibungsnetzes, das wiederum einen Dienst repräsentiert, der aufgerufen werden soll. Dieser Dienst kann entweder lokal Funktionalität bereitstellen, oder er kann als Callback dienen und den Aufrufer des ursprünglichen Dienstes über Ereignisse informieren. Das Übertragen von Funktionalität ist eng verwandt mit der Migration von Agenten in Multiagentensystemen. Somit ließen sich durch diese Möglichkeit auch petrinetzbasierte Multiagentensysteme wie Mulan (vgl Rölke, 2004) ergänzen und stärker formalisieren.

Für die Modellierung von Geschäftsprozessen ergibt sich eine weitere Ergänzung. Geschäftsprozesse sind im Allgemeinen an Ressourcen gebunden, so dass eine stär-

kere Einbeziehung von Ressourcen bei der Analyse sinnvoll ist. Diese müssten dann ebenfalls im Netz modellierbar sein. Erste Ansätze hierzu wurden in Abschnitt 7.4 diskutiert, jedoch verlangen Ressourcen eine weitergehende Analyse und eine differenziertere Betrachtung der Nebenläufigkeit (vgl. van Hee u. a., 2005). Dies führt jedoch in Verbindung mit Daten unter Umständen zu nicht mehr entscheidbaren Problemen, so dass hierfür geeignete Teilsysteme gefunden werden müssen, anhand derer sich eine Analyse durchführen lässt.

Verwandt hiermit ist eine detailliertere Betrachtung des Objektlebenszyklus über die in Abschnitt 6.4 dargestellten Möglichkeiten hinaus. Auch die Analyse der Interaktion zweier Dienste ist durch das Verschmelzen von Netzen möglich. In beiden Bereichen lassen sich die Ergebnisse von Reisig (2009) und Wolf (2009a,b) übernehmen. Die Ansätze dort legen den Fokus zwar auf Dienstverhalten, es lassen sich jedoch aufgrund der Nähe des Dienstverständnisses der Autoren zu Objektlebenszyklen die Analysemethoden vielfach übertragen.

Bei der Analyse von Dienstbeschreibungsnetzen ist interessant, in wie weit sich über elementare Free-Choice-Dienstbeschreibungsnetze hinausgehend eine Analyse durchführen lässt. Die Beweise in Kapitel 5 und das dort dargestellte Vorgehen zur Analyse der Free-Choice-Workflownetze beziehen sich sehr stark auf die S- und T-Komponenten. Dies legt die Vermutung nahe, dass sich die Ergebenisse eventuell auf Workflownetze übertragen lassen, in deren Abschluss sowohl das Subnetz jeder minimalen S-Invariante eine S-Komponente als auch das Subnetz jeder minimalen T-Invariante eine T-Komponente ist. Allerdings müsste hierbei statt des Rang-Theorems und der Analyse mittels der Cluster eine andere Herangehensweise gefunden werden, um beim Hinzufügen der minimalen T-Invarianten analysieren zu können, ob ein Netz korrekt bleibt. Eine solche Klasse von zugrundeliegenden Workflownetzen würde die Ausdruckskraft der zu analysierenden Dienstbeschreibungsnetze noch einmal erhöhen, da die Free-Choice-Workflownetze eine Subklasse dieser Netze wären (vgl. Lemma 5.14).

Mittels Techniken der symbolischen Analyse (vgl. Jhala und Majumdar, 2009) lässt sich über die Vorgehensweise für elementare Dienstbeschreibungsnetze hinaus zudem auch für nicht-elementare Dienstbeschreibungsnetze zum Teil die schwache Korrektheit nachweisen. Hier ist die Frage interessant, in wie weit sich dann auch die Sequentialisierung übertragen lässt, so dass sich die Klasse der analysierbaren Netze ausweiten lässt. Auf die Sequentialisierung selbst lassen sich zudem Konzepte aus dem Compilerbau, wie sie etwa von Muchnick (1997) beschrieben werden, anwenden. Auch die Analyse mittels der Termersetzung (vgl. Baader und Nipkow, 1998; Goguen und Lin, 2003; Bezem u. a., 2003) bietet hier vielfach interessante Möglichkeiten zur Vereinfachung der zu analysierenden Netze und zu ihrer Analyse selbst.

Im Bereich der Verwendung von Dienstbeschreibungsnetzen sind einerseits Fragestellungen zur intuitiveren Notation und andererseits Fragestellungen zur Integration in bestehende Ansätze und Werkzeuge von Interesse. Dienstbeschreibungs-

netze basieren auf Petrinetzen als formalem Modell. Häufig ist für den praktischen Einsatz jedoch eine andere Notation für den Modellierer intuitiver. Somit ist es unter Umständen sinnvoll, eine Notation mit größerer Verbreitung in der Welt der Prozessmodellierung wie BPMN auf Dienstbeschreibungsnetze abzubilden, um so diese Netzklasse einem größeren Publikum verfügbar zu machen. Die Abbildung von BPMN auf Petrinetze wird beispielsweise von Decker u. a. (2008) und von Dijkman u. a. (2008) diskutiert. Ähnliche Konstrukte finden sich in der Sprache YAWL (Yet Another Workflow Language) (vgl. YAWL, 2008), wie sie etwa Russell u. a. (2007) anhand von gefärbten Netzen beschreibt. Ob die dort beschriebenen Konstrukte, die nicht mit einfachen Workflownetzen darstellbar sind, wie etwa OR-Splits und -Joins, in praktischen Fällen bei der Verwendung von Dienstbeschreibungsnetzen notwendig sind, wäre ebenfalls zu klären, da durch sie schnell die Möglichkeiten der Analyse eingeschränkt werden.

Der Einsatz von Dienstbeschreibungsnetzen ist insbesondere im Zusammenhang mit der Modellierung dienstbasierter Systeme und darüber hinausgehend bei der Modellierung agentenbasierter Systeme interessant. Da Dienste prinzipiell auch durch Agenten bereitgestellt werden können, stellen dienstbasierte Systeme eine Teilmenge der agentenbasierten Systeme dar. Letztere erlauben zusätzlich noch die Migration von Agenten, was durch Netze-in-Netzen unterstützt werden kann. Hier lassen sich Dienstbeschreibungsnetze im Kontext petrinetzbasierter Multiagentensysteme wie Mulan (Rölke, 2004) einsetzen. Betrachtet man Dienstbeschreibungsnetze als Austauschformat für die Semantik von Diensten, so ließen sich durch Dienstbeschreibungsnetze auch funktionale Schnittstellen austauschen, wie dies im Kontext von Agenten zur verteilten Planung interessant ist. Ansätze, die Semantik von Prozessen durch Prozess-Ontologien zu beschreiben, finden sich durch OWL-S (2006), deren Prozessspezifikation durch Moldt und Ortmann (2004a) auf Referenznetze übertragen wurde. Eine weitergehende Anordnung von Dienstschnittstellen anhand einer Ontologie erfordert sowohl eine Klassifizierung anhand der Schnittstelle als auch anhand der Funktionalität eines Dienstes. Hier wäre ebenfalls der Einsatz von Dienstbeschreibungsnetzen interessant. So könnten unterspezifizierte Dienstbeschreibungsnetze als Schnittstelle genutzt werden, um zu beschreiben, welche Funktionalität in jedem Fall notwendig ist. Diese Netze können dann weiter verfeinert werden, um eine Implementation der Schnittstelle zu erhalten. Eine einfache Klassifizierung wird dabei bereits dadurch erreicht, dass alle Dienste mit demselben Dienstprotokoll einer Klasse zugerechnet werden können. Darüber hinaus ließe sich noch eine weitergehende Klassifizierung anhand von Eigenschaften erreichen, die auf jeden Fall in einem Dienstprotokoll vorkommen müssen und anhand von Eigenschaften, die niemals in einem solchen Protokoll vorkommen dürfen.

An dieses Forschungsfeld schließt sich die Planung von Abläufen durch Agenten an. Aufgrund der präzise definierten Semantik und der Möglichkeit, Netze symbolisch auszuführen, können unter Verwendung von Dienstbeschreibungsnetzen komplexere Abläufe aus einfacheren Abläufen geplant werden. Hier wäre es

interessant, Ansätze wie den von Castilho u. a. (2004) weiterzuverfolgen, wo Petrinetze als Grundlage eines Planungsverfahrens eingesetzt werden. Im Agentenkontext ist hierbei die Verwendung von hierarchischen Planern naheliegend, wie sie im Bereich der Hierarchical Task Networks eingesetzt werden (vgl. LaValle, 2006; Ghallab u. a., 2004). Verwandt mit der Planung ist der Bereich der Simulation von Szenarien. Hierzu wäre es interessant, Dienstbeschreibungsnetze um Elemente stochastischer Petrinetze (vgl. Haas, 2002) zu ergänzen, so dass simuliert werden kann, wie lange eine Transition zum Schalten braucht, und mit welcher Wahrscheinlichkeit eine aktivierte Transition auch feuert. Für die Implementation umfangreicher Systeme müsste zudem die Möglichkeit in Dienstbeschreibungsnetzen bereitgestellt werden, die Analyse auch auf die Reservierung von Lokationen auszudehnen, wodurch sich auch ein transaktionales Verhalten analysieren ließe. Für elementare Dienstbeschreibungsnetze ist dieses zwar irrelevant, da sie nicht nebenläufig auf dieselben Lokationen zugreifen, allgemein dürfte diese Fragestellung jedoch interessant sein. Hier ließen sich Ergebnisse aus dem Bereich der Datenbanken etwa von Weikum und Vossen (2002) oder von Jessen und Valk (1987) übernehmen, um eine Variable dann über verschiedene Transitionen hinweg als (transaktional) geschützt zu betrachten.

Neben dem Planen ist auch das Auffinden von wiederkehrenden Abläufen in Organisationen interessant, was meist unter dem Stichwort Process Mining (vgl. Schimm, 2004) subsumiert wird. Dienstbeschreibungsnetze bieten durch die Modellierung von Rollen hier den Vorteil, dass auch Veränderungen von Objekten in Abläufen identifiziert werden können und so gegebenenfalls Prozessen zugeordnet werden können. Ansätze des petrinetzbasierten Process Mining finden sich bereits bei de Medeiros u. a. (2007) und bei Cabac und Denz (2008). Durch die explizite Darstellung von Objektlebenszyklen kann einerseits identifiziert werden, wie sich Objekte verändern und andererseits identifiziert werden, wie sich das Zusammenspiel mehrerer Objekte in einem Prozess verhält.

Der praktische Einsatz von Dienstbeschreibungsnetzen ist darüber hinaus in verschiedenen Bereichen denkbar. So wäre insbesondere die Überprüfung von Plausibilitätseigenschaften in virtuellen Dienstnetzwerken, wie sie Zirpins (2007) darstellt, interessant. Hier könnte mittels einer formalen Analyse überprüft werden, ob den virtuellen Partnern gegenüber gewisse Garantien gemacht werden können.

Für eine weitergehende Betrachtung wäre auch der Einsatz von Dienstbeschreibungsnetzen zur Beschreibung von Schnittstellen von Plugin-Systemen vorstellbar. Eine petrinetzbasierte Architektur wurde hier bereits von Duvigneau (2009) vorgestellt. Dies lässt sich zum Teil auch auf komponentenbasierte Systeme übertragen, bei denen beispielsweise Griffel (2001) ebenfalls die gleichen Schnittstellen identifiziert, die auch durch Dienstbeschreibungsnetze betrachtet werden, wobei der Autor dort das Verhalten durch Automaten spezifiziert.

Die Möglichkeit der Simulation von Dienstbeschreibungsnetzen lässt sich darüber hinaus als didaktisches Mittel verwenden. Erste Ansätze, soziologische Theorien im

Rahmen der Sozionik für den E-Learning-Kontext aufzubereiten, finden sich bei Moldt u. a. (2004a). Die dort verwendeten Modelle basieren auf Vorarbeiten von von Lüde u. a. (2003). Dienstbeschreibungsnetze sind zwar weniger zur Repräsentation allgemeiner soziologischer Sachverhalte geeignet, sie lassen sich jedoch einsetzen, um organisationale Strukturen und Abläufe darzustellen und zu simulieren. Darüber hinaus lassen sich Modelle nebenläufiger Systeme mit Hilfe von Petrinetzen lehren (vgl. Kristensen und Jensen, 2008).

Zusammenfassend bieten Dienstbeschreibungsnetze eine formale Basis, auf deren Grundlage sich eine Vielzahl von weitergehenden Forschungsfeldern betrachten lassen. Da sie auf Petrinetzen und Algebren basieren, lässt sich ihr Verhalten präzise angeben, so dass das Verhalten weiterer (graphischer) Prozessbeschreibungssprachen auf ihrer Grundlage spezifiziert werden und eine Analyse dieser Sprachen erfolgen kann.

Literaturverzeichnis

[van der Aalst 1997] Aalst, Wil M. P. van der: Verification of Workflow Nets. In: Azéma, Pierre (Hrsg.) ; Balbo, Gianfranco (Hrsg.): *Application and Theory of Petri Nets 1997*. Berlin u.a. : Springer-Verlag, 1997 (Lecture Notes in Computer Science 1248), S. 407–426 81

[van der Aalst 1999a] Aalst, Wil M. P. van der: Formalization and verification of event-driven process chains. In: *Information & Software Technology* 41 (1999), Nr. 10, S. 639–650 25

[van der Aalst 1999b] Aalst, Wil M. P. van der: Interorganizational Workflows: An Approach based on Message Sequence Charts and Petri Nets. In: *Systems Analysis - Modelling - Simulation* 34 (1999), Nr. 3, S. 335–367 20

[van der Aalst 2000] Aalst, Wil M. P. van der: Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques. In: Desel, Jörg (Hrsg.) ; Oberweis, Andreas (Hrsg.): *Business Process Management, Models, Techniques, and Empirical Studies*. Berlin u.a. : Springer-Verlag, 2000, S. 161–183 21, 22, 47, 69, 82, 146, 162

[van der Aalst 2002] Aalst, Wil M. P. van der: Inheritance of Interorganizational Workflows to Enable Business-to-Business. In: *Electronic Commerce Research* 2 (2002), Nr. 3, S. 195–231 283, 284

[van der Aalst 2003] Aalst, Wil M. P. van der: Inheritance of Business Processes: A Journey Visiting Four Notorious Problems. In: (Ehrig u. a., 2003), S. 383–408 283, 284

[van der Aalst u. a. 2000] Aalst, Wil M. P. van der ; Barros, Alistair P. ; Hofstede, Arthur H. M. ter ; Kiepuszewski, Bartek: Advanced Workflow Patterns. In: Etzion, O. (Hrsg.) ; Scheuermann, P. (Hrsg.): *7th International Conference on Cooperative Information Systems (CoopIS 2000)* Bd. 1901. Berlin u.a. : Springer-Verlag, 2000, S. 18–29. – WWW-Dokument†: http://is.tm.tue.nl/staff/wvdaalst/publications/p105.pdf 303

[van der Aalst und Basten 1997] Aalst, Wil M. P. van der ; Basten, Twan: Life-Cycle Inheritance: A Petri-Net-Based Approach. In: Azéma, Pierre (Hrsg.) ; Balbo, Gianfranco (Hrsg.): *ICATPN* Bd. 1248. Berlin u.a. : Springer-Verlag, 1997, S. 62–81 285

[van der Aalst und Basten 2002] Aalst, Wil M. P. van der ; Basten, Twan: Inheritance of workflows: an approach to tackling problems related to change. In: *Theor. Comput. Sci.* 270 (2002), Nr. 1-2, S. 125–203 285

[van der Aalst u. a. 2002] Aalst, Wil M. P. van der ; Desel, Jörg ; Kindler, Ekkart: On the semantics of EPCs: A vicious circle. In: Nüttgens, Markus (Hrsg.) ; Rump, Frank J. (Hrsg.): *EPK*, GI-Arbeitskreis Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, 2002, S. 71–79 25

[van der Aalst und van Hee 2002] Aalst, Wil M. P. van der ; Hee, Kees van: *Workflow management : models, methods, and systems.* 1. Aufl. Cambridge, Mass. : MIT Press, 2002 2, 17

[van der Aalst und van Hee 2004] Aalst, Wil M. P. van der ; Hee, Kees van: *Workflow Management: Models, Methods, and Systems.* Cambridge, MA : The MIT Press, 2004 159, 163

[van der Aalst u. a. 2003] Aalst, Wil M. P. van der ; Hofstede, Arthur H. M. ter ; Kiepuszewski, Bartek ; Barros, Alistair P.: Workflow Patterns. In: *Distributed and Parallel Databases* 14 (2003), Nr. 1, S. 5–51 22

[van der Aalst u. a. 2005] Aalst, Wil M. P. van der ; Jørgensen, Jens B. ; Lassen, Kristian B.: Let's Go All the Way: From Requirements Via Colored Workflow Nets to a BPEL Implementation of a New Bank System. In: Meersman, Robert (Hrsg.) ; Tari, Zahir (Hrsg.) ; Hacid, Mohand-Said (Hrsg.) ; Mylopoulos, John (Hrsg.) ; Pernici, Barbara (Hrsg.) ; Babaoglu, Özalp (Hrsg.) ; Jacobsen, Hans-Arno (Hrsg.) ; Loyall, Joseph P. (Hrsg.) ; Kifer, Michael (Hrsg.) ; Spaccapietra, Stefano (Hrsg.): *OTM Conferences (1)* Bd. 3760. Berlin u.a. : Springer-Verlag, 2005, S. 22–39 9

[van der Aalst u. a. 1999] Aalst, Wil M. P. van der ; Moldt, Daniel ; Valk, Rüdiger ; Wienberg, Frank: Enacting Interorganizational Workflows Using Nets in Nets. In: Becker, Jörg (Hrsg.) ; Mühlen, Michael zur (Hrsg.) ; Rosemann, Michael (Hrsg.): *Proceedings of the 1999 Workflow Management Conference Workflow-based Applications, Münster, Nov. 9th 1999.* Münster : University of Münster, Department of Information Systems, Steinfurter Str. 109, 48149 Münster, 1999 (Working Paper Series of the Department of Information Systems), S. 117–136. – Working Paper No. 70 97

[Ahlrichs und Knuppertz 2006] Ahlrichs, Frank ; Knuppertz, Thilo: *Controlling von Geschäftsprozessen. Prozessorientierte Unternehmenssteuerung umsetzen.* Stuttgart : Schäffer-Poeschel, 2006 1

[Allweyer 2005] Allweyer, Thomas: *Geschäftsprozessmanagement : Strategie, Entwurf, Implementierung, Controlling*. Herdecke : W3L-Verlage, 2005 25, 312, 314

[Andrews u. a. 2005] Andrews, Tony ; Curbera, Francisco ; Dholakia, Hitesh ; Goland, Yaron ; Klein, Johannes ; Leymann, Frank ; Liu, Kevin ; Roller, Dieter ; Smith, Doug ; Thatte, Satish ; Trickovic, Ivana ; Weerawarana, Sanjiva: Business Process Execution Language for Web Services (BPEL4WS) Version 1.1 / IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems. 2005. – Forschungsbericht. WWW-Dokument†: http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/ 303

[Antoniou u. a. 2005] Antoniou, Grigoris ; Damásio, Carlos V. ; Grosof, Benjamin ; Horrocks, Ian ; Kifer, Michael ; Maluszynski, Jan ; Patel-Schneider, Peter F.: Combining Rules and Ontologies. A survey / Rewerse. 2005. – Forschungsbericht. Report I3-D3. WWW-Dokument†: http://rewerse.net/deliverables.html 10

[Baader u. a. 2002] Baader, Franz (Hrsg.) ; Calvanese, Diego (Hrsg.) ; McGuinness, Deborah (Hrsg.) ; Nardi, Daniele (Hrsg.) ; Patel-Schneider, Peter F. (Hrsg.): *Description Logic Handbook: Theory, Implementation and Applications*. Cambridge (Mass.) : Cambridge University Press, 2002 39

[Baader und Nipkow 1998] Baader, Franz ; Nipkow, Tobias: *Term Rewriting and All That*. Cambridge, UK : Cambridge University Press, 1998 339

[Bajaj u. a. 2006] Bajaj, Siddharth ; Box, Don ; Chappell, Dave ; Curbera, Francisco ; Daniels, Glen ; Hallam-Baker, Phillip ; Hondo, Maryann ; Kaler, Chris ; Langworthy, Dave ; Nadalin, Anthony ; Nagaratnam, Nataraj ; Prafullchandra, Hemma ; Riegen, Claus von ; Roth, Daniel ; Schlimmer, Jeffrey ; Sharp, Chris ; Shewchuk, John ; Vedamuthu, Asir ; Yalçýnalp Ümit ; Orchard, David: *Web Services Policy Framework (WSPolicy)*. March 2006. – gemeinsame Spezifikation von BEA, IBM, Microsoft, SAP, Verisign u.a. WWW-Dokument†: http://www-128.ibm.com/developerworks/library/specification/ws-polfram/ 33, 34

[Barkaoui u. a. 2007] Barkaoui, Kamel ; Ayed, Rahma B. ; Sbaï, Zohra: Workflow Soundness Verification based on Structure Theory of Petri Nets. In: *International Journal of Computing and Information Sciences - (IJCIS)* 5 (2007), April, Nr. 1. – WWW-Dokument†: http://ijcis.info/International Journal of Computing and Information Sciences_files/Vol4N1.htm 251, 273

[Battiston u. a. 2001] Battiston, Eugenio ; Chizzoni, A. ; Cindio, Fiorella de: CLOWN as a Testbed for Concurrent Object-Oriented Concepts. In: Agha, Gul

(Hrsg.) ; Cindio, Fiorella de (Hrsg.) ; Rozenberg, Grzegorz (Hrsg.): *Concurrent Object-Oriented Programming and Petri Nets* Bd. 2001. Berlin u.a. : Springer-Verlag, 2001, S. 131–163 96

[Becker u. a. 2005] Becker, Jörg ; Kugeler, Martin ; Rosemann, Michael: *Prozessmanagement – Ein Leitfaden zur prozessorientierten Organisationsgestaltung.* Berlin u.a. : Springer-Verlag, 2005 15

[Bernstein 1966] Bernstein, A. J.: Program Analysis for Parallel Processing. In: *IEEE Trans. on Electronic Computers* EC-15 (1966), Nr. 5, S. 757–762 139, 167

[Berthelot 1986] Berthelot, Gérard: Checking Properties of Nets Using Transformations. In: Rozenberg, Grzegorz (Hrsg.): *Advances in Petri Nets 1985* Bd. 222. Berlin u.a. : Springer-Verlag, 1986, S. 19–40 xvi, 204, 205, 206

[Berthelot u. a. 1980] Berthelot, Gérard ; Roucairol, Gérard ; Valk, Rüdiger: Reductions of Nets and Parallel Programs. In: Brauer, Wilfried (Hrsg.): *Advanced Course: Net Theory and Applications* Bd. 84. Berlin u.a. : Springer-Verlag, 1980, S. 277–290 204

[Bezem u. a. 2003] Bezem, Marc (Hrsg.) ; Klop, Jan W. (Hrsg.) ; Vrijer, Roel de (Hrsg.): *Term rewriting systems.* Cambridge, UK : Cambridge University Press, 2003 (Cambridge Tracts in Theoretical Computer Science 55) 339

[Biberstein u. a. 2001] Biberstein, O. ; Buchs, D. ; Guelfi, N.: Object-Oriented Nets with Algebraic Specifications: The CO-OPN/2 Formalism. In: Agha, G.A. (Hrsg.) ; Cindio, F., De (Hrsg.) ; Rozenberg, G. (Hrsg.): *Concurrent Object-Oriented Programming and Petri Nets, Advances in Petri Nets* Bd. 2001. Berlin u.a. : Springer-Verlag, 2001, S. 73–130 96

[Biske 2008] Biske, Todd: *SOA Governance.* Packt Publishing, 2008 37

[Booch u. a. 2006] Booch, Grady ; Rumbaugh, James ; Jacobson, Ivar: *Das UML Benutzerhandbuch.* München : Addison Wesley, 2006 2

[Booth u. a. 2004] Booth, David ; Haas, Hugo ; McCabe, Francis ; Newcomer, Eric ; Champion, Michael ; Ferris, Chris ; Orchard, David: Web Services Architecture / W3C. 2004. – Forschungsbericht. W3C Working Group Note 11 February 2004, WWW-Dokument[†]: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211 27, 28, 287

[Börger und Bolognesi 2003] Börger, Egon ; Bolognesi, Tommaso: Abstract State Machines. In: Börger, Egon (Hrsg.) ; Gargantini, Angelo (Hrsg.) ; Riccobene, Elvinia (Hrsg.): *Abstract State Machines – Advances in Theory and*

Applications 10th International Workshop, ASM 2003 – Taormina, Italy, March 2003 Bd. 2589. Berlin u.a. : Springer-Verlag, 2003, S. 22–32 44

[Börger und Huggins 1998] Börger, Egon ; Huggins, James K.: Abstract State Machines 1988-1998: Commented ASM Bibliography. In: *CoRR* cs.SE/9811014 (1998). – WWW-Dokument†: http://arxiv.org/abs/cs.SE/9811014 45

[Börger und Stärk 2003] Börger, Egon ; Stärk, Robert: *Abstract State Machines – A Method for High-Level System Design and Analysis*. Berlin u.a. : Springer-Verlag, 2003 4, 6, 9, 44, 91, 94

[BPMN 2008] Group, Object M.: *Business Process Modelling Notation*. 2008. – WWW-Dokument†: http://www.bpmn.org/ 2, 22, 25, 314

[BPMN 1.2 2008] Group, Object M.: *Business Process Modelling Notation (BPMN) 1.2*. 2008. – WWW-Dokument†: http://www.omg.org/spec/BPMN/1.2/ 25, 172

[Brachman und Schmolze 1985] Brachman, Ronald J. ; Schmolze, James G.: An Overview of the KL-ONE Knowledge Representation System. In: *Cognitive Science* 9 (1985), Nr. 2, S. 171–216 39

[Brandon 2002] Brandon, Daniel: CRUD matrices for detailed object oriented design. In: *Journal of Computing Sciences in Colleges* 18 (2002), Nr. 2, S. 306–322 5

[Brauer u. a. 1989] Brauer, Wilfried ; Gold, Robert ; Vogler, Walter: A survey of behaviour and equivalence preserving refinements of Petri nets. In: (Rozenberg, 1991), S. 1–46 204

[Bridgeland und Zahavi 2008] Bridgeland, David M. ; Zahavi, Ron: *Business Modeling: A Practical Guide to Realizing Business Value*. San Francisco, CA, USA : Publisher: Morgan Kaufmann, 2008 25

[de Bruijn und Heymans 2008] Bruijn, Jos de ; Heymans, Stijn: On the Relationship between Description Logic-based and F-Logic-based Ontologies. In: *Fundam. Inform.* 82 (2008), Nr. 3, S. 213–236 10

[de Bruijn u. a. 2004] Bruijn, Jos de ; Martín-Recuerda, Francisco ; Manov, Dimitar ; Ehrig, Marc: State-of-the-art Survey on Ontology Merging and Aligning V1 / SEKT. 2004 (D4.2.1). – Deliverable. WWW-Dokument†: http://www.debruijn.net/publications/sekt-d4.2.1-mediation-survey-final.pdf 10

[Busi 2002] Busi, Nadia: Analysis issues in Petri nets with inhibitor arcs. In: *Theor. Comput. Sci.* 275 (2002), Nr. 1-2, S. 127–177 140

[Bussler 2002] Bussler, Christoph: The Application of Workflow Technology in Semantic B2B Integration. In: *Distributed and Parallel Databases* 12 (2002), Nr. 2/3, S. 163–191 283

[Cabac 2010] Cabac, Lawrence: *Modeling Petri Net-Based Multi-Agent Applications*. Vogt-Kölln Str. 30, 22527 Hamburg, Universität Hamburg, Fakultät für Mathematik, Informatik und Naturwissenschaften, Department Informatik, Dissertation, Januar 2010. – eingereicht 2, 11, 292

[Cabac und Denz 2008] Cabac, Lawrence ; Denz, Nicolas: Net Components for the Integration of Process Mining into Agent-Oriented Software Engineering. In: (Jensen u. a., 2008), S. 86–103 341

[Cabac u. a. 2008] Cabac, Lawrence ; Dörges, Till ; Duvigneau, Michael ; Moldt, Daniel ; Reese, Christine ; Wester-Ebbinghaus, Matthias: Agent Models for Concurrent Software Systems. In: Bergmann, Ralph (Hrsg.) ; Lindemann, Gabriela (Hrsg.) ; Kirn, Stefan (Hrsg.) ; Pechoucek, Michal (Hrsg.): *MATES* Bd. 5244. Berlin u.a. : Springer-Verlag, 2008, S. 37–48 2

[Cabac und Moldt 2004] Cabac, Lawrence ; Moldt, Daniel: Formal Semantics for AUML Agent Interaction Protocol Diagrams. In: Odell, James (Hrsg.) ; Giorgini, Paolo (Hrsg.) ; Müller, Jörg P. (Hrsg.): *AOSE* Bd. 3382. Berlin u.a. : Springer-Verlag, 2004, S. 47–61 292

[Cabac u. a. 2003] Cabac, Lawrence ; Moldt, Daniel ; Rölke, Heiko: A Proposal for Structuring Petri Net-Based Agent Interaction Protocols. In: Aalst, Wil M. P. van der (Hrsg.) ; Best, Eike (Hrsg.): *ICATPN* Bd. 2679. Berlin u.a. : Springer-Verlag, 2003, S. 102–120 292, 303

[Calegari und Ciucci 2007] Calegari, Silvia ; Ciucci, Davide: Fuzzy Ontology, Fuzzy Description Logics and Fuzzy-OWL. In: Masulli, Francesco (Hrsg.) ; Mitra, Sushmita (Hrsg.) ; Pasi, Gabriella (Hrsg.): *WILF* Bd. 4578. Berlin u.a. : Springer-Verlag, 2007, S. 118–126 41

[Campos u. a. 1991] Campos, Javier ; Chiola, Giovanni ; Silva, Manuel: Properties and Performance Bounds for Closed Free Choice Synchronized Monoclass Queueing Networks. In: *IEEE Transactions on Automatic Control* 36 (1991), Nr. 12, S. 1368–1382 77

[Castilho u. a. 2004] Castilho, Marcos ; Guedes, André ; Lima, Tiago ; Marynowski, João ; Montaño, Razer ; Künzle, Luis ; Silva, Fabiano: A Petri net based representation for planning problems. In: *Booklet of the International Planning Competition (IPC)*, 2004, S. 27–29. – IPC was hosted at the International Conference on Automated Planning and Scheduling (ICAPS) 2004. WWW-Dokument†:

http://www.irit.fr/recherches/LILAC/Pers/Santos/publications/Castilho_etal-2004-IPC.pdf 341

[Chandrasekaran u. a. 1999] Chandrasekaran, B. ; Josephson, J. R. ; Benjamins, V. R.: What Are Ontologies, and Why Do We Need Them. In: *IEEE Intelligent Systems* 14 (1999), Nr. 1, S. 20–26 38

[Christensen und Hansen 1993] Christensen, Søren ; Hansen, Niels D.: Coloured Petri Nets Extended with Place Capacities, Test Arcs and Inhibitor Arcs. In: (Marsan, 1993), S. 186–205. – WWW-Dokument†: http://www.daimi.au.dk/CPnets/publ/full-papers/ChrHan1993.pdf 94, 140, 298

[Christensen u. a. 2001] Christensen, Søren ; Kristensen, Lars M. ; Mailund, Thomas: A Sweep-Line Method for State Space Exploration. In: (Margaria und Yi, 2001), S. 450–464 227

[Clarke Jr. u. a. 2001] Clarke Jr., Edmund M. ; Grumberg, Orna ; Peled, Doron A.: *Model Checking*. 3. Aufl. Cambridge, Mass. : The MIT Press, 2001 227

[Coad und Mayfield 1997] Coad, Peter ; Mayfield, Mark: *Java design : building better apps and applets*. Upper Saddle River, NJ : Prentice Hall, 1997 43

[Cook 1971] Cook, Stephen A.: The complexity of theorem-proving procedures. In: *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*. New York, NY, USA : ACM, 1971, S. 151–158 172, 174

[Coulondre und Libourel 2002] Coulondre, Stéphane ; Libourel, Thérèse: An integrated object-role oriented database model. In: *Data Knowl. Eng.* 42 (2002), Nr. 1, S. 113–141 42, 43

[Davis 2001] Davis, Robert E.: *Business process modelling with ARIS*. Berlin u.a. : Springer-Verlag, 2001 24

[Decker u. a. 2008] Decker, Gero ; Dijkman, Remco M. ; Dumas, Marlon ; García-Bañuelos, Luciano: Transforming BPMN Diagrams into YAWL Nets. In: Dumas, Marlon (Hrsg.) ; Reichert, Manfred (Hrsg.) ; Shan, Ming-Chien (Hrsg.): *BPM* Bd. 5240, Springer, 2008, S. 386–389. – ISBN 978-3-540-85757-0 340

[Dehnert und Rittgen 2001] Dehnert, Juliane ; Rittgen, Peter: Relaxed Soundness of Business Processes. In: Dittrich, Klaus R. (Hrsg.) ; Geppert, Andreas (Hrsg.) ; Norrie, Moira C. (Hrsg.): *CAiSE* Bd. 2068. Berlin u.a. : Springer-Verlag, 2001, S. 157–170 69, 70

[Desel 1996] Desel, Jörg: Basic Linear Algebraic Techniques for Place or Transition Nets. In: (Reisig und Rozenberg, 1998), S. 257–308 68

[Desel und Esparza 1993] Desel, Jörg ; Esparza, Javier: Reachability in Cyclic Extended Free-Choice Systems. In: *Theor. Comput. Sci.* 114 (1993), Nr. 1, S. 93–118. – WWW-Dokument†: http://www7.in.tum.de/um/bibdb/esparza/reach_in_cyclic.pdf 176

[Desel und Esparza 1995] Desel, Jörg ; Esparza, Javier: *Free Choice Petri nets*. Cambridge(UK) : Cambridge University Press, 1995 (Cambridge Tracts in Theoretical Computer Science 40) 47, 62, 67, 68, 74, 75, 76, 77, 78, 79, 80, 163, 164, 176, 180, 185, 197, 202, 204, 205, 217, 218, 219

[Desel und Frank 2005] Desel, Jörg (Hrsg.) ; Frank, Ulrich (Hrsg.): *Enterprise Modelling and Information Systems Architectures, Proceedings of the Workshop in Klagenfurt, October 24-25, 2005*. Bd. *75*. GI, 2005. (LNI) 368, 372

[Desel und Silva 1998] Desel, Jörg (Hrsg.) ; Silva, Manuel (Hrsg.): *Application and Theory of Petri Nets 1998, 19th International Conference, ICATPN '98, Lisbon, Portugal, June 22-26, 1998, Proceedings*. Bd. *1420*. Berlin u.a. : Springer-Verlag, 1998. (Lecture Notes in Computer Science) 360, 370

[Desel 1992] Desel, Jörg: *Struktur und Analyse von Free-Choice-Petrinetzen*. Wiesbaden : Deutscher Universitäts-Verlag, 1992 68, 77

[Dhurjati und Adve 2006] Dhurjati, Dinakar ; Adve, Vikram S.: Efficiently Detecting All Dangling Pointer Uses in Production Servers. In: *DSN*, IEEE Computer Society, 2006, S. 269–280 110

[Dijkman u. a. 2007] Dijkman, Remco M. ; Dumas, Marlon ; Ouyang, Chun: Formal Semantics and Analysis of BPMN Process Models / Queensland University of Technology. 2007. – Forschungsbericht. WWW-Dokument†: http://eprints.qut.edu.au/archive/00007115/01/7115.pdf 26

[Dijkman u. a. 2008] Dijkman, Remco M. ; Dumas, Marlon ; Ouyang, Chun: Semantics and analysis of business process models in BPMN. In: *Inf. Softw. Technol.* 50 (2008), Nr. 12, S. 1281–1294 26, 340

[van Dongen u. a. 2005] Dongen, Boudewijn F. van ; Aalst, Wil M. P. van der ; Verbeek, H. M. W.: Verification of EPCs: Using Reduction Rules and Petri Nets. In: Pastor, Oscar (Hrsg.) ; Cunha, Joao F. e (Hrsg.): *CAiSE* Bd. 3520. Berlin u.a. : Springer-Verlag, 2005, S. 372–386 25

[Dostal u. a. 2005] Dostal, Wolfgang ; Jeckle, Mario ; Melzer, Ingo ; Zengler, Barbara: *Service-orientierte Architekturen mit Web Services: Konzepte - Standards - Praxis*. München : Elsevier, Spektrum Akad. Verl., 2005 30

[Dumas und ter Hofstede 2001] Dumas, Marlon ; Hofstede, Arthur H. M. ter: UML Activity Diagrams as a Workflow Specification Language. In: Gogolla, Martin (Hrsg.) ; Kobryn, Cris (Hrsg.): *UML* Bd. 2185. Berlin u.a. : Springer-Verlag, 2001, S. 76–90 2

[Dunkel u. a. 2008] Dunkel, Jürgen ; Eberhart, Andreas ; Fischer, Stefan ; Kleiner, Carsten ; Koschel, Arne: *Systemarchitekturen für Verteilte Anwendungen – Client-Server, Multi-Tier, SOA, Event-Driven Architectures, P2P, Grid, Web 2.0.* München : Carl Hanser Verlag, 2008 36

[Duvigneau 2002] Duvigneau, Michael: *Bereitstellung einer Agentenplattform für petrinetzbasierte Agenten.* Vogt-Kölln Str. 30, 22527 Hamburg, Universität Hamburg, Fachbereich Informatik, Diplomarbeit, Dezember 2002 293, 324

[Duvigneau 2009] Duvigneau, Michael: *Konzeptionelle Modellierung von Plugin-Systemen mit Petrinetzen.* Vogt-Kölln Str. 30, 22527 Hamburg, Universität Hamburg, Fakultät für Mathematik, Informatik und Naturwissenschaften, Department Informatik, Dissertation, 2009 11, 341

[Ehrig und Mahr 1985] Ehrig, Hartmut ; Mahr, Bernd: *Equations and initial semantics; Fundamentals of algebraic specification.* Berlin u.a. : Springer-Verlag, 1985 (EATCS Monographs on Theoretical Computer Science) 50

[Ehrig u. a. 2001] Ehrig, Hartmut ; Mahr, Bernd ; Cornelius, Felix ; Grosse-Rhode, Martin ; Zeitz, Philip: *Mathematisch-strukturelle Grundlagen der Informatik.* 2. Aufl. Berlin u.a. : Springer-Verlag, 2001 47, 50, 56, 57

[Ehrig u. a. 2003] Ehrig, Hartmut (Hrsg.) ; Reisig, Wolfgang (Hrsg.) ; Rozenberg, Grzegorz (Hrsg.) ; Weber, Herbert (Hrsg.): *Petri Net Technology for Communication-Based Systems - Advances in Petri Nets.* Bd. *2472.* Berlin u.a. : Springer-Verlag, 2003. (Lecture Notes in Computer Science) 343, 351

[Emmerich und Gruhn 1991] Emmerich, Wolfgang ; Gruhn, Volker: FUNSOFT nets: a Petri-net based software process modeling language. In: *IWSSD '91: Proceedings of the 6th international workshop on Software specification and design.* Los Alamitos, CA, USA : IEEE Computer Society Press, 1991, S. 175–184 97

[Erl 2005] Erl, Thomas: *Service-oriented architecture: concepts, technology, and design.* Upper Saddle River, NJ u.a. : Prentice Hall, 2005 30

[Eshuis und Wieringa 2003] Eshuis, Rik ; Wieringa, Roel: Comparing Petri Net and Activity Diagram Variants for Workflow Modelling - A Quest for Reactive Petri Nets. In: (Ehrig u. a., 2003), S. 321–351 23

[Esparza 1998a] Esparza, Javier: Decidability and Complexity of Petri Net Problems - An Introduction. In: (Reisig und Rozenberg, 1998), S. 374–428 129, 175

[Esparza 1998b] Esparza, Javier: Reachability in Live and Safe Free-Choice Petri Nets is NP-Complete. In: *Theor. Comput. Sci.* 198 (1998), Nr. 1-2, S. 211–224 163

[Fettke 2008] Fettke, Peter: Business Process Modeling Notation. In: *WIRTSCHAFTSINFORMATIK* 50 (2008), Dezember, Nr. 6, S. 504–507 25

[Forgy 1982] Forgy, Charles L.: RETE:A fast algorithm for the many pattern / many objectpattern-match problems. In: *Artificial Intelligence* (1982), Nr. 19, S. 17–37 309

[Franceschinis und Wolf 2009] Franceschinis, Giuliana (Hrsg.) ; Wolf, Karsten (Hrsg.): *Applications and Theory of Petri Nets, 30th International Conference, PETRI NETS 2009, Paris, France, June 22-26, 2009. Proceedings.* Bd. *5606*. Berlin u.a. : Springer-Verlag, 2009. (Lecture Notes in Computer Science) 359, 366

[Freund und Götzer 2008] Freund, Jakob ; Götzer, Klaus: *Vom Geschäftsprozess zum Workflow. Ein Leitfaden für die Praxis.* München : Hanser Wirtschaft, 2008 22

[Gartner 2001] Smith, David ; Correia, Joanne ; Pring, Ben ; Plummer, Daryl: *The Future of Web Services: Dynamic Business Webs.* 2001. – Gartner's Internet Strategies Research Note M-13-3593, 9 April 2001; WWW-Dokument†: http://www.gartner.com/webletter/bowstreet/art4/art4.html 27

[Genrich 1986] Genrich, Hartmann J.: Predicate/Transition Nets. In: Brauer, Wilfried (Hrsg.) ; Reisig, Wolfgang (Hrsg.) ; Rozenberg, Grzegorz (Hrsg.): *Advances in Petri Nets* Bd. 254. Berlin u.a. : Springer-Verlag, 1986, S. 207–247 96

[Genrich und Lautenbach 1981] Genrich, Hartmann J. ; Lautenbach, Kurt: System Modelling with High-Level Petri Nets. In: *Theor. Comput. Sci.* 13 (1981), S. 109–136 96

[Ghallab u. a. 2004] Ghallab, Malik ; Nau, Dana ; Traverso, Paolo: *Automated planning : theory and practice.* Amsterdam : Elsevier, Kaufmann, 2004 341

[Gierhake 1998] Gierhake, Olaf: *Integriertes Geschäftsprozeßmanagement : effektive Organisationsgestaltung mit Workflow-, Workgroup- und Dokumentenmanagement-Systemen.* Braunschweig : Vieweg, 1998 16

[Girault und Valk 2003] Girault, Claude ; Valk, Rüdiger: *Petri Nets for Systems Engineering*. Berlin u.a. : Springer-Verlag, 2003 62

[Glässer 1997] Glässer, Uwe: Combining Abstract State Machines with Predicate Transition Nets. In: Pichler, F. (Hrsg.) ; Moreno-Diaz, R. (Hrsg.): *Computer Aided Systems Theory - EUROCAST'97 (Proceedings of the 6th International Workshop on Computer Aided Systems Theory, Las Palmas de Gran Canaria, Spain, Feb. 1997*. Berlin u.a. : Springer-Verlag, 1997 (LNCS 1333), S. 108–122 96

[Glässer u. a. 2004] Glässer, Uwe ; Gurevich, Yuri ; Veanes, Margus: Abstract Communication Model for Distributed Systems. In: *IEEE Trans. Softw. Eng.* 30 (2004), Nr. 7, S. 458–472 9

[Glausch und Reisig 2006] Glausch, Andreas ; Reisig, Wolfgang: Distributed Abstract State Machines and Their Expressive Power / Humboldt-Universität zu Berlin. jan 2006 (196). – Informatik-Berichte. WWW-Dokument†: http://edoc.hu-berlin.de/series/informatik-berichte/196/PDF/196.pdf 91

[Godefroid und Wolper 1994] Godefroid, Patrice ; Wolper, Pierre: A Partial Approach To Model Checking. In: *Information and Computation* 110 (1994), May, Nr. 2, S. 305–326 227

[Goguen 1991] Goguen, Joseph A.: Types as Theories. In: Reed, George M. (Hrsg.) ; Roscoe, Andrew W. (Hrsg.) ; Wachter, Ralph F. (Hrsg.): *Topology and Category Theory in Computer Science*, Oxford University, 1991, S. 357–390. – Proceedings of a Conference held at Oxford, June 1989 61

[Goguen 1999] Goguen, Joseph A.: Hidden Algebra for Software Engineering. In: Calude, Cristian (Hrsg.) ; Dinneen, Michael (Hrsg.): *Combinatorics, Computation and Logic, Proceedings, Conference on Discrete Mathematics and Theoretical Computer Science, (University of Auckland, New Zealand, 18-21 January 1999)* Bd. 21. Berlin u.a. : Springer-Verlag, 1999, S. 35–59. – WWW-Dokument†: http://cseweb.ucsd.edu/~goguen/pps/dmtcs.ps 98

[Goguen und Lin 2003] Goguen, Joseph A. ; Lin, Kai: Behavioral Verification of Distributed Concurrent Systems with BOBJ. In: *QSIC*, IEEE Computer Society, 2003, S. 216– 339

[Goguen und Malcolm 2000] Goguen, Joseph A. ; Malcolm, Grant: A hidden agenda. In: *Theor. Comput. Sci.* 245 (2000), Nr. 1, S. 55–101 98

[Goguen und Meseguer 1992] Goguen, Joseph A. ; Meseguer, José: Order-sorted algebra I: equational deduction for multiple inheritance, overloading, exceptions and partial operations. In: *Theor. Comput. Sci.* 105 (1992), Nr. 2, S. 217–273 47, 50, 57, 59, 60, 61, 62, 260

[Goguen und Rosu 2004] Goguen, Joseph A. ; Rosu, Grigore: Composing Hidden Information Modules over Inclusive Institutions. In: Owe, Olaf (Hrsg.) ; Krogdahl, Stein (Hrsg.) ; Lyche, Tom (Hrsg.): *Essays in Memory of Ole-Johan Dahl* Bd. 2635. Berlin u.a. : Springer-Verlag, 2004, S. 96–123 98

[Griffel 2001] Griffel, Frank: *Verteilte Anwendungssysteme als Komposition-klassifizierter Softwarebausteine*, Universität Hamburg, Fachbereich Informatik, Verteilte Systeme und Informationssysteme, Dissertation, 7 2001 341

[Gross und Koch 2007] Gross, Tom ; Koch, Michael: *Computer-Supported Cooperative Work*. München u.a. : Oldenbourg, 2007 17

[Große-Rhode 2004] Grosse-Rhode, Martin: *Semantic Integration of Heterogeneous Software Specifications*. Berlin u.a. : Springer-Verlag, 2004 (Monographs in Theoretical Computer Science) 61, 91

[Gruber 1993] Gruber, Tom R.: A translation approach to portable ontology specifications. In: *Knowledge Acquisition* 5 (1993), Nr. 2 38

[Guarino 1998] Guarino, Nicola: Formal Ontology and Information Systems. In: Guarino, Nicola (Hrsg.): *Formal Ontology in Information Systems. Proceedings of FOIS '98*. Amsterdam : IOS-Press, 1998, S. 3–15. – WWW-Dokument†: http://www.loa-cnr.it/Papers/FOIS98.pdf 38

[Guarino und Welty 2000] Guarino, Nicola ; Welty, Christopher A.: Ontological Analysis of Taxonomic Relationships. In: *ER*, 2000, S. 210–224 42

[Gurevich 1988] Gurevich, Yuri: *Algorithms in the World of Bounded Resources*. S. 407–416. In: Herken, R. (Hrsg.): *The Universal Turing Machine – A Half-Century Story*, Oxford University Press, 1988 44, 90

[Gurevich 1991] Gurevich, Yuri: Evolving Algebras: a tutorial introduction. In: *Bulletin of the European Association for Theoretic Computer Science* 43 (1991), S. 264–284 4, 44, 90

[Gurevich 1995] Gurevich, Yuri: *Evolving Algebras 1993: Lipari Guide*. S. 9–36. In: Börger, Egon (Hrsg.): *Specification and Validation Methods*. Oxford / UK : Oxford University Press, 1995 90

[Haas 2002] Haas, Peter J.: *Stochastic Petri Nets: Modelling, Stability, Simulation*. Berlin u.a. : Springer-Verlag, 2002 (Springer Series in Operations Research) 341

[Hack 1974] Hack, Michael: The recursive equivalence of the reachability problem and the liveness problem for Petri nets and vector addition systems / Massachusetts Institute of Technology – Laboratory for Computer Science. 1974. – Forschungsbericht. WWW-Dokument†: http://csg.csail.mit.edu/pubs/memos/Memo-107/Memo-107.pdf 204

[Hapner u. a. 2002] Hapner, Mark ; 22:50, Rich 2. ; Sharma, Rahul ; Fialli, Joseph ; Stout, Kate: *Java Message Service (JMS)*. March 2002. – WWW-Dokument†: http://java.sun.com/products/jms/ 33

[van Hee u. a. 2005] Hee, Kees M. van ; Serebrenik, Alexander ; Sidorova, Natalia ; Voorhoeve, Marc: Soundness of Resource-Constrained Workflow Nets. In: Ciardo, Gianfranco (Hrsg.) ; Darondeau, Philippe (Hrsg.): *ICATPN* Bd. 3536. Berlin u.a. : Springer-Verlag, 2005, S. 250–267 339

[van Hee u. a. 2004] Hee, Kees M. van ; Sidorova, Natalia ; Voorhoeve, Marc: Generalised Soundness of Workflow Nets Is Decidable. In: Cortadella, Jordi (Hrsg.) ; Reisig, Wolfgang (Hrsg.): *ICATPN* Bd. 3099. Berlin u.a. : Springer-Verlag, 2004, S. 197–215 129

[Hollingsworth 1995] Hollingsworth, David: The Workflow Reference Model / Workflow Management Coalition. Jan 1995. – Forschungsbericht. Document Number TC00-1003, Document Status - Issue 1.1; WWW-Dokument†: http://www.wfmc.org/standards/docs/tc003v11.pdf 16, 18, 19

[Hondo u. a. 2008] Hondo, Maryann ; Portier, Bertrand ; Potepan, Franco: SOA Policy Management. In: *IBM Redpaper (ibm.com/redbooks)* (2008). – WWW-Dokument†: http://www.redbooks.ibm.com/abstracts/redp4463.html?Open 37

[Hotho u. a. 2003] Hotho, Andreas ; Maedche, Alexander ; Staab, Steffen ; Zacharias, Valentin: On Knowledgeable Unsupervised Text Mining. In: J. Franke, I. R. (Hrsg.): *Text Mining. Theoretical Aspects and Applications*. Berlin u.a. : Springer-Verlag, 2003, S. 131–152. – WWW-Dokument†: http://www.aifb.uni-karlsruhe.de/WBS/aho/pub/txt_mining_ws_2002.pdf 39, 41

[Huth und Wieland 2008] Huth, Stefan ; Wieland, Thomas: *Geschäftsprozessmodellierung mittels Software-Services auf Basis der EPK*. S. 61–76. In: *Service-orientierte Architekturen*. Berlin u.a. : Springer-Verlag, 2008 2

[INA 2003] Starke, Peter H. ; Roch, Stephan: *INA – Integrated Net Analyzer*. 2003. – Dokumentation als WWW-Dokument†: http://www2.informatik.hu-berlin.de/ starke/ina.html 262

[Ingalls 1981] Ingalls, Daniel H. H.: Design Principles Behind Smalltalk. In: *Byte* (1981), August. – WWW-Dokument†: http://carlstrom.com/stanford/cs242/readings/10.pdf 9

[ISO 15909 2002] ISO/IEC: *High-level Petri Nets - Concepts, Definitions and Graphical Notation.* 2002. – Final Draft International Standard ISO/IEC 15909-1, Version 4.7.3, May 10, 2002, WWW-Dokument†: http://www.petrinets.info/docs/pnstd-4.7.4.pdf 138

[ISO 16262 2002] ISO/IEC: *International Standard ISO/IEC16262:2002 – ECMAScript Language Specification.* 2002. – WWW-Dokument†: http://www.iso.org/iso/catalogue_detail.htm?csnumber=33835 oder WWW-Dokument†: http://webstore.iec.ch/preview/info_isoiec16262{ed2.0}en.pdf 9

[Jablonski u. a. 1997] Jablonski, Stefan (Hrsg.) ; Böhm, Markus (Hrsg.) ; Schulze, Wolfgang (Hrsg.): *Workflow-Management : Entwicklung von Anwendungen und Systemen; Facetten einer neuen Technologie.* Heidelberg : dpunkt-Verlag, 1997 16, 20

[Jacob 2002] Jacob, Thomas: *Implementierung einer sicheren und rollenbasierten Workflowmanagement-Komponente für ein Petrinetzwerkzeug.* Vogt-Kölln Str. 30, 22527 Hamburg, Universität Hamburg, Fachbereich Informatik, Diplomarbeit, 2002 23

[Jacob u. a. 2001] Jacob, Thomas ; Kummer, Olaf ; Moldt, Daniel: Persistent Petri Net Execution. In: *Petri Net Newsletter* (2001), Oktober, Nr. 61, S. 18–26 23

[Jakob und Weiß 2004] Jakob, Ralf ; Weiss, Gerhard: *Agentenorientierte Softwareentwicklung: Methoden und Tools.* Springer-Verlag, 2004 291

[Jänich 1998] Jänich, Klaus: *Lineare Algebra.* Berlin u.a. : Springer-Verlag, 1998 179, 185

[Jensen 1981] Jensen, Kurt: Coloured Petri Nets and the Invariant-Method. In: *Theor. Comput. Sci.* 14 (1981), S. 317–336 2

[Jensen 1983] Jensen, Kurt: High Level Petri Nets. In: Pagoni, A. (Hrsg.) ; Rozenberg, G. (Hrsg.): *Applications and Theory of Petri Nets.* Berlin u.a. : Springer-Verlag, 1983 (Informatik Fachberichte 66), S. 166–180 71, 95

[Jensen 1992] Jensen, Kurt: *EATCS monographs on theoretical computer science.* Bd. 1: *Coloured Petri nets, Basic Methods, Analysis Methods and Practical Use.* Berlin u.a. : Springer-Verlag, 1992 71

[Jensen 1994] Jensen, Kurt: An introduction to the theoretical aspects of Coloured Petri Nets. In: Bakker, J. W. de (Hrsg.) ; Roever, W.-P. de (Hrsg.) ; Rozenberg, G (Hrsg.): *A Decade of Concurrency Reflections and Perspectives*. Berlin u.a. : Springer-Verlag, 1994 (Lecture Notes in Computer Science 803), S. 230–272 2

[Jensen u. a. 2008] Jensen, Kurt (Hrsg.) ; Aalst, Wil M. P. van der (Hrsg.) ; Billington, Jonathan (Hrsg.): *Transactions on Petri Nets and Other Models of Concurrency I*. Bd. *5100*. Berlin u.a. : Springer-Verlag, 2008. (Lecture Notes in Computer Science) 348, 359

[Jensen u. a. 2009] Jensen, Kurt (Hrsg.) ; Billington, Jonathan (Hrsg.) ; Koutny, Maciej (Hrsg.): *Transactions on Petri Nets and Other Models of Concurrency III*. Bd. *5800*. Berlin u.a. : Springer-Verlag, 2009. (Lecture Notes in Computer Science) 361, 367

[Jensen und Kristensen 2009] Jensen, Kurt ; Kristensen, Lars M.: *Coloured Petri Nets*. Berlin u.a. : Springer-Verlag, 2009 71, 95

[Jess 2008] *Jess, the rule engine for the Java Platform*. 2008. – Dokumentation und Download. WWW-Dokument†: http://www.jessrules.com/ ist nicht von jedem ISP erreichbar, alternativ: WWW-Dokument†: http://herzberg.ca.sandia.gov/ 309

[Jessen und Valk 1987] Jessen, Eike ; Valk, Rüdiger: *Rechensysteme; Grundlagen der Modellbildung*. Berlin u.a. : Springer-Verlag, 1987 17, 47, 62, 95, 137, 148, 341

[Jhala und Majumdar 2009] Jhala, Ranjit ; Majumdar, Rupak: Software model checking. In: *ACM Comput. Surv.* 41 (2009), Nr. 4, S. 1–54 225, 263, 339

[Jodlowski u. a. 2003] Jodlowski, Andrzej ; Habela, Piotr ; Plodzien, Jacek ; Subieta, Kazimierz: Extending OO Metamodels towards Dynamic Object Roles. In: Meersman, Robert (Hrsg.) ; Tari, Zahir (Hrsg.) ; Schmidt, Douglas C. (Hrsg.): *CoopIS/DOA/ODBASE* Bd. 2888. Berlin u.a. : Springer-Verlag, 2003, S. 1032–1047 43

[Johnsonbaugh und Murata 1981] Johnsonbaugh, Richard ; Murata, Tadao: Additional Methods for Reduction and Expansion of Marked Graphs. In: *IEEE Transaction on Circuit and Systems* CAS-28 (1981), Oktober, Nr. 10, S. 1009–1014 204

[Jones und Morris 2005] Jones, Steve ; Morris, Mike: *A Methodology for Service Architectures*. 2005. – OASIS, WWW-Dokument†: http://www.oasis-open.org/committees/download.php/15071/ 290

[Jørgensen u. a. 2008] Jørgensen, Jens B. ; Lassen, Kristian B. ; Aalst, Wil M. P. van der: From task descriptions via colored Petri nets towards an implementation of a new electronic patient record workflow system. In: *STTT* 10 (2008), Nr. 1, S. 15–28 9

[Keller u. a. 1992] Keller, Gerhard ; Nüttgens, Markus ; Scheer, August-Wilhelm: Semantische Prozeßmodellierung auf der Grundlage Ëreignisgesteuerter Prozeßketten (EPK)" / Institut für Wirtschaftsinformatik Universität Saarbrücken. 1992 (Heft 89). – Forschungsbericht. WWW-Dokument†: http://www.iwi.uni-sb.de/Download/iwihefte/heft89.pdf 2, 22, 24

[Kemper und Eickler 1999] Kemper, Alfons ; Eickler, Andre: *Datenbanksysteme*. 3. Aufl. München : Oldenbourg, 1999 110, 122

[Kemper 1993] Kemper, Peter: Linear Time Algorithm to Find a Minimal Deadlock in a Strongly Connected Free-Choice Net. In: (Marsan, 1993), S. 319–338 163, 186, 193, 195, 196, 217

[Kemper 1994] Kemper, Peter: O($|P||T|$)-algorithm to compute a cover of S-components in EFC-nets / Fachbereich Informatik der Universität Dortmund. 1994 (543). – Forschungsbericht. WWW-Dokument†: http://ls4-www.informatik.uni-dortmund.de/QM/MA/pk/publication_ps_files/FB_543.ps 216, 217

[Kharbili und Stein 2008] Kharbili, Marwane E. ; Stein, Sebastian: Policy-Based Semantic Compliance Checking for Business Process Management. In: Loos, Peter (Hrsg.) ; Nüttgens, Markus (Hrsg.) ; Turowski, Klaus (Hrsg.) ; Werth, Dirk (Hrsg.): *MobIS Workshops* Bd. 420, CEUR-WS.org, 2008, S. 178–192 291

[Kiehn 1988] Kiehn, Astrid: Petri Net systems and their closure properties. In: Rozenberg, Grzegorz (Hrsg.): *European Workshop on Applications and Theory in Petri Nets* Bd. 424. Berlin u.a. : Springer-Verlag, 1988, S. 306–328 250

[Kifer u. a. 1995] Kifer, Michael ; Lausen, Georg ; Wu, James: Logical Foundations of Object-Oriented and Frame-Based Languages. In: *J. ACM* 42 (1995), Nr. 4, S. 741–843 39

[Kim und Lochovsky 1989] Kim, Won (Hrsg.) ; Lochovsky, Frederick H. (Hrsg.): *Object-Oriented Concepts, Databases, and Applications*. ACM Press and Addison-Wesley, 1989 365, 369

[Koehler u. a. 2008] Koehler, Jana ; Hauser, Rainer ; Küster, Jochen ; Ryndina, Ksenia ; Vanhatalo, Jussi ; Wahler, Michael: The Role of Visual Modeling and Model Transformations in Business-driven Development. In: *Electron. Notes Theor. Comput. Sci.* 211 (2008), S. 5–15 290

[Köhler u. a. 2001] Köhler, Michael ; Moldt, Daniel ; Rölke, Heiko: Modelling the Structure and Behaviour of Petri Net Agents. In: Colom, J.M. (Hrsg.) ; Koutny, M. (Hrsg.): *Proceedings of the 22nd Conference on Application and Theory of Petri Nets* Bd. 2075. Berlin u.a. : Springer-Verlag, 2001, S. 224–241 295

[Köhler und Ortmann 2005] Köhler, Michael ; Ortmann, Jan: Formal Aspects for Semantic Service Modeling Based on High-Level Petri Nets. In: *CIMCA/IAWTIC*, IEEE Computer Society, 2005, S. 107–112 85, 91, 156, 308

[Köhler-Bußmeier 2009a] Köhler-Bussmeier, Michael: *Formale Agentenorganisationen*. Kap. 10, S. 377–414. Siehe (von Lüde u. a., 2009) 311

[Köhler-Bußmeier 2009b] Köhler-Bussmeier, Michael: Hornets: Nets within Nets Combined with Net Algebra. In: (Franceschinis und Wolf, 2009), S. 243–262 338

[Köhler-Bußmeier 2009c] Köhler-Bussmeier, Michael: *Interaktion verteilter Systeme*. Kap. 8, S. 285–336. Siehe (von Lüde u. a., 2009) 274, 276, 277

[Koniewski u. a. 2006] Koniewski, Ryszard ; Dzielinski, Andrzej ; Amborski, Krzysztof: Use of Petri Nets and Business Processes Management Notation in Modelling and Simulation of Multimodal Logistics Chains. In: *20th European Conference on Modelling and Simulation ECMS 2006*, 2006. – WWW-Dokument†: http://www.scs-europe.net/services/ecms2006/ecms2006%20pdf/84-cs.pdf 26

[Kovalyov 1996] Kovalyov, Andrei: An O(|S|×|T|)- Algorithm to Verify if a Net is Regular. In: Billington, Jonathan (Hrsg.) ; Reisig, Wolfgang (Hrsg.): *Application and Theory of Petri Nets* Bd. 1091. Berlin u.a. : Springer-Verlag, 1996, S. 366–379 176

[Krafzig u. a. 2005] Krafzig, Dirk ; Banke, Karl ; Slama, Dirk: *Enterprise SOA*. 4. Aufl. Prentice Hall, 2005 30, 33, 34, 35

[Kreplin 2006] Kreplin, Klaus-Dieter: *Konkordanz englischer und deutscher Begriffe des Workflow Management*. 2006. – WWW-Dokument†: http://www.wfmc.org/Download-document/Terminology-and-Glossary-German.html 14

[Kriha und Schmitz 2009] Kriha, Walter ; Schmitz, Roland: *Sichere Systeme: Konzepte, Architekturen und Frameworks*. Berlin u.a. : Springer-Verlag, 2009 5

[Kristensen und Jensen 2008] Kristensen, Lars M. ; Jensen, Kurt: Teaching Modelling and Validation of Concurrent Systems Using Coloured Petri Nets. In: (Jensen u. a., 2008), S. 19–34 342

[Kristensen und Valmari 1998] Kristensen, Lars M. ; Valmari, Antti: Finding Stubborn Sets of Coloured Petri Nets Without Unfolding. In: (Desel und Silva, 1998), S. 104–123 226

[Kristensen und Valmari 2000] Kristensen, Lars M. ; Valmari, Antti: Improved Question-Guided Stubborn Set Methods for State Properties. In: Nielsen, Mogens (Hrsg.) ; Simpson, Dan (Hrsg.): *ICATPN* Bd. 1825, Springer-Verlag, 2000, S. 282–302 227

[Kummer 2002] Kummer, Olaf: *Referenznetze*. Vogt-Kölln Str. 30, 22527 Hamburg, Universität Hamburg, Fachbereich Informatik, Dissertation, 2002 2, 90, 97, 129, 338

[Kwong 1977] Kwong, Y. S.: On Reduction of Asynchronous Systems. In: *Theor. Comput. Sci.* 5 (1977), Nr. 1, S. 25–50 204

[Lakos 1996] Lakos, Charles: The Consistent Use of Names and Polymorphism in the Definition of Object Petri Nets. In: Billington, Jonathan (Hrsg.) ; Reisig, Wolfgang (Hrsg.): *Application and Theory of Petri Nets* Bd. 1091. Berlin u.a. : Springer-Verlag, 1996, S. 380–399 96

[Langner u. a. 1998] Langner, Peter ; Schneider, Christoph ; Wehler, Joachim: Petri Net Based Certification of Event-Driven Process Chains. In: (Desel und Silva, 1998), S. 286–305 25

[Lautenbach 2002] Lautenbach, Kurt: Reproducibility of the Empty Marking. In: Esparza, Javier (Hrsg.) ; Lakos, Charles (Hrsg.): *ICATPN* Bd. 2360. Berlin u.a. : Springer-Verlag, 2002, S. 237–253 67

[LaValle 2006] LaValle, Steven M.: *Planning Algorithms*. Cambridge, USA : Cambridge University Press, 2006 341

[Lehmann 2003] Lehmann, Kolja: *Analyse und Bewertung von Agentenprotokollen auf Basis von Petrinetzen*. Vogt-Kölln Str. 30, 22527 Hamburg, Universität Hamburg, Fachbereich Informatik, Diplomarbeit, Oktober 2003 274

[Loeckx u. a. 1997] Loeckx, Jacques ; Ehrich, Hans-Dieter ; Wolf, Markus: *Specification of abstract data types*. New York, NY, USA : John Wiley & Sons, Inc., 1997 50

[Luckham 2008] Luckham, David: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. In: Bassiliades, Nick (Hrsg.) ; Governatori, Guido (Hrsg.) ; Paschke, Adrian (Hrsg.): *RuleML* Bd. 5321. Berlin u.a. : Springer-Verlag, 2008, S. 3 36, 37

[von Lüde u. a. 2003] Lüde, Rolf von (Hrsg.) ; Moldt, Daniel (Hrsg.) ; Valk, Rüdiger (Hrsg.): *Reihe: Wirtschaft – Arbeit – Technik*. Bd. 2: *Sozionik: Modellierung soziologischer Theorie*. Berlin u.a. : Lit-Verlag, 2003 294, 342

[von Lüde u. a. 2009] Lüde, Rolf von (Hrsg.) ; Moldt, Daniel (Hrsg.) ; Valk, Rüdiger (Hrsg.): *Selbstorganisation und Governance in künstlichen und sozialen Systemen. Abschlusbericht des DFG Projektes „Emergenz in dynamishen Prozessen - Dirigismus und symbolische Politik“ im Rahmen des DFG-Schwerpunktprogrammes „Sozionik“*. Berlin u.a. : Lit Verlag, 2009 294, 359

[Ly u. a. 2008] Ly, Linh T. ; Göser, Kevin ; Rinderle-Ma, Stefanie ; Dadam, Peter: Compliance of Semantic Constraints – A Requirements Analysis for Process Management Systems. In: Sadiq, S. (Hrsg.) ; Indulska, M. (Hrsg.) ; Muehlen, M. zur (Hrsg.) ; Franch, X. (Hrsg.) ; Hunt, E. (Hrsg.) ; Coletta, R. (Hrsg.): *Proc. 1st Int'l Workshop on Governance, Risk and Compliance - Applications in Information Systems (GRCIS'08), Montpellier, France*, CEUR-WS.org, 2008, S. 31–45. – WWW-Dokument†: http://dbis.eprints.uni-ulm.de/156/291

[MacKenzie u. a. 2006] MacKenzie, C. M. ; Laskey, Ken ; McCabe, Francis ; Brown, Peter ; Metz, Rebekah: Reference Model for Service Oriented Architecture / OASIS. 2006. – Forschungsbericht. Committee Draft 1.0, 7 February 2006 WWW-Dokument†: http://www.oasis-open.org/committees/download.php/16587/wd-soa-rm-cd1ED.pdf 27

[Mäkelä 2000] Mäkelä, Marko: Maria: Modular Reachability Analyzer for High-Level Petri Nets. In: *The 5th Workshop on Discrete Event Systems (WODES 2000)*. Ghent, Belgium : Kluwer Academic Publishers, Boston, MA, USA, August 2000, S. 477–478 262

[Mans u. a. 2009] Mans, Ronny S. ; Russell, Nick C. ; Aalst, Wil M. P. van der ; Bakker, Piet J. M. ; Moleman, Arnold J. ; Bisgaard Lassen, Kristian ; Bæk Jørgensen, Jens: From Requirements via Colored Workflow Nets to an Implementation in Several Workflow Systems. In: (Jensen u. a., 2009), S. 25–49 10

[Mans u. a. 2008] Mans, Ronny S. ; Russell, Nick C. ; Aalst, Wil M. P. van der ; Moleman, Arnold J. ; Bakker, Piet J. M.: Augmenting a Workflow Management System with Planning Facilities using Colored Petri Nets. In: Jensen, Kurt (Hrsg.): *Proceedings of the Nineth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2008)* Bd. 588. Aarhus, Denmark : University of Aarhus, 2008, S. 143–162. – WWW-Dokument†: http://www.daimi.au.dk/CPnets/workshop08/cpn/papers/Paper09.pdf 9

[Margaria und Yi 2001] Margaria, Tiziana (Hrsg.) ; Yi, Wang (Hrsg.): *Tools and Algorithms for the Construction and Analysis of Systems, 7th International Conference, TACAS 2001 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2001 Genova, Italy, April 2-6, 2001, Proceedings*. Bd. *2031*. Berlin u.a. : Springer-Verlag, 2001. (Lecture Notes in Computer Science) 349, 373

[Marsan 1993] Marsan, Marco A. (Hrsg.): *Application and Theory of Petri Nets 1993, 14th International Conference, Chicago, Illinois, USA, June 21-25, 1993, Proceedings*. Bd. *691*. Berlin u.a. : Springer-Verlag, 1993. (Lecture Notes in Computer Science) 349, 358

[Martens 2005] Martens, Axel: Analyzing Web Service Based Business Processes. In: Cerioli, Maura (Hrsg.): *FASE* Bd. 3442. Berlin u.a. : Springer-Verlag, 2005, S. 19–33 308

[Martin u. a. 2004a] Martin, David ; Burstein, Mark ; Hobbs, Jerry ; Lassila, Ora ; McDermott, Drew ; McIlraith, Sheila ; Narayanan, Srini ; Paolucci, Massimo ; Parsia, Bijan ; Payne, Terry ; Sirin, Evren ; Srinivasan, Naveen ; Sycara, Katia: *OWL-S: Semantic Markup for Web Services*. 2004. – W3C Member Submission 22 November 2004: WWW-Dokument†: http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/ 34, 304

[Martin u. a. 2004b] Martin, David L. ; Paolucci, Massimo ; McIlraith, Sheila A. ; Burstein, Mark H. ; McDermott, Drew V. ; McGuinness, Deborah L. ; Parsia, Bijan ; Payne, Terry R. ; Sabou, Marta ; Solanki, Monika ; Srinivasan, Naveen ; Sycara, Katia P.: Bringing Semantics to Web Services: The OWL-S Approach. In: Cardoso, Jorge (Hrsg.) ; Sheth, Amit P. (Hrsg.): *SWSWPC* Bd. 3387. Berlin u.a. : Springer-Verlag, 2004, S. 26–42 34

[Martinez und Silva 1981] Martinez, J. ; Silva, Manuel: A Simple and Fast Algorithm to Obtain All Invariants of a Generalized Petri Net. In: Girault, Claude (Hrsg.) ; Reisig, Wolfgang (Hrsg.): *Selected Papers from the First and the Second European Workshop on Application and Theory of Petri Nets* Bd. 52. Berlin u.a. : Springer-Verlag, 1981, S. 301–310 164

[Massuthe u. a. 2005] Massuthe, Peter ; Reisig, Wolfgang ; Schmidt, Karsten: An Operating Guideline Approach to the SOA. In: *Annals of Mathematics, Computing & Teleinformatics* 1 (2005), Nr. 3, S. 35–43. – WWW-Dokument†: http://www.informatik.hu-berlin.de/top/download/publications/MassutheRS2005_amct.pdf 274, 276

[Maude 2009] Laboratory, SRI Computer S.: *Maude 2.4 Manual and Examples*. 2009. – WWW-Dokument†: http://maude.cs.uiuc.edu/maude2-manual/ 262

[Mayr 1984] Mayr, E. W.: An Algorithm for the General Petri Net Reachability Problem. In: *SIAM J. Comput.* 13 (1984), August, Nr. 3, S. 441–460 129

[McGovern u. a. 2006] McGovern, James (Hrsg.) ; Sims, Oliver (Hrsg.) ; Jain, Ashish (Hrsg.) ; Little, Mark (Hrsg.): *Enterprise service oriented architectures: concepts, challenges, recommendations*. Berlin u.a. : Springer-Verlag, 2006 37

[de Medeiros u. a. 2007] Medeiros, Ana Karla A. de ; Pedrinaci, Carlos ; Aalst, Wil M. P. van der ; Domingue, John ; Song, Minseok ; Rozinat, Anne ; Norton, Barry ; Cabral, Liliana: An Outlook on Semantic Business Process Mining and Monitoring. In: Meersman, Robert (Hrsg.) ; Tari, Zahir (Hrsg.) ; Herrero, Pilar (Hrsg.): *OTM Workshops (2)* Bd. 4806, Springer, 2007, S. 1244–1255 341

[Mendelzon u. a. 1994] Mendelzon, Alberto O. ; Milo, Tova ; Waller, Emmanuel: Object Migration. In: *PODS*, ACM Press, 1994, S. 232–242 42

[Merlin und Faber 1976] Merlin, P. M. ; Faber, D. J.: Recoverability of communication protocols. In: *IEEE Trans. Commun. COM* 24 (1976), Nr. 9, S. 1036–1043 149

[Merz 1999] Merz, Michael: *Elektronische Dienstemärkte: Modelle und Mechanismen des electronic commerce*. Berlin u.a. : Springer-Verlag, 1999 14, 27, 28, 29

[Meseguer und Montanari 1990] Meseguer, José ; Montanari, Ugo: Petri Nets Are Monoids. In: *Information and Computation* 88 (1990), S. 105–155 262

[Meyer 2001] Meyer, Carl D.: *Matrix Analysis and Applied Linear Algebra*. Philadelphia : SIAM: Society for Industrial and Applied Mathematics, 2001 179

[Mitchell u. a. 2007] Mitchell, Brice ; Kristensen, Lars M. ; Zhang, Lin: Formal specification and state space analysis of an operational planning process. In: *STTT* 9 (2007), Nr. 3-4, S. 255–267 227

[Mitra 2005] Mitra, Tilak: *Business-driven development*. 2005. – IBM developerWorks, WWW-Dokument[†]: http://www.ibm.com/developerworks/webservices/library/ws-bdd/index.html 290

[Moldt 1996] Moldt, Daniel: *Höhere Petrinetze als Grundlage für Systemspezifikationen*. Vogt-Kölln Str. 30, 22527 Hamburg, Universität Hamburg, Fachbereich Informatik, Dissertation, August 1996 96, 263

[Moldt 2006] Moldt, Daniel: PAOSE: A Way to Develop Distributed Software Systems Based on Petri Nets and Agents. In: Barjis, Joseph (Hrsg.) ; Ultes-Nitsche, Ulrich (Hrsg.) ; Augusto, Juan C. (Hrsg.): *MSVVEIS*, INSTICC Press, 2006 2, 292

[Moldt u. a. 2004a] Moldt, Daniel ; Nyssen, Peter ; Ortmann, Jan: Modellierung soziologischer Theorien – Ein Petrinetz basierter Ansatz. In: *Modellierung 2004, Workshop W1: "Intelligente Lehr-/Lernsysteme: Modellierung als Schlüsselkonzept in intelligenten Lehr-/Lernsystemen", Marburg*, 2004 342

[Moldt u. a. 2004b] Moldt, Daniel ; Offermann, Sven ; Ortmann, Jan: A Proposal for Petri Net Based Web Service Application Modeling. In: Koch, Nora (Hrsg.) ; Fraternali, Piero (Hrsg.) ; Wirsing, Martin (Hrsg.): *ICWE* Bd. 3140. Berlin u.a. : Springer-Verlag, 2004, S. 93–97 10, 295

[Moldt u. a. 2005] Moldt, Daniel ; Offermann, Sven ; Ortmann, Jan: A Petri Net-Based Architecture for Web Services. In: *The Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE'2005), Utrecht, Niederlande*, 2005, S. 33–40. – Ausgetragen zusammen mit der AAMAS'2005 (Autonomous Agents and Multi Agent Systems), Juli 2005. 11, 295, 296, 303

[Moldt und Ortmann 2004a] Moldt, Daniel ; Ortmann, Jan: A Conceptual and Practical Framework for Web-Based Processes in Multi-Agent Systems. In: *AAMAS*, IEEE Computer Society, 2004, S. 1464–1465. – ISBN 1-58113-864-4 340

[Moldt und Ortmann 2004b] Moldt, Daniel ; Ortmann, Jan: DaGen: A Tool for Automatic Translation from DAML-S to High-level Petri Nets. In: Wermelinger, Michel (Hrsg.) ; Margaria-Steffen, Tiziana (Hrsg.): *Fundamental Approaches to Software Engineering, 7th International Conference, FASE 2004* Bd. 2984. Berlin u.a. : Springer-Verlag, 2004, S. 209–213 303

[Moldt und Ortmann 2009] Moldt, Daniel ; Ortmann, Jan: Agent-Oriented Petri Net Based Service Composition. In: *DigiBiz 2009 Workshop W1: Workshop on Technological trends in Enterprise Systems for SMEs and large enterprises: Heading towards the Future Internet*, 2009 293

[Moldt und Rölke 2003] Moldt, Daniel ; Rölke, Heiko: Pattern Based Workflow Design Using Reference Nets. In: Aalst, Wil M. P. van der (Hrsg.) ; Hofstede, Arthur ter (Hrsg.) ; Weske, Mathias (Hrsg.): *Proc. of International Conference on BUSINESS PROCESS MANAGEMENT, Eindhoven, NL* Bd. 2678. Berlin u.a. : Springer-Verlag, 2003, S. 246 – 260 303

[Motik u. a. 2002] Motik, Boris ; Maedche, Alexander ; Volz, Raphael: A Conceptual Modeling Approach for Semantics-Driven Enterprise Applications.

In: Meersman, Robert (Hrsg.) ; Tari, Zahir (Hrsg.): *CoopIS/DOA/ODBASE* Bd. 2519. Berlin u.a. : Springer-Verlag, 2002, S. 1082–1099 39

[Muchnick 1997] Muchnick, Steven S.: *Advanced Compiler Design and Implementation*. Morgan Kaufmann, August 1997 339

[Mühl u. a. 2006] Mühl, Gero ; Fiege, Ludger ; Pietzuch, Peter R.: *Distributed Event-Based Systems*. Berlin u.a. : Springer-Verlag, August 2006. – ISBN 3-540-32651-0 10, 36

[Murata 1989] Murata, Tadao: Petri Nets: Properties, Analysis and Applications. In: *Proceedings of the IEEE* 77 (1989), April, Nr. 4, S. 541–580 xvi, 75, 164, 184, 204, 205

[Neubauer u. a. 2004] Neubauer, Bertram ; Ritter, Tom ; Stoinski, Frank: *CORBA Komponenten : effektives Software-Design und Programmierung*. Berlin u.a. : Springer-Verlag, 2004 (Xpert.press) 10

[Newcomer und Lomow 2005] Newcomer, Eric ; Lomow, Greg: *Understanding SOA with Web services*. Addison-Wesley, 2005 30

[Nierstrasz 1989] Nierstrasz, Oscar: A Survey of Object-Oriented Concepts. In: (Kim und Lochovsky, 1989), S. 3–21 43

[Odell u. a. 2000] Odell, James ; Parunak, H. Van D. ; Bauer, Bernhard: Representing Agent Interaction Protocols in UML. In: Ciancarini, Paolo (Hrsg.) ; Wooldridge, Michael (Hrsg.): *AOSE* Bd. 1957. Berlin u.a. : Springer-Verlag, 2000, S. 121–140 292, 309

[Offermann 2003] Offermann, Sven: *Ein Kompositionsmodell für Mehrwertdienste auf Basis von Referenznetzen*, Universität Hamburg, Fachbereich Informatik, Verteilte Systeme und Informationssysteme, Diplomarbeit, 11 2003 10

[Orriëns u. a. 2005] Orriëns, Bart ; Yang, Jian ; Papazoglou, Mike P.: A Rule Driven Approach for Developing Adaptive Service Oriented Business Collaboration. In: Benatallah, Boualem (Hrsg.) ; Casati, Fabio (Hrsg.) ; Traverso, Paolo (Hrsg.): *ICSOC* Bd. 3826. Berlin u.a. : Springer-Verlag, 2005, S. 61–72 291, 292

[OWL-S 2006] OWL Services Coalition: *OWL-S 1.2 Pre-Release*. 2006. – WWW-Dokument†: http://www.daml.org/services/owl-s/1.1/ 323, 340

[Papazoglou u. a. 2006] Papazoglou, Michael P. ; Traverso, Paolo ; Dustdar, Schahram ; Leymann, Frank ; Krämer, Bernd J.: Service-Oriented Computing: A Research Roadmap. In: Cubera, Francisco (Hrsg.) ; Krämer,

Bernd J. (Hrsg.) ; Papazoglou, Michael P. (Hrsg.): *Service Oriented Computing (SOC)*, Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006 (Dagstuhl Seminar Proceedings 05462). – WWW-Dokument†: http://drops.dagstuhl.de/opus/volltexte/2006/524 3, 33

[Petri 1962] Petri, Carl A.: *Kommunikation mit Automaten*. Bonn, Rheinisch-Westfälisches Institut für Instrumentelle Mathematik an der Universität Bonn, Dissertation, 1962 48

[Piccinelli u. a. 2003] Piccinelli, Giacomo ; Zirpins, Christian ; Lamersdorf, Winfried: The FRESCO Framework: An Overview. In: *2003 Symposium on Applications and the Internet Workshops (SAINT 2003 Workshops)*, IEEE Computer Society, 1 2003, S. 120–123 292

[Popien u. a. 1996] Popien, Claudia ; Schürmann, Gerd ; Weiss, Karl-Heinz: *Verteilte Verarbeitung in Offenen Systemen: Das ODP-Referenzmodell*. Stuttgart : Teubner, 1996 26

[Priese und Wimmel 2008] Priese, Lutz ; Wimmel, Harro: *Petri-Netze*. 2. Aufl. Berlin u.a. : Springer-Verlag, 2008 (eXamen.press) 229

[Ramos u. a. 2006] Ramos, Isaac C. ; Di Bucchianico, Alessandro ; Lusine, Hakobyan ; Hee, Kees van: Synthesis and reduction of state machine workflow nets / Technische Universiteit Eindhoven. 2006. – Forschungsbericht. WWW-Dokument†: http://alexandria.tue.nl/extra1/wskrap/publichtml/200618.pdf 207

[Reese 2009] Reese, Christine: *Prozess-Infrastruktur für Agentenanwendungen*. Vogt-Kölln Str. 30, 22527 Hamburg, Universität Hamburg, Fakultät für Mathematik, Informatik und Naturwissenschaften, Department Informatik, Dissertation, 2009 11, 23

[Reisig 1991a] Reisig, Wolfgang: Petri Nets and Algebraic Specifications. In: Jensen, K. (Hrsg.) ; Rozenberg, G. (Hrsg.): *High-level Petri Nets – Theory and Application*. Berlin u.a. : Springer-Verlag, 1991, S. 137–170 2, 71

[Reisig 1991b] Reisig, Wolfgang: Petri Nets and Algebraic Specifications. In: *Theoretical Computer Science* 80 (1991), S. 1–34 4, 9, 47, 95

[Reisig 2009] Reisig, Wolfgang: Simple Composition of Nets. In: (Franceschinis und Wolf, 2009), S. 23–42 339

[Reisig und Rozenberg 1998] Reisig, Wolfgang (Hrsg.) ; Rozenberg, Grzegorz (Hrsg.): *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996*. Bd. *1491*. Berlin u.a. : Springer-Verlag, 1998. (Lecture Notes in Computer Science) 47, 350, 352, 370

[Renew 2005] Renew – *The Reference Net Workshop*. WWW-Dokument[†]: http://www.renew.de/. 2005. – Referenzen auf das Programm, den Quellcode und die Dokumentation von Renew. 97

[Richardson und Ruby 2007] Richardson, Leonard ; Ruby, Sam: *RESTful web services*. O'Reilly, 2007 10

[Rölke 1999] Rölke, Heiko: *Modellierung und Implementation eines Multi-Agenten-Systems auf der Basis von Referenznetzen*, Universität Hamburg, Diplomarbeit, 1999 293

[Rölke 2004] Rölke, Heiko: *Modellierung von Agenten und Multiagentensystemen – Grundlagen und Anwendungen*. Vogt-Kölln Str. 30, 22527 Hamburg, Universität Hamburg, Fachbereich Informatik, Dissertation, 2004 2, 11, 293, 294, 295, 296, 297, 324, 338, 340

[Rozenberg 1991] Rozenberg, Grzegorz (Hrsg.): *Advances in Petri Nets 1990 [10th International Conference on Applications and Theory of Petri Nets, Bonn, Germany, June 1989, Proceedings]*. Bd. *483*. Berlin u.a. : Springer-Verlag, 1991. (Lecture Notes in Computer Science) 347, 370

[Russell u. a. 2007] Russell, Nick ; Hofstede, Arthur H. ter ; Aalst, Wil M. van der: newYAWL: Specifying a Workflow Reference Language using Coloured Petri Nets. In: Jensen, K. (Hrsg.): *Eighth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, Aarhus, Denmark, October 2007*. University of Aarhus, 2007 340

[Russell und van der Aalst 2009] Russell, Nick C. ; Aalst, Arthur H. M. van der: Designing a Workflow System Using Coloured Petri Nets. In: (Jensen u. a., 2009), S. 1–24 9

[Scheer 2002] Scheer, August-Wilhelm: *ARIS - vom Geschäftsprozess zum Anwendungssystem*. Berlin u.a. : Springer-Verlag, 2002 20

[Schimm 2004] Schimm, Guido: *Workflow Mining: Verfahren zur Extraktion von Workflow-Schemata aus ereignisbasierten Daten*. Oldenburg, Universität Oldenburg, Dissertation, 2004 341

[Schmelzer und Sesselmann 2004] Schmelzer, Hermann J. ; Sesselmann, Wolfgang: *Geschäftsprozessmanagement in der Praxis*. München, Wien : Hanser, 2004 15

[Schmidt 1999] Schmidt, Karsten: Stubborn Sets for Standard Properties. In: Donatelli, Susanna (Hrsg.) ; Kleijn, H. C. M. (Hrsg.): *ICATPN* Bd. 1639. Berlin u.a. : Springer-Verlag, 1999, S. 46–65 227

[Schmidt 2000] Schmidt, Karsten: How to Calculate Symmetries of Petri Nets. In: *Acta Inf.* 36 (2000), Nr. 7, S. 545–590 227

[Schmidt 2003] Schmidt, Karsten: Distributed Verification with LoLA. In: *Fundam. Inform.* 54 (2003), Nr. 2-3, S. 253–262 227, 262

[Schmidt 2005] Schmidt, Karsten: Controllability of Open Workflow Nets. In: (Desel und Frank, 2005), S. 236–249 3, 308

[Shankar 2000] Shankar, Natarajan: Symbolic Analysis of Transition Systems. In: Gurevich, Yuri (Hrsg.) ; Kutter, Philipp W. (Hrsg.) ; Odersky, Martin (Hrsg.) ; Thiele, Lothar (Hrsg.): *Abstract State Machines* Bd. 1912. Berlin u.a. : Springer-Verlag, 2000, S. 287–302 263

[Shneiderman und Plaisant 2010] Shneiderman, Ben ; Plaisant, Catherine: *Designing the user interface : strategies for effective human-computer interaction.* 5. Aufl. Upper Saddle River, NJ : Addison-Wesley/Pearson, 2010 10

[SOAP 2003] Gudgin, Martin ; Hadley, Marc ; Mendelsohn, Noah ; Moreau, Jean-Jacques ; Nielsen, Henrik F.: *SOAP: Simple Object Access Protocol Version 1.2 Part 1: Messaging Framework.* 2003. – WWW-Dokument†: http://www.w3.org/TR/2003/REC-soap12-part1-20030624/ 10

[Sowa 2000] Sowa, John F.: *Knowledge Representation: logical, philosophical, and computational foundations.* Pacific Grove : Brooks/Cole, 2000 38

[Staab und Studer 2004] Staab, Steffen (Hrsg.) ; Studer, Rudi (Hrsg.): *Handbook on Ontologies.* Berlin u.a. : Springer-Verlag, 2004 (International Handbooks on Information Systems) 10, 38

[Stahl 2005] Stahl, Christian: A Petri Net Semantics for BPEL / Humboldt-Universität zu Berlin. Juli 2005 (188). – Informatik-Berichte. WWW-Dokument†: http://www2.informatik.hu-berlin.de/Institut/struktur/systemanalyse/preprint/stahl188.pdf 303

[Stahl u. a. 2007] Stahl, Thomas ; Völter, Markus ; Efftinge, Sven ; Haase, Arno: *Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management.* 2. Heidelberg : dpunkt-Verlag, 2007 2

[Stehr 2002] Stehr, Mark-Oliver: *Programming, Specification, and Interactive Theorem Proving – Towards a Unified Language based on Equational Logic, Rewriting Logic, and Type Theory*, Fachbereich Informatik, Universität Hamburg, Dissertation, 2002. – WWW-Dokument†: http://formal.cs.uiuc.edu/stehr/thesis.pdf 262

[Steimann 2000] Steimann, Friedrich: On the representation of roles in object-oriented and conceptual modelling. In: *Data Knowl. Eng.* 35 (2000), Nr. 1, S. 83–106 42, 43

[Stein u. a. 1989] Stein, Lynn A. ; Lieberman, Henry ; Ungar, David: A Shared View of Sharing: The Treaty of Orlando. In: (Kim und Lochovsky, 1989), S. 31–48 43

[Stein und Ivanov 2007] Stein, Sebastian ; Ivanov, Konstantin: Vorgehensmodell zur Entwicklung von Geschäftsservicen. In: Fähnrich, Klaus-Peter (Hrsg.) ; Thränert, Maik (Hrsg.): *Integration Engineering – Motivation, Begriffe, Methoden und Anwendungsfälle*. Leipzig, Germany : Eigenverlag Leipziger Informatik-Verbund (LIV), 2007 (Leipziger Beiträge zur Informatik VI) 290, 291

[Su 1997] Su, Jianwen: Dynamic Constraints and Object Migration. In: *Theor. Comput. Sci.* 184 (1997), Nr. 1-2, S. 195–236. – WWW-Dokument†: http://citeseer.ist.psu.edu/su91dynamic.html 42

[Suzuki und Murata 1983] Suzuki, Ichiro ; Murata, Tadao: A Method for Stepwise Refinement and Abstraction of Petri Nets. In: *Journal of Computer and System Sciences* 27 (1983), Nr. 1, S. 51–76 204

[Taylor u. a. 2009] Taylor, Hugh ; Yochem, Angela ; Phillips, Les ; Martinez, Frank: *Event-Driven Architecture: How SOA Enables the Real-Time Enterprise*. Boston, MA : Addison-Wesley, 2009 36

[Thomas u. a. 2009] Thomas, Oliver ; Leyking, Katrina ; Scheid, Michael: *Vorgehensmodelle zur Entwicklung serviceorientierter Softwaresysteme*. S. 181–190. In: Hansen, H. R. (Hrsg.) ; Karagiannis, D. (Hrsg.) ; Fill, H.-G. (Hrsg.): *Business Services: Konzepte, Technologien, Anwendungen: 9. Internationale Tagung Wirtschaftsinformatik, 25.-27. Februar 2009, Wien. Band 1*. Wien : Österreichische Computer Gesellschaft, 2009. – WWW-Dokument†: http://www.dfki.de/web/forschung/publikationen/renameFileForDownload?filename=ReSubmit_WI_2009_SOA-VM_081115_tt3.pdf&file_id=uploads_137 289, 290

[Thuraisingham 1998] Thuraisingham, Bhavani M.: *Handbook of Data Management, 1998 Edition*. Boca Raton, FL, USA : CRC Press, Inc., 1998 5

[Ullrich 1977] Ullrich, Gernot: Der Entwurf von Steuerstrukturen für parallele Abläufe mit Hilfe von Petri-Netzen / Institut für Informatik der Universität Hamburg. 1977. – Forschungsbericht. Bericht Nr. 36. (IFI-HH-B-36/77) 184, 209

[UML 2.0 2005] Object Management Group: UML 2.0 Superstructure Specification / Object Management Group. 2005. – Forschungsbericht. UML 2.0 Superstructure FTF convenience document 'formal/05-07-04', WWW-Dokument†: http://www.omg.org/cgi-bin/doc?formal/05-07-04 22

[Valette 1979] Valette, Robert: Analysis of Petri Nets by Stepwise Refinements. In: *Journal of Computer and System Sciences* 18 (1979), Nr. 1, S. 35–46 204

[Valk 1978] Valk, Rüdiger: Self-Modifying Nets, a Natural Extension of Petri Nets. In: Ausiello, Giorgio (Hrsg.) ; Böhm, Corrado (Hrsg.): *ICALP* Bd. 62. Berlin u.a. : Springer-Verlag, 1978, S. 464–476 129

[Valk 1986] Valk, Rüdiger: Nets in Computer Organisation. In: Brauer, Wilfried (Hrsg.) ; Reisig, Wolfgang (Hrsg.) ; Rozenberg, Grzegorz (Hrsg.): *Advances in Petri Nets* Bd. 255. Berlin u.a. : Springer-Verlag, 1986, S. 218–233 97, 226

[Valk 1991] Valk, Rüdiger: Modelling Concurrency by Task/Flow EN Systems. In: Best, Eike (Hrsg.) ; Rozenberg, Grzegorz (Hrsg.): *GMD-Studien Nr. 191; Hildesheimer Informatik-Berichte 6/91; 3rd Workshop on Concurrency and Compositionality, 1991, Goslar, Germany.* St. Augustin, Germany : Gesellschaft für Mathematik und Datenverarbeitung mbH — Universität Hildesheim (Germany), Institut für Informatik, Mai 1991, S. 207–215 97, 226

[Valk 1998] Valk, Rüdiger: Petri Nets as Token Objects: An Introduction to Elementary Object Nets. In: (Desel und Silva, 1998), S. 1–25 97, 226, 228, 263, 264, 267, 338

[Valmari 1989] Valmari, Antti: Stubborn sets for reduced state space generation. In: (Rozenberg, 1991), S. 491–515 227

[Valmari 1996] Valmari, Antti: The State Explosion Problem. In: (Reisig und Rozenberg, 1998), S. 429–528 226

[Verbeek u. a. 2001] Verbeek, H. M. W. E. ; Basten, Twan ; Aalst, Wil M. P. van der: Diagnosing Workflow Processes using Woflan. In: *Comput. J.* 44 (2001), Nr. 4, S. 246–279 181

[Verbeek 2004] Verbeek, Henricus M.: *Verification of WF-nets.* Eindhoven, Technische Universiteit, Dissertation, 2004. – WWW-Dokument†: http://alexandria.tue.nl/extra2/200411300.pdf 69, 82

[Vogels 2003] Vogels, Werner: Web Services are not Distributed Objects. In: *IEEE Internet Computing* 7 (2003), Nr. 6, S. 59–66. – WWW-Dokument†: http://www.allthingsdistributed.com/historical/archives/000343.html 28

[Vogler 1987] Vogler, Walter: Behaviour preserving refinements in Petri nets. In: *Proceedings fo the 12th International Workshop on Graph Theoretic Concepts in Computer Science, München, 1986* Bd. 246. Berlin u.a. : Springer-Verlag, 1987, S. 82 – 93 204

[Vogler 1997] Vogler, Walter: Efficiency of Asynchronous Systems and Read Arcs in Petri Nets. In: Degano, Pierpaolo (Hrsg.) ; Gorrieri, Roberto (Hrsg.) ; Marchetti-Spaccamela, Alberto (Hrsg.): *ICALP* Bd. 1256. Berlin u.a. : Springer-Verlag, 1997, S. 538–548 149

[Vogler 2002] Vogler, Walter: Efficiency of asynchronous systems, read arcs, and the MUTEX-problem. In: *Theor. Comput. Sci.* 275 (2002), Nr. 1-2, S. 589–631 149

[Weikum und Vossen 2002] Weikum, Gehrard ; Vossen, Gottfried: *Transactional Information Systems – theory, algorithms, and the practice of concurrency control and recovery*. San Francisco, CA, USA : Morgan Kaufmann, 2002 341

[Weske u. a. 2005] Weske, Mathias ; Vossen, Gottfried ; Puhlmann, Frank: Workflow and Service Composition Languages. In: Bernus, P. (Hrsg.) ; Mertins, K. (Hrsg.) ; Schmidt, G. (Hrsg.): *Handbook on Architectures of Information Systems*. 2. Aufl. Berlin u.a. : Springer-Verlag, 2005 (International Handbooks on Information Systems), S. 369–390 20

[Wester-Ebbinghaus und Moldt 2008a] Wester-Ebbinghaus, Matthias ; Moldt, Daniel: Modelling Multi-Agent Systems with Organizations in Mind. In: Ultes-Nitsche, Ulrich (Hrsg.) ; Moldt, Daniel (Hrsg.) ; Augusto, Juan C. (Hrsg.): *MSVVEIS*, INSTICC PRESS, 2008, S. 81–90 294

[Wester-Ebbinghaus und Moldt 2008b] Wester-Ebbinghaus, Matthias ; Moldt, Daniel: Structure in threes: modelling organization-oriented software architectures built upon multi-agent systems. In: Padgham, Lin (Hrsg.) ; Parkes, David C. (Hrsg.) ; Müller, Jörg (Hrsg.) ; Parsons, Simon (Hrsg.): *AAMAS (3)*, Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), 2008, S. 1307–1310 294

[Wetzel und Klischewski 2004] Wetzel, Ingrid ; Klischewski, Ralf: Serviceflow beyond workflow? IT support for managing inter-organizational service processes. In: *Inf. Syst.* 29 (2004), Nr. 2, S. 127–145 26

[Wienberg 2001] Wienberg, Frank: *Feature/Structure-Netze*. Vogt-Kölln Str. 30, 22527 Hamburg, Universität Hamburg, Fachbereich Informatik, Dissertation, 2001 97

[Wieringa u. a. 1995] Wieringa, Roel ; Jonge, Wiebren de ; Spruit, Paul: Using dynamic classes and role classes to model object migration. In: *Theor. Pract. Object Syst.* 1 (1995), Nr. 1, S. 61–83 43, 305

[Willems und Wolper 1996] Willems, Bernard ; Wolper, Pierre: Partial-Order Methods for Model Checking: From Linear Time to Branching Time. In: *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, 27-30 July 1996*, IEEE Computer Society Press, 1996, S. 294–303. – WWW-Dokument†: http://www.montefiore.ulg.ac.be/~pw/papers/psfiles/WW96.ps 227

[Wolf 2009a] Wolf, Karsten: Does My Service Have Partners? In: Jensen, Kurt (Hrsg.) ; Aalst, Wil M. P. van der (Hrsg.): *Transactions on Petri Nets and Other Models of Concurrency II, Special Issue on Concurrency in Process-Aware Information Systems* Bd. 5460. Berlin u.a. : Springer-Verlag, 2009, S. 152–171 339

[Wolf 2009b] Wolf, Karsten: A theory of service behavior. In: Kopp, Oliver (Hrsg.) ; Lohmann, Niels (Hrsg.): *ZEUS* Bd. 438, CEUR-WS.org, 2009, S. 1–7. – WWW-Dokument†: http://CEUR-WS.org/Vol-438/paper1.pdf 339

[Wolff u. a. 2005] Wolff, Frank ; Oberle, Daniel ; Lamparter, Steffen ; Staab, Steffen: Economic Reflections on Managing Web Service Using Semantics. In: (Desel und Frank, 2005), S. 194–207 287

[Wong u. a. 1997] Wong, Raymond K. ; Chau, H. L. ; Lochovsky, Frederick H.: A Data Model and Semantics of Objects with Dynamic Roles. In: Gray, W. A. (Hrsg.) ; Larson, Per-Åke (Hrsg.): *ICDE*, IEEE Computer Society, 1997, S. 402–411. – WWW-Dokument†: http://citeseer.ist.psu.edu/96227.html 43

[Wooldridge u. a. 2000] Wooldridge, Michael ; Jennings, Nicholas R. ; Kinny, David: The Gaia Methodology for Agent-Oriented Analysis and Design. In: *Autonomous Agents and Multi-Agent Systems* 3 (2000), Nr. 3, S. 285–312 291

[Workflow Management Coalition 1999] Workflow Management Coalition: *Terminology & Glossary.* Feb 1999. – Document Number WFMC-TC-1011 – Document Status - Issue 3.0; WWW-Dokument†: http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf 15, 17

[WSDL 2001] Christensen, Erik ; Curbera, Francisco ; Meredith, Greg ; Weerawarana, Sanjiva: WSDL: Web Services Definition Language 1.1 / W3C. 2001. – Forschungsbericht. WWW-Dokument†: http://www.w3.org/TR/2001/NOTE-wsdl-20010315 304

[Yavuz-Kahveci u. a. 2001] Yavuz-Kahveci, Tuba ; Tuncer, Murat ; Bultan, Tevfik: A Library for Composite Symbolic Representations. In: (Margaria und Yi, 2001), S. 52–66 263

[YAWL 2008] *YAWL – Yet Another Workflow Language*. 2008. – Darstellung der Sprache und weiterführende Artikel, WWW-Dokument[†]: http://http://www.yawlfoundation.org/ 340

[Zirpins 2007] Zirpins, Christian: *Interaktionsorientierte Komposition virtueller Dienstleistungsprozesse*, Universität Hamburg, Fachbereich Informatik, Verteilte Systeme und Informationssysteme, Dissertation, 6 2007 10, 292, 341

[Zirpins und Piccinelli 2004] Zirpins, Christian ; Piccinelli, Giacomo: Evolution of Service Processes by Rule Based Transformation. In: Lamersdorf, Winfried (Hrsg.) ; Tschammer, Volker (Hrsg.) ; Amarger, Stéphane (Hrsg.): *I3E*, Kluwer, 2004 (IFIP Conference Proceedings), S. 287–305 292

[Zúñiga 2001] Zúñiga, Gloria L.: Ontology: its transformation from philosophy to information systems. In: *2nd International Conference on Formal Ontology in Information Systems, FOIS 2001, Ogunquit, Maine, USA, October 17-19, 2001*, ACM, 2001, S. 187–197 38

[†]Sämtliche verwendeten Links auf Webseiten sind vom Stand 1. Feb. 2010.

Index