Strukturen und Werkzeuge für eine durch das Anlagenpersonal modifizierbare und ergänzbare Petrinetzanlagensteuerung

Automatische Generierung übersichtlicher Petrinetze und optimaler Abläufe durch Aufspaltung in eine vom Systemingenieur zu implementierende Anlagenbeschreibung und eine vom Facharbeiter vorgebbare Anlagennutzung

Vom Fachbereich 18
der Technischen Hochschule Darmstadt
zur Erlangung der Würde eines
Doktor-Ingenieurs
(Dr.-Ing.)
genehmigte
D i s s e r t a t i o n

von

Dipl.-Ing. Bernd Strege,
geboren am 7. Februar 1966 in Hofheim am Taunus.

Referent:	Prof. Dr. rer. nat. Dipl.-Ing. H. Tolle
Korreferent:	Prof. Dr.-Ing. J. Lunze
Tag der Einreichung:	15. April 1996
Tag der mündlichen Prüfung:	24. Juni 1996

D 17
Darmstädter Dissertationen
1996

Strukturen und Werkzeuge für eine durch das Anlagenpersonal modifizierbare und ergänzbare Petrinetzanlagensteuerung

Automatische Generierung übersichtlicher Petrinetze und optimaler Abläufe durch Aufspaltung in eine vom Systemingenieur zu implementierende Anlagenbeschreibung und eine vom Facharbeiter vorgebbare Anlagennutzung

Vom Fachbereich 18
der Technischen Hochschule Darmstadt
zur Erlangung der Würde eines
Doktor-Ingenieurs
(Dr.-Ing.)
genehmigte
D i s s e r t a t i o n

von

Dipl.-Ing. Bernd Strege,
geboren am 7. Februar 1966 in Hofheim am Taunus.

Referent:	Prof. Dr. rer. nat. Dipl.-Ing. H. Tolle
Korreferent:	Prof. Dr.-Ing. J. Lunze
Tag der Einreichung:	15. April 1996
Tag der mündlichen Prüfung:	24. Juni 1996

D 17
Darmstädter Dissertationen
1996

Vorwort

Die in der vorliegenden Dissertation zusammengefaßten Forschungsergebnisse entstanden während meiner Mitarbeit als Assistent von Herrn Professor Tolle am Fachgebiet Regelsystemtheorie und Robotik des Instituts für Regelungstechnik der Technischen Hochschule Darmstadt.

Mein tief empfundener Dank gilt Herrn Prof. Dr. rer. nat. Dipl.-Ing. H. Tolle für die Unterstützung und Förderung dieser Arbeit.

Herrn Professor Dr.-Ing. J. Lunze danke ich für die Übernahme des Korreferats.

Der Siemens AG sowie der Deutschen Forschungsgemeinschaft (DFG) gilt mein Dank für ihre finanzielle Unterstützung. Insbesondere danke ich den Herren Dipl.-Ing. Höh, Dipl.-Ing. Roth, Dr.-Ing. Niedermayr, Dr.-Ing. Lumer und Dipl.-Ing. Koglin - alle Mitarbeiter der Siemens Zentralabteilung Produktion und Logistik - für die Förderung bzw. fachliche Unterstützung dieser Arbeit.

Herr Dr. Pham und Herr Dr. He haben als Gastwissenschaftler durch ihre engagierte Mitarbeit einen wichtigen Beitrag zu dieser Arbeit geleistet und damit eine besondere Würdigung verdient. Für die finanzielle Unterstützung des Gastaufenthaltes der vorgenannten Herren danke ich der Technischen Hochschule Darmstadt bzw. dem Graduiertenkolleg ISIA (Intelligente Systeme für die Informations- und Automatisierungstechnik).

Dank auch meinem Vater Hans W. Strege für die Durchsicht des Manuskriptes.

Darmstadt, im Frühjahr 1996

Bernd Strege

Begriffe:

(alphabetisch geordnet)

Auftragsgraph:	Werkzeug zur Vorgabe von Abläufen zur Herstellung eines Produktes
Auftragsmodell:	Vereinigung aller verschiedenen Auftragsgraphen in einem Modell
Ausgangsbereich einer Transition t:	Alle Stellen, zu denen eine Kante von t existiert
Deadlock/Verklemmung:	Markierung eines Petrinetzes, in der keine Transition Konzession hat. Eine gewünschte Zielmarkierung, in der ebenfalls keine Transition Konzession besitzt, wird nicht als Deadlock im Sinne einer zu vermeidenden Markierung bezeichnet
Eingangsbereich einer Transition t:	Alle Stellen, von denen eine Kante zu t existiert
IDEF-Graph:	Der zu einem Auftragsgraph korrespondierende Graph in IDEF-Notation (zu jeder Operation werden die benötigten Ressourcen benannt)
IDEF-Modell:	Vereinigung aller IDEF-Graphen
Job, Auftrag:	Ablauf zur Herstellung eines Produktes (Produkttypen). Zu n verschiedenen Produkten gehören n verschiedene Aufträge
Konzession:	Eine Transition t hat Konzession, wenn sie feuerbar ist, d.h. alle Stellen im Eingangsbereich von t ausreichend markiert sind und

	alle Stellen im Ausgangsbereich die beim Feuern von t abzulegenden Marken aufnehmen können
Makespan:	Fertigungszeit, die zur Herstellung einer vorgegebenen Anzahl von Produkten benötigt wird. Beispiel: Sollen 100 Fernseher montiert werden, ist der Makespan durch den Zeitraum vom Beginn der Produktion bis zur Fertigstellung des letzten der 100 Fernseher gegeben
Maximaler Schritt:	Sowohl bei zeitbewerteten als auch nicht zeitbewerteten Petrinetzen eine Teilmenge der konzessionierten Transitionen, die bei Hinzunahme nur einer einzigen weiteren konzessionierten Transition die Eigenschaft der nebenläufigen Schaltbarkeit verliert
Schedule:	Schaltfolge π, die eine Anfangs- in eine Zielmarkierung überführt
Scheduling:	Ermittlung von optimierten Schaltfolgen auf der Basis zeitbewerteter Petrinetze

1 Einleitung

1.1 Anforderung: Veränderung von Fertigungsabläufen durch das Anlagenpersonal vor Ort

Heutzutage werden in der Industrie komplexe Fertigungssysteme eingesetzt, in denen teil- bzw. vollautomatisiert Produkte gefertigt werden. Die Hardware der Systeme ist im allgemeinen modular aufgebaut, d.h. sie ist aus über Transporteinrichtungen verbundenen Fertigungseinheiten - sogenannten Fertigungszellen - zusammengesetzt. Gesteuert und überwacht werden die Produktionsabläufe durch Fertigungsleitsysteme, die aus einer übergeordneten Produktionsplanungsebene und einer operativen Ebene bestehen. Die Planungsebene bestimmt für einen längeren Planungshorizont Produktionspläne, während die operative Ebene (Zellensteuerung) die Abläufe ereignisorientiert steuert und koordiniert.

Die Erfahrung aus der industriellen Praxis ist, daß während des Betriebs von Fertigungsanlagen sowohl die Hardware als auch die Steuerungssysteme auf der operativen Ebene häufig geändert werden müssen. Das ist z.B. der Fall, wenn das Fertigungsprogramm um zusätzliche Produkttypen oder eine Fertigungszelle zur Kapazitätserhöhung um zusätzliche Komponenten erweitert wird. Es kommt auch vor, daß eine Steuerung um zunächst noch nicht berücksichtigte Fehlerbehandlungsabläufe ergänzt werden muß. Da das in einer Fertigungsanlage verfügbare Personal - Techniker, Facharbeiter, Betriebsingenieure - im allgemeinen nicht über die zur Änderung der Ablaufsteuerungen notwendigen Kenntnisse verfügt, muß eine Softwarefirma bzw. der Instandhaltungsservice des Systemlieferanten mit der Änderung beauftragt werden. Dadurch entstehen einerseits ein hoher Aufwand und Zeitverzüge und andererseits trägt dies nicht zur Motivation der Mitarbeiter vor Ort bei.

Deshalb wären anwenderfreundlichere Steuerungssysteme wünschenswert, die eine so einfache Bedienerschnittstelle besitzen, daß die Ablaufänderungen von dem Anlagenpersonal vor Ort in einer seiner Denkweise entsprechenden Form vorgegeben werden können und die Umsetzung in die Steuerprogramme vom System automatisch erledigt wird. Eine solche Möglichkeit würde das heutzutage in der Industrie verfolgte Konzept der Übertragung von Verantwortung an die Fertigungsarbeitsgruppen voll unterstützen und neben der Reduktion

des Änderungsaufwandes im Zuge einer verbesserten Mensch-Technik-Kooperation die Motivation der Mitarbeiter erheblich steigern. Voraussetzung hierfür ist aber, daß die Anwender eines solchen Systems weder überfordert noch in ihren Eingriffsmöglichkeiten zu stark eingeschränkt werden. D.h. die Bedienerschnittstelle darf nicht zu detailliert bzw. kompliziert sein. Sie sollte aber ebenfalls nicht so abstrakt sein, daß sie den Anwender zu stark in der Vorgabe von Abläufen einschränkt und damit dessen Eingriffsmöglichkeiten de facto nur gering sind.
Gesucht werden also Systeme, die sich schnell und mit geringem Aufwand modifizieren lassen. Man bezeichnet Systeme, die diese Anforderung erfüllen, als ***Agile Fertigungssysteme***[1] ***(AFS)***.

In der Automatisierungstechnik haben sich Petrinetze als eine wichtige Methode zur Ablaufsteuerung etabliert ([Schnieder 92], [Glüer und Schmidt 88], [Lunze 92], [Kluwe et al. 95], [Bredebusch et al. 94]). Neben der grafischen und strukturierten Modellierung liegt ihr Hauptvorteil darin, daß sie über alle Phasen eines Steuerungsentwurfes (Modellbildung, Implementierung und Betrieb) als einheitliche Methode angewendet werden können ([Schnieder 92]). Darüberhinaus können sie mittels Umsetzungsprogrammen direkt in Code für speicherprogrammierbare Steuerungen (SPS) umgesetzt werden ([Schnieder 92], [Jörns et al. 95], [EN 61131-3]). Damit sind sie praktisch zu der heute verwendeten Standard-Hard- und Software kompatibel.
Da das an einer Fertigungsanlage normalerweise verfügbare Personal nur wenige bzw. keine Petrinetzkenntnisse besitzt, ergibt sich somit konkret die Anforderung, abstrakt und einfach vorgegebene Ablaufspezifikationen in Petrinetze umzusetzen. Befragungen von Testpersonen haben ergeben, daß Personen mit der Qualifikation eines Facharbeiters Fertigungsabläufe fast intuitiv in der groben und einfachen Form

„Transportiere Werkstück von Palette 1 in Maschine 1. Dann führe eine Bohroperation aus. Wenn die Bohroperation mit dem Fehlerkode x beendet wurde, transportiere das Werkstück zur Nachbearbeitung in ....."

formulieren.

[1] Im Englischen lautet die Bezeichnung ***Agile Manufacturing Systems***, abgekürzt ***AMS***.

Es ist kein System bekannt, das solche Anweisungen automatisch in Petrinetze umsetzt. Die Lösung dieses Problems ist Aufgabenstellung dieser Arbeit.

Die automatisch erzeugten Petrinetze, für die eine Anfangsmarkierung und eine gewünschte Zielmarkierung bekannt sind, können aber nicht unmittelbar zur Ablaufsteuerung verwendet werden, da Verklemmungen auftreten können und zur optimierten Ressourcennutzung das Schalten einer optimierten Schaltfolge wünschenswert wäre. Deshalb muß zunächst durch eine Ablaufplanung, wofür meist die englische Bezeichnung Scheduling benutzt wird, ein verklemmungsfreier und optimierter Ablauf ermittelt werden, bevor ein automatisch erzeugtes Petrinetzmodell zur Ablaufsteuerung verwendet werden kann. Ein mögliches Optimierungskriterium ist z.B. die Minimierung der Zyklus- oder auch Gesamtfertigungszeit[2].

Die vorliegende Arbeit hat damit zwei Zielsetzungen, die sich aus der generellen Anforderung einer einfachen Bedienerschnittstelle ergeben. Das erste Ziel ist die automatische Generierung von Petrinetzen. Daraus folgt die zweite Zielsetzung der Ermittlung optimierter und verklemmungsfreier Abläufe (Schaltfolgen) bei bekannten Zeitdauern der einzelnen Operationen.

Bevor in Abschnitt 1.3 eine Konkretisierung der Zielsetzungen erfolgt und die Gliederung der Arbeit vorgestellt wird, diskutiert der folgende Abschnitt allgemeine Forschungsergebnisse, die zur Lösung der genannten Ziele einen relevanten Beitrag liefern.

[2] engl.: *Makespan*

1.2 Stand der Forschung

Im Bereich der Automatisierung komplexer Handhabungsaufgaben mit Robotern wurden eine Reihe von Forschungsarbeiten mit der Zielsetzung durchgeführt, die Mensch-Maschine-Schnittstelle in Richtung eines höheren, aufgabenorientierten Abstraktionsniveaus zu verschieben ([Kegel 90], [Simon 91], [Hörmann 92], [Pitschelsrieder 93], [Matthiesen 95]). In [Simon 91] wird z.B. ein regelbasiertes Robotersteuerungssystem beschrieben, das die Fähigkeit besitzt, grobe, problemorientierte Ablaufanweisungen in sinnvolle Handlungssequenzen umzusetzen und diese unter Verwendung der verfügbaren Sensorik von einem Roboter ausführen zu lassen.
In [Kegel 90] wird ein objektorientierter Ansatz verfolgt, bei dem grobe Handlungsanweisungen nach einem objektorientierten Prinzip schrittweise in kommandierbare Handlungssequenzen für ein Robotersystem umgesetzt werden. Matthiesen beschreibt in [Matthiesen 95] ein wissensbasiertes System, in dem Abläufe auf der Entscheidungsebene durch Graphen repräsentiert werden und das in der Lage ist, komplexe Handhabungsaufgaben mit Hilfe eines Roboters und Sensorik autonom auszuführen.
Die genannten Ansätze erzielen bei der Steuerung von Robotern bzw. bei speziellen Anwendungen wie Montagesystemen eine erhebliche Vereinfachung der Mensch-Maschine-Kommunikation, sind aber nicht einfach auf größere Fertigungssysteme mit einer Vielzahl von Robotern, Maschinen und Transporteinrichtungen übertragbar.

Da eine breitere Betrachtung der Literatur zu weit von der konkreten Zielsetzung der vorliegenden Arbeit wegführt, beschränkt sich der Literaturüberblick im folgenden konkret erstens auf die Petrinetzgenerierung und zweitens auf die Scheduling-Aufgabe.

Petrinetzgenerierung:

Die zu diesem Punkt relevanten Forschungsarbeiten lassen sich zwei grundsätzlichen Ansätzen zuordnen.

Der erste Ansatz besteht darin, Methoden und Prinzipien zur Vereinfachung und Systematisierung des Petrinetzentwurfes für Petrinetzspezialisten herauszuarbeiten. Hierzu gibt es zwei Methoden:

1. Eine Methode besteht darin, von dem Prinzip der Verfeinerung bzw. Vergröberung von Petrinetzen Gebrauch zu machen. Die Idee hierbei ist, entweder nach dem Top-Down- oder dem Bottom-Up-Entwurfsprinzip Petrinetze schrittweise entweder so zu verfeinern oder so aus elementaren Strukturen zu synthetisieren, daß ein Gesamt-Petrinetz entsteht, welches z.B. das Kriterium der Lebendigkeit erfüllt. Die Top-Down-Methode wird z.B. in [Zhou et al. 89] und [Valette 79] beschrieben, während Bottom-Up-Verfahren z.B. in [Jeng 92], [Koh 91] oder [Chao und Wang 95] zu finden sind. Die Methode stellt zwar eine wichtige Hilfestellung für Petrinetzexperten dar, löst aber das Problem der Umsetzung einfacher Spezifikationen in Petrinetze nicht.
2. Eine andere Methode verwendet Petrinetzmodule als Basis und verknüpft diese zu Gesamt-Petrinetzen (vgl. hierzu auch [Glüer und Schmidt 88] oder [Stulle 95]). Diese Vorgehensweise führt zu einer erheblichen Vereinfachung des Petrinetzentwurfes. Fertigungsfachleute dürften die zur manuellen Verknüpfung der Module notwendigen Petrinetzkenntnisse allerdings nicht in ausreichendem Umfang besitzen.

Der zweite grundsätzliche Ansatz besteht in einer getrennten Modellierung von Fertigungssystem und Fertigungsstrategie. Ezpeleta et al. modellieren die Fähigkeiten eines Fertigungssystems zunächst in Form eines nicht colorierten Basis-Petrinetzes. ([Ezpeleta et al. 93 a und b]). Dieses enthält Informationen über die möglichen Reihenfolgen der Transport- und Maschinenoperationen. Die Arbeitspläne der verschiedenen Jobs werden anschließend als colorierte Marken repräsentiert und bilden zusammen mit entsprechend ergänzten Kantenanschriften ein komplettes Modell der Fertigungsabläufe. Die Anpassung der Steuerung an Hardwaremodifikationen sowie grundlegende Ablaufänderungen geschieht über die Anpassung des Basis-Petrinetzes und erfordert somit Petrinetzkenntnisse.

Camurri und Franchi modellieren ein Fertigungssystem in einer strukturierten Wissensbasis und erzeugen hieraus ein Petrinetz ([Camurri und Franchi 90]). Die wissensbasierte Repräsentation erhöht die Transparenz sowie die Änderungsfreundlichkeit. Die von Camurri und Franchi vorausgesetzte Beschränkung auf Maschinen- und Pufferoperationen mit praktisch als unbeschränkt zu betrachtenden Werkstückpuffern begrenzt die Einsatzmöglichkeiten des Systems in der Praxis allerdings sehr stark.
In [Dungern 90 a und b] wird für die spezielle Anwendung der Montageablaufsteuerung ein System dargelegt, in dem durch eine produktbezogene Ablaufplanung zunächst ein Operationsnetz erzeugt wird, dessen Knoten anschließend einzeln zu als Petrinetze darstellbaren sog. Aktionsnetzen verfeinert werden.

Aus dem Literaturüberblick geht hervor, daß kein System bekannt ist, das Anweisungen von Facharbeitern in Petrinetz-Anlagensteuerungen umsetzt, ohne deren Einsatzmöglichkeiten auf spezielle Fertigungsstrukturen zu beschränken.

Scheduling-Aufgabe:

Es ist bekannt, daß zum Lösen von Scheduling-Aufgaben selbst bei einfach strukturierten Fertigungssystemen ein hoher Aufwand erforderlich ist. In vielen Fällen handelt es sich um NP-vollständige Optimierungsprobleme (vgl. auch [Moser 93]), deren Rechenaufwand stärker als polynomial mit dem Aufgabenumfang wächst. Einen Überblick über die verschiedenen Verfahren findet man z.B. in [Wagner 69] oder [Blacewicz et al. 91].
In der industriellen Anwendung zerfällt das Scheduling-Problem häufig in zwei Phasen: In der ersten Phase werden die Transportwege und Bearbeitungsvorgänge der Werkstücke bzw. Aufträge bestimmt (Bestimmung der Routen). In der zweiten Phase erfolgt die Zuweisung von Ressourcen zu Transport- und Bearbeitungsvorgängen. Unter Umständen werden die Schritte auf unterschiedlichen Rechnersystemen durchgeführt, d.h. die Festlegung der Routen erfolgt auf einer übergeordneten Planungsebene mit einem PC bzw. einer Workstation, während die Ressourcenzuweisung auf einer untergeordneten Ebene durch speicherprogrammierbare Steuerungen durchgeführt wird (vgl. Anwendungsbericht in [Kraft und Krüger 94]). Gerade in flexiblen Systemen wäre zur vollen Ausnutzung der Flexiblität eine Zusammenfassung der beiden Schritte sinnvoll.

Zur Reduktion des Rechenaufwandes werden häufig einfache heuristische Schedulingregeln angewendet (einen Überblick über Prioritätsregeln findet man z.B. in [Moser 93] oder [Panwalker und Iskander 77]). Die heuristischen Schedulingregeln sind jeweils auf bestimmte Problemstellungen zugeschnitten und repräsentieren keine generell für ereignisdiskrete Prozesse erfolgreiche Methode. Auch werden zur Leistungsanalyse flexibler Fertigungssysteme Warteschlangennetzwerke oder Simulationsmodelle herangezogen (vgl. hierzu [Tempelmeier und Kuhn 93]).

Die in der Literatur vorgeschlagenen und über die Prioritätsregeln hinausgehenden Verfahren haben Probleme mit der einfachen Formulierung bzw. Modellierung der Scheduling-Aufgaben und/oder mit der Rechenzeit bzw. dem Speicherbedarf. Für eine mathematische Formulierung als binäres Optimierungsproblem existieren für spezielle Problemstellungen effiziente Lösungsverfahren ([Hoitomt 90], [Domschke et al. 93]). Es treten aber Modellierungsprobleme und ein hoher Rechenaufwand bei flexiblen Fertigungssystemen mit konkurrierend angeforderten Ressourcen[3], Routing-Flexibilität, variablen Losgrößen, Montage- bzw. Demontagesituationen und komplexen Ressourcenanforderungssituationen auf. Im allgemeinen sind diese Scheduling-Probleme NP-vollständig und können nur in Spezialfällen mit polynomialem Aufwand gelöst werden ([Blazewicz et al. 91], [Rutten 93], [Lee und DiCesare 93]). Verfahren wie Branch&Bound oder Dynamische Programmierung sind bei bestimmten Problemstellungen effizient (vgl. hierzu [Wagner 69], [Muth und Thompson 63] oder [Lee 94]). Heuristische Suchverfahren wie der A*-Algorithmus benötigen zuweilen viel Rechenspeicher. Netzpläne werden heutzutage meist für die Projektplanung verwendet (vgl. auch [Domschke und Drexl 95] oder [Neumann und Morlock 93]). Sie können aber auch zur Kapazitätsplanung und damit z.B. für die Maschinenbelegungsplanung verwendet werden (vgl. auch [Drexl 90]). Bei der Anwendung auf Fertigungssysteme mit einer Vielzahl von Aufträgen variabler Losgröße entstehen hierbei allerdings umfangreiche Netzpläne und auch längere Rechenzeiten, zumal bei der Lösung eine Formulierung als binäres Optimierungsproblem zugrundegelegt wird.
Neuere Ansätze zum Scheduling auf der Basis der Max-Plus-Algebra ([Moßig und Rehkopf 96]) benötigen relativ viel Rechenzeit und greifen letztendlich im Kern auf Prioritätsregeln zurück (siehe hierzu [Rehkopf 92]).

[3] der übliche englische Ausdruck hierfür ist *Shared Resources*

Mit Petrinetzen steht ein allgemein anwendbares Werkzeug zur Formulierung von Scheduling-Aufgaben für ereignisdiskrete Systeme zur Verfügung ([Schnieder 92]). Insbesondere können Aspekte wie Routing-Flexibilität, Shared Resources, variable Losgrößen und Nebenläufigkeiten effizient modelliert werden. Eine Beschränkung auf bestimmte Systemstrukturen ist nicht erforderlich. Da Petrinetze sowieso zur Steuerung erstellt werden müssen - die auf Werkstattebene steuernden SPS-Programme repräsentieren im Grunde eine Klasse von Petrinetzen - ensteht kein wesentlicher zusätzlicher Modellierungsaufwand. Im Vergleich zu Scheduling-Verfahren, die die Zeitpunkte von Aktivitäten ermitteln, ist ein Petrinetz-basierter Scheduling-Ansatz unempfindlich gegenüber zeitlichen Verzögerungen im Ablauf. Zwar kann die Güte der Abläufe infolge der vom Modell abweichenden tatsächlichen Ausführungszeiten abhängig von dem konkreten Petrinetz mehr oder weniger stark abnehmen, die Abläufe selbst aber bleiben ausführbar. Sollen die Startzeitpunkte für bestimmte Aktivitäten vorgegeben werden können, sind unter Umständen andere Ansätze wie Constraint-basiertes Scheduling (vgl. [Fox 94]) oder Netzplan-Methoden besser geeignet.
Es gibt nur wenige Arbeiten über direkt auf Petrinetzen basierende Scheduling-Verfahren. Zunächst kann die Erreichbarkeitsanalyse zur Bestimmung eines optimalen Ablaufes herangezogen werden. Im allgemeinen wachsen aber die Erreichbarkeitsgraphen exponentiell mit der Anzahl der Transitionen, Stellen und Marken, d.h. es kommt zur kombinatorischen Explosion ([Hanisch 92 a], [Shen et al. 92]).

Als Ersatz für die Erreichbarkeitsanalyse werden in der Literatur unterschiedliche Wege beschritten:

Der erste und einfachste Weg besteht in der Simulation der Petrinetze ([Ramamoorthy und Ho 80], [Lin und Lee 94]).

Der zweite Ansatz basiert auf der Anwendung heuristischer Suchverfahren. Dabei werden unterschiedliche Heuristiken zur Einschränkung des Suchraumes benutzt. In [Huang und Zhang 94] wird ein auf der Auswertung von T-Invarianten basierendes Verfahren beschrieben, bei dem aus alternativen T-Invarianten und damit aus alternativen Abläufen mittels einfacher Heuristiken ausgewählt wird. Allerdings wird hier nur eine sehr einfache heuristische Regel verwendet. Lee und DiCesare bestimmen für Petrinetze, die Fertigungssysteme mit Maschinen und unbegrenzten Puffern modellieren, optimierte Schaltfolgen durch einen A*-

Algorithmus ([Lee und DiCesare 93], [Lee 94]). Dieser A*-Algorithmus wird von Cheng et al. um die Anwendbarkeit auf Fertigungssysteme mit in ihrer Kapazität beschränkten Puffern sowie Routing-Flexibilität erweitert ([Cheng et al. 94]). Die Verfahren von Lee und DiCesare sowie Cheng et al. benötigen immer noch viel Speicher. So benötigt der A*-Algorithmus von Lee und DiCesare bei einem Scheduling-Problem mit 3 Maschinen, 5 Jobs und einer Losgröße von 10 mehr als 16MB Programmspeicher. Shih und Sekiguchi schlagen vor, Petrinetze zu simulieren und Konflikte durch einen Beam-Search-Algorithmus zu lösen ([Shih und Sekiguchi 91]). Der Beam-Search-Algorithmus untersucht ausgehend von einem Konfliktzustand Teile des Erreichbarkeitsgraphen mit einer vorgegebenen Beam-Tiefe und ermittelt so lokal die beste Alternative. Diese Methode greift allerdings bei komplexen Aufgaben unter Umständen zu kurz und optimiert nur begrenzt.

Ein dritter Lösungsweg besteht in der künstlichen Beschränkung des Zustandsraumes der Petrinetze durch die Einführung von Fakten-Transitionen ([Hanisch 92 a]). Fakten repräsentieren verbotene Zustände (Markierungen). Beim Aufbau des Erreichbarkeitsgraphen wird ein Pfad abgebrochen, wenn eine Faktentransition feuern kann. Auf diese Weise wird der untersuchte Teil des Erreichbarkeitsgraphen reduziert. Der Nachteil dieser Vorgehensweise ist unmittelbar einsichtig: Die Ergänzung eines Petrinetzes durch Fakten-Transitionen erfordert genaue Prozeß- und Petrinetzkenntnisse.

Zentrale Idee eines vierten Ansatzes ist die Unterteilung eines Petrinetzes in Teilnetze. Shen et al. bestimmen Schedules für ein Gesamtnetz, indem sie das Gesamtnetz in Teilnetze zerlegen und für diese Teilnetze einzeln mittels Branch&Bound-Verfahren optimierte Abläufe ermitteln ([Shen et al. 92]). Neben der Tatsache, daß jede Transition nur einmal feuern kann und verklemmungsfreie Petrinetze vorausgesetzt werden, ist ein weiterer Nachteil dieses Verfahrens ein fehlender Algorithmus zur Zerlegung des Gesamtnetzes. Die Zerlegung muß manuell erfolgen und erfordert Petrinetzkenntnisse sowie ausreichend Erfahrung.

Zusammenfassend ist anzumerken, daß für spezielle Scheduling-Probleme Lösungsverfahren bekannt sind, die mit polynomialem Aufwand unter Umständen sogar optimale Lösungen ermitteln können. Da mit Petrinetzen im allgemeinen aber beliebige, ergänzbare und erweiterbare Fertigungssysteme modelliert werden, sind auch allgemein anwendbare Verfahren gesucht. Die einfache Petrinetz-Simulation bzw. die Anwendung von heuristischen

Prioritätsregeln können im Einzelfall und abhängig von der konkreten Aufgabenstellung zu guten Ergebnissen führen. Die Lösungen können aber auch sehr unbefriedigend sein. Die bislang entwickelten Petrinetz-basierten Scheduling-Verfahren benötigen wie die A*-Suche - hierbei je nach Parametrisierung der heuristischen Kostenabschätzungsfunktion - viel Rechenspeicher oder verlangen Spezialkenntnisse über Petrinetze und den zu steuernden Prozeß (vgl. Ansatz mit Faktentransitionen) und lassen somit den Wunsch nach Verfahren offen, die optimierte Abläufe auf der Basis einer einfachen Vorgabe von Zeitdauern der Operationen mit weniger Rechenspeicher und -aufwand ermitteln.

1.3 Zielsetzung und Gliederung der Arbeit

Die Zielsetzung dieser Arbeit ist ein System, das den Rückgriff auf den Systementwickler für den Nutzer einer Fertigungsanlage auch bei Produktänderungen, Maschinenveränderungen und neu einzufügender automatischer Fehlerhandhabung möglichst weitgehend überflüssig macht.

Das System soll auf die in der Praxis vorkommenden Fertigungsstrukturen anwendbar sein. Dazu gehören Fertigungsanlagen mit bzw. ohne Routing-Flexibilität, mit über beschränkte Puffer oder unmittelbar zwischen Maschinen realisiertem Werkstücktransport, mit mehreren Auftragstypen bzw. Produkt-Varianten, mit Montage- bzw. Demontageaktivitäten sowie mit fehlerbehafteten Abläufen.

Aus der Anforderung der Umsetzung einfacher Spezifikationen in Petrinetze folgt die Zielsetzung der Abbildung des Wissens eines Petrinetzexperten in die Form eines (wissensbasierten) Programmes. Die grundsätzliche Realisierbarkeit eines solchen Programmes erscheint deshalb plausibel, weil ein Petrinetzexperte bei der Generierung eines Petrinetzes aus Spezifikationen der Fertigungsfachleute nach bestimmten Generierungsregeln vorgeht. Die Schwierigkeit ist, daß die von einem Petrinetzfachmann „benutzten" Regeln und sonstigen Modelle zunächst nur implizit als Expertenwissen vorhanden sind. Eine Umsetzung in ein Programm erfordert die Ermittlung der Regeln in expliziter Form und eine Modellierung des nicht-regelbasierten deskriptiven Wissens sowie eine Strukturierung dieses Wissens sowie der Petrinetzmodellierung. D.h. zunächst muß gefragt werden, wie wir als Petrinetzexperten aus einer Spezifikation eines Facharbeiters oder Betriebsingenieurs ein Petrinetz

generieren. Die Ergebnisse der hierzu durchgeführten Untersuchungen und das daraus entwickelte und implementierte Programmsystem sind Gegenstand von Kapitel 2, ebenso wie ein exemplarischer Fertigungsprozeß, eine Softwareduplizierzelle (Testzelle), die zur Verifizierung der Ansätze und zur Illustration der Einzelschritte in dieser Arbeit dient. Das in Kapitel 2 dargestellte Programmsystem generiert übersichtliche Petrinetze basierend auf einer Aufspaltung in eine vom Systemingenieur zu implementierende Anlagenbeschreibung und eine vom Facharbeiter vorgebbare Anlagennutzung. Es werden die auf der Ebene der Anlagenbeschreibung und der Ebene der Petrinetz-Anlagensteuerung notwendigen Informationen und geeigneten Basis-Strukturen erläutert. Darüberhinaus wird zur verbesserten Strukturierung und Steigerung der Übersichtlichkeit der Petrinetz-Anlagensteuerung in Kapitel 3 ein neu entwickeltes Verfahren zur automatischen Bildung hierarchischer Petrinetze erläutert.

Scheduling-Algorithmen und Verklemmungsbehebungsverfahren für die generierten Petrinetze werden anschließend in Kapitel 4 beschrieben. Hier werden lokale Suchverfahren erläutert und mit einigen heuristischen Prioritätsregeln verglichen, die optimierte Schaltfolgen zur Überführung eines Petrinetzes von einer Anfangs- in eine Zielmarkierung ermitteln. Das in dieser Arbeit gewählte Optimierungskriterium ist der Makespan. Als Basis dienen Petrinetzmodelle, die für das Scheduling durch das Petrinetzgenerierungssystem ebenfalls automatisch erzeugt werden und sich von den Petrinetzen zur Steuerung im wesentlichen dadurch unterscheiden, daß die den Fertigungs- und Transportoperationen zugeordneten Ausführungszeiten im Petrinetz modelliert und somit die Rückmeldungen der unterlagerten Ebenen an die Petrinetzsteuerung (z.B. „Operation fertig") durch die Zeitbewertungen ersetzt werden.

Kapitel 5 faßt die Arbeit zusammen und bewertet die Ergebnisse.

2 Umsetzung abstrakter Bedieneranweisungen in Petrinetz-Anlagensteuerungen

Zunächst wird die zugrundeliegende Steuerungs- und Bedienarchitektur vorgestellt. Danach wird die Testzelle kurz beschrieben. Es folgen in den Abschnitten 2.3 bis 2.7 ausführliche Erläuterungen zu den einzelnen Komponenten des Petrinetzgenerierungssystems. Dieses System besteht aus dem die Fertigungsressourcen beschreibenden und vom Systemingenieur zu implementierenden Ressourcen- und Operationenmodell sowie dem die Fertigungsabläufe beschreibenden und vom Facharbeiter vorgebbaren Auftragsmodell. Das Ressourcenmodell (Abschnitt 2.4) ist übersichtlich und objektorientiert konzipiert. In Abschnitt 2.5 wird die Grundstrukturierung der Petrinetzebene erläutert. Abschnitt 2.6 befaßt sich mit der Auftragsmodellierung mittels Auftragsgraphen. Abschnitt 2.7 fokussiert auf die Petrinetzgenerierung. Eine Erweiterung dieses Systems um Verfahren zur über die Grundstrukturierung hinausgehenden automatischen Bildung hierarchischer Petrinetze ist dann Untersuchungsgegenstand des nachfolgenden dritten Kapitels.

2.1 Die Steuerungs- und Bedienarchitektur

Die Architektur besteht aus einem hierarchischen Grundgerüst sowie Erweiterungen hierzu, die Forschungsgegenstand dieser Arbeit sind (Bild 1). Das Grundgerüst entspricht einer Steuerungsarchitektur nach dem Stand der Forschung (siehe hierzu z.B. [Rehkopf und Krebs 92], [Kusiak 87]). Es umfaßt die in Bild 1 unschraffiert gezeichneten Elemente. Zentraler Bestandteil der Erweiterungen ist ein System zur Umsetzung von Bedieneranweisungen in Petrinetze (im folgenden Petrinetzgenerierungssystem genannt).

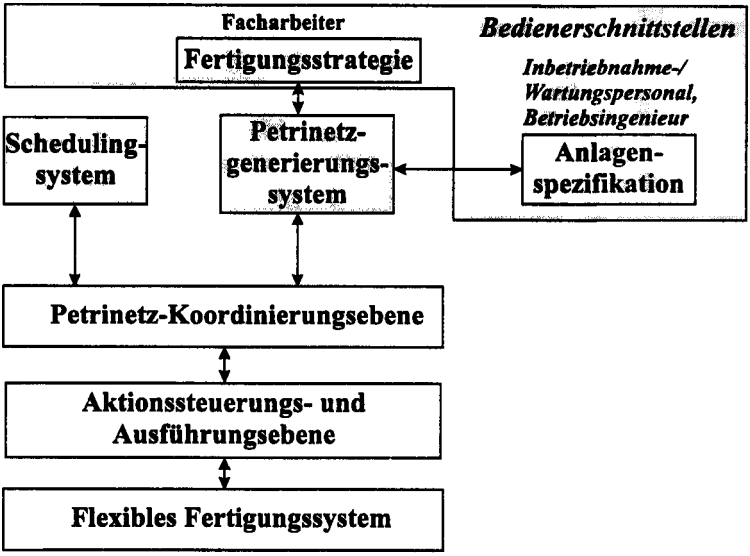


Bild 1: Die zugrundegelegte Steuerungsarchitektur. Die hervorgehobenen Komponenten sind Untersuchungsgegenstand der Arbeit.

Das Grundgerüst besteht auf der untersten Ebene aus einem flexiblen Fertigungssystem. Darüber liegt eine Aktionssteuerungs- und Ausführungsebene. Die Ausführungsebene umfaßt Steuerprogramme zur Ausführung von Elementaroperationen (Primitiva) wie z.B. „Bewegung eines Objektes von einer Palette in eine Maschine durch einen Roboter". Die Elementaroperationen werden von einer Koordinierungsebene kommandiert und koordiniert. Eine Koordination ist notwendig, da zwischen den Aktionen im allgemeinen ein kausaler Zusammenhang besteht und die im Fertigungssystem vorhandenen Betriebsmittel wie z.B. Roboter oder Maschinen - diese werden im folgenden als Ressourcen bezeichnet - in ihrer Kapazität beschränkt sind. Auf der Koordinierungsebene werden Petrinetzmodelle verwendet. Die Koordinierungsebene steuert auch fehlerbehaftete Abläufe. Dazu werden auf der Ausführungsebene klassifizierte Fehlerzustandsmeldungen in die Ablaufkoordinierung einbezogen.

Das Scheduling-System bestimmt für ein gegebenes (zeitbehaftetes) Petrinetz mit Anfangs- und Zielmarkierung eine verklemmungsfreie Schaltfolge mit möglichst geringem Makespan. Das Scheduling-System geht hierbei von einem fehlerfreien Fertigungsablauf aus, d.h. im Petrinetz eventuell modellierte und bei Auftreten von Fehlerereignissen bei der Steuerung aktivierte Fehlerbehandlungsabläufe werden nicht berücksichtigt. Die für ein solches Scheduling-System neu entwickelten Verfahren werden in Kapitel 4 behandelt.

Die Petrinetzgenerierungskomponente ist über Bedienerschnittstellen mit dem Anwender verbunden und setzt dessen Spezifikationen in Petrinetze um. In Bild 1 wird von der realistischen Annahme ausgegangen, daß Anwender unterschiedlicher Qualifikation dieses System bedienen. Die einfache Vorgabe von Abläufen - d.h. die Eingabe der Operationssequenzen für die Fertigungsaufträge bzw. Jobs [4]- kann von einem Anwender mit geringer Qualifikation durchgeführt werden (**Facharbeiter/Meister** aus einer Fertigungsgruppe).
Die Spezifikation der Systemhardware, die in die Petrinetzgenerierung eingeht und immer dann geändert werden muß [5], wenn die Hardware einer Anlage modifiziert wird, ist von einem Anwender höherer Qualifikation durchzuführen. Dieser muß aber weder Petrinetz- noch Programmierkenntnisse in einer klassischen Programmiersprache wie z.B. C oder C++ besitzen. Die Systembenutzer mit einer solchen Qualifikation werden im folgenden unter dem Begriff **Inbetriebnahme-/Wartungspersonal** bzw. **Systemingenieur** zusammengefaßt. **Systemspezialisten** kennen sich darüberhinaus mit Petrinetzen und im Detail mit dem Petrinetzgenerierungssystem aus und sind vor Ort normalerweise nicht verfügbar.
In dieser Arbeit wird bei der Beschreibung verschiedener Aspekte der Bedienerschnittstellen jeweils die erforderliche Mindestqualifikation angegeben. Diese Angaben sind aufwärtskompatibel, d.h. die Bediener mit höherer Qualifikation können die Aufgaben der weniger qualifizierten Anwender übernehmen.

Die schraffierten Komponenten in Bild 1 sind Hauptgegenstand der durchgeführten Untersuchungen und werden deshalb näher betrachtet. Zur Erläuterung der Petrinetze der

[4]Z.B. folgende Sequenz: (1): Greife Werkstück von Puffer und lege es in die Aufnahme von Förderband. (2): Transportiere Werkstück auf Förderband zur Station 1. (3): Bestücke Station 1. usw.
[5]Hardwaremodifikationen werden beispielsweise zur Erhöhung der Fertigungskapazität (Erweiterung einer Maschinengruppe) oder im Zuge eines kontinuierlichen Verbesserungsprozesses (KVP) durchgeführt. Der modulare Aufbau flexibler Fertigungssysteme erleichtert solche Erweiterungen. Die räumlichen Randbedingungen für Erweiterungen müssen schon bei der Neuplanung einer Anlage berücksichtigt werden.

Koordinierungsebene wird auf die Literatur verwiesen ([Abel 90], [Schnieder 92], [Hanisch 92 b], [Quäck 88]).

2.2 Eine Testzelle als Erläuterungsbeispiel

Zur Erläuterung der Konzepte und Verfahren wird in dieser Arbeit eine Testzelle herangezogen, deren Aufbau und Arbeitsweise hier grob angegeben wird. Es handelt sich um eine zu Demonstrationszwecken im Rahmen dieser Arbeit aufgebaute Softwareduplizierzelle. Der Unterschied zwischen der in Bild 2 skizzierten und der tatsächlich vorhandenen Zelle besteht lediglich darin, daß in der realen Anlage ein Roboter die Aufgaben der beiden Roboter *rob1* und *rob2* übernimmt (vgl. [Strege und Tolle 93]). Die reale Anlage besteht aus Hard- und Softwarekomponenten der Firma Siemens.

Die Testzelle arbeitet in folgender Weise:
3,5''-Disketten befinden sich in Puffer1 der Duplizierzelle. Der Roboter *rob1* legt die Disketten einzeln auf die im Eingangspuffer von Förderband 1 (EP1) liegenden Werkstückträger. Über Förderband 1 werden die Disketten zum Eingang von Förderband 2 (EP2) transportiert. Die Sensoren S_1 bzw. S_2 erkennen leere Werkstückträger am Eingangspuffer von Förderband 1 bzw. volle Werkstückträger am Eingang von Förderband 2. Förderband 1 hat eine Kapazität von 2, d.h. es können maximal 2 Werkstückträger gleichzeitig auf dem Band befördert werden. Roboter 2 transportiert die Disketten in die Dupliziermaschinen M1 bzw. M2. Das Starten der wartenden Werkstückträger an EP1 bzw. EP2 erfolgt durch pneumatisch betätigte Aktoren A1 bzw. A2, die durch entsprechende Signale der Koordinierungsebene angesteuert werden.

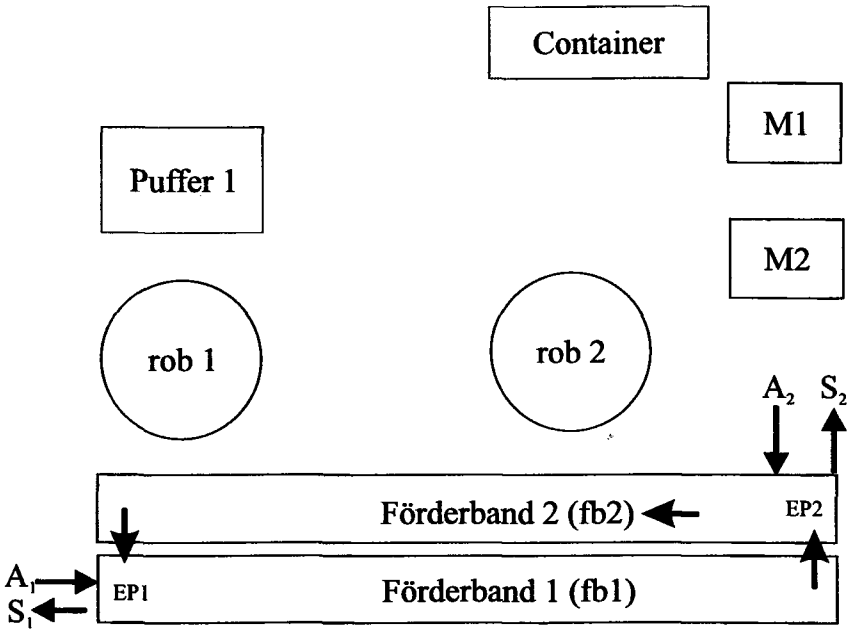


Bild 2: Die Testzelle. Disketten werden auf Förderband 1 von EP1 zum Eingangspuffer EP2 von Förderband 2 befördert. D.h. das Umsetzen von fb1 auf fb2 erfolgt automatisch.

2.3 Die Bedienungsfunktionen und Systemkomponenten im Überblick

Bild 3 veranschaulicht die Funktionsweise des Gesamtsystems anhand eines Flußdiagramms. Bei der Inbetriebnahme einer Fertigungszelle entwirft der Systemingenieur ein hierarchisches Modell der Fertigungszelle. Er orientiert sich dabei an der physikalischen Struktur der Zelle und faßt die elementaren, ausführenden Ressourcen (Roboter, Maschinen etc.) zu Ressourcengruppen zusammen und entwickelt so ein hierarchisches, übersichtliches Modell der Fertigungszelle. Anschließend ordnet er die so definierten Ressourcen jeweils einer Klasse in einem a priori vorhandenen Klassenmodell der Ressourcen zu.

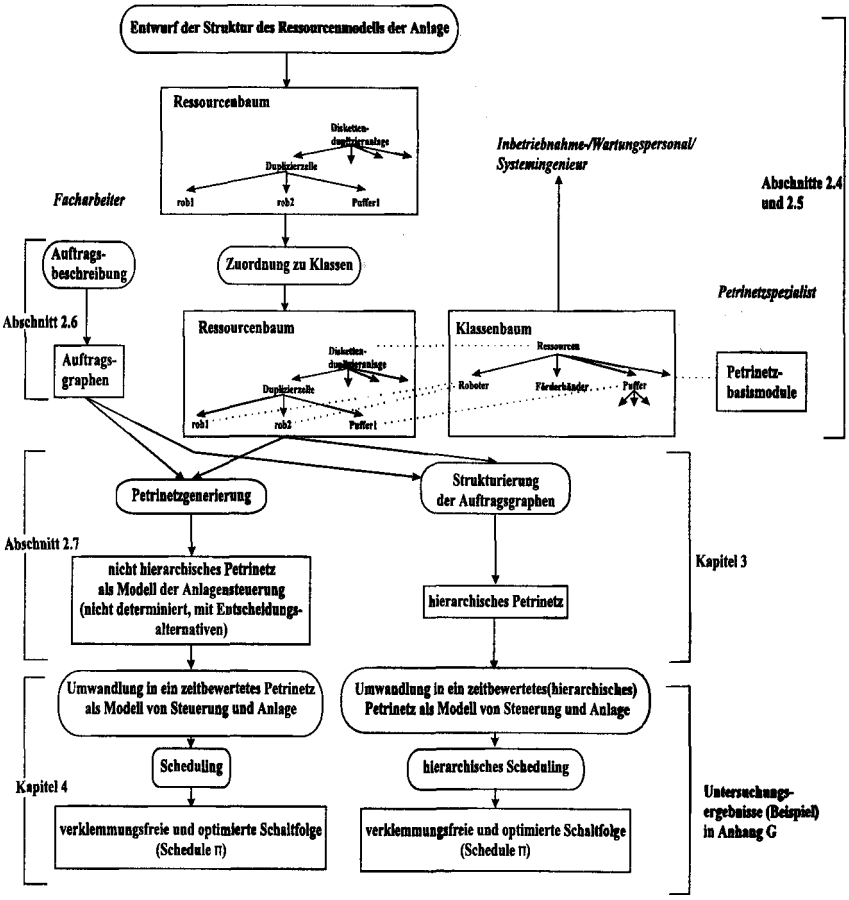


Bild 3: Die Bedienung und Funktionsweise des Systems im Überblick. Petrinetzgenerierung bezeichnet den Vorgang der automatischen Bildung von Petrinetzen unter Zugriff auf die angegebenen Modelle.

Die Ressourcen im Ressourcenmodell erben die Eigenschaften der Klassen. Darüberhinaus können den Ressourcen im Ressourcenbaum individuelle Eigenschaften zugewiesen werden. Zu den aus dem Klassenmodell vererbten Eigenschaften zählen auch die von den Ressourcen ausführbaren Operationen (wie z.B. *Objekt greifen und ablegen* oder *Werkstück bearbeiten*),

die jeweils wiederum einem Operationstypen zugewiesen sind, der ihre Repräsentation im Petrinetz in Form sog. Petrinetzbasismodule vorgibt. Ein Facharbeiter kann die auszuführenden Fertigungsaufträge auftragsorientiert in Form von Auftragsgraphen[6] beschreiben und nach der Inbetriebnahme der Anlage bei Bedarf ändern.

Ein Petrinetzgenerierungsprogramm synthetisiert anschließend entweder ein nicht hierarchisches oder ein transparenteres, für einen Petrinetzexperten einfacher lesbares hierarchisches Petrinetz der Anlagensteuerung. Vor der Synthese eines hierarchischen Petrinetzes erfolgt eine hierarchische Strukturierung der Auftragsgraphen. In den generierten Petrinetzen enthaltene Entscheidungsalternativen werden nach der Umwandlung in zeitbewertete Petrinetze als Modelle von Steuerung und Anlage durch die Scheduling-Komponente gelöst, indem vorab eine optimierte Schaltfolge ermittelt und anschließend zur Anlagensteuerung geschaltet wird.

Die zur Petrinetzsynthese erforderlichen Komponenten werden in den folgenden Abschnitten einzeln und ausführlich erläutert. Die zur Bestimmung optimierter Schaltfolgen entwickelten Scheduling-Verfahren sind Gegenstand von Kapitel 4.

[6]Ein Auftrag bzw. Job beschreibt z.B. die zur Herstellung eines Produktes auszuführenden Arbeitsabläufe.

2.4 Beschreibung der Ressourcen

In dem implementierten System besteht datenmäßig die in Bild 4 dargestellte Abhängigkeit der genannten Modelle: Das Ressourcenmodell geht durch Instanziierung aus dem Klassenmodell hervor. Die im Ressourcenmodell definierten oder aus dem Klassenmodell geerbten Operationen werden in einem Operationenmodell gesammelt. Aus diesem kann der Facharbeiter Operationen auswählen und zu einem Auftragsmodell verknüpfen.

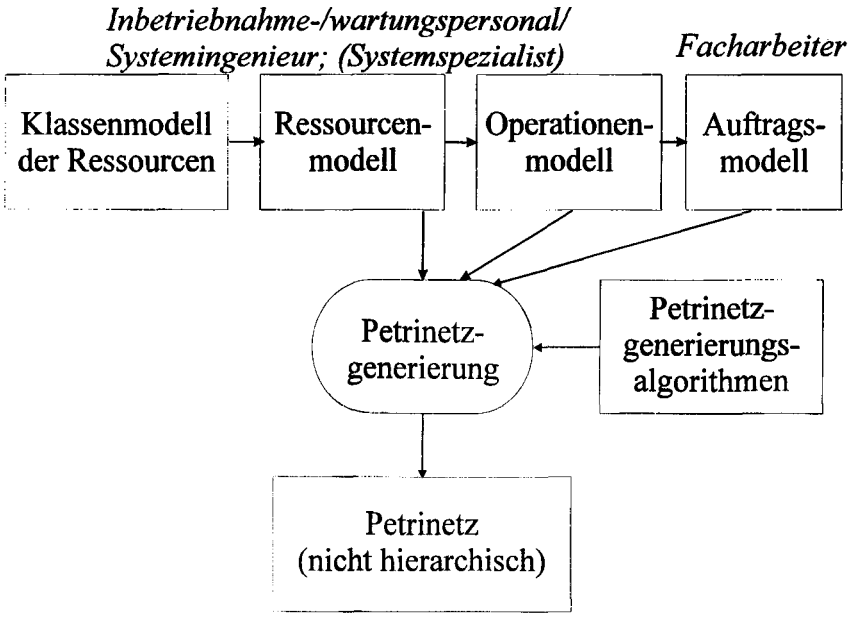


Bild 4: Die Komponenten des Petrinetzgenerierungssystems in ihrer datenmäßigen Abhängigkeit.

2.4.1 Klassenmodell der Ressourcen

In Fertigungsanlagen werden eine Vielzahl unterschiedlicher Transport- und Bearbeitungseinrichtungen eingesetzt. Diese Ressourcen lassen sich hinsichtlich bestimmter Kriterien in Klassen einteilen. Zu diesen Klassen gehören jeweils Ressourcen mit gemeinsamen Eigenschaften bzw. Attributen. Bei genauerer Betrachtung sind Klassenhierarchien erkennbar. *Fräsmaschinen* bilden z.B. mit ihrem die ausführbare Operation beschreibenden Attribut *Operation=Fräsen* eine Unterklasse der Klasse der *Maschinen*, die alle das Attribut *Operation=Werkstück bearbeiten* besitzen.
Eine hierarchische Klassifizierung der Ressourcen erhöht die Transparenz der Modellierung und reduziert die Mehrfachspeicherung gleicher Information. Darüberhinaus entspricht sie den Vorstellungen über die Wissensrepräsentation eines Petrinetzexperten. Weiterhin ist davon auszugehen, daß sich ein Petrinetzexperte bei der Modellierung einer Fertigungsaktivität bzw. der Zustandsmodellierung einer Ressource an der Klassenzugehörigkeit der an der Aktivität beteiligten Ressourcen orientiert.

Die in Bild 5 gezeigten Basisklassen können durch Ableitung weiterer Klassen zu einer anwendungsspezifischen Klassenbibliothek erweitert werden. Jeder Basisklasse sind standardmäßig *Attribut-Wert-Paare* zugeordnet, die zur Petrinetzgenerierung benötigte Eigenschaften festlegen. Die Attribute werden durch das Klassenmodell hindurch vererbt (der Vererbungsmechanismus wird bei der Erläuterung des Ressourcenmodells genauer erläutert). Zur Petrinetzgenerierung werden folgende Attribute benötigt: *Zustand*, *Anfangszustand*, *Operation* und *Zeitdauer*. Durch das Attribut *Zustand* werden die für die Steuerung benötigten Zustände definiert (z.B. Maschine ist bereit, eine Operation auszuführen).

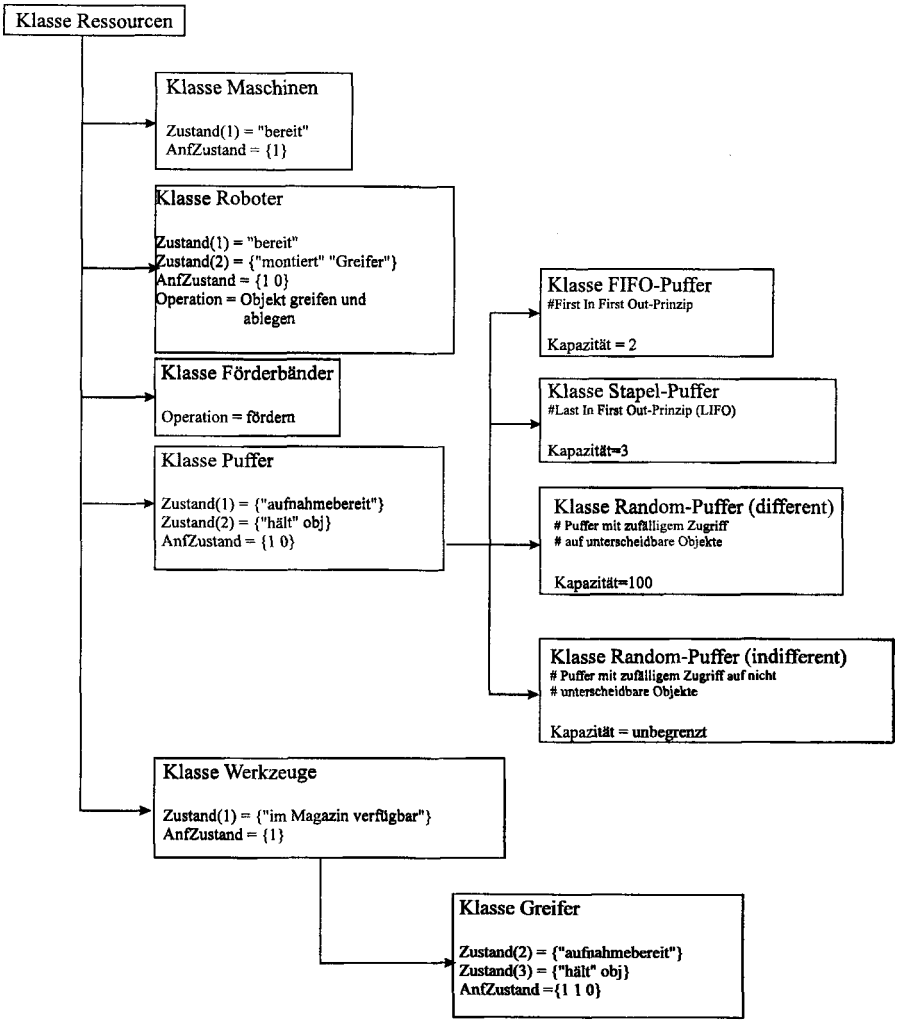


Bild 5: Klassenmodell der Ressourcen: Hier sind exemplarisch die für die Testzelle implementierten Basisklassen abgebildet. Das Zeichen '#' leitet jeweils Kommentarzeilen ein. *Indifferente Random-Puffer* können auch eine begrenzte Kapazität besitzen. Alle Attribute sind über ein Bediener-Menü definier- und änderbar. Bei dem Attribut *Zustand* handelt es sich um einen Vektor. Das Attribut *AnfZustand* ist ein Vektor mit derselben Dimension wie *Zustand*.

Die Werte des Attributvektors *Anfangszustand* legen die Anfangswerte dieser Zustände fest, die im Einzelfall angepaßt werden können. Das Attribut *Operation* kann mehrere Werte annehmen und beschreibt die Operationen, die eine Ressourcenklasse ausführen kann. Der Wert des Attributes *Zeitdauer* spezifiziert die Ausführungszeiten von Operationen.
Die Attribute mit der Bezeichnung *Zustand* werden zunächst von einem Petrinetzexperten im Klassenmodell definiert. Zustandsattribute können über einen Bedienerdialog auch durch Wartungspersonal ergänzt werden. Wird z.B. der Werkzeugsatz eines Roboters erweitert, ergänzt der Bediener ein Zustandsattribut in der entsprechenden Roboter-Ressource. *Operationen* können von Inbetriebnahme- und Wartungspersonal definiert werden. Die Werte des Attributvektors *AnfZustand* sowie die *Zeitdauern* kann ein Facharbeiter eingeben.
Der Anwender (Inbetriebnahme/Wartung) kann bei Bedarf beliebige andere Attribute definieren und diesen Werte zuweisen. Diese Attribute gehen nicht in die Petrinetzgenerierung ein, können aber als Information im Klassenmodell im Sinne einer besseren Lesbarkeit nützlich sein (z.B. Information darüber, welche Art von Disketten in einer bestimmten Duplizierstation bearbeitet werden können).

Die Pfeile des Klassenmodells zeigen jeweils von der übergeordneten zur abgeleiteten Klasse. Die Klasse *Maschinen* umfaßt Ressourcen, die Werkstücke bearbeiten können (z.B. Fräsmaschinen oder Bohrmaschinen). Zur Klasse *Förderbänder* gehören z.B. FIFO-Fördersysteme. *Roboter* sind flexible Fördersysteme, die Werkzeuge aus der Klasse *Werkzeuge* benötigen. *Greifer* für Roboter bilden eine Unterklasse der Klasse Werkzeuge, da sie im Unterschied zu normalen Werkzeugen Werkstücke halten bzw. aufnehmen können. Die Klasse *Puffer* umfaßt Werkstückspeicherkomponenten wie Lager, Container, Paletten oder Werkstückträger. Unter Beachtung der Tatsache, daß Werkstücke in beliebiger Reihenfolge, nach dem FIFO- oder Stapelprinzip gepuffert werden können, wurden die Klassen *Random-*, *FIFO-* und *Stapel-Puffer* definiert. Die Klasse der *Random*-Puffer, die dadurch charakterisiert ist, daß Objekte in beliebiger Reihenfolge gespeichert und entnommen werden, spaltet sich in die beiden Klassen *differente* und *indifferente Random*-Puffer auf. In differenten Random-Puffern ist der Anlagensteuerung immer bekannt, wo genau sich welches Objekt befindet und es kann die Entnahme eines bestimmten Objektes kommandiert werden. Bei indifferenten Random-Puffern ist es der Anlagensteuerung nur möglich, irgendein Objekt zu entnehmen.

Dabei bleibt die Identität des Objektes unbekannt. Ein Beispiel für einen solchen Puffer ist ein Feeder für Schrauben oder aber auch ein Ausgangspuffer einer Zelle, in den (Teil-)Produkte abgelegt werden, deren Reihenfolge bzw. Plazierung steuerungsseitig keine Rolle spielt. *Indifferenten Random*-Puffern kann eine unbegrenzte Kapazität zugeordnet werden (den anderen Puffer-Klassen nicht). Dies ist zwar physikalisch nicht möglich, aber steuerungstechnisch unter Umständen sinnvoll (z.B. Ausgangspuffer, der laufend geleert wird und damit praktisch eine unbegrenzte Aufnahmekapazität besitzt).
Das Klassenmodell ist in seiner Struktur ein Klassenbaum. Eine Ressourcenklasse kann höchstens von einer übergeordneten Klasse abgeleitet werden.

2.4.2 Ressourcenmodell

Das Ressourcenmodell beschreibt eine konkrete Fertigungsanlage. Es besteht aus einer Hierarchie von Objekten, wobei jedes Objekt einer im Klassenmodell definierten Ressourcenklasse zugeordnet ist. Das Ressourcenmodell besitzt wie das Klassenmodell eine Baumstruktur, d.h. ein Ressourcenobjekt kann direkt nur zu einem übergeordneten Ressourcenobjekt gehören. Die Objekte des Ressourcenmodells repräsentieren entweder Ressourcengruppen, elementare (ansteuerbare) Ressourcen oder Subressourcen von elementaren Ressourcen.
In Bild 6 ist das Ressourcenmodell der Testzelle skizziert. Die Ressourcengruppen werden durch normale Rechtecke dargestellt. In der Ressourcengruppe *Duplizierzelle 1* werden alle Ressourcen der Testzelle in Form eines übergeordneten Objektes zusammengefaßt. Alle übrigen Objekte sind elementare Ressourcen bzw. deren Subressourcen. Jeder elementaren Ressource bzw. Subressource wird eine Identifikationsnummer zugewiesen (ID_Nr). Das Schalten von Aktoren von Förderbändern wird in der bei *fb1* dargestellten Weise durch Regeln beschrieben. Ein aus mehreren Aufnahmeplätzen zusammengesetzter Puffer (*Puffer 1* bzw. die beiden Förderbänder) für Werkstücke wird im Ressourcenmodell als eine aus Subressourcen zusammengesetzte elementare Ressource abgebildet. Die Subressourcen von *Puffer 1* müssen nicht explizit vom Bediener eingegeben werden. Die Angabe der Kapazität genügt. Die explizite Modellierung der Subressourcen ist z.B. dann sinnvoll, wenn es sich um von Robotern anfahrbare Pufferplätze handelt.

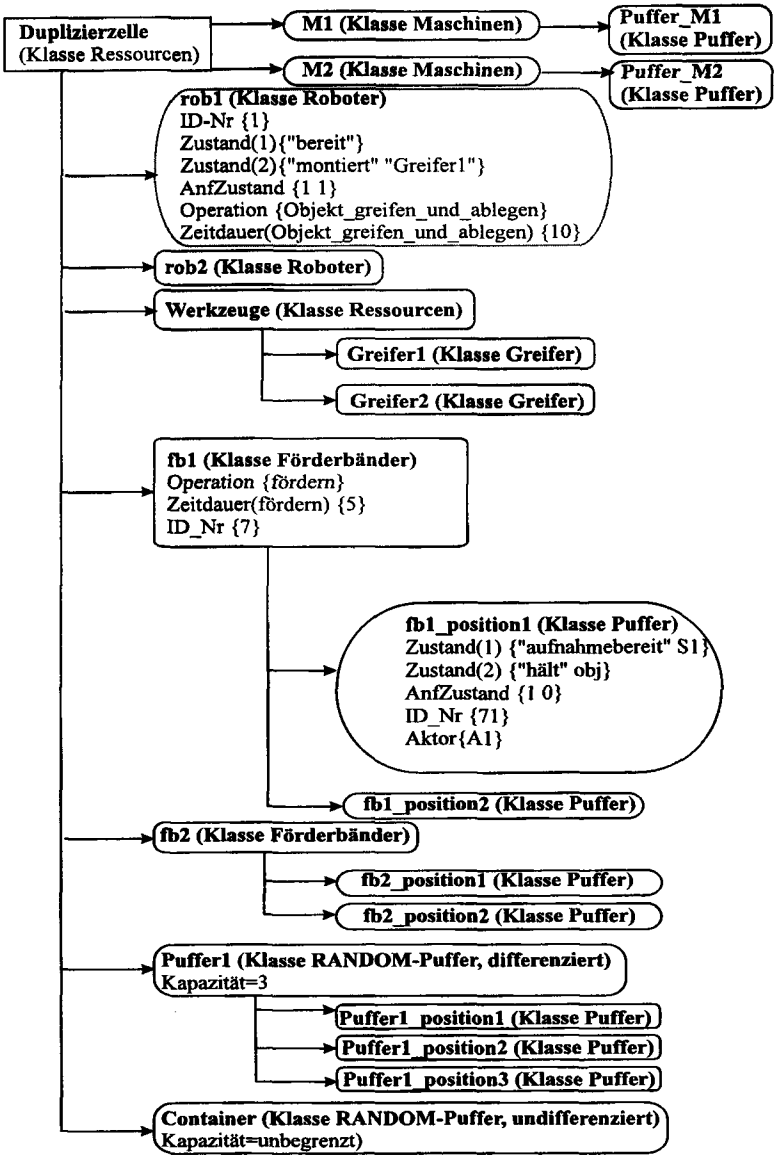


Bild 6: Ressourcenmodell zur Testzelle. Einige Objekte dieses Modells sind exemplarisch ausführlicher dargestellt. Die Spezifikation der Attribute *Zustand*, *AnfZustand*, *Operation* und *Zeitdauer* folgt der gezeigten Syntax.

Elementare Ressourcen und deren Subressourcen werden durch abgerundete Rechtecke symbolisiert. Die Maschinen M1 und M2 sind z.B. elementare Ressourcen, die jeweils eine Subressource der Klasse *Puffer* besitzen. Diese Subressourcen repräsentieren die Werkstückpufferplätze, die zur Diskettenaufnahme dienen.

Vererbungsmechanismus und a priori im System definierte Attribute

Zur Veranschaulichung des Vererbungsmechanismus werden das Klassen- und Ressourcenmodell aus Bild 7 herangezogen.
Die Objekte des Ressourcenmodells erben sowohl die Attribute der im Ressourcenmodell übergeordneten Objekte als auch die Attribute ihrer Klasse, die wiederum von übergeordneten Klassen vererbt sein können. Deshalb wird bei der Vererbung zunächst das Klassenmodell und anschließend das Ressourcenmodell betrachtet. Klasse C und D erben das Attribut K=5 von Klasse B. Klasse D erbt zusätzlich S=600 von Klasse C.

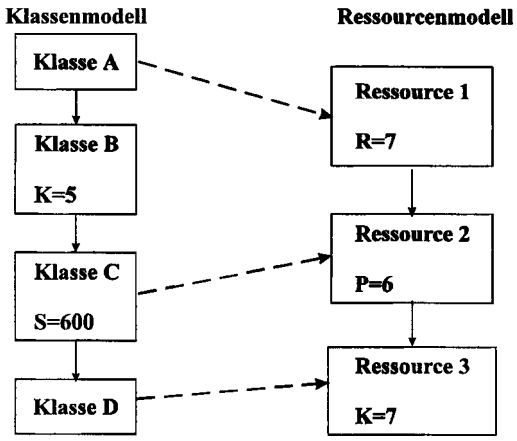


Bild 7: Zur Erläuterung des Vererbungsmechanismus. Im Ressourcenmodell nehmen die Subressourcen von elementaren Ressourcen eine Sonderstellung ein. Sie erben nicht von der im Ressourcenmodell übergeordneten elementaren Ressource (Beispiel: Die Subressource Pufferplatz einer Maschine erbt nicht die Attribute der Maschine).

Ressource 1 erbt die Attribute von Klasse A (im Beispiel keine). Ressource 2 erbt R=7 von Ressource 1 sowie S=600 und K=5 von Klasse C. Ressource 3 erbt R=7, S=600 und K=5 und P=6 von Ressource 2. Der Wert des Attributes K wird in Ressource 3 auf 7 geändert.

Im Klassenmodell bzw. Ressourcenmodell gibt es die bereits erwähnten Attribute mit fest vorgegebener Bedeutung. Die Werte dieser Attribute werden bei der Petrinetzgenerierung benötigt bzw. ausgewertet. Der Systembenutzer muß die Bedeutung der Attribute kennen und darf deren Namen nicht in einem anderen Zusammenhang verwenden. Die Bedeutung der Attribute sowie ihre genaue Spezifikation lautet im Detail:

Operation: Das Attribut legt fest, welche Operationen eine Ressource ausführen kann. Einer Ressourcenklasse bzw. einem Objekt im Ressourcenmodell werden über dieses Attribut die ausführbaren Operationen zugewiesen. Diese Operationen sind durch die Konstruktion der Hardwarekomponenten wie Roboter usw. vorgegeben. Das Attribut kann mehrere Werte gleichzeitig annehmen. Dies entspricht der Tatsache, daß eine Ressource mehrere Operationen ausführen kann.

Zustand: Das Attribut Zustand ist ein Vektor und definiert Zustände von Ressourcen. Hierbei kann es sich um rein logische Zustände oder um direkt sensorüberwachte Zustände handeln. Wird die Anwesenheit eines Objektes auf einem Pufferplatz durch z.B. eine Lichtschranke überwacht, so wird dies bei der Zustandsdefinition entsprechend berücksichtigt. Die Syntax der Zustandsdefinition ist *Zustand <Nummer><Prädikat><Objekt><Sensor>*. Die in Klammern gesetzten Elemente sind Platzhalter. *Prädikat* beschreibt den Zustand (z.B. *Puffer hält). Objekt* und *Sensor* müssen nicht definiert werden. Der Eintrag *Objekt* gibt an, ob sich das Prädikat auf Flußobjekte[7] bezieht, und durch die Angabe von *Sensor* wird der den Zustand überwachende Sensor spezifiziert.

AnfZustand: Dieses Attribut definiert den Anfangszustand von Ressourcen. Der Wert ist hier ein Vektor $[z_1, z_2, ...z_n]$, dessen Komponenten 1 oder 0 sind, je nachdem, ob der betreffende Zustand erfüllt ist oder nicht. Die Komponenten dieses Vektors legen die Anfangswerte der im Attribut *Zustand* definierten Ressourcenzustände fest. Als Beispiel wird der Roboter *rob1*

[7]Flußobjekte sind z.B. Werkstücke

mit montierbarem Greifer betrachtet. Der Eintrag AnfZustand {1 1} bedeutet, daß der Roboter zu Beginn bereit und Greifer 1 montiert ist.

Zeitdauer: Durch das Attribut *Zeitdauer* werden die Ausführungsdauern der im Attribut *Operation* definierten Operationen in Zeiteinheiten angegeben. Für jede Operation ist im Ressourcenobjekt folgendes einzutragen: *Zeitdauer* (<*Operationsname*>) {<*Wert*>}.

ID_Nr: Dies ist die Identifikationsnummer einer Ressource. Die Nummern können durchgängig bis in die Ausführungsebene hinein benutzt werden. Beispielsweise kann die Identifikationsnummer eines Puffers direkt als Parameter einer Steuerungsprozedur der Ausführungsebene verwendet werden. Die Identifikationsnummern sollten sich deshalb an der Realisierung der Ausführungsebene orientieren.

Kapazität: Dieses Attribut definiert die Aufnahmekapazität von Puffern. Der Standardwert beträgt 1, wenn der Systemingenieur den Wert nicht explizit angibt und einem Puffer keine Subressourcen-Puffer zugeordnet sind. Fehlt die explizite Angabe und gibt es Subressourcen, ergibt sich die Kapazität aus der Anzahl der Sub-Pufferplätze des zusammengesetzten Puffers.

2.5 Operationenmodell und Petrinetzbasismodule

Im vorangehenden Abschnitt wurde ein objektorientiertes Prinzip der Modellierung von Fertigungsressourcen vorgestellt. Das Attribut *Operation* nimmt dabei eine Sonderstellung ein. Es beschreibt die Operationen, die von einer Ressource ausgeführt werden können und beinhaltet somit die Grundlage dessen, was der Bediener (Facharbeiter) bei der Ablaufvorgabe von Fertigungsaufträgen als Aktivitäten spezifizieren kann.

Um die Transparenz des Systems zu erhöhen, werden die im Ressourcenmodell definierten Operationen in eine Operationenliste übernommen, die auch als Operationenmodell bezeichnet wird. Jede Operation ist einem Operationstypen zugeordnet. Diese Zuweisung bestimmt, wie die Operationen im Petrinetz in Form von sogenannten Petrinetzbasismodulen abgebildet werden. Es wurde eine Grundmenge von Typen definiert, die durch einen Systemspezialisten erweitert werden kann. Die Operationstypen der Grundmenge sind im einzelnen:

1. Objekt greifen
2. Objekt ablegen
3. Objekt greifen und ablegen
4. Werkzeug montieren
5. Werkzeug demontieren
6. Werkzeug wechseln
7. Werkstück bearbeiten
8. Objekt demontieren
9. FIFO-Speichern
10. LIFO-Speichern

Der Aufbau der den Operationstypen zugewiesenen Petrinetzbasismodule ist für den Bediener aus einer Fertigungsgruppe nicht sichtbar. Ein Wartungsmann kann sich darüber informieren, welche Zustände der Ressourcen mit einem Operationstypen verknüpft und somit verändert werden.

Die Operationen aus dem Operationenmodell stehen dem Bediener zur Beschreibung von Abläufen in Form sog. *Auftragsgraphen* zur Verfügung. Bevor in Abschnitt 2.6 auf diese Beschreibungsebene eingegangen wird, erläutert dieser Abschnitt die Petrinetzbasismodule im Detail.

Die Darstellung der Petrinetzbasismodule erfolgt deshalb so ausführlich, weil ihre Definition Ergebnis grundlegender Überlegungen und eine zentrale Aufgabe bei der Petrinetzgenerierung ist. Zunächst wird von fehlerfreien Abläufen ausgegangen. Am Ende des Abschnitts wird das Prinzip der Erweiterung der Module zur Berücksichtigung von Fehlerzuständen behandelt. Da die Einbeziehung von Zeit zur Simulation und bei bestimmten Systemen auch zur Steuerung von Abläufen (z.B. bei der Überwachung von Prozeßzeiten) notwendig ist, wird die Integration der Zeit in die zunächst zeitfreien Petrinetzbasismodule ebenfalls erörtert.

Die Ausführlichkeit der Erläuterung der Module in diesem Abschnitt ist zunächst zur Darstellung des Modellierungsprinzips groß und nimmt dann bei den weiter unten behandelten Modulen ab.

Das Prinzip, Petrinetze aus einzelnen Petrinetzmodulen zusammenzusetzen, ist bekannt und wird z.B. in [Glüer und Schmidt 88], [Al-Jaar et al. 90], [Camurri und Franchi 90], [Ahmed et al. 93], [Stulle 95], [Zhou und DiCesare 90] und [Zhou und DiCesare 96] beschrieben.

Die genaue Betrachtung von Fertigungssystemen zeigt, daß die konkurrierende Beanspruchung von begrenzt vorhandenen Ressourcen einer ihrer zentralen Aspekte ist. In Petrinetzen kann dieser Aspekt durch das in Bild 8 dargestellte Prinzip des *wechselseitigen Ausschlusses*[8] modelliert werden (siehe hierzu auch [Zhou und DiCesare 91], [Glüer und Schmidt 88]). Bild 8 zeigt den Fall, daß mehrere Prozesse eine Ressource *R* benötigen. Das Prinzip ist auch dann anwendbar, wenn die Ressourcen verschachtelt von verschiedenen Prozessen angefordert werden. Der Platz R modelliert die Verfügbarkeit einer Ressource und besitzt die Kapazität 1. Er wird jeweils mit den Transitionen der Jobs durch eine Prekante verbunden, die zum Schalten eine Marke aus R brauchen.

[8]Häufig wird für dieses Prinzip der englische Ausdruck *mutual exclusion* verwendet.

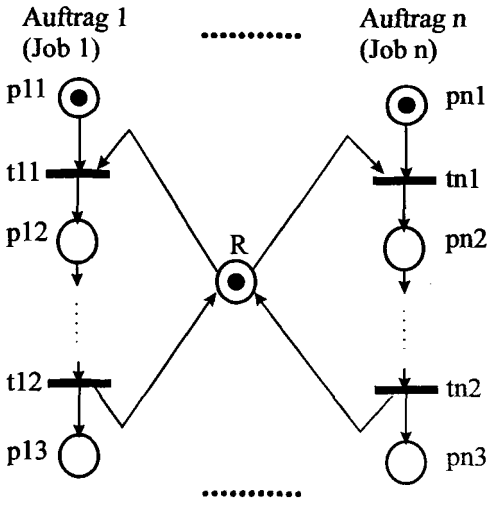


Bild 8: Prinzip des Wechselseitigen Ausschlusses (R kann nicht gleichzeitig von mehr als einem Prozeß benutzt werden (siehe auch [Glüer und Schmidt 88] sowie [Zhou und DiCesare 91]). Als Prozeß wird hier eine sich auf den Jobplätzen *pi* bewegende Auftrags- bzw. Jobmarke verstanden.

Die Anwendung des Prinzips des wechselseitigen Ausschlusses führt zu einer Unterscheidung zwischen Job- und Ressourcenplätzen. Jobplätze repräsentieren die (Zwischen-)Zustände der Jobs (Aufträge). Ressourcenplätze modellieren die Verfügbarkeit bzw. den Zustand von Ressourcen.

Im entwickelten Petrinetzgenerierungssystem wurde das dargestellte Prinzip konsequent umgesetzt. Den elementaren Ressourcen bzw. deren Subressourcen werden entsprechend Bild 9 Ressourcenstellen im Petrinetz zugeordnet.

Bild 9: Zuordnung von Petrinetz-Ressourcenstellen zu den elementaren Ressourcen und deren Subressourcen. Ein als Subressource einer Maschine zugewiesener Puffer (vgl. Bild 6) wird im Petrinetz wie *Puffer1_position_i* mit zwei Stellen modelliert. Eine Ressource der Klasse Förderband muß mindestens eine Subressource der Klasse *Puffer* besitzen.

Die Zuordnung der Ressourcenplätze hängt von den im Klassen- und Ressourcenmodell vorgenommenen Zustandsdefinitionen (Bild 5 und Bild 6) ab. In Bild 9 sind Ressourcen der im System standardmäßig vorhandenen Basisressourcenklassen die zugeordneten Petrinetzstellen gegenübergestellt. Da es sich bei den Blättern des Ressourcenbaumes um elementare Ressourcen oder Subressourcen handelt und diese von einer Basisklasse abgeleitet sein müssen, ist eine Zuweisung von Petrinetzstellen immer eindeutig möglich.

Die zu einer elementaren Ressource bzw. Subressource gehörenden Ressourcenstellen existieren im Petrinetz genau einmal. Ihre Verknüpfung mit den Transitionen des Petrinetzes wird durch die Petrinetzbasismodule vorgegeben, die im folgenden näher erläutert werden.
Die Ressourcenstellen werden für jede Ressource in der gezeigten Weise numeriert. Die Stelle *Roboter.1* bedeutet, daß der Roboter bereit ist. Die Stelle *Roboter.2* gibt an, daß der Roboter ein Werkzeug montiert hat (zu weiteren Werkzeugen gehören die Stellen *Roboter 3* usw.). Jedem Werkzeug wird eine *Stelle Werkzeug.1* zugeordnet, deren Markierung angibt, ob das Werkzeug im Magazin verfügbar ist oder nicht. Ein Greifer wird durch die drei Zustände *Greifer.1* (Greifer im Magazin verfügbar), *Greifer.2* (Greifer ist leer) und *Greifer.3* (Greifer hält Objekt) abgebildet. Die *Stellen Greifer.1* und *Greifer.2* können schwarze, d.h. die von den Stellen/Transitionen-Netzen bekannten, nicht unterscheidbaren Marken aufnehmen, während die Stelle *Greifer.3* Marken aufnehmen kann, die Flußobjekte (Werkstücke) identifizieren. Ein Puffer wird durch zwei Zustände *Puffer.1* und *Puffer.2* modelliert. Die Stelle *Puffer.1* kann schwarze Marken aufnehmen und bedeutet, daß der Puffer frei ist. Die Stelle *Puffer.2* nimmt wie die Stelle *Greifer.3* Flußobjekt-Marken auf.
Die Stellen *Roboter.2*, *Greifer.3*, *Förderband_position_i.2* und *Puffer_position_i.2* sind zur Koordinierung der Abläufe und zum Scheduling (siehe Kapitel 4) nicht unbedingt erforderlich, sondern werden zu Beobachtungszwecken (vgl. [Stulle 95]) und zur Konsistenzprüfung bei der Petrinetzgenerierung (siehe folgenden Abschnitt) benötigt. Mit Einführung der Stellen *Greifer.3*, *Förderband_position_i.2* und *Puffer_position_i.2* ist zu jedem Steuerzeitpunkt unmittelbar ersichtlich, wo sich die Flußobjekte befinden.

Das verwendete Petrinetzparadigma:

Das Petrinetzparadigma entspricht dem der Prädikaten/Transitionen-Netze (siehe z.B. [Baumgarten 90], [Hanisch 92 b]). Die Kanten der in den folgenden Abbildungen enthaltenen Petrinetze sind teilweise mit Variablennamen beschriftet. Es gilt die Vereinbarung, daß Variablen mit gleichen Namen im Vor- und Nachbereich einer Transition sich auf dieselbe Marke beziehen. Alle Kantengewichte betragen eins.
Über die unbeschrifteten Kanten wandern schwarze Marken. Bei den unterscheidbaren Marken handelt es sich um strukturierte Marken. So wird in den durch die Kantenanschrift <Job> bezeichneten Marken die Prozeßhistorie hinterlegt, d.h. es werden z.B. auftretende Fehler protokolliert.
Die mit p_i bezeichneten Stellen sind Jobstellen und besitzen unbeschränkte Kapazität. Die Ressourcenstellen werden mit ***Ressourcenname.Nummer*** bezeichnet und können nur eine Marke aufnehmen. Mit der Zahl ***.Nummer*** werden die zu einer elementaren Ressource gehörenden Ressourcenstellen entsprechend Bild 9 numeriert und damit eindeutig bezeichnet.
Die Prekanten mit einem ausgefüllten Kreis am Ende sind Testkanten (siehe hierzu z.B. [Lunze 92]).
Die Kanten mit den leeren Kreisen am Ende sind Inhibitorkanten ([Lunze 92]). Diese verhindern das Schalten einer Transition, wenn die verbundene Stelle markiert ist.

Jeder Transition ist eine erweiterte Feuerbedingung sowie eine Aktion zugeordnet. Die erweiterten Feuerbedingungen sind logische Bedingungen, deren Wahrheitswert zur Laufzeit ermittelt wird. Die erweiterten Feuerbedingungen können sich z.B. auf die durch den Aufruf von Schnittstellenfunktionen erfaßten Zustände von Sensoren oder auf Rückmeldungen der Ausführungsebene beziehen. Durch das Ausführen der Transitionsaktionen können Operationen bzw. Primitiva der Ausführungsebene angestoßen werden.

Bevor das Petrinetzmodul aus Bild 10 in seinem Aufbau erläutert wird, noch einige Bemerkungen zur Definition von Petrinetzbasismodulen:

1. Eine der Problemstellungen bei der Definition der Petrinetzbasismodule liegt in der Sicherstellung der Konsistenz und der hinreichenden Mächtigkeit zur Modellierung der in praxi vorkommenden Fertigungsaufgaben. Konsistenz meint, daß die Module so miteinander

verknüpft werden, daß physikalisch unerwünschte Zustände nicht auftreten können. Hierzu ein Beispiel: Man könnte einen physikalischen Puffer P1 nur durch eine Ressourcenstelle *P1.2* modellieren und vereinbaren, daß eine Marke auf dieser Stelle der Tatsache entspricht, daß sich ein Objekt auf P1 befindet. Eine Roboteroperation, durch die ein Objekt auf diesen Puffer bewegt wird, würde eine Marke von einem anderen Pufferplatz abziehen und hätte die Voraussetzung, daß die Stelle *P1.2* nicht markiert ist (Inhibitorkante). Eine solche Modellierung wäre nicht konsistent und würde unerwünschte Zustände erlauben: So könnten zwei Roboter gleichzeitig ein Objekt zum Puffer *P1* transportieren und damit kollidieren.
Zur Vermeidung einer solchen Situation wird deshalb jeder Puffer durch zwei Petrinetzstellen modelliert. Eine Marke auf *Puffer.1* bedeutet, daß der Puffer frei ist und ein Transportvorgang mit dem Ziel Puffer gestartet werden kann. Fehlt die Marke, kann ein solcher Transport nicht gestartet werden. Die Stelle *Puffer.2* beschreibt, ob sich ein Objekt auf dem Puffer befindet (die Marke selbst identifiziert dieses Objekt).

2. Sowohl das verwendete Petrinetzparadigma als auch die Gestaltung der Petrinetzbasismodule sind keineswegs eindeutig. Ein Charakteristikum der Petrinetze ist, daß es zu einer Aufgabenstellung stets eine Reihe möglicher Lösungsmodelle mit gleicher Funktionalität und gleichem Verhalten gibt, die ineinander überführbar sind. Obwohl die Petrinetze automatisch generiert werden und damit ihr Umfang zunächst nicht unmittelbar ins Gewicht fällt, wurde in dieser Arbeit auf die Übersichtlichkeit der Modelle Wert gelegt, wobei die Feinheit der Modellierung aus der Erfahrung mit der implementierten Testzelle heraus ausreichend ist.

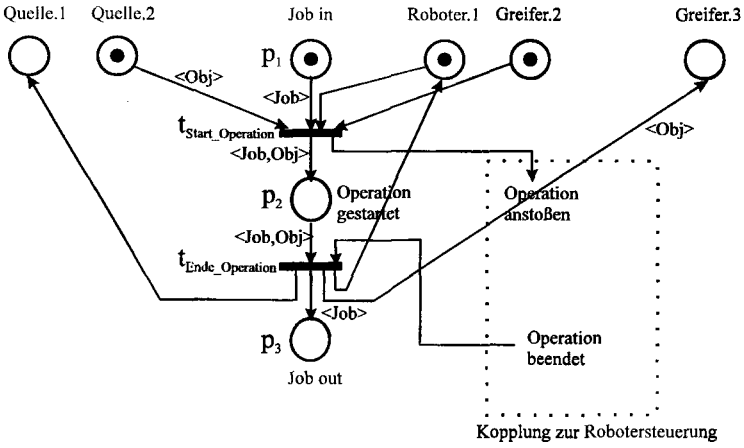
3. Es kann nicht ausgeschlossen werden, daß sich bei der Anwendung des Petrinetzgenerierungssystems auf andere Fertigungszellen herausstellt, daß neue, noch nicht definierte Petrinetzmodule benötigt werden. Dieser Tatsache wird in dem implementierten System dadurch Rechnung getragen, daß es einfach ergänzt werden kann. In einem menügestützten Dialog legt man dazu die Parameter von zusätzlichen Operationstypen und damit die Struktur der zugehörigen Petrinetzmodule fest. Dies kann nur von System- und Petrinetzspezialisten vorgenommen werden.

4. Die Module werden zunächst so dargestellt, wie sie später im Petrinetzmodell der Steuerung verwendet werden. Bei der Generierung zur Konsistenzprüfung zusätzlich in

Betracht gezogene Kanten (dies sind Test- und Inhibitorkanten) werden im folgenden Abschnitt bei der Darstellung des Generierungsvorgangs erläutert.

Petrinetzmodul *Objekt greifen* (Bild 10)

Die Operation *Objekt greifen* wird in diesem Modul durch die Transition $t_{Start\_Operation}$ gestartet. Zum Schalten von $t_{Start\_Operation}$ ist eine Marke auf dem Platz *Quelle.2* erforderlich (Objekt befindet sich in dem mit *Quelle* bezeichneten Puffer)[9]. Darüberhinaus muß der Roboter bereit sein (*Roboter.1* markiert) und der Greifer muß leer sein (*Greifer.2* markiert).



Objekt greifen

Bild 10: Petrinetzbasismodul zum Operationstypen *Objekt greifen*. *pi*: Jobstellen, auf denen strukturierte Marken liegen, in denen die Prozeßhistorie gespeichert wird (Rückmeldungen der unterlagerten Steuerungen). Auf den Stellen *Quelle.2* und *Greifer.3* liegen Marken, die Flußobjekte identifizieren. Die übrigen Stellen nehmen schwarze Marken auf. Über die Kanten *<Job,Obj>* fließen zusammengesetzte Marken, die beim Schalten der Transition $t_{Ende\_Operation}$ wieder in ihre ursprünglichen Bestandteile *Job* und *Obj* zerlegt werden. Die Überprüfung, ob das Werkzeug montiert ist (Kante von *Roboter.2* zur Starttransition), erfolgt bei der Petrinetzgenerierung (vgl. Abschnitt 2.7) und ist im Steuerungsmodell nicht erforderlich.

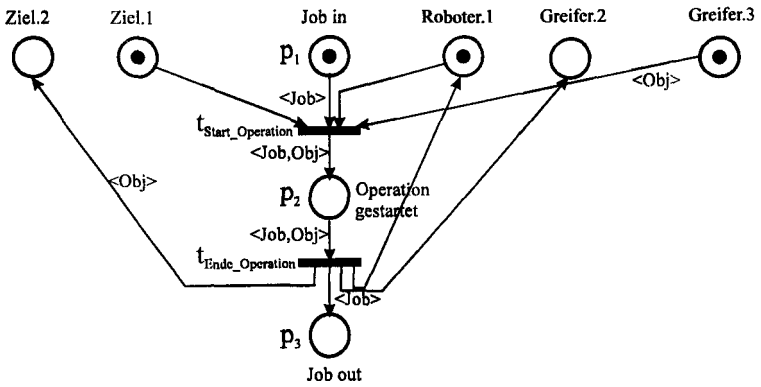
[9] Zur Bezeichnung der Plätze vgl. auch Bild 9.

Beim Schalten von $t_{Start\_Operation}$ werden die Marken aus den Plätzen *Quelle.2*, *p1*, *Greifer.2* und *Roboter.1* entfernt und es wird eine Marke <Job, Obj> auf der Merkerstelle p_2 abgelegt. p_2 modelliert den Zustand, daß ein Objekt (*Obj*) gerade vom Roboter mit einem Greifer gegriffen wird. Schaltet $t_{Start\_Operation}$, wird ein Kommando an die Robotersteuerung gesendet, welches die Greifoperation mit Parametern startet. Nach der Ausführung der Operation und entsprechender Rückmeldung der Robotersteuerung feuert die Transition $t_{Ende\_Operation}$.

Nach der Ausführung der Operation *Objekt greifen* ist das in Bild 10 dargestellte Teilnetz folgendermaßen markiert: Die zuvor in *Quelle.2* vorhandene Marke liegt auf dem Platz *Greifer.3* (entsprechend der physikalischen Situation, daß der Greifer das Objekt hält). Der Platz *Quelle.1* ist markiert, d.h. der Pufferplatz kann wieder beladen werden. Der Roboter ist für eine weitere Operation bereit (*Roboter.1* markiert). Die zuvor auf Platz p_1 vorhandene Marke liegt auf p_3. Die Stellen *Quelle.2*, *Greifer.2* und p_1 sind unmarkiert.

Petrinetzmodul *Objekt ablegen*:

Bild 11 zeigt das Petrinetzmodul *Objekt ablegen* und modelliert den Vorgang, daß ein sich im Greifer eines Roboters befindliches Objekt auf einen Puffer mit der Bezeichnung Ziel abgelegt wird.



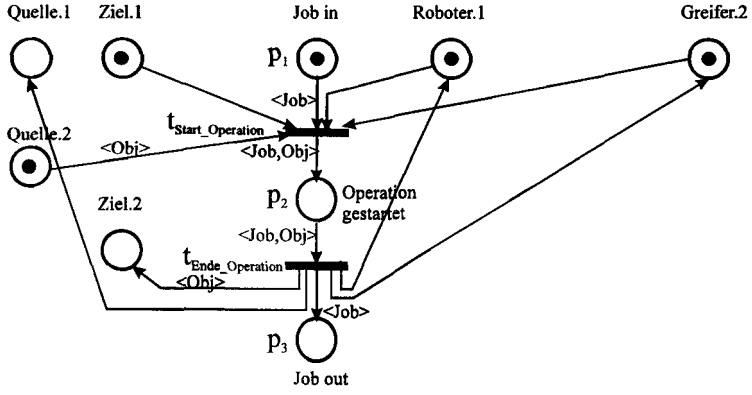
Objekt ablegen

Bild 11: Petrinetzbasismodul zum Operationstypen *Objekt ablegen*.

Voraussetzung für das Schalten der Starttransition $t_{Start\_Operation}$ ist, daß der Zielpuffer frei ist (*Ziel.1* markiert), der Roboter bereit (*Roboter.1* markiert), der Greifer ein Objekt hält (*Greifer.3* markiert) und die Operation angefordert wird (p_1 markiert).

Petrinetzmodul *Objekt greifen und ablegen:*

Wird das Greifen und Ablegen eines Objektes in einer Fertigungszelle durch zwei verschiedene Kommandos von der Koordinierungsebene an die Ausführungsebene kommandiert, ergibt sich die entsprechende Petrinetzdarstellung durch Zusammenschalten der Petrinetzmodule *Objekt greifen* und *Objekt ablegen*. Kann der *Objekt greifen- und ablegen*-Vorgang durch ein einziges Kommando gestartet werden, wird das in Bild 12 gezeigte Petrinetzmodul verwendet. Die Interpretation der Netzelemente erfolgt entsprechend Bild 10 und Bild 11.



Objekt greifen und ablegen

Bild 12: Petrinetzbasismodul *Objekt greifen und ablegen.*

Anhang A enthält die Petrinetzbasismodule zu weiteren Operationstypen.

2.6 Auftragsgraphen zur Vorgabe von Fertigungsabläufen

Fertigungsabläufe lassen sich auf einer abstrakten Ebene unter Vernachlässigung der zur Koordinierung der Abläufe notwendigen und in Petrinetzmodellen abgebildeten Detailinformationen mit Hilfe von Graphen sehr anschaulich beschreiben und vorgeben (siehe hierzu u.a. [Ezpeleta et al. 93 a und b] oder [Matthiesen 95]). Graphen haben gegenüber einer regelbasierten Beschreibung den Vorteil einer höheren Übersichtlichkeit. Sie liegen deshalb z.B. den für Anwender ohne Programmierkenntnisse konzipierten IDEF-Modellen als Beschreibungssprache zugrunde (IDEF: **I**ntegrated **C**omputer-Aided **M**anufacturing **DEF**inition, vgl. [Mayer et al. 92] und [U.S. Airforce 81]).

In dieser Arbeit werden durch sog. *Auftragsgraphen* Abläufe grob vorgegeben. Die zur Erfüllung eines Auftrages bzw. zur Herstellung eines Produktes notwendigen Arbeitsabläufe werden getrennt durch je einen Auftragsgraphen spezifiziert (siehe zu der getrennten Vorgabe auch [Ezpeleta et al. 93 a und b] und [Reuter 95]). Die Knoten der Auftragsgraphen sind Operationen und die Kanten zeigen jeweils auf die Nachfolgeoperationen (Bild 13).

Es gibt praktisch keinen Zweifel darüber, daß ein Facharbeiter Abläufe durch die Verknüpfung von Grundoperationen zu Graphen beschreiben kann.
Bild 14 zeigt als Beispiel einen Auftragsgraphen, der einen in der Testzelle auszuführenden, fehlerbehafteten Ablauf beschreibt. Den Operationen können menügestützt Parameter zugeordnet werden. Bei den Operationen *Objekt greifen und ablegen* eines Roboters sind dies der Quellpuffer, von dem ein Werkstück gegriffen und der Zielpuffer, auf dem es abgelegt werden soll. Hierbei handelt es sich um im Ressourcenmodell modellierte Pufferplätze (vgl. Bild 6). Auch zur Operation *Fördern* eines Förderbandes werden der Quell- und Zielpuffer angegeben. Diese Pufferplätze sind als Subressourcen des Förderbandes oder eines sich anschließenden Bandes (vgl. Bild 2) modelliert. In dem implementierten System ist es möglich, daß ein Förderband auf den Startpuffer eines Folgebandes fördern kann (siehe Operation 2 in Bild 14).

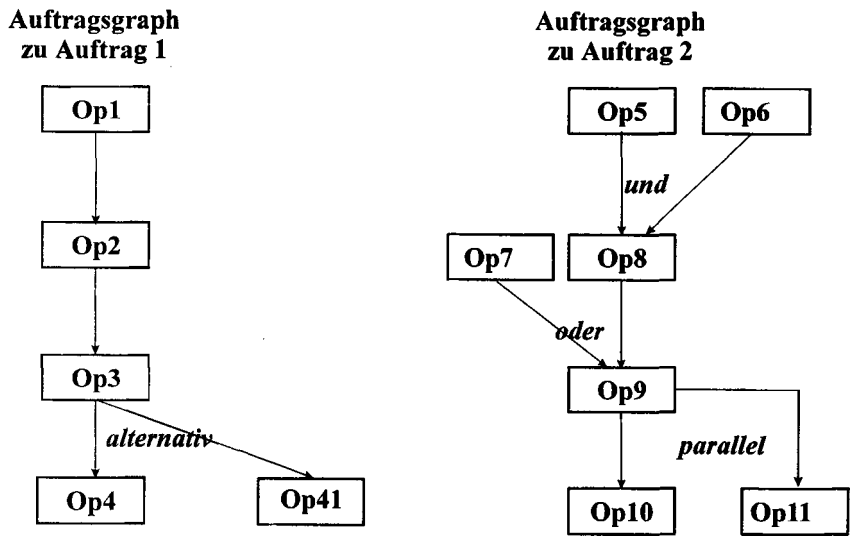


Bild 13: Für jeden Auftrag (Produkt) wird der Arbeitsablauf durch je einen Auftragsgraphen vorgegeben. Op5 und Op6 müssen beide vor Op8 ausgeführt worden sein (*und*-Verknüpfung). Nach Op9 können Op10 und Op11 ausgeführt werden (*parallel*). Op9 kann gestartet werden, wenn Op7 oder Op8 beendet wurden (*oder*-Verknüpfung). Nach Op3 können *alternativ* entweder Op4 oder Op41 ausgeführt werden.

Dies ermöglicht die Vorgabe von Transportrouten in einem Transportsystem, das aus beliebig verknüpften Transportbändern zusammengesetzt ist. Auch der Transport auf einer Teilstrecke eines Bandes von einer Bearbeitungsstation zur nächsten ist auf diese Weise einfach zu spezifizieren.

Mögliche Fehlerereignisse werden an den Postkanten der fehlerbehafteten Operationen im Auftragsgraphen notiert. Hier können auch logische Verknüpfungen mehrerer Fehlerereignisse angegeben werden. Die bei der Ausführung einer Operation möglichen Fehlerereignisse sind im Ressourcenmodell den Operationen zugeordnet. Wenn diese Zuordnung vom Betriebsingenieur bzw. Inbetriebnahmepersonal nicht vollständig erfolgt ist und zusätzliche, bislang unberücksichtigte Fehlerereignisse in das Steuerungssystem übernommen werden sollen, kann ein Facharbeiter vor Ort die entsprechenden Listen ergänzen.

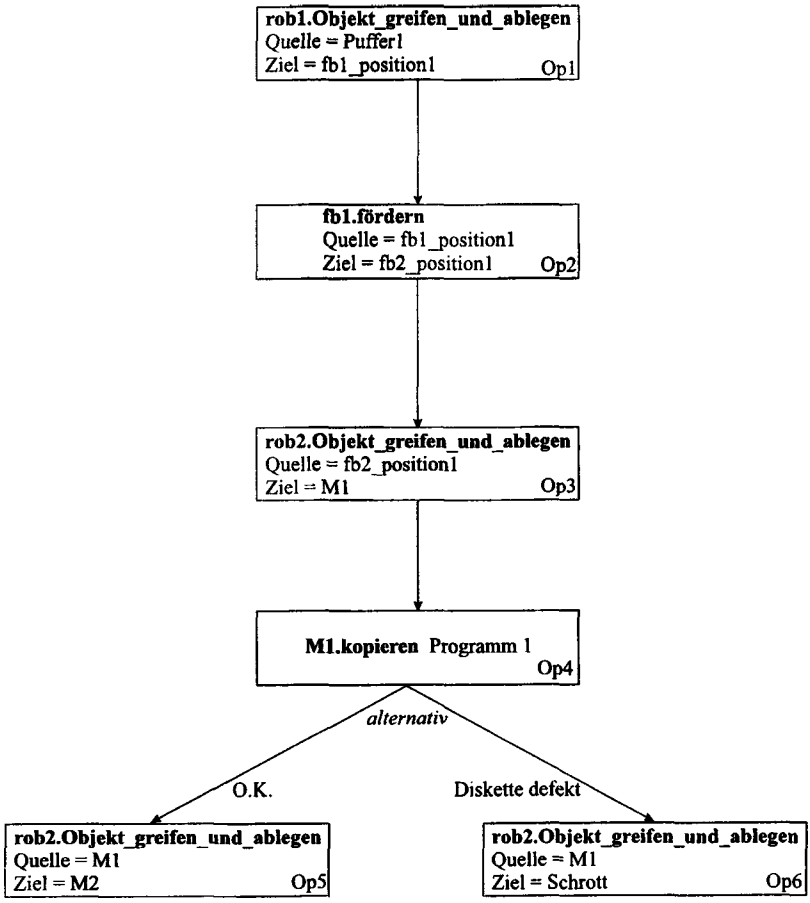


Bild 14: Ein Beispiel für die Vorgabe eines Auftrags durch einen Auftragsgraphen. Der Bediener kann durch entsprechende Baumstrukturen auch fehlerbehaftete Abläufe spezifizieren. Eine solche Form der Darstellung genügt nach den praktischen Erfahrungen den steuerungstechnischen und bedienungsergonomischen Anforderungen. Die Kantenanschriften (*O.K.* bzw. *Diskette defekt*) können über ein Menü ausgewählt werden. Hinter den fett gedruckten Operationsnamen, die sich aus den Namen der ausführenden Ressourcen und den im Ressourcenmodell definierten Werten der Attribute *Operation* zusammensetzen, sind die Operationsparameter notiert.

Voraussetzung hierfür ist, daß die Fehlerereignisse im Klartext angezeigt und in dieser Form auch in die Petrinetzanlagensteuerung übernommen werden.

Der Parameter *Quelle* von Operation 1 aus Bild 14 ist der aus einzelnen Pufferplätzen zusammengesetzte *Puffer 1*. Die Möglichkeit einer solchen Parametrisierung von Transportoperationen wurde zur Vereinfachung der Ablaufvorgabe im Auftragsmodell gewählt. Bei der Angabe von *Quelle* bzw. *Ziel* einer Transportoperation kann ein konkreter Puffer oder ein zusammengesetzter Puffer spezifiziert werden. Wenn ein Werkstück von einem bestimmten Platz eines Zwischenlagers entnommen werden soll, erfolgt die Angabe der entsprechenden Subressource des Zwischenlagers. Soll ein beliebiges Werkstück aus dem Zwischenlager entnommen werden, ist der einfache Verweis auf die zusammengesetzte Ressource Zwischenlager ausreichend.

2.7 Petrinetzgenerierung

Aus den zuvor dargestellten Komponenten Ressourcen-, Operationen- und Auftragsmodell wird ein Petrinetz zur Ablaufsteuerung synthetisiert. Dabei werden Petrinetzbasismodule entsprechend den Vorgaben in den Auftragsgraphen zu einem Gesamtnetz verknüpft. Die Petrinetzbasismodule bilden die Grundstruktur eines nicht hierarchischen Petrinetzes. Eine weitergehende Strukturierung in Form einer Hierarchiebildung wird erst im folgenden Abschnitt betrachtet.

Das entwickelte Petrinetzgenerierungsprogramm untersucht die generierten Petrinetze auftragsweise hinsichtlich ihrer Konsistenz. Dazu werden während der Generierung in einer Zwischenstufe erweiterte Petrinetzbasismodule verwendet. Die Erweiterungen werden exemplarisch anhand des Moduls *Objekt greifen und ablegen* erläutert. Die Ergänzungen bei den übrigen Modulen ergeben sich entsprechend. Im Modul aus Bild 15 wurden drei Prekanten zu der Starttransition ergänzt. Durch die Inhibitorkante von der Stelle *Quelle.1* zur Starttransition wird überprüft, ob ein Widerspruch zwischen den Markierungen der Plätze *Quelle.1* und *Quelle.2* besteht. Ebenso wird durch die Testkante von *Roboter.2* (Werkzeug montiert) geprüft, ob ein zur Ausführung einer Roboteroperation benötigtes Werkzeug montiert ist. Die Inhibitorkante von der Stelle *Greifer.1* (Werkzeug befindet sich im Magazin) prüft, ob das benötigte Werkzeug sich noch im Magazin befindet. Ist dies der Fall, liegt eine inkonsistente Markierung vor. Die Inhibitorkante von der Stelle *Greifer.3* stellt sicher, daß der Greifer leer ist. Der Systembediener wird durch das Petrinetzgenerierungsprogramm auf Verletzungen der implementierten Konsistenzregeln aufmerksam gemacht und zur Korrektur bzw. Modifikation der Auftragsgraphen aufgefordert. Die auftragsweise Konsistenzprüfung unter Beachtung einer gegebenen Anfangsmarkierung gewährleistet zusammen mit den im vorangehenden Abschnitt definierten Petrinetzbasismodulen die Konsistenz des gesamten Petrinetzmodells. Im Petrinetzmodell der Steuerung können die zusätzlich eingeführten Kanten dann weggelassen werden. Diese Aussage kann man sich anhand verschiedener Beispiele verdeutlichen.

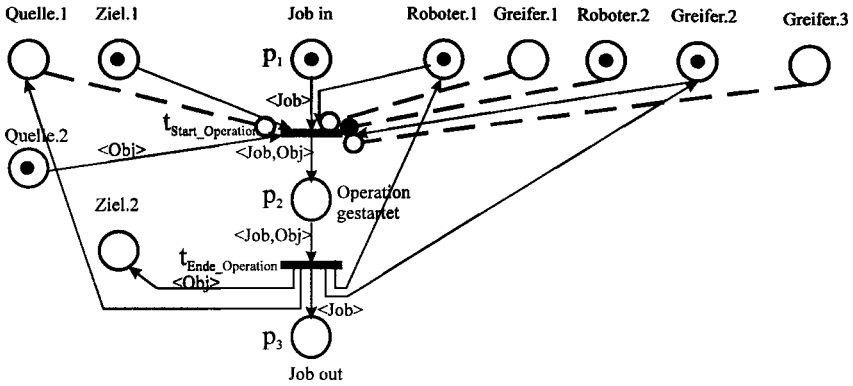


Bild 15: Während des Generierungsvorgangs werden die Module um Kanten zur Konsistenzprüfung erweitert (gestrichelt hervorgehoben). Diese Kanten treten in den erzeugten Netzen nicht mehr auf. Hier als Beispiel das Modul *Objekt greifen und ablegen.*

Obwohl die aufgrund fehlerhafter Vorgaben bzw. Anfangsmarkierungen möglichen Verklemmungen - die z.B. durch eine infolge des Fehlens von Werkzeugen auftretende Blockierung einer Transition hervorgerufen werden können - durch die Konsistenzprüfung abgefangen werden, sind die generierten Petrinetze nicht zwangsläufig verklemmungsfrei. Wird z.B. für das in Bild 16 gezeigte Fertigungssystem ein Petrinetz erzeugt, sind Verklemmungen möglich. Die Verhinderung von physikalisch tatsächlich auftretenden Verklemmungen obliegt in einem solchen Fall der Scheduling-Komponente (vgl. Abschnitt 4.6).

Die Petrinetzsynthese wird im folgenden am Beispiel des Auftragsgraphen aus Bild 14 erläutert.

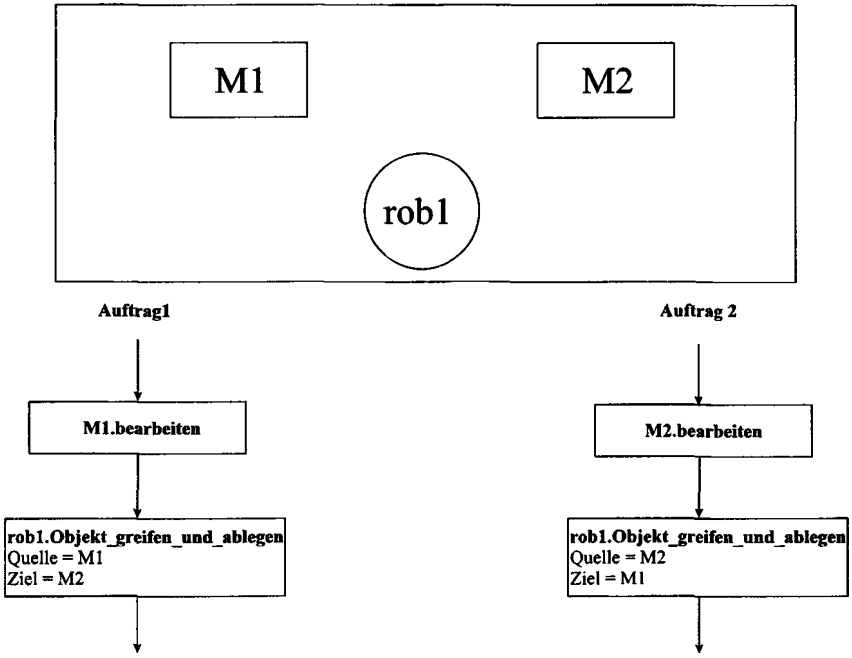


Bild 16: Für das dargestellte System und die vorgegebenen Auftragsgraphen erzeugt das Petrinetzgenerierungssystem ein Petrinetz, in dem es bereits bei einer Losgröße von 1 zu unerwünschten Verklemmungen kommen kann. Die Verhinderung von Verklemmungen obliegt bei einem solchen Petrinetz der in Kapitel 4 beschriebenen Scheduling-Komponente, die vorab eine verklemmungsfreie Schaltfolge ermittelt (z.B. zuerst alle Werkstücke von Auftrag 1 bearbeiten und nachfolgend diejenigen von Auftrag 2).

Erzeugung und Verknüpfung der Petrinetzmodule:

Zur Petrinetzgenerierung werden die Auftragsgraphen nacheinander durchlaufen. Zu jeder Operation eines Auftragsgraphen wird das zugehörige Petrinetzbasismodul instanziiert und mit den bereits erzeugten Modulen verknüpft. Das in Bild 17 gezeigte Petrinetzmodul zur ersten Operation des Auftragsgraphen aus Bild 14 ist aus dem generischen Modul *Objekt greifen und ablegen* durch Instanziierung hervorgegangen. Die Ressourcenstellen *Quelle.1*, *Quelle.2*, *Ziel.1* und *Ziel.2* wurden aus der Parametrisierung der Operation 1 ermittelt. Diese

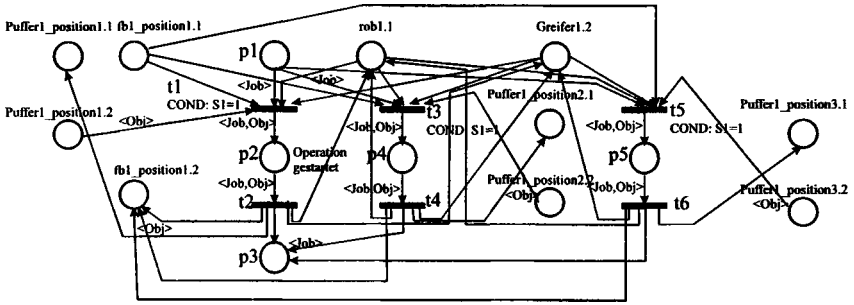


Bild 17: Das zur ersten Operation des Auftragsgraphen aus Bild 14 erzeugte Petrinetzmodul. Da es sich bei Puffer1 um einen differenten Random-Puffer handelt, besteht die Alternative, ein Objekt von dem Pufferplatz *Puffer1_position1* (Transition *t1*), *Puffer1_position2* (Transition *t3*) oder *Puffer1_position3* (Transition *t5*) zu entnehmen. Den Transitionen *t1*, *t3* und *t5* ist jeweils die erweiterte Feuerbedingung „Sensor S1 meldet Aufnahmebereitschaft von fb1_position1" (S1=1) zugeordnet.

sind im Beispiel die konkreten Pufferressourcenstellen *Puffer1_position_i.1*, *Puffer1_position_i.2*, *fb1_position1.1* und *fb1_position1.2*. Die zunächst als Platzhalter angegebenen Ressourcenstellen *Roboter.1* und *Greifer.2* wurden durch die entsprechenden Stellen der ausführenden Ressourcen *rob1* und *Greifer1* ersetzt.

Anschließend werden die Petrinetzmodule der nachfolgenden Operationen auf die gleiche Weise in das Gesamtnetz eingebunden (vgl. Bild 18: Darstellung für die ersten drei Operationen des Auftragsgraphen aus Bild 14). Die Verknüpfung der Module erfolgt über die Jobeingangs- bzw. -ausgangsstellen entsprechend der im Auftragsgraphen vorgegebenen Reihenfolgen der Operationen. Im Auftragsgraphen spezifizierte *und-*, *oder-*, *parallel-* und *alternativ*-Verknüpfungen (vgl. Bild 13) werden im Petrinetz entsprechend abgebildet.

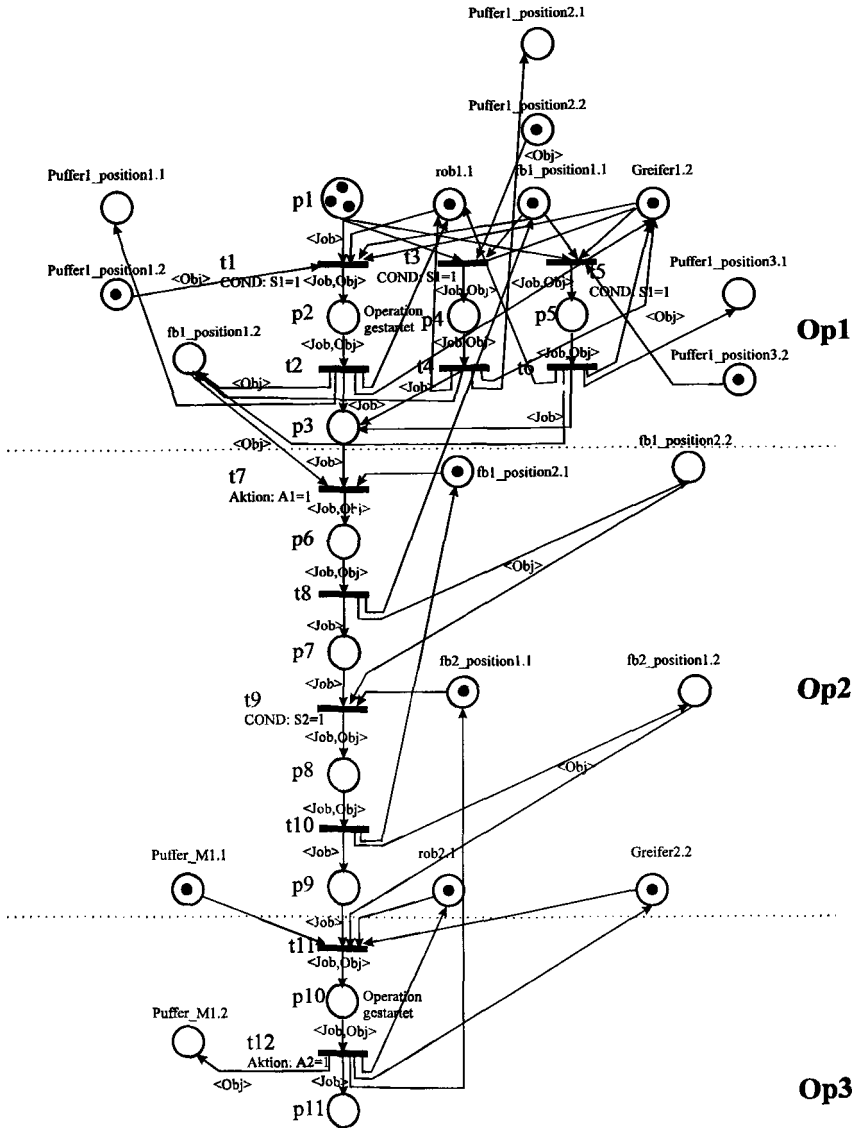


Bild 18: Das aus der Verknüpfung der Petrinetzmodule der ersten drei Operationen des Auftragsgraphen aus Bild 14 entstandene Petrinetz.

Die Petrinetzsynthese vollzieht sich nach dem im folgenden dargestellten Algorithmus, zu dessen Erläuterung einige Definitionen vorgenommen werden:

$i = 1.....n_A$:	Nummer der Auftragsgraphen
Op:	Operation in einem Auftragsgraphen
Typ(Op):	Operationstyp der Operation Op
Ende(Op):	Ende-Jobstelle des Petrinetzmoduls zur Operation Op
Start(Op)[]:	Liste der Start-Jobstellen des Petrinetzmoduls zur Operation Op
n(Start(Op)[]):	Anzahl der Elemente in der Liste Start(Op)[]
PE(Op)[]:	Liste der Vorgänger-Operationen der Operation Op im Auftragsgraphen
SE(Op)[]:	Liste der Nachfolger-Operationen der Operation Op im Auftragsgraphen
n(PE(Op)[]), n(SE(Op)[]):	Anzahl der Elemente in den Listen PE(Op)[] bzw. SE(Op)[]
Res(Op):	Aus dem Ressourcenmodell abgeleitete Ressourcenstellen, die mit den Transitionen des Petrinetzmoduls zur Operation Op zu verbinden sind (diese Liste wird ermittelt, indem die in den Petrinetzbasismodulen vorgenommene klassenorientierte Spezifikation der Ressourcenstellen durch eine konkrete Angabe der Ressourcen aus dem Ressourcenmodell ersetzt wird)
Quellpuffer(Op)[]:	Liste der Puffer-Ressourcen eines Puffers der Klasse *diff. Random-Puffer*, von denen ein Werkstück alternativ entnommen werden kann
n(Quellpuffer(Op)[]):	Anzahl der Ressourcen in Quellpuffer(Op)[]
Zielpuffer(Op)[]:	Liste der Puffer-Ressourcen eines Puffers der Klasse *diff. Random-Puffer*, auf die ein Werkstück alternativ abgelegt werden kann
n(Zielpuffer(Op)[]):	Anzahl der Ressourcen in Zielpuffer(Op)[]

Der Synthesealgorithmus lautet:

```
/*Start*/
für alle Auftragsgraphen (von i = 1 ..... n_A)
        {
        Op = Startoperation(i)
        Erzeuge_Startstelle(i)  /* erste Stelle erzeugen*/
        für alle Operationen des Auftragsgraphen i
                {
                Start(Op)[]   Bestimme_Startstellen(Op)
```

```
        Erzeuge_neue_Res(Res(Op))/*Diejenigen Ressourcenstellen erzeugen, die mit
                                  dem Modul zu Op zu verbinden, aber noch nicht
                                  im Petrinetz vorhanden sind*/
        Wenn ( Typ(Op) == Objekt_greifen_und_ablegen || Objekt_ablegen ||
                          Objekt_greifen)
                {
                Quellpuffer(Op)[] = Bestimme_Quellpuffer(Op)

                Zielpuffer(Op)[] = Bestimme_Zielpuffer(Op)

                für (k = 1; k <= n(Quellpuffer(Op)[]); k++)
                        {
                        für (l = 1; l <= n(Zielpuffer(Op)[]); l++)
                                {
                                Erzeuge_Basismodul(Op, Start(Op)[],Ende(Op),
                                Res(Op), Quellpuffer(Op)[k], Zielpuffer(Op)[l], Obj, Obj
                                )
                                }
                        }
                }
        sonst:
                {
                Wenn (Typ(Op) == Objekt_demontieren)
                        {
                        Obj1,Obj2 = Bestimme_demontierte_Objekte(Op) /*ermittelt
                        aus den Parametern von Op die Bezeichnung der nach der
                        Demontage vorlliegenden Objekte */

                        Erzeuge_Basismodul(Op, Start(Op)[],Ende(Op),Res(Op),
                        Quellpuffer(Op)[1], Ziellpuffer(Op)[1], Obj1, Obj2 )
                        }
                sonst:
                        {
                        Erzeuge_Basismodul(Op, Start(Op)[],Ende(Op),Res(Op),
                        Quellpuffer(Op)[1], Ziellpuffer(Op)[1], Obj, Obj )

                        }

                }
        Wenn (nach Op eine Parallelverzweigung erfolgt)
                {
                Erzeuge_Parallel_Verzweigung(Op) /*Transition an Endestelle
                anhängen und an diese soviele Stellen anhängen, wie nachfolgende,
                parallele Operationen vorgegeben sind*/
                }
        Op = Nächste_Op(Op) /*nächste Op bestimmen: Entweder nachfolgende
        Operation oder eine Operation, die auf eine schon behandelte Operation folgt
        und noch nicht in das Petrinetz integriert ist*/
        }

Petrinetz_Markieren()        /*Anfangsmarkierung setzen*/
```

```
        Simuliere_Auftrag()          /*Konsistenzprüfung: Markenspieler starten und das zum
                                     Auftrag i erzeugte Petrinetz simulieren*/
    }
/*Ende*/
```

Die Funktion *Erzeuge_Basismodul* führt im einzelnen folgende Schritte aus:

```
Erzeuge_Basismodul(Op, Pstart, Pende,Res(Op), Quellpuffer, Ziellpuffer, Obj1, Obj2 )
{

Erzeuge_Stellen_Transitionen_Kanten() /*Jobstellen, Transitionen, Kanten und Kantenan-
schriften enstprechend der Petrinetzbasismoduldefinition erzeugen*/

Wenn ( Sensordefinition(Zielpuffer) == TRUE)
    {
    Erzeuge_erweiterte_Feuerbedingung(„COND : Sensor_1 == 1 & Sensor_2 = 1 &...&
    Sernsor_ns = 1")     /*UND-Verknüpfung aller ns für Zielpuffer definierten Sensoren
                         als Transitionsinschrift der Transition ergänzen, die Prekanten
                         von Zielpuffer.1 besitzt*/
    }

Wenn (Typ(Op) == fördern)
    {
    Wenn (Aktordefinition(Quellpuffer) == TRUE)
        {
        Erzeuge_Aktion(„A1=1 & A2=1 & ... & An=1") /*für alle n auf Quellpuffer
        definierten Aktoren*/
        }
    }

Wenn (den Kanten zu SE(Op) im AG Fehlerbedingungen zugeordnet sind)
    {
    Erzeuge_Alternativverzweigung(Op) /* verzweigt entsprechend der
    Modellierung fehlerbehafteter Abläufe gemäß Bild 19, das im folgenden
    erläutert wird*/
    }
Wenn (den Kanten zu PE(Op) im AG Fehlerbedingungen zugeordnet sind)
    {
    Übernehme_Fehlerbedingung(Op)  /*Übernehmen der Fehlerbedingungen aus den
    Prekanten zu Op im AG in die erweiterte Feuerbedingung COND der Starttransition
    des Moduls zu Op: Wenn Prekanten im AG ODER-verknüpft: ODER-Verknüpfung
    der Fehlerbedingungen in COND;
    Wenn Prekanten im AG UND-verknüpft: UND-Verknüpfung der Fehlerbedingungen
    n COND*/
    }
}/*Ende*/
```

Bild 19 zeigt die Modellierung von fehlerbehafteten Abläufen im Petrinetz. Dabei wird davon ausgegangen, daß der Bediener (Facharbeiter) an den Postkanten der Operationen in den

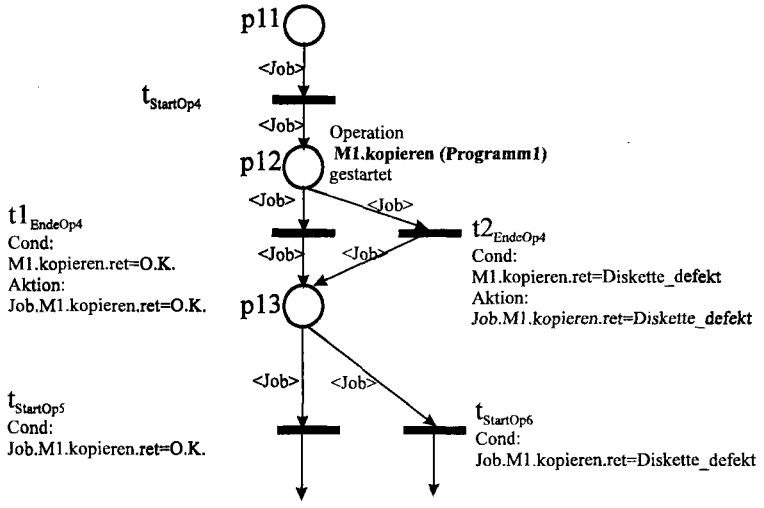


Bild 19: Abbildung fehlerbehafteter Abläufe im Petrinetz am Beispiel der Operationen Op4, *Op5* und *Op6* aus *Bild* 14.

Auftragsgraphen, bei denen Fehler auftreten können, die entsprechenden Fehlerereignisse eingetragen hat (vgl. Bild 14). Die Rückmeldungen der Ausführungsebene werden in die Datenstrukturen der Jobmarken übernommen (*Aktionen* der Transitionen *t1EndeOp4* und *t2EndeOp4*) und in den erweiterten Feuerbedingungen der Starttransitionen der nachfolgenden Operationen abgefragt.

Generierung von Petrinetzen als Modell von Steuerung und Anlage zur Simulation:

Bislang wurde die Synthese von Petrinetzen als Modelle der Steuerung betrachtet. Für Simulationen und die in Kapitel 4 behandelten Scheduling-Verfahren werden als Modelle der Steuerung *und* der Anlage selbst zeitbewertete Petrinetze benötigt. Dazu werden bei der Petrinetzgenerierung die erweiterten Feuerbedingungen, die auf Rückmeldungen der Ausführungsebene und Sensorwerte bezug nehmen, weggelassen und statt dessen die Operationsdauern als Zeitbewertungen in die Petrinetze übernommen. Es wird dabei von

fehlerfreien Abläufen ausgegangen. Die Operationsdauern repräsentieren feste Werte, die gemessen oder geschätzt wurden.

Bild 20 illustriert die Abbildung der Operationen in Form zeitbewerteter Petrinetzmodule.

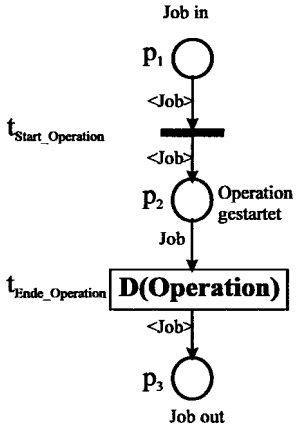


Bild 20: Modellierung der Operationsdauern in zeitbehafteten Petrinetzen. Diese Modellierung wird für die Scheduling-Verfahren aus Kapitel 4 verwendet.[10]

Markierung der Stellen:

Nach der Synthese des Gesamtpetrinetzes wird die Anfangsmarkierung der Ressourcenstellen aus den im Ressourcenmodell zu jeder Ressource vorhandenen Attribut *Anfangszustand* bestimmt. Der Bediener (Facharbeiter) kann den Anfangszustand der Ressourcen im Steuerungssystem (Petrinetz) über ein Dialogfenster mit dem tatsächlichen Zustand der Ressourcen abgleichen. Die Zustände *Roboter.bereit* usw. beschreibt er dazu mit *0* oder *1* im Sinne von *nicht erfüllt* oder *erfüllt*. Eine Besonderheit ist zu den Zuständen anzumerken, die sich darauf beziehen, ob ein Puffer ein Flußobjekt hält. Der Bediener gibt eine 0 an, wenn der Puffer leer ist. Er gibt den Pufferzustand mit 1 an, wenn der Puffer ein Objekt hält. Auf den

[10]Grundsätzlich hätte hier auch eine Zuordnung der Zeiten zu Kanten oder Stellen erfolgen können (zur Kante zwischen *p2* und $t_{Ende\_Operation}$ oder zur Stelle *p2*, vgl. [Stulle 95] oder [Hanisch 92 a und b]). Da die in Kapitel 4 erläuterten Scheduling-Verfahren auf zeitbewerteten Transitionen aufsetzen, erfolgt hier die Zeitbewertung von Transitionen.

entsprechenden Petrinetz-Stellen (*Puffer.2*, *Greifer.3* usw.) wird dann eine Marke mit einer automatisch vergebenen Nummer abgelegt. Der Bediener kann aber auch zusätzlich zur Angabe einer 1 das auf einem Puffer liegende Flußobjekt mit einem Text beschreiben (z.B. „Gehäuse_1"). Die den Startzustand der Aufträge abbildenden Jobstellen (im Bild 18 die Stelle *p1*) erhält eine der Losgröße des Auftrages entsprechende Anzahl Marken (im Beispiel beträgt die Losgröße 3). Die Losgröße ist als die Häufigkeit der Ausführung der Startoperation eines Auftrages definiert und wird vom Facharbeiter vorgegeben.

2.8 Zusammenfassung

Die in Kapitel 2 ausführlich dargestellten Untersuchungsergebnisse lauten zusammengefaßt: Das zur Petrinetzgenerierung benötigte Wissen läßt sich in eine an der Hardware bzw. den Ressourcen orientierte Beschreibung der Anlage und eine an den Abläufen ausgerichtete Beschreibung der Anlagennutzung aufspalten. Die Beschreibung bzw. die Vorgabe der Anlagennutzung, die im Vergleich zur Ressourcenbeschreibung in praxi häufigeren Änderungen unterworfen ist, kann in Form von zu Auftragsgraphen verknüpften Operationen (Aktivitäten) von Facharbeitern bzw. Betriebsingenieuren - also hauptsächlich mit technischen Abläufen befaßten Personen - vorgenommen werden.

Die Beschreibung einer Fertigungsanlage, die ein weitergehendes Verständnis in bezug auf die Bedienung des Systems und die Streuerungsdetails voraussetzt, ist in eine von einem Systemspezialisten zu implementierende, generische Basismodellierung und eine anwendungsspezifische, von einem Systemingenieur zunächst bei der Inbetriebnahme zu installierende und später änder- bzw. erweiterbare Ressourcenmodellierung unterteilbar. Diese Aufspaltung erhöht die Flexibilität und Überschaubarkeit des Systems und ermöglicht darüberhinaus eine weitere Verlagerung des Schwerpunkts des insgesamt zur Petrinetzgenerierung notwendigen Wissens auf eine objekt- und technikorientierte, abstraktere Ebene. Dadurch wird das vor Ort unmittelbar verfügbare Personal in die Lage versetzt, nicht nur die Anlagennutzung zu ändern, sondern auch die durch Hardwaremodifikationen bzw. -erweiterungen induzierten und notwendigen Steuerungsanpassungen selbständig durchzuführen. Allerdings ist hierzu ein normaler Facharbeiter nicht geeignet, sondern eine Qualifikation als Systemingenieur oder eine auf die Systembedienung hin ausgerichtete Qualifikation als Wartungsfachmann Voraussetzung.

Die generische Basismodellierung, die ein variables und erweiterbares Ressourcenklassenmodell sowie eine Grundstrukturierung der Petrinetzkoordinierungs- bzw. -steuerungsebene in Form von Petrinetzbasismodulen umfaßt, ist Grundbestandteil des Systems und kann von Petrinetzspezialisten evolutionär ausgebaut werden.

Die grundsätzliche Praxistauglichkeit der Vorgehensweise wurde neben zahlreichen simulativen Experimenten auch durch exemplarische Untersuchungen anhand der Testzelle nachgewiesen.

3 Automatische Strukturierung zu hierarchischen Petrinetzen

In diesem Abschnitt wird eine Erweiterung des Petrinetzgenerierungssystems um eine Komponente beschrieben, die automatisch hierarchische Petrinetze erzeugt. D.h. es werden über mehrere Ebenen hinweg verteilte Petrinetzmodelle gebildet, welche exakt dieselben Abläufe steuern bzw. modellieren wie die nach dem in Abschnitt 2.7 erläuterten Verfahren generierten, nicht-hierarchischen Petrinetze.

Die Hierarchisierung von Petrinetzen ist aus folgenden Gründen sinnvoll und wünschenswert:

1. Die Petrinetzmodelle werden übersichtlicher. Zumindest besteht die Möglichkeit, von abstrakteren Ebenen in Detail-Netze zu wechseln. Dies kann nützlich sein, wenn bei der Anlagensteuerung auf der Basis automatisch erzeugter Petrinetze Probleme auftreten und ein Petrinetzspezialist die Petrinetze überprüfen muß.
2. Die Bestimmung einer optimalen Schaltfolge für ein Gesamtnetz kann auf die Bestimmung der Schaltfolgen von Teilnetzen und anschließende Aggregation zurückgeführt werden (siehe hierzu die in [He et al. 96] dokumentierten Schedulingergebnisse[11] oder vgl. auch [Shen et al. 92]). Abhängig vom vorliegenden Gesamt-Petrinetz sind die aggregierten Schaltfolgen zumeist nicht schlechter als die durch die Anwendung der Scheduling-Verfahren auf das Gesamtnetz ermittelten, können aber in kürzerer Rechenzeit bestimmt werden[12].
3. Die Dekomposition eines umfangreichen Petrinetzes kann als Strukturvorgabe für die Implementierung einer Steuerung auf verteilten Rechnersystemen genutzt werden.

Eine Hierarchisierung von Petrinetzen ist nur dann sinnvoll und von praktischem Nutzen, wenn die Teil- oder Unternetze völlig oder weitgehend unabhängig voneinander sind. D.h. zwischen den auf einer übergeordneten Ebene jeweils vergröbert dargestellten Teilnetzen

[11] Für das Scheduling wurden die in Kapitel 4 dieser Arbeit erläuterten Scheduling-Verfahren benutzt. Im Anhang G sind die Ergebnisse abgedruckt. Es wird empfohlen, hier nur die Ergebnisse zur Kenntnis zu nehmen und die Details aus Anhang G erst nach der Lektüre von Kapitel 4 nachzuvollziehen.

[12] bis zu 90% Rechenzeitersparnis. Allerdings ist Vorsicht geboten: Nicht selten traten bei den Untersuchungen Verschlechterungen des Makespans bis zu 30% auf, wenn entsprechende Verkopplungen der Teilnetze bestehen.

sollten möglichst wenige Verkopplungen bestehen - im Idealfall nur sequentielle Kopplungen jeweils über die nächst höhere Ebene.

Die nähere Betrachtung von Petrinetzmodellen für Fertigungssysteme zeigt, daß neben der sequentiellen Kopplung, die durch den Produktionsablauf vorgegeben ist, die Kopplung der Petrinetzelemente im wesentlichen über die Ressourcenstellen erfolgt (vgl. hierzu die Petrinetze aus Abschnitt 2.5 oder auch [Zhou und DiCesare 91]). Aus diesem Grund ist eine Zerlegung wünschenswert, bei der die Elemente der Teilnetze jeweils mit disjunkten Teilmengen der Ressourcenstellen verbunden sind.

3.1 Ablaufdekomposition

Aus den genannten Überlegungen heraus wurde eine Komponente zur Bildung hierarchischer Petrinetze als eine Erweiterung des in Bild 4 dargestellten Petrinetzgenerierungssystems mit der Zielsetzung realisiert, hierarchische, aus untereinander nicht ressourcenverkoppelten Teilnetzen zusammengesetzte Petrinetze zu erzeugen (vgl. auch Übersichtsdarstellung in Bild 3).
Wird dieses Ziel vollständig erreicht, dann sind die Teilnetze nur sequentiell über die jeweils nächst höhere Ebene verkoppelt. Sie repräsentieren so von der höheren Ebene aus betrachtet in sich abgeschlossene (atomare) Abläufe.

Im rechten Teil von Bild 21 sind die in diesem Abschnitt betrachteten Erweiterungen skizziert. Die Bildung hierarchischer Petrinetze vollzieht sich in den folgenden Schritten:
Zunächst wird jeder Auftragsgraph des Auftragsmodells in einen IDEF-Graphen umgewandelt. Die IDEF-Graphen unterscheiden sich von den Auftragsgraphen dadurch, daß zu jeder Operation die benötigten Ressourcen angegeben werden. Zur Umsetzung der Auftragsgraphen in IDEF-Graphen werden Informationen aus dem Ressourcen- und Operationenmodell herangezogen. Es wird davon ausgegangen, daß die Auftragsgraphen zyklenfrei sind (dies ist Voraussetzung für die nachfolgende Zerlegung und bedeutet lediglich eine Anweisung an den Bediener, die seine Vorgabemöglichkeiten nicht grundsätzlich einschränkt).
Die IDEF-Graphen werden zusammen als Gesamtgraph betrachtet (im folgenden einfach IDEF-Modell genannt) und durch ein Gruppierungsverfahren in Teil-IDEF-Graphen zerlegt. Ein Teil-IDEF-Graph kann also aus Operationen verschiedener Aufträge bestehen.
Bei dem zur Dekomposition verwendeten sog. *sequentiellen Gruppierungsalgorithmus* (SGA) handelt es sich um eine Erweiterung des in [Kusiak 90] angegebenen Gruppierungsalgorithmus[13]. Die so ermittelten Teil-IDEF-Graphen werden dann durch die bereits bekannten Petrinetzgenerierungsalgorithmen nach dem in Abschnitt 2.7 dargestellten Prinzip in Teil-Petrinetze umgesetzt.

[13]Engl.: Cluster Identification Algorithm (CI-Algorithm)

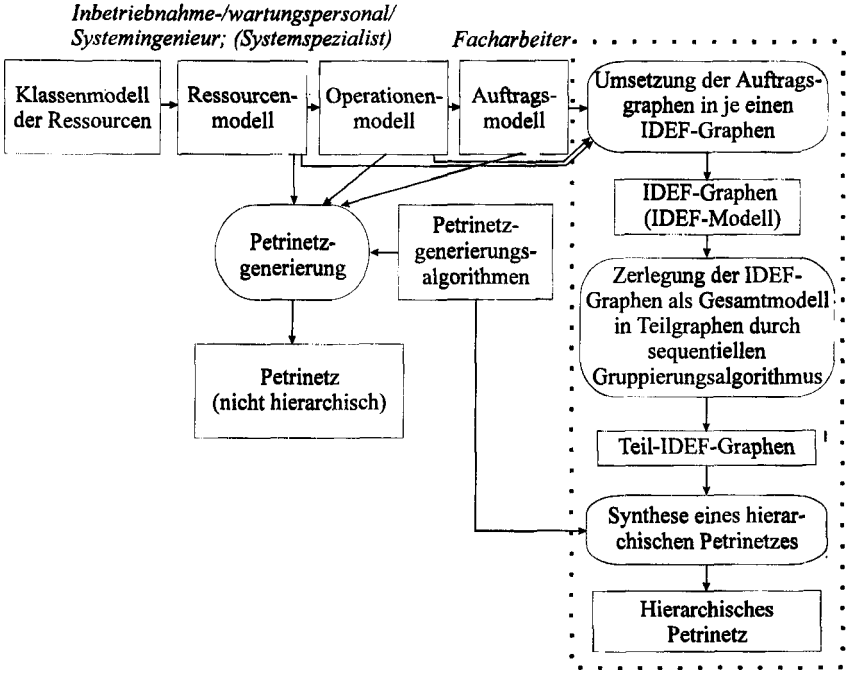


Bild 21: Die Erweiterung des in Bild 4 gezeigten Systems um eine Komponente zur Hierarchiebildung. Die Auftragsgraphen werden einzeln in IDEF-Graphen und diese als Gesamt-IDEF-Graph (im folgenden allgemein als IDEF-Modell bezeichnet) in Teilgraphen zerlegt. Aus den Teilgraphen werden mit den bereits erläuterten Verfahren Teil-Petrinetze gebildet und diese anschließend zu einem hierarchischen Petrinetz verknüpft. Die Petrinetzgenerierungsalgorithmen wurden dazu entsprechend erweitert.

Diese zunächst nicht verbundenen Teilnetze werden anschließend zu einem hierarchischen Petrinetz verknüpft (die in Bild 21 dargestellten Petrinetzgenerierungsalgorithmen umfassen im Vergleich zu Bild 4 dafür noch zusätzliche Programme zur Hierarchiebildung). Im folgenden werden die genannten Schritte ausführlich erläutert. Dabei wird die Hierarchisierung exemplarisch zunächst auf nur zwei Ebenen betrachtet. Durch die wiederholte Anwendung des sequentiellen Gruppierungsalgorithmus auf bereits ermittelte Teil-IDEF-Graphen können Petrinetzhierarchien über mehr als zwei Ebenen gebildet werden.

Umsetzung der Auftragsgraphen in IDEF-Graphen:

Die Umsetzung der Auftrags- in IDEF-Graphen erfolgt durch Zugriff auf im Operationen- und Ressourcenmodell gespeicherte Informationen. Aus den Auftragsgraphen sowie dem Operationen- und Ressourcenmodell können zu jeder Operation eines Auftragsgraphen die benötigten Ressourcen bestimmt werden. Dies erfolgt durch Ermittlung der in den Petrinetz-basismodulen angeforderten (Prekanten zu den Starttransitionen) und freigegebenen Ressourcen (Postkanten der Endetransitionen). In Bild 22 ist ein Auftragsgraph seinem korrespondierenden IDEF-Graphen gegenübergestellt.

Die Operationen-Ressourcen-Matrix zum IDEF-Modell aus Bild 22 lautet:

Operationen \ Ressourcen	0	1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1									
2			1		1	1							
3						1	1	1					1
4								1					
5							1	1			1		1
6						1	1		1				1
7									1				
8							1		1	1			1

Der SGA besteht aus folgenden Schritten:

Schritt 0: Setze den Iterations-Zählindex k = 1. Initialisiere die Gruppen G und G' als leere Gruppen. Setze $A^{(k)} = A$. $A^{(k)}$ ist die Matrix A in der k-ten Iteration.

Schritt 1: Selektiere die erste Zeile von $A^{(k)}$ und ordne sie der Gruppe G' zu. Markiere die erste Zeile von $A^{(k)}$ durch eine horizontale Linie $\mathbf{h}_1$.

Schritt 4: Für jede Zeile aus G führe folgendes aus:

(1): Für jedes nur vertikal durchgestrichene $a_{ij} = 1$ ziehe eine horizontale Linie h_i.

(2): Für jedes nur horizontal durchgestrichene $a_{ij} = 1$ ziehe eine vertikale Linie v_j.

Schritt 5: Wiederhole die Schritte 3 und 4, bis alle Zeilen aus G keine einfach durchgestrichenen Einträge mit dem Wert 1 besitzen. Die horizontal und vertikal gestrichenen Elemente in $A^{(k)}$ bilden die Operationengruppe k und die Ressourcengruppe k. Die Operationengruppe k repräsentiert den Teilgraphen $TG^{(k)}$.

Schritt 6: Die Matrix $A^{(k)}$ wird durch Streichen aller Zeilen und Spalten, die keine einfach gestrichenen Einträge mit dem Wert 1 enthalten, in die Matrix $A^{(k+1)}$ überführt.

Schritt 7: Wenn die Matrix $A^{(k+1)}$ die Dimension 0 hat oder alle Elemente den Wert 0 haben, dann beende den Algorithmus. Sonst: Setze $G = G' = \varnothing$, $k = k + 1$ und beginne erneut mit Schritt 1.

Die Erfüllung der ***Sequenzenbedingung*** ist zur Vermeidung von unzusammenhängenden und damit für die Petrinetzmodellierung ungünstigen Zerlegungen notwendig. Die Entscheidung darüber, ob die Sequenzenbedingung erfüllt ist, erfolgt gemäß den Reihenfolgen der Operationen in den IDEF-Graphen nach folgenden Regeln (jede Regel ist in Bild 23 durch ein Beispiel illustriert):

Regel 1: Wenn alle direkten Vorgängeroperationen einer Operation aus G' zur Gruppe G gehören, dann ist die Sequenzenbedingung erfüllt.

Sonst:

Regel 2: Wenn alle direkten Nachfolger einer Operation aus G' zur Gruppe G gehören, dann ist die Sequenzenbedingung erfüllt.

Sonst:

Es wird ausgehend von der betrachteten Operation aus G' im IDEF-Graphen ab jeder direkten Vorgängeroperation, die nicht zu G gehört, solange in Rückwärtsrichtung gesucht, bis eine

Operation gefunden wird, die zu G gehört oder alle erreichbaren Startoperationen des IDEF-Modells erreicht wurden.

Regel 3: Wenn eine zu G gehörende Operation gefunden wurde, dann ist die Sequenzenbedingung nicht erfüllt.

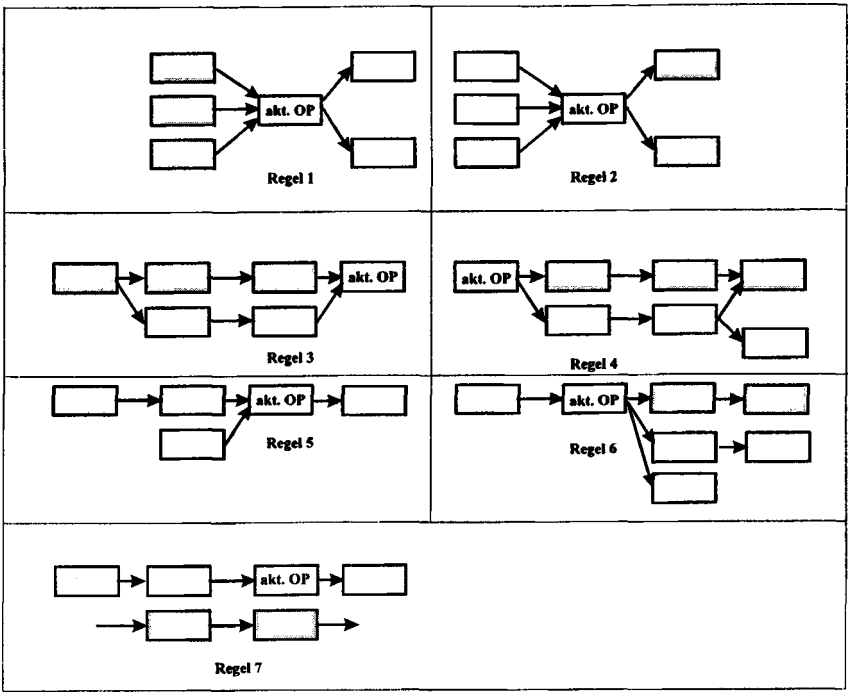


Bild 23: Zur Illustration der Regeln. *akt.OP* ist die Operation aus der Gruppe G', die gerade auf die Erfüllung der Sequenzenbedingung hin überprüft wird. Die schraffierten Operationen sind bereits der Operationengruppe k (Gruppe G = Teilgraph k) zugeordnet, die unschraffierten nicht.

Sonst:

Es wird ausgehend von der betrachteten Operation aus G' im IDEF-Graphen ab jeder direkten Nachfolgeoperation, die nicht zu G gehört, solange in Vorwärtsrichtung gesucht, bis eine

Operation gefunden wird, die zu G gehört oder alle erreichbaren Endoperationen des IDEF-Graphen erreicht wurden.

Regel 4: Wenn eine Operation gefunden wurde, die zu G gehört, dann ist die Sequenzenbedingung nicht erfüllt.

Sonst:

Regel 5: Wenn mindestens eine Vorgängeroperation zur Gruppe G gehört, dann ist die Sequenzenbedingung erfüllt.

Sonst:

Regel 6: Wenn mindestens eine Nachfolgeoperation zur Gruppe G gehört, dann ist die Sequenzenbedingung erfüllt.

Sonst:

Regel 7: Die Sequenzenbedingung ist erfüllt.

Die Funktionsweise des sequentiellen Gruppierungsalgorithmus wird in Anhang C anhand eines Beispiels ausführlich erläutert.

Zerlegung über mehrere Ebenen durch mehrfache Anwendung des SGA:

Die einmalige Anwendung des SGA auf ein IDEF-Modell führt zu einer Zerlegung der Tiefe 1 (eine oberste Ebene 0 und eine aus einer Zerlegungsiteration gebildete Ebene 1). Zur weiteren Zerlegung kann der SGA nun weiter auf die Teil-IDEF-Graphen angewendet werden, wobei für die erneute Anwendung eine Mindestgröße der Teilgraphen festgelegt wurde. Man erhält so eine Zerlegung der Tiefe 2 usw.

Das funktioniert allerdings nur, wenn vor Anwendung des SGA auf die Teil-IDEF-Graphen diese durch heuristische Regeln modifiziert werden. Denn wäre eine weitere Zerlegung der Teil-IDEF-Graphen durch unmittelbare Anwendung des SGA auf einen Teil-IDEF-Graphen möglich, hätte sie bereits in der vorangehenden Zerlegungsiteration stattgefunden.

Vor der Anwendung des SGA auf die (Teil)-IDEF-Graphen werden die entsprechenden Operationen-Ressourcen-Matrizen mit Hilfe zweier Regeln modifiziert. Die erste Regel ist die *sequentielle Ressourcennutzungs*-Regel und die zweite die *häufigste Ressourcennutzungs*-Regel. Mit Hilfe dieser Regeln werden immer vor der Anwendung des SGA auf einen Teil-IDEF-Graphen in einer weiteren Zerlegungsiteration die entsprechenden Operationen-Ressourcen-Matrizen modifiziert.

Nach der ersten Regel werden diejenigen Ressourcen aus einer Operationen-Ressourcen-Matrix entfernt, die mindestens von einer definierten Anzahl s_{max} direkt aufeinanderfolgender Operationen benutzt werden. Die Vorgabe von s_{max} basiert auf der Erfahrung eines Petrinetzexperten.

Nach der zweiten Regel werden diejenigen Ressourcen aus der Operationen-Ressourcen-Matrix entfernt, deren Nutzungsquotient n einen definierten Wert n_{max} überschreitet. Der Nutzungsquotient einer Ressource wird als Quotient aus der Anzahl der Operationen, die die Ressource benötigen, und der Gesamtzahl der Operationen des (Teil-)IDEF-Graphen berechnet. Die aus einer Matrix entfernten Ressourcen sind potentielle sogenannte *kritische* Ressourcen, sofern sie in verschiedenen Teil-IDEF-Graphen benötigt werden[14].

[14] Für s_{max} und n_{max} sind im System feste Werte vorgeben (z.B. $s_{max} = 3$, $n_{max} = 0.6$).

3.2 Synthese hierarchischer Petrinetze

Zunächst werden durch Anwendung des in Abschnitt 2.7 erläuterten Programms aus den zu den Teil-IDEF-Graphen korrespondierenden Teil-Auftragsgraphen Teil-Petrinetze erzeugt.
Zwischen diesen Teil-Petrinetzen bestehen Reihenfolgebeziehungen, die durch die ursprünglichen, nicht zerlegten Graphen vorgegeben sind. Bild 24 zeigt hierzu als Beispiel die aus einer Zerlegungsiteration resultierende Unterteilung des IDEF-Graphen (IDEF-Modell) aus Bild 22 in die zwei Teilgraphen TG1 und TG2.
Eine Möglichkeit der Bildung eines hierarchischen Petrinetzes bestünde darin, die Teilnetze auf einer übergeordneten Ebene zu Transitionen zu vergröbern (vgl. hierzu auch [Dittrich 93 und 95]). Bild 25 zeigt die daraus resultierende Struktur des hierarchischen Petrinetzes, das aus den beiden Teilnetzen, die aus den IDEF-Graphen TG1 und TG2 erzeugt wurden, zusammengesetzt ist.
Neben der Tatsache, daß eine solche Vergröberung eine Modifikation des Markenspielers nach sich ziehen muß, wird anhand von Bild 25 ein weiterer Nachteil deutlich. Bei der dargestellten Markierung würde, von der übergeordneten Ebene aus betrachtet, nach der klassischen Petrinetzschaltregel die Transition TG2 feuern und alle Marken würden von den Eingangsplätzen p1 und p2 abgezogen und auf die Ausgangsplätze gelegt werden. Tatsächlich kann es in diesem Netz aber passieren, daß beim Feuern von TG2 nur eine Marke von der Stelle p1 abgezogen wird, die andere Marke auf der Eingangsstelle p2 liegenbleibt und zunächst keine Marke im Ausgangsbereich landet. Da dies zur Verwirrung beiträgt, wurde dieses Hierarchisierungskonzept nicht gewählt.

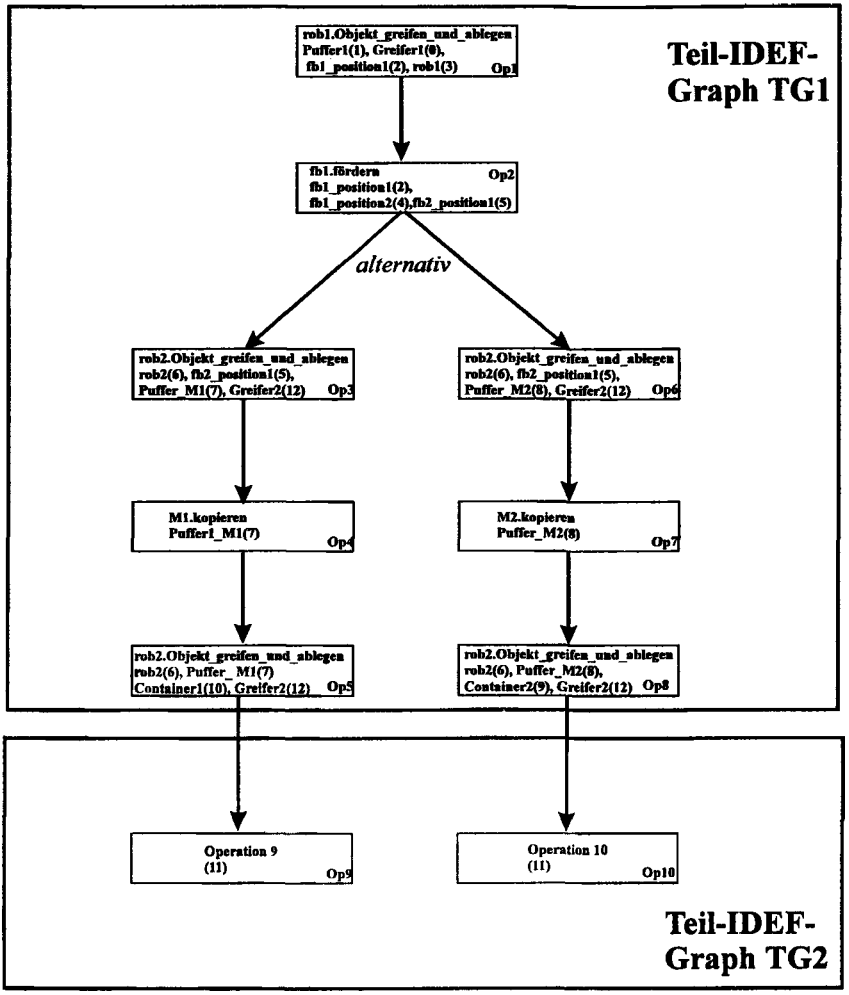


Bild 24: Der zerlegte IDEF-Graph aus Bild 22. Die Teilgraphen sind eingerahmt. Die Operationen 9 und 10 (TG2) wurden noch zur Erweiterung des Beispiels hinzugenommen. Zu den Einzelschritten der Zerlegung siehe Anhang C.

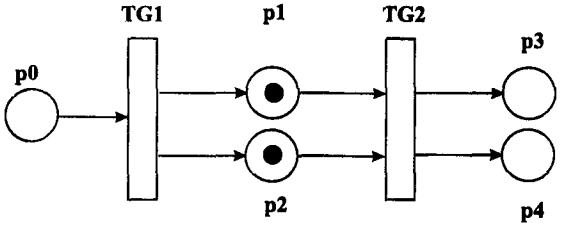


Bild 25: Die oberste Ebene einer auf Vergröberung basierenden Hierarchie. TG1 und TG2 sind vergröberte Transitionen und damit Repräsentanten von im Bild nicht dargestellten und auf einer untergeordneten Ebene liegenden Transitions-berandeten Teilnetzen.

Vielmehr wird zwischen einer *echten Hierarchisierung* und einer *Modularisierung* unterschieden. Die echte Hierarchisierung erfolgt nach dem in Bild 26 dargestellten Prinzip immer dann, wenn ein Teil-IDEF-Graph mit einer Operation beginnt und endet (vgl. auch „blocks" in [Valette 79]). Danach wird ein Teilnetz an eine übergeordnete Petrinetzebene durch einen Synchronisationsmechanismus gekoppelt. Die klassische Schaltregel behält ihre Gültigkeit. Ein Teilnetz wird von der oberen Ebene durch das Schalten einer Transition gestartet (Transition *t1*). Dadurch wird auf dem Startplatz des untergeordneten Netzes (*p3*) eine Marke abgelegt. Zwischen dem Startplatz und dem Endplatz (*p4*) kann sich ein Netz mit beliebiger Struktur befinden. Auf der Merkerstelle *pM* befindet sich nach einmaligem Schalten von *t1* solange eine Marke, bis nach Markierung der Stelle *p4* die Transition *t2* gefeuert hat. Die Stelle *pM* besitzt unbeschränkte Kapazität und die Anzahl der auf ihr liegenden Marken gibt an, wie oft das darunterliegende Teilnetz gestartet wurde.

Eine atomare Hierarchisierung liegt vor, wenn ein Teilnetz nach außen keine weiteren Verbindungen besitzt als die Kanten zur Start- und Endetransition (in Bild 26 *t1* und *t2*). Existieren Kanten zu Ressourcenstellen, auf die auch andere Teilnetze zurückgreifen - diese können auf einer höheren, niedrigeren oder der gleichen Ebene liegen (wenn den Ebenen von oben mit 0 beginnend eine Tiefe zugeordnet wird) -, dann wird in dieser Arbeit von Quasi-Hierarchisierung gesprochen. Diese kann auftreten, wenn verschiedene Teilnetze infolge der Anwendung der Regeln über die *sequentielle* und *häufigste Ressourcennutzung* Verbindungen zu gemeinsamen, kritischen Ressourcenstellen aufweisen. Die kritischen Ressourcenstellen

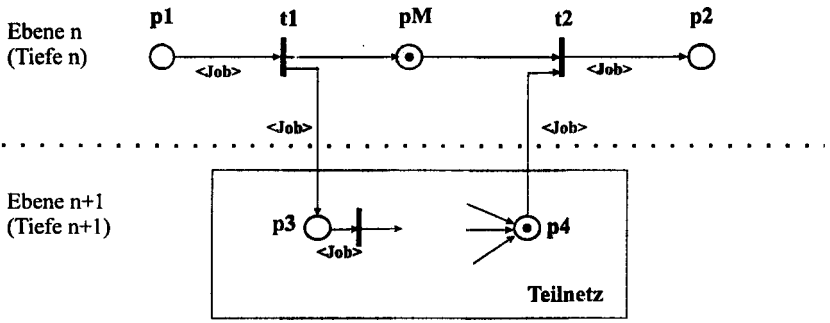


Bild 26: Hierarchiebildung über Interfaces zwischen den Ebenen. Bei der echten Hierarchisierung ist das Teilnetz auf der Ebene n+1 nur über die Stellen p3 und p4 an die übergeordnete Ebene gekoppelt.

werden jeweils derjenigen Ebene im Petrinetz zugeordnet, die über allen tieferen Ebenen liegt, mit deren Transitionen sie verbunden sind.

Kann das dargestellte Prinzip nicht angewendet werden, ohne das dynamische Verhalten des hierarchischen Petrinetzes gegenüber dem des nicht hierarchischen z.B. durch zusätzlich eingeführte Synchronisationen zu modifizieren, erfolgt eine Strukturierung in Form von *Modulen*. Hierbei wird ein Teilnetz auf der nächst höheren Ebene durch ein Modulsymbol repräsentiert. In diesem Symbol befinden sich die Randstellen des Teilnetzes (beliebig viele Start- und Endestellen). Bild 27 zeigt die Struktur einer modularen Verknüpfung der beiden zu den Teilgraphen TG1 und TG2 gehörenden Teil-Petrinetze. Entsprechend der Tatsache, daß der Graph TG1 mit einer Operation beginnt und mit zwei Operationen endet, besitzt das zugehörige Teilnetz einen Start- und zwei Endestellen, die auf der übergeordneten Ebene im Modul zu sehen sind. Zur Verknüpfung der Module werden auf der übergeordneten Ebene die zusätzlichen Transitionen *tk1* und *tk2* automatisch ergänzt. Die Module sind nicht als Transitionen oder Stellen zu interpretieren, sondern stellen lediglich eine Strukturierung dar. Sie ermöglichen einem Petrinetzexperten eine Betrachtung auf verschiedenen Ebenen unterschiedlichen Detaillierungsgrades.

Entsprechend der Definition bei der Hierarchisierung wird hier von einer Quasi-Modularisierung gesprochen, wenn die Module über kritische Ressourcenstellen verkoppelt sind.

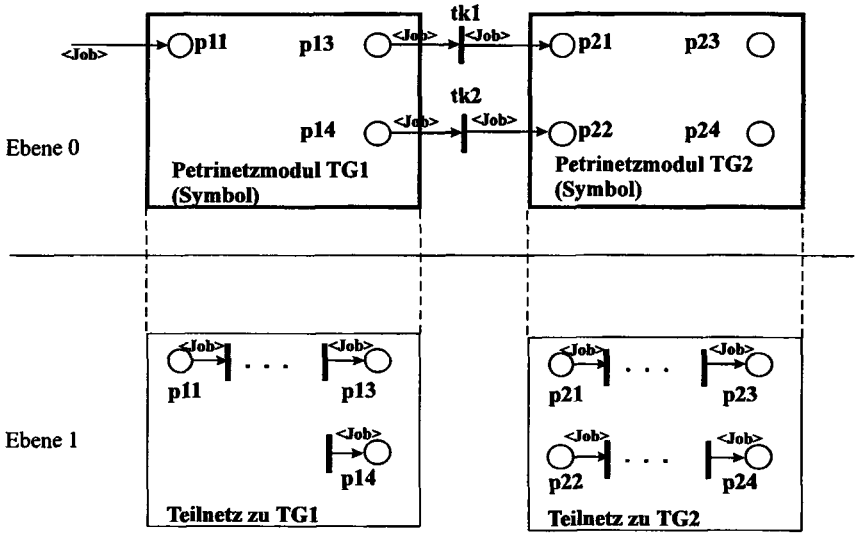


Bild 27: Hierarchiebildung durch Modularisierung. Die Eingangsplätze (*p11*; *p21*, *p22*) und Ausgangsplätze (p13; p14; p23; p24) werden auf die übergeordnete Ebene gezogen und dort in einem Modulsymbol angezeigt. Die Module werden auf der übergeordneten Ebene über die Transitionen *tk1* und *tk2* verbunden, die automatisch ergänzt werden. Die Randstellen der Module werden mehrfach angezeigt, existieren aber tatsächlich nur einmal.

Sowohl auf einer Ebene als auch über mehrere Ebenen hinweg können die Hierarchisierung und Modularisierung gemischt auftreten.

Im Anhang D ist in Bild 45 hierzu ein hierarchisches Petrinetz als Beispiel zu sehen, das aus dem IDEF-Modell von Bild 22 automatisch erzeugt wurde.

4 Ermittlung zeitoptimaler Abläufe auf der Basis generierter Petrinetze

Dieses Kapitel erläutert die neu entwickelten Petrinetz-basierten Verfahren zur Ermittlung von Abläufen mit minimiertem Makespan (siehe hierzu auch [Strege und Pham 95] oder [Strege et al. 96]).
Diese Aufgabe wird allgemein auch als Planung bzw. Scheduling bezeichnet. Eine Schaltfolge zu einem Petrinetz, welche eine Anfangs- in eine Zielmarkierung überführt, wird als Schedule π bezeichnet.

Zunächst wird in Abschnitt 4.1 die Schedulingaufgabe auf der Basis zeitbewerteter Petrinetze definiert. Gegenstand der Abschnitte 4.2 bis 4.5 sind die Schedulingalgorithmen Monte Carlo-Simulation, lokale Suche, lokale Suche mit simulierter Abkühlung sowie heuristische Prioritätsregelverfahren. Die heutzutage in der Praxis häufig verwendeten heuristischen Prioritätsregeln wurden zum Zwecke des Vergleichs implementiert.

Vorab ist anzumerken, daß der von den implementierten Verfahren benötigte Speicherplatz lediglich linear mit der Anzahl der Netzelemente und Marken steigt und nicht von der Größe des Zustandsraumes der Petrinetze bestimmt wird. Hinsichtlich ihrer Anwendbarkeit auf verschiedene Fertigungssystemtypen existieren praktisch keine Einschränkungen: Die Verfahren können auf Petrinetze angewendet werden, die flexible Fertigungssysteme mit Routing-Alternativen, konfliktbehafteten Ressourcen, gepuffertem und ungepuffertem Werkstücktransport, Jobvarianten mit verschiedenen Losgrößen, Montage- und Demontage-vorgängen sowie beliebig verschachtelten Ressourcenanforderungen modellieren.
In Abschnitt 4.6 wird auf die Behandlung bzw. Verhinderung von Verklemmungen eingegangen.

4.1 Zielsetzung: Bestimmung optimierter Schaltfolgen mit geringem Rechen- und Zeitaufwand: Alternative Methode zur Erreichbarkeitsanalyse

Zur Erläuterung der Scheduling-Aufgabe werden das in Bild 28 dargestellte einfache System und das zugehörige Petrinetz betrachtet. Zur Erfüllung von Auftrag 1 soll der Roboter Werkstücke von einem Eingangspuffer EP1 in die Maschine M1 befördern und nach deren Bearbeitung in den Ausgangspuffer AP1 legen. Für den Auftrag 2 ist der gleiche Ablauf mit den Puffern EP2 bzw. AP2 und Maschine M2 auszuführen.

Die Ausführungszeiten der Transport- und Bearbeitungsoperationen sind in Bild 28 bei den Operationen aufgeführt. Im Petrinetz sind diese Ausführungszeiten den Transitionen zugeordnet (siehe Abschnitt 2.5). Zeitbehaftete Transitionen sind durch Rechtecke dargestellt, während normale, unendlich schnell schaltende Transitionen durch einfache Balken repräsentiert sind. Schaltet eine Transition t mit der Dauer D(t), dann werden die Marken im Eingangsbereich von t sofort abgezogen und landen aber erst nach D(t) Zeiteinheiten auf den Ausgangsstellen (siehe [Quäck 88], [Starke 90]). Das Petrinetz in Bild 28 ist gegenüber der Darstellung aus Abschnitt 2.5 in reduzierter Form abgebildet. Operationen werden durch eine zeitbehaftete Transition repräsentiert. Die zu Beobachtungszwecken eingeführten Ressourcenstellen sowie Kantenanschriften werden nicht dargestellt, da sie beim Scheduling nicht berücksichtigt werden müssen. Die Scheduling-Algorithmen basieren also auf Stellen/Transitionen-Netzen, in denen zwischen Jobstellen mit unbegrenzter Kapazität und Ressourcenstellen mit der Kapazität 1 unterschieden wird. Die Kantengewichte haben den Wert 1.

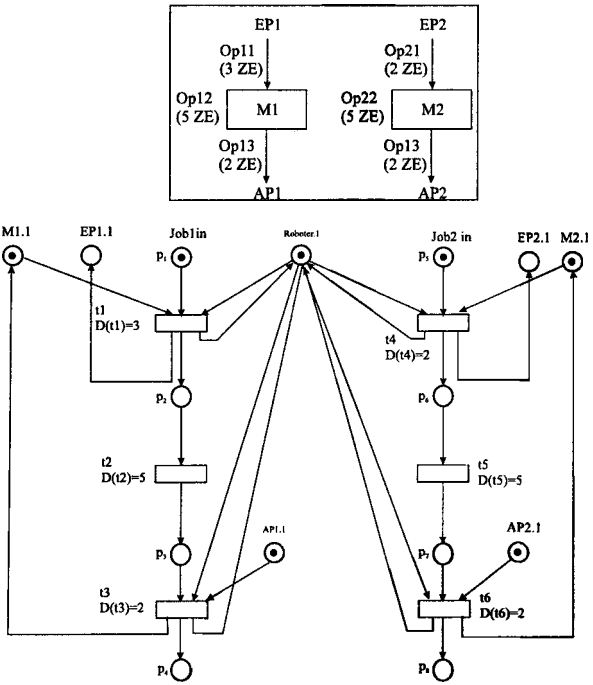


Bild 28: Zur Erläuterung der Scheduling-Aufgabe. Dargestellt ist ein zeitbewertetes Netz als Modell von Steuerung und Anlage. Der lokale Schedule für den Roboter $\pi_{Roboter.1}$ = [*t1*, *t4*, *t3*, *t6*] legt einen Pfad im Dynamikgraphen des Petrinetzes und damit in diesem Fall den gesamten Schedule des Netzes fest. Die Transitionen *t2* und *t5* treten in $\pi_{Roboter.1}$ nicht auf, weil sie nicht mit anderen Transitionen in Konflikt stehen können. *t2* bzw. *t5* können sofort schalten, wenn die Stellen *p2* bzw. *p6* markiert sind. Es wird in maximalen Schritten geschaltet[15]. Die Schlingen (*Roboter.1* ist z.B. Eingangs- und Ausgangsplatz von Transition *t1*) enstehen durch die komprimierte Darstellung.

[15]Ein maximaler Schritt ist eine Teilmenge nebenläufig (d.h. gleichzeitig) schaltbarer Transitionen aus der Menge der Transitionen, die bei einer Markierung Konzession haben und die durch Hinzunehmen einer weiteren konzessionierten Transition nicht mehr alle nebenläufig schalten können (siehe hierzu [Hanisch 92 b]). Bei der in Bild 28 angegebenen Markierung haben t1 und t4 Konzession. Da t1 und t4 um die Stelle *Roboter.1* in Konflikt stehen, kann entweder Schritt 1 bestehend aus t1 oder Schritt 2 bestehend aus t2 ausgeführt werden. Die maximalen Schritte werden unter Beachtung der Zeitdauern der Transitionen immer sofort ausgeführt, wenn die Bedingungen für Konzession erfüllt sind (d.h. Vorbereich markiert und Nachbereich kann Marken aufnehmen).

Aus Bild 28 ist unmittelbar ersichtlich, daß der Roboter eine konfliktbehaftete Ressource ist. Die zugehörige konfliktbehaftete Ressourcenstelle im Petrinetz ist *Roboter.1*. Die Menge der Transitionen, für die *Roboter.1* Eingangsstelle ist, besteht aus den Transitionen *t1*, *t3*, *t4* und *t6*. Diese Menge wird als Konfliktmenge $K_{Roboter.1}$ der Stelle *Roboter.1* bezeichnet.

Die Aufgabe des Scheduling für das dargestellte Petrinetz besteht konkret in der Zuweisung eines die Reihenfolge des Feuerns der Transitionen aus $K_{Roboter.1}$ festlegenden lokalen Schedules $\pi_{Roboter.1}$ zu der konfliktbehafteten Stelle *Roboter.1*. Bei dem speziellen Petrinetz aus Bild 28 gibt der lokale Schedule $\pi_{Roboter.1}$ gleichzeitig den Schedule π für das gesamte Netz vor.

Allgemein gilt, daß ein Schedule π eines Petrinetzes eindeutig durch die den konfliktbehafteten Job- *und* Ressourcenstellen zugeordneten lokalen Reihenfolgen festgelegt wird, wobei in den lokalen Reihenfolgen alle Transitionen, für die die betreffende Stelle im Eingangsbereich liegt, entsprechend der Häufigkeit ihres Feuerns enthalten sind. Die Begründung dieser Aussage kann folgendermaßen skizziert werden: Ein Schedule wird eindeutig durch eine Folge von maximalen Schritten bestimmt. Maximale Schritte enthalten nebenläufig schaltbare Transitionen, von denen nur ein Teil oder alle bei einer Markierung, in der sie geschaltet werden, mit anderen Transitionen um Ressourcen- oder Jobplätze in Konflikt stehen. Ist in diesen Plätzen konsistent hinterlegt, welche Transition an der Reihe ist, wird dadurch die Menge der konzessionierten Transitionen auf den zu schaltenden maximalen Schritt reduziert. Die Konsistenz wird dadurch erreicht, daß bei der Ermittlung der Reihenfolgen selbst simuliert wird, also nur ausführbare und mögliche Schrittfolgen betrachtet werden.
Man erkennt am Beispiel des Netzes aus Bild 28, daß die Reihenfolgen der nicht konfliktbehafteten Ressourcenstellen - z.B. $\pi_{Mt.1}$ - trivial sind und nur aus einer Folge einer einzigen Transition bestehen.
Die Definition konfliktbehafteter Jobstellen entspricht derjenigen für konfliktbehaftete Ressourcenstellen. D.h. konfliktbehaftete Jobstellen besitzen mehr als eine Ausgangstransition. Dies tritt bei Systemen mit Routing-Alternativen auf und muß daher berücksichtigt werden. Bild 29 zeigt ein Beispiel: Ein auf Puffer EP liegendes Werkstück kann entweder durch Roboter1 in Maschine1 oder durch Roboter2 in Maschine2 transportiert werden.

4.3 Lokale Suche

Die Zielsetzung des lokalen Suchverfahrens ist die schrittweise Verbesserung eines mit einem anderen Verfahren - z.B. Monte Carlo Simulation - ermittelten Anfangsschedules. Dieses Verfahren wird als lokale Suche bezeichnet, weil es für ein gegebenes determiniertes Petrinetz (Schedule π) eine Nachbarschaftsbeziehung definiert und hierin lokale Veränderungen vornimmt. Das Verfahren vollzieht sich in folgenden Schritten:

Schritt 1: Es wird zunächst z.B. mittels Monte Carlo Simulation ein ausführbarer Schedule π bestimmt. Dieser Schedule ist zu Beginn der beste Schedule π_{best}.

Schritt 2: Dem aktuell besten Schedule π_{best} wird eine Nachbarschaftsstruktur $LNS_{\pi_{best}}$ zugeordnet. Diese besteht aus allen benachbarten und dieselben Eingangsressourcenstellen besitzenden Paaren von Transitionen auf dem sogenannten kritischen Pfad. Der kritische Pfad eines Petrinetzes ist der in bezug auf die Zeit längste Pfad (vergleichbar mit der Definition bei der CPM-Methode, siehe hierzu [Domschke und Drexl 95]). Die Länge des kritischen Pfades entspricht der Zeit zur Erreichung der Zielmarkierung des Petrinetzes und damit dem Makespan des Schedules. Die Bestimmung des kritischen Pfades erfordert folgendes Vorgehen: Zunächst werden die Stellen des Petrinetzes mit der spätesten Ankunftszeit der Marken (AZM) identifiziert. Rückwärts werden dann diejenigen Eingangstransitionen der Plätze bzw. Eingangsplätze der Transitionen dem kritischen Pfad zugeordnet, die die späteste Feuerzeit (FZ) bzw. die späteste Ankunftszeit der Marken aufweisen. Zur Erläuterung wird das Petrinetz aus Bild 31 herangezogen.

Die Stelle p0 besitzt die späteste Ankunftszeit einer Marke (AZM=21) und bildet das Ende des kritischen Pfades. Da p0 nur die Eingangstransition t_1 besitzt, gehört diese zum kritischen Pfad. Im Eingangsbereich von t_1 liegen die drei Plätze p_1, p_2 und R_1. R_1 weist die späteste Ankunftszeit einer Marke auf und liegt auf dem kritischen Pfad usw. Im Beispiel besteht der kritische Pfad aus der Folge [p_6, t_3, p_4, t_2, R_1, t_1, p_0] und die Menge LNS_π enthält das Transitionenpaar (t_2, t_1). Das Paar (t_3, t_2) gehört nicht zur Menge LNS_π, weil die in den Eingangsbereichen von t_3 und t_2 enthaltenen Ressourcenstellen nicht alle übereinstimmen.

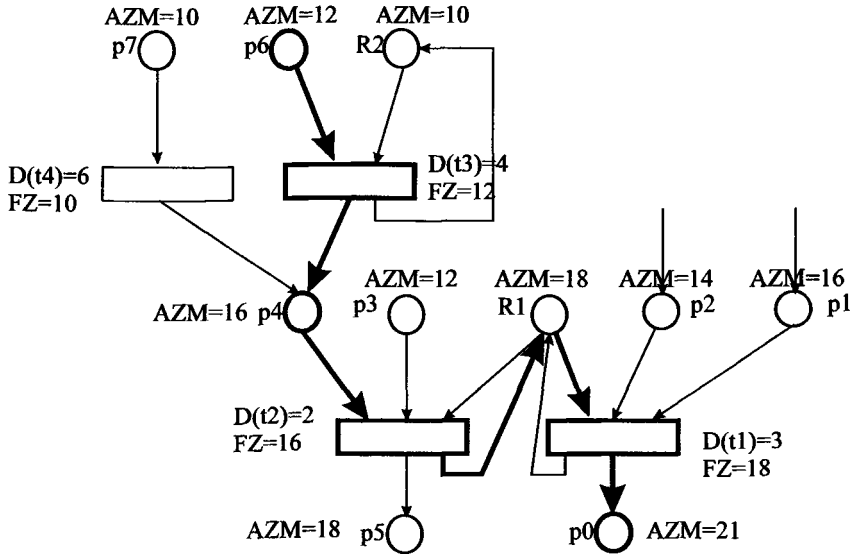


Bild 31: Zur Ermittlung des kritischen Pfades (im Netz hervorgehoben). *pi*: Jobstellen; *Ri*: Ressourcenstellen.

Schritt 3: Aus der Menge der noch nicht untersuchten Transitionenpaare aus $LNS_{\pi_{best}}$ wird ein Transitionenpaar zufällig ausgewählt und dessen Feuerreihenfolge umgekehrt. Dazu werden die Transitionensequenzen der beteiligten Ressourcenstellen aus dem Schedule π_{best} entsprechend modifiziert. Man erhält so einen neuen Schedule π_{neu}.

Schritt 4: Es wird die Zielfunktion $f(\pi_{neu})$ für den modifizierten Schedule durch Simulation ermittelt. Tritt während der Simulation eine Verklemmung auf, kann eine von mehreren möglichen Verklemmungsbehebungsprozeduren gestartet werden. Die Verklemmungsbehebungsprozeduren werden weiter unten in Abschnitt 4.6 erläutert. Finden diese Prozeduren keinen verklemmungsfreien, ausführbaren Schedule, wird die zuletzt in Schritt 3 durchgeführte Modifikation verworfen. Andernfalls werden π_{neu} und $f(\pi_{neu})$ in einer Liste KL (Kandidatenliste) zwischengespeichert.

Die Schritte 3 und 4 werden für alle Transitionenpaare aus LNS_{π} ausgeführt.

Schritt 5: Es wird der Schedule mit minimalem $f(\pi_{neu})$ aus der Liste KL bestimmt und mit dem bislang besten Schedule verglichen. Gilt $f(\pi_{neu,best}) < f(\pi_{best})$, wird $\pi_{neu,best}$ zum neuen π_{best} und die Prozedur verzweigt erneut zu Schritt 2. Andernfalls ist keine Verbesserung mehr möglich und die Suche wird beendet.

4.4 Lokale Suche mit simulierter Abkühlung (Simulated Annealing)

Die lokale Suchprozedur wurde in ein meta-heuristisches Verfahren der simulierten Abkühlung eingebettet (vgl. auch [Domschke und Drexl 95]. Der Grund hierfür lag in der unbefriedigenden Tatsache, daß das lokale Suchverfahren in vielen Fällen nur weit von den optimalen Lösungen entfernte lokale Minima ermittelte. Grundgedanke des Verfahrens des simulierten Abkühlens ist es, während des Suchvorgangs vorübergehend auch schlechtere Lösungen zu akzeptieren. Diese Strategie hat den positiven Effekt, daß in vielen Fällen eine Befreiung aus lokalen Minima in Richtung besserer Lösungen erreicht wird. Die Leistungsfähigkeit des simulierten Abkühlens hat sich in verschiedenen Anwendungen erwiesen (vgl. z.B. [Lutton und Philippart 96] oder [Sadeh und Thangiah 96]). Eine anderes lokales Verbesserungsverfahren ist die meta-heuristische Tabu-Suche (Tabu Search, vgl. [Glover und Laguna 95]), die allerdings in dieser Arbeit nicht implementiert wurde.

Das in dieser Arbeit implementierte Verfahren der simulierten Abkühlung benutzt ein in [Kirkpatrick et al. 83] angegebenes Prinzip. Verschlechterungen der Lösungen bzw. der Schedules π werden bei diesem Suchverfahren nach folgendem Schema zugelassen: In jedem Suchschritt wird für einen neuen Schedule π_{neu} die Gütefunktion $f(\pi_{neu})$ berechnet. Liegt eine Verbesserung vor, wird der Schedule π_{neu} akzeptiert. Andernfalls wird der neue Schedule mit der Wahrscheinlichkeit $P(\pi_{neu}) = \exp([f(\pi_{neu})-f(\pi)]/k_B T)$ akzeptiert. k_B ist die Boltzmannkonstante, T die veränderliche Temperatur. Im folgenden wird statt T die Ersatztemperatur $T'=k_B T$ verwendet.

Das lokale Suchverfahren mit simulierter Abkühlung besteht im Detail aus den folgenden Schritten, wobei die Temperaturen T'_{Start} und T'_{Ende} sowie die Abkühlungskonstante $\gamma \in (0..1)$ als gegeben vorausgesetzt werden ($T'_{Start} > T'_{Ende}$):

Schritt 1: Mit Hilfe der Monte Carlo-Simulation bzw. eines anderen Eröffnungsverfahrens wird ein ausführbarer Schedule π_{best} ermittelt. $T' = T'_{Start}$.

Schritt 2: Es wird auf die gleiche Weise wie bei dem lokalen Suchverfahren eine lokale Nachbarschaftsstruktur $LNS_{\pi_{best}}$ bestimmt. Der zur Begrenzung der Anzahl lokaler Änderungen pro Temperaturniveau auf L_{max} benötigte Laufindex k wird mit 0 initialisiert.

Schritt 3: Ist $LNS_{\pi_{best}}$ leer, wird der Algorithmus beendet. Die Reihenfolge eines zufällig ausgewählten Transitionenpaares aus $LNS_{\pi_{best}}$ wird umgekehrt und dieses Paar aus $LNS_{\pi_{best}}$ entfernt. Die Gütefunktion $f(\pi_{neu})$ wird durch Simulation ermittelt. Tritt während der Simulation eine Verklemmung auf, kann eine Behebungsprozedur aufgerufen werden. Erweist sich π_{neu} als nicht ausführbar, wird aus der Menge $LNS_{\pi_{best}}$ solange ein neues Transitionenpaar ausgewählt, bis ein ausführbarer Schedule gefunden ist. Wird kein ausführbarer Schedule π_{neu} gefunden, wird der Algorithmus beendet. Sonst wird k um eins erhöht.

Schritt 4: Es wird $\Delta f = f(\pi_{neu}) - f(\pi_{best})$ berechnet. Bei einer Verbesserung ($\Delta f < 0$), wird π_{neu} akzeptiert ($\pi_{best} = \pi_{neu}$). Liegt eine Verschlechterung vor, erfolgt die Akzeptanz von π_{neu} mit einer Wahrscheinlichkeit $P(\pi_{neu}) = \exp(\Delta f/T')$. Die Entscheidung über die Annahme von π_{neu} wird mit Hilfe eines im Intervall [0..1] gleichverteilten Zufallsprozesses Random[0..1] getroffen. Ist die mit Hilfe dieses Zufallsprozesses gewürfelte Zahl kleiner oder gleich $\exp(\Delta f/T')$, wird π_{neu} als neuer bester Schedule π_{best} übernommen. Ist $k=L_{max}$, dann wird zu Schritt 5 verzweigt. Sonst: Wurde π_{neu} akzeptiert, erfolgt eine Verzweigung zu Schritt 2. Wurde π_{neu} nicht akzeptiert, weiter mit Schritt 3.

Schritt 5: Die Temperatur T' wird nach der in [Kirkpatrick et al. 83] angegebenen Abkühlvorschrift verringert. Diese Abkühlvorschrift hat sich in bezug auf die Konvergenz im Vergleich zu anderen Vorschriften bewährt (siehe hierzu [Zheng und Behmam 91]). Die neue Temperatur wird berechnet zu $T' = \gamma T'$. Solange T' größer als die vorgegebene Endtemperatur T_{Ende} ist, beginnt der Algorithmus erneut mit Schritt 2. Andernfalls wird die Suche beendet.

4.5 Prioritätsregelverfahren

In der industriellen Anwendung werden verbreitet heuristische Prioritätsregeln ([Panwalker und Iskander 77], [Haupt 89]) benutzt, da sie praktisch einen vernachlässigbaren Rechenzeit- und Rechenspeicherbedarf aufweisen und zudem häufig relativ gute Lösungen ermitteln (ein umfassender Vergleich für den Anwendungsfall der Maschinenbelegungsplanung findet man z.B. in [Moser 93]).

Zu Vergleichszwecken wurde in dieser Arbeit eine Auswahl der Prioritätsregeln implementiert. Die Prioritätsregeln werden während der Petrinetzsimulation zur Konfliktlösung herangezogen, d.h. sie entscheiden, welche der um eine Stelle in Konflikt stehenden, feuerbaren Transitionen tatsächlich feuern darf.

Zur Anwendung der Regeln müssen die Ankunftszeiten der Marken auf den Petrinetzstellen bekannt sein. Die Ankunftszeiten der Marken (AZM) werden deshalb während der Simulation gespeichert.

Folgende Regeln wurden implementiert:

First In First Out (FIFO):
Es feuert diejenige Transition zuerst, deren Job-Eingangsstellen zuerst markiert waren. Da Jobstellen mehrere Marken aufnehmen können, wird jeweils der Ankunftszeitpunkt der „frühesten" Marke berücksichtigt. Der früheste Zeitpunkt, zu dem alle Jobstellen des Eingangsbereichs ausreichend markiert waren, wird Anforderungszeitpunkt genannt.

Last In First Out (*LIFO*):
Es feuert die Transition mit dem spätesten Anforderungszeitpunkt.

Minimale verbleibende Zeit (MinvZ):
Zuerst feuert die Transition, die zu dem Auftrag mit der minimalen noch verbleibenden Zeit bis zu seinem vorgegebenen Fertigstellungstermin gehört. Die verbleibende Zeit bis zur

Fertigstellung errechnet sich als Differenz zwischen dem Fertigstellungstermin[16] und der Jobanforderungszeit[17]. Auch wenn die errechnete Differenz negativ ist, erhält die Transition mit der kleinsten Differenz Vorrang.

Maximale verbleibende Zeit (MaxvZ):

Hier erhält die Transition mit der maximalen verbleibenden Zeit bis zum Fertigstellungstermin Priorität.

Kürzeste Operationszeit (KO):

Die Transition mit der kürzesten Zeitdauer der Beanspruchung der konfliktbehafteten Ressourcen feuert zuerst. Die Zeitdauer der Ressourcenbeanspruchung wird aus der Summe der Zeitdauern der Transitionen berechnet, die bis zur Freigabe der Ressource feuern werden.

Kritischer Quotient (KQ):

Diese Regel priorisiert die Transition des Auftrags mit dem kleinsten Quotienten aus noch verbleibender Zeit bis zur Fertigstellung und der Operationszeit.

[16]engl. „Due Date"

[17]Die Fertigstellungstermine der Jobs werden in dem implementierten Programm mit Standardwerten belegt (z.B. 0). In einer Erweiterung könnten sie aber vom Bediener (Facharbeiter) über das Petrinetzgenerierungssystem eingegeben und an die Schedulingkomponente übergeben werden. Die Zugehörigkeit der Transitionen (und Plätze) zu den Aufträgen wird vom Petrinetzgenerierungssystem als Information in den Elementen hinterlegt.

4.6 Verhinderung von Verklemmungen

Ein Petrinetz befindet sich im Zustand einer Verklemmung, wenn keine Transition Konzession hat. Ein solcher Zustand, in dem alle Abläufe zum Stillstand kommen, ist zu verhindern, sofern es sich nicht um einen gewünschten Zielzustand handelt. In Kapitel 2 wurde dargelegt, daß es in den automatisch erzeugten Petrinetzen auch bei korrekter Anfangs- und Zielmarkierung zu Verklemmungen kommen kann. Die Gründe hierfür liegen in den mit der Anforderung von Ressourcenstellen verbundenen Situationen des *wechselseitigen Ausschlusses*, des *Haltens und Wartens* und der *Prozeßautonomie* (vgl. [Banaszak und Krogh 90]). *Halten und Warten* bedeutet, daß ein Prozeß (repräsentiert durch eine Jobmarke im Petrinetz) eine Ressource hält und erst freigibt, wenn eine andere Ressource verfügbar bzw. durch ihn belegt worden ist. *Prozeßautonomie* meint, daß ein Prozeß durch einen anderen nicht zur Freigabe von Ressourcen gezwungen werden kann. Offensichtlich sind diese Voraussetzungen für das Auftreten von Verklemmungen gegeben.

Die Bestimmung verklemmungsfreier Schaltfolgen (Schedules π) kann mit Hilfe der Erreichbarkeitsanalyse durchgeführt werden (vgl. [Hanisch 92 b]). Die T-Invarianten-Analyse liefert notwendige Bedingungen für die Verklemmungsfreiheit von Petrinetzen. Lediglich bei der speziellen Netzklasse der Synchronisationsgraphen[18] ist eine auf der Auswertung von P-Invarianten basierende notwendige und hinreichende Bedingung für die Verklemmungsfreiheit bekannt. Zur Bestimmung verklemmungsfreier Schaltfolgen liefern die Petrinetz-Analyse-Werkzeuge keinen Beitrag.

Wenn die Erreichbarkeitsanalyse aus Rechenzeit- oder -speichergründen umgangen werden soll, sind andere, schnellere, on-line-fähige Verfahren gefragt.

Der in [Viswanadham et al. 90] dargestellte Vorschlag lautet dazu, für Petrinetze mit umfangreichem Dynamikgraphen jeweils von einer aktuellen Markierung ausgehend einen Teil des Dynamikgraphen der Tiefe n zu untersuchen und anschließend diejenige Transition zu feuern, deren mögliche Folgezustände keinen Hinweis auf eine später mögliche Verklemmung geben. Diese nicht immer erfolgreiche Strategie ist bei schwachen Deadlockproblemen anwendbar.

Ein in [Banaszak und Krogh 90] beschriebenes Verklemmungsausweichprinzip garantiert verklemmungsfreie Abläufe dadurch, daß die Menge feuerbarer Transitionen abhängig von der Netzmarkierung eingeschränkt wird. Die verwendeten Verklemmungsverhinderungsregeln

[18] engl.: Marked Graphs

sind plausibel und sinnvoll und wurden deshalb in die Scheduling-Komponente dieser Arbeit integriert.

In den folgenden drei Unterabschnitten werden drei Verklemmungsverhinderungsverfahren beschrieben: Das erste besteht darin, eine *Rücksetz-Ausweich-Strategie* anzuwenden, wenn während der Anwendung der Scheduling-Prozeduren aus den Abschnitten 4.2 bis 4.5 Verklemmungen auftreten. Die Rückwärts-Ausweich-Strategie ist somit eine Ergänzung der bereits erläuterten Scheduling Verfahren. Das zweite Verfahren benutzt bei gleicher Ausweich- eine modifizierte Rücksetzstrategie. Das dritte Verfahren basiert auf der Verklemmungsverhinderungsheuristik aus [Banaszak und Krogh 90].

Die Rücksetz-Ausweich-Strategien wurden in alle bislang erläuterten Scheduling-Algorithmen integriert und können automatisch aufgerufen werden, wenn Verklemmungen auftreten.

4.6.1 Rücksetz-Ausweich-Strategie (RA-Strategie)

Befindet sich ein Petrinetz bei der Simulation - im Rahmen der Monte Carlo- und Prioritätsregel-Simulation sowie der Ermittlung von $f(\pi)$ bei den lokalen Suchverfahren (vgl. Schritt 4 der lokalen Suche bzw. Schritt 3 bei der simulierten Abkühlung) - in einer Verklemmung (vgl. Markierung im Beispiel in Bild 32), wird durch die RA-Strategie zunächst eine zurückliegende, vergangene Markierung durch das Rücksetzen von Marken wiederhergestellt und anschließend versucht, der Verklemmung mit einer alternativen Schaltfolge auszuweichen. Beim Rücksetzen wird von der zuletzt gefeuerten Transition ausgegangen (im Beispiel *t22*) und das Schalten der Transitionen des betreffenden Auftrages (hier: Auftrag 2) soweit rückwärts rückgängig gemacht, bis eine konfliktbehaftete Transition (im Beispiel *t21*) angetroffen wird. Bei dem sich anschließenden Ausweichen in Vorwärtsrichtung - realisiert als Monte Carlo Simulation - wird die konfliktbehaftete Transition aus der Liste der feuerbaren Transitionen gestrichen.

Das Verfahren führt nur bei schwach verklemmungsbehafteten Petrinetzen zum Erfolg. Bild 33 zeigt ein Petrinetz, bei dem die Deadlockauflösung mit der RA-Strategie nicht möglich ist.

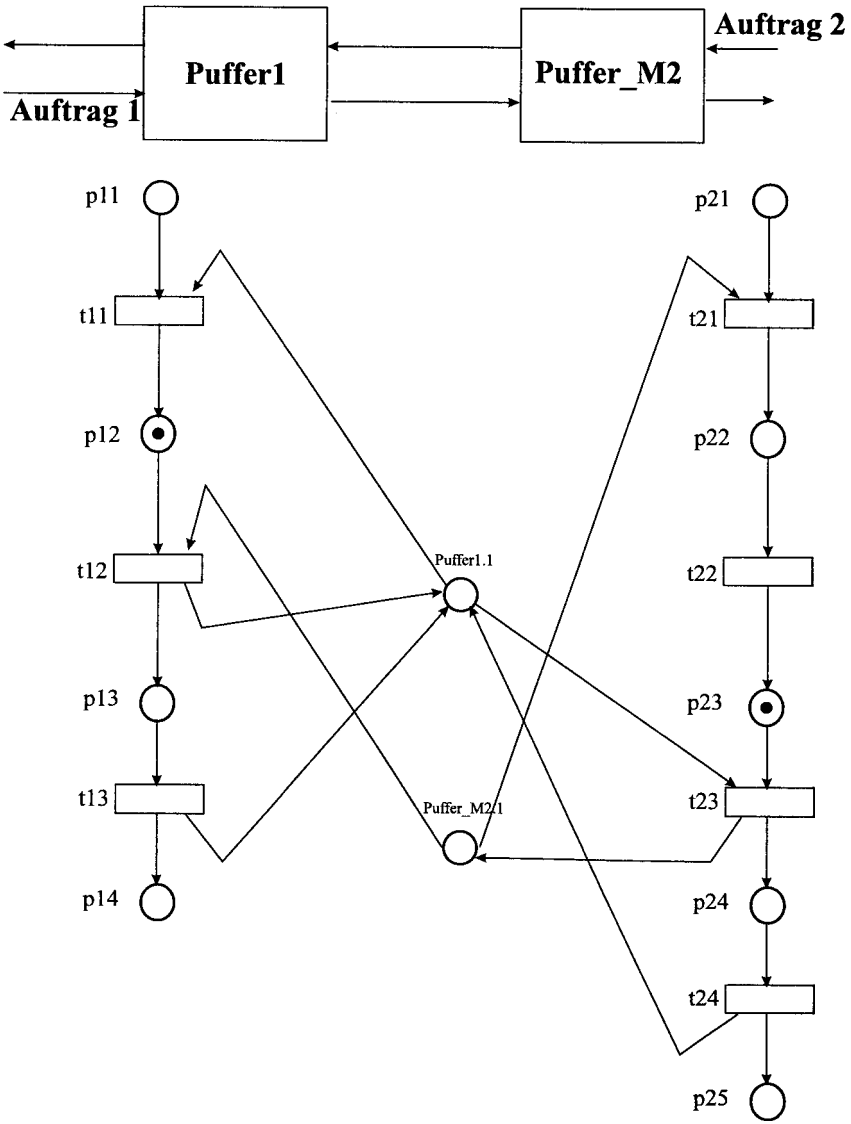


Bild 32: Petrinetz in einer Verklemmung, die beim Scheduling auftreten kann. Die gewünschte Zielmarkierung (Stellen p14 und p25 markiert) kann mit Hilfe des RA-Verfahrens ausgehend von der Verklemmung erreicht werden.

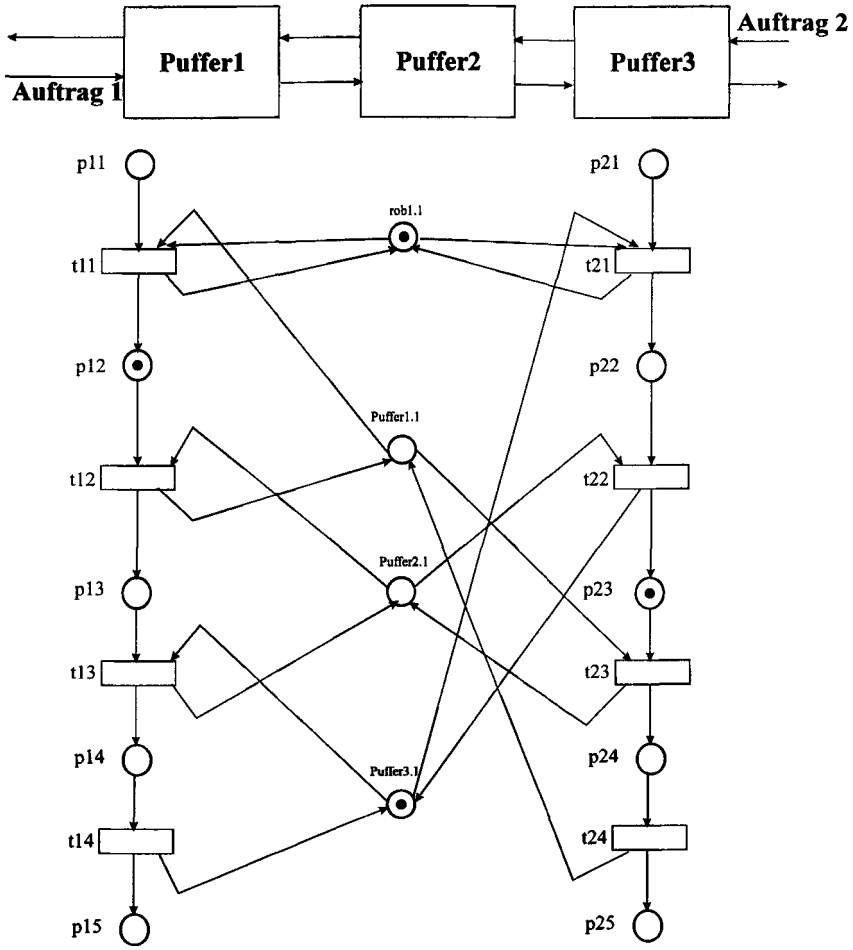


Bild 33: Die einfache RA-Strategie scheitert beim Versuch der Befreiung aus der gezeigten Verklemmung. Die erweiterte RA-Strategie ist erfolgreich.

Die Simulation ergab, daß die RA-Strategie in einem Zyklus hängen bleibt. Die zur Verbesserung der Erfolgsquote durchgeführte Erweiterung der RA-Strategie wird im folgenden erläutert.

4.6.2 Erweiterte RA-Strategie

Bei der erweiterten RA-Strategie werden während der Simulation eines Petrinetzes in einem Kurzzeitspeicher einstellbarer Länge (Standardwert 10) zurückliegende, konfliktbehaftete Markierungen abgelegt. Die bei den Markierungen ausgeführten Schritte werden ebenfalls gespeichert. Ausgehend von einer Verklemmung wird die Petrinetzmarkierung auf die im Kurzzeitspeicher hinterlegten Markierungen zurückgesetzt. Dabei erfolgt solange ein Rücksetzen auf weiter zurückliegende Markierungen, bis ein Ausweichen in Vorwärtsrichtung möglich ist oder alle gespeicherten Markierungen probiert wurden (d.h. das Verfahren scheitert).

Simulationen mit den RA-Strategien ergaben, daß das erweiterte eine größere Erfolgsquote besitzt als das einfache RA-Verfahren. Allerdings scheiterte auch die erweiterte RA-Strategie bei stark verklemmungsbehafteten Petrinetzen. Für solche Petrinetze wurde ein auf der Verklemmungsverhinderungsheuristik aus [Banaszak und Krogh 90] basierendes Verfahren implementiert. Bei diesem Verfahren werden Verklemmungen nicht durch eine Ausweichstrategie aufgelöst, sondern durch zusätzliche Scheduling-Regeln grundsätzlich vermieden.

4.6.3 Verhinderung von Verklemmungen durch zusätzliche Scheduling-Regeln

Banaszak und Krogh haben Regeln zur Verhinderung von Verklemmungen in einer Petrinetzsteuerung formuliert, die in das Monte Carlo-Simulationsverfahren und die Scheduling-Programme mit heuristischen Prioritätsregeln integriert wurden. Diese zusätzlichen Regeln werden dazu benutzt, die Menge der feuerbaren Transitionen so zu reduzieren, daß während der Monte Carlo-Simulation bzw. der Simulation mit Prioritätsregeln keine Verklemmungen auftreten können. Dadurch wird garantiert, daß eine verklemmungsfreie und optimierte Schaltfolge (Schedule π) gefunden wird.

Die Funktionsweise der Verklemmungsverhinderungsregeln wird im folgenden erläutert. Dazu wird von einem zeitbehafteten Petrinetzmodell ausgegangen, das aus Auftragsgraphen mit Baumstruktur hervorgegangen ist. Ausgehend von den Startstellen werden dann *kritische* bzw. *unkritische* Sequenzen ermittelt, in denen *kritische* oder *unkritische* Ressourcen benötigt werden. Bei der Ermittlung der *kritischen* bzw. *unkritischen* Sequenzen wird das Petrinetz auftragsweise betrachtet. Ressourcen sind *unkritisch*, wenn sie nur von dem gerade betrachteten Auftrag genau einmal angefordert werden oder wenn zwischen ihrer Anforderung und Freigabe keine weitere Ressource angefordert wird (d.h. keine *Halten und Warten*-Situation möglich ist). *Kritische* Ressourcen werden im Petrinetz mehrfach nach dem *Halten und Warten-Prinzip* angefordert und wieder freigegeben. Eine *kritische* Sequenz beginnt mit einer Transition und besteht aus einer Kette von aufeinanderfolgenden Transitionen und Stellen, in der *kritische* Ressourcen benötigt werden. Bild 34 zeigt ein Beispiel. Obwohl die Ressource R1 von Auftrag 1 und 2 angefordert wird, ist sie keine kritische Ressource, da zwischen ihrer Anforderung und Freigabe keine weitere Ressource benötigt wird (R7 und R1 werden gleichzeitig belegt; dies ist unproblematisch). Der zum Auftrag 1 korrespondierende Teil des Petrinetzes besitzt eine kritische Sequenz zwischen den Transitionen t14 und t16. Der Grund liegt in der Anforderung der Ressourcen R4 und R5, die ebenfalls von Auftrag 2 nach dem *Halten und Warten*-Prinzip angefordert werden. Die kritische Sequenz des zu Auftrag 2 korrespondierenden rechten Teils des Petrinetzes liegt zwischen den Transitionen t23 und t25.

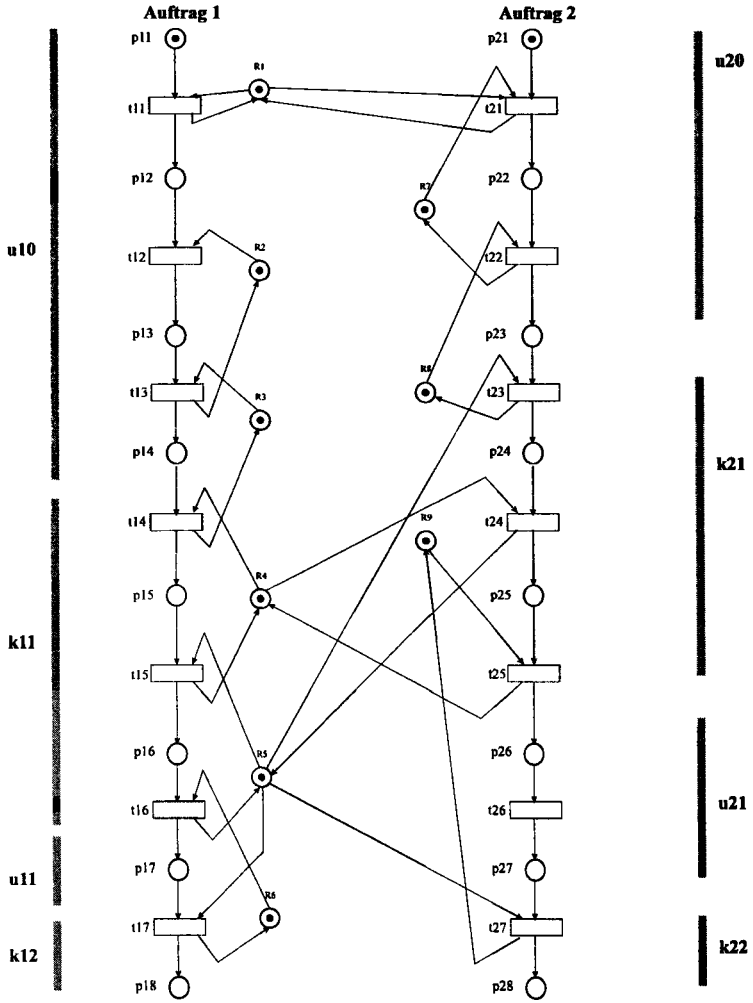


Bild 34: Beispiel zur Ermittlung von kritischen und unkritischen Sequenzen in einem Petrinetz. Der zu Auftrag 1 korrespondierende Teil des Petrinetzes enthält eine Zone z11=k11,u11; der zu Auftrag 2 gehörende Teil enthält die Zone z21=k21,u21.

Eine Folge aus einer kritischen und einer unkritischen Sequenz wird als Zone bezeichnet. Das Petrinetz aus Bild 34 enthält demnach zwei Zonen: Eine Zone (Auftrag 1) besteht aus den Sequenzen k11 und u11, die andere (Auftrag 2) aus den Sequenzen k21 und u21.

Zur Ermittlung verklemmungsfreier und optimierter Schaltfolgen wurden das Monte Carlo-Verfahren sowie die Prioritätsregelverfahren durch Verklemmungsverhinderungsregeln ergänzt. Die Verklemmungsverhinderungsregeln schränken die konzessionierten Transitionen in folgender Weise ein:

Verklemmungsverhinderungsregel 1: Eine Jobmarke von Auftrag i darf nur dann in eine neue Zone $z_{ij}=k_{ij},u_{ij}$ eintreten, wenn die Anzahl der Jobmarken in der Zone kleiner als die Kapazität der unkritischen Sequenz u_{ij} ist.

Die Kapazität einer unkritischen Sequenz wird dabei durch die Verknüpfung mit den Ressourcenstellen vorgegeben und entspricht der maximalen Anzahl von Jobmarken, die sich in der Sequenz befinden können. Die Kapazität der unkritischen Zonen u11 und u21 aus Bild 34 beträgt jeweils 1.

Verklemmungsverhinderungsregel 2: Eine kritische Ressource darf in einer Zone z_{ij} nur belegt werden, wenn alle übrigen kritischen Ressourcen der Zone verfügbar sind (d.h. die entsprechenden Ressourcenstellen markiert sind).

Während der Monte Carlo-Simulation und der Simulation mit Prioritätsregeln werden bei jeder Markierung zunächst wie in den Abschnitten 4.2 und 4.5 die Menge der konzessionierten (feuerbaren) Transitionen bestimmt. Anschließend wird diese Menge durch Anwendung der Verklemmungsverhinderungsregeln ggfls. reduziert. Danach erfolgt das Schalten eines maximalen Schrittes. Bei Konflikten erfolgt die Auswahl des Schrittes zufällig (Monte Carlo) oder entsprechend den Prioritätsregeln. Dabei ist zu beachten, daß im Vergleich zur klassischen Petrinetz-Schaltregel durch die Verklemmungsverhinderungsregeln zusätzliche Konflikte entstehen können.

In Anhang F sind für verschiedene Beispiele die erzielten Ergebnisse dokumentiert. Dort sind die Simulationsergebnisse aufgeführt, die mit einer Kombination aus Schedulingverfahren und Rücksetz-Ausweich-Strategien sowie einer Kombination aus Schedulingverfahren mit den Verklemmungsverhinderungsregeln erzielt wurden. Die Ergebnisse verdeutlichen, daß bei Anwendung der RA-Strategien bei schwach verklemmungsbehafteten Petrinetzen Schedules mit geringerem Makespan ermittelt werden können. Der Grund hierfür ist, daß die Verklemmungsverhinderungsregeln Restriktionen repräsentieren und somit die optimierte Ressourcennutzung erschweren. Bei stark verklemmungsbehafteten Petrinetzen scheitern die RA-Strategien allerdings in den meisten Fällen. Hier kann bei Anwendung der Verklemmungsverhinderungsregeln immer eine verklemmungsfreie Schaltfolge bestimmt werden.

4.7 Bewertung der Scheduling-Verfahren

Exemplarische Untersuchungen ergaben, daß das stochastische Simulated Annealing-Verfahren im allgemeinen bessere Ergebnisse als das deterministische lokale Suchverfahren oder auch die Prioritätsregeln liefert. Allerdings besteht eine Abhängigkeit der mit Simulated Annealing erzielten Güte von der Start- und Endtemperatur, der Abkühlkonstanten und der maximalen Anzahl von Veränderungen pro Temperaturniveau sowie von der Anfangslösung. Das Simulated Annealing-Verfahren lieferte bei verschiedenen Fischer/Thompson-Benchmarks jeweils die beste Lösung und verfehlte die optimale Lösung beim 6x6-Problem um 1,8%, beim 10x10-Problem um 5,6% und beim 20x5-Problem um 4,7% (vgl. Anhang E). Es erwies sich als möglich, die Verhinderung von Verklemmungen, die in praxi in Job-Shop-Systemen mit limitierten Puffern auftreten können, in Kombination mit der Scheduling-Aufgabe zu lösen. Dabei ist im Sinne einer Makespan-Minimierung zunächst die Anwendung der ad-hoc-Verklemmungsausweichstrategien indiziert. Nur beim Scheitern dieser Methoden bei stärker verklemmungsbehafteten Petrinetzen sind die garantiert erfolgreichen, aber restriktiveren und im allgemeinen in einem höheren Makespan resultierenden Verklemmungsverhinderungsregeln anzuwenden. Wenn weder Rechenzeit- noch -speicherprobleme entgegenstehen, ist eine vollständige Erreichbarkeitsanalyse beiden Ansätzen vorzuziehen.
Beim hierarchischen Scheduling (Anhang G) ist unbedingt auf die Kompatibilität der auf die Teilnetze angewendeten Verfahren untereinander und mit dem in der Aggregierungsphase auf die kritischen Ressourcen applizierten Verfahren zu achten. D.h. es ist keineswegs so, daß die Ermittlung der jeweils besten Teil-Schedules (die unter Umständen auch durch eine Erreichbarkeitsanalyse durchführbar ist) nach der Aggregierung immer zu einem guten Gesamt-Makespan führt. Vielmehr ist es z.B. sinnvoller, sowohl auf die Teilnetze als auch bei der anschließenden Aggregierung einheitlich die FIFO-Regel anzuwenden. Weiterführende Forschungsarbeiten könnten sich z.B. mit einer unter Umständen topologieabhängigen Auswahl geeigneter Scheduling-Verfahren sowie mit der Entwicklung leistungsfähiger Aggregierungsmethoden beschäftigen. Auch bedarf die Frage nach einem 'optimalen' Hierarchisierungsgrad bei der Zerlegung aus Abschnitt 3 als trade-off zwischen der Unterteilung der Petrinetze (Übersichtlichkeit, Rechenzeit) und der Scheduling-Güte weiterer Forschungsarbeiten.

5 Zusammenfassung

Die vorliegende Arbeit wurde durch den Wunsch der Fertigungsanlagenbetreiber nach anwenderfreundlichen Steuerungssystemen motiviert, die eine volle Ausschöpfung der heutzutage häufig vorhandenen Fertigungsanlagen-immanenten Flexibilität, kontinuierliche Verbesserungsprozesse sowie eine Strategievorgabe bzw. -änderung in bezug auf die Anlagennutzung vor Ort ohne den Rückgriff auf den Steuerungsentwickler unterstützen. Die Erfüllung dieses mit der Reduktion des Änderungsaufwandes verbundenen Wunsches erfordert ein agiles Steuerungssystem, das durch eine der Mutabilität der Fertigungsumgebung adäquate Flexibilität ausgezeichnet ist und vor allem die Belange einer verbesserten Mensch-Technik-Kooperation berücksichtigt.

Ausgehend von der Erkenntnis, daß mit Petrinetzen zwar die Aufgaben der Anlagensteuerung allgemein lösbar sind, eine Anlagensteuerung selbst als Petrinetzmodell realisierbar ist und die genannten Ablaufsteuerungsänderungen eine Modifikation der Petrinetze erfordern, das an der Anlage verfügbare Personal (Facharbeiter, Wartungspersonal) aber mit dem Petrinetzkonzept im allgemeinen überhaupt nicht oder in zu geringem Umfang vertraut ist und damit auch überfordert wäre, bestand die konkret zu untersuchende Aufgabe in der automatischen Umsetzung einfacher Vorgaben bzw. Änderungen der bestehenden Spezifikation in Petrinetze bzw. deren Modifikationen.

Das aus dieser Anforderung entwickelte und in der Arbeit anwendungsorientiert, in seinen einzelnen Aspekten anhand einer Testzelle exemplarisch explizierte Steuerungssystem ermöglicht nicht nur die Änderungen bzw. Erweiterungen durch das technische Personal vor Ort und reduziert damit die Wartungskosten, sondern führt schlußendlich, eine evolutionäre Verbesserung und Erweiterung bzw. Konsolidierung des Petrinetzgenerierungssystems selbst vorausgesetzt, auch zu einer Reduktion der Inbetriebnahme- bzw. Steuerungsentwicklungskosten. Darüberhinaus erwies sich die Scheduling-Aufgabe, d.h. die Ermittlung von auf die Minimierung des Makespan hin optimierten Schaltfolgen in indeterminierten Petrinetzmodellen der Steuerung, die manuell selbst für Petrinetzexperten praktisch unmöglich ist und für die das Petrinetzkonzept die Rechenzeit- und -speicher-intensive Erreichbarkeitsanalyse vorsieht, als durch direkt Petrinetz-basierte Verfahren lösbar, die bei hoher Güte einen praktisch vernachlässigbaren Rechenspeicherbedarf besitzen und bei denen die Anzahl von Such-Iterationen vorgebbar ist. Dabei sind zur Erzielung einer hohen Güte bei umfangreicheren Problemen entsprechend mehr Iterationen vorzusehen.

Es konnte gezeigt werden, daß das Problem der automatischen Hierarchisierung von Petrinetzen, dessen Lösbarkeit zunächst zweifelhaft war, tatsächlich mit Erfolg gelöst werden kann. Dabei erwies sich die dem entwickelten Hierarchisierungsverfahren zugrundeliegende Fokussierung auf die ressourcenmäßigen und im Petrinetz durch das *Mutual Exclusion*-Prinzip modellierten Interdependenzen der Aktivitäten bei gleichzeitiger Beachtung der durch den Produktionsablauf vorgeschriebenenen und nicht änderbaren sequentiellen kausalen Abhängigkeiten als effektiv und tragfähig. Der Erfolg liegt nicht nur in der Tatsache begründet, daß die generierten, hierarchischen und damit strukturierten Petrinetze für einen aufgrund von unerwarteten Problemen vor die Notwendigkeit der Überprüfung der generierten Petrinetzmodelle gestellten Petrinetzfachmann einfacher zu handhaben sind. Vielmehr impliziert die Hierarchisierung im Hinblick auf ein Scale-Up des Konzepts auf umfangreiche Systeme eine Strukturvorgabe zur Implementierung der Gesamtsteuerung in verteilten Rechnersystemen und bildet die Basis für eine hierarchische und damit den Rechenaufwand noch weiter reduzierende Scheduling-Methodik.

Die Grenzen der Änderbarkeit des Systems ohne Programmierung liegen bei der Anpassung der Kommunikationsschnittstellen zu den unterlagerten Ebenen. Diese dürfte in Zukunft verbreitet über Standardprotokolle stattfinden, wobei erst auf den unterlagerten Steuerungsebenenen eine Umsetzung in spezifische Kommandos erfolgt. Insofern ist die in dieser Arbeit vorausgesetzte Standardschnittstelle zwischen der Petrinetzkoordinierungs- und der Ausführungsebene und damit eine über eine einfache Änderung der Ressourcenmodellierung mögliche Anpassung der Schnittstellen ohne Programmierung realistisch.

6 Literatur

[Abel 90]: D. Abel: *Petri-Netze für Ingenieure*, Springer, 1990

[Ahmed et al. 93]: S. Ben Ahmed, M. Moalla, P. Esteban, M. Courvoisier: *A modular approach for the specification and validation the production flexible systems command*, IEEE Conference on Systems, Man and Cybernetics, 1993

[Al-Jaar et al. 90]: Rert Y. Al- Jaar, Alan A. Desrochers: *Performance Evaluation of Automated Manufacturing Systems using Generalized Stochastic Petri Nets*, IEEE Transactions on Robotics and Automation, Vol. 6, No. 6, Dezember 1990

[Banaszak und Krogh 90]: Zbignew Banaszak, Bruce Krogh: *Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows*, IEEE Transactions on Robotics and Automation, Vol. 6, No. 6, 1990

[Baumgarten 90]: B. Baumgarten: *Petri-Netze*, BI Wissenschaftsverlag, 1990

[Blazewicz et al. 91]: J. Blazewicz, M. Dor, J. Weglarz: *Mathematical programming formulations for machine scheduling: A survey*, European Journal of Operational Research 51, 1991, S. 283 - 300

[Bredebusch et al. 94]: A. Bredebusch, J. Lunze, H. Richter: *A Petri-Net representation of the qualitative behaviour of a dynamical continuous-time system*, International Conference on Intelligent Systems Engineering, 5. - 9. September 1994, Hamburg

[Camurri und Franchi 90]: A. Camurri, P. Franchi: *An approach to the design and implementation of the hierarchical control system of FMS, combining structured knowledge representation formalisms and high-level petri nets*, IEEE Int. Conference on Robotics and Automation, 1990

[Chao und Wang 95]: D. Y. Chao, D. T. Wang: *Knitting technique with TP-PT generations for Petri Net synthesis*, IEEE Conference on Systems, Man and Cybernetics, Vancouver, 1995

[Cheng et al. 94]: C. W. Cheng, T.H. Sun, L.C. Fu: *Petri-Net based modeling and scheduling of a Flexible manufacturing system*, IEEE Conference on Robotics and Automation, 1994

[Dittrich 95]: G. Dittrich: *Modeling of complex systems using hierarchically represented Petri nets*, IEEE Conference on Systems, Man and Cybernetics, Vancouver, 1995

[Dittrich 93]: G. Dittrich: *Modellierung komplexer Systeme mit hierarchischen Petrinetzen*, 3. Fachtagung Entwurf komplexer Automatisierungssysteme, Braunschweig, 1993

[Domschke und Drexl 95]: W. Domschke, A. Drexl: *Einführung in Operations Research*, 3. Auflage, Springer, 1995

[Domschke et al. 93]: W. Domschke, A. Scholl, S. Voß: *Produktionsplanung*, Springer, 1993

[Drexl 90]: Andreas Drexl: *Fließbandaustaktung, Maschinenbelegung und Kapazitätsplanung in Netzwerken*, Zeitschrift für Betriebswirtschaft, 60, 1990, S. 53 - 69

[Dungern 90 a]: O. v. Dungern, G. Schmidt: *Vorbereitende und begleitende Ablaufplanung für flexible Montagezellen in industrieller Umgebung*, Robotersysteme 6, 1990, S. 225 - 235

[Dungern 90 b]: O. v. Dungern: *Planungs- und Autonomiefunktionen zur Steuerung flexibler Montagezellen*, VDI Fortschrittsberichte, Reihe 8, Nr. 235, 1990

[EN 61131-3]: Deutsche Fassung der internationalen Norm IEC 1131-3: Speicherprogrammierbare Steuerungen, 1994

[Ezpeleta et al. 93 a]: J. Ezpeleta, J. Martinez, J.M. Colom: *Synthesis of live high level models for a class of FMS*, Int. Conference on Systems, Man and Cybernetics, 1993

[Ezpeleta et al. 93 b]: J. Ezpeleta, J. Martinez: *Synthesis of live models for a class of FMS*, Int. Conference on Robotics and Automation, 1993

[Fischer und Thompson 63]: H. Fischer, G.L. Thompson: *Probabilistic learning combinations of local job-shop scheduling rules*, in: J.F. Muth, G. Thompson: *Industrial Scheduling*, Prentice-Hall, 1963

[Fox 94]: M. Fox: *ISIS: A retrospective*, In: M. Zweben und M. Fox: *Intelligent Scheduling*, Morgan Kaufmann, San Francisco, 94, S. 3 - 28

[Giffler et al. 63]: B. Giffler, G.L. Thompson, V. van Ness: *Numerical experience with linear and Monte Carlo algorithms for solving production scheduling problems*, in: J.F. Muth, G. Thompson: *Industrial Scheduling*, Prentice-Hall, 1963

[Glover und Laguna 95]: *Tabu Search*, In: C.R. Reeves: *Modern heuristic techniques for combinatorial problems*, 1995, S. 70 - 150

[Glüer und Schmidt 88]: D. Glüer, G. Schmidt: *Die Anwendung von Petri-Netzen zu Modellbildung, Simulation und Steuerungsentwurf bei flexiblen Fertigungssystemen*, at 12/1988, S. 463 - 471

[Hanisch 92 a]: H.-M. Hanisch: *Berechnung optimaler diskreter Koordinierungssteuerungen auf der Grundlage zeitbewerteter Petri-Netze*, at 40, 10/1992

[Hanisch 92 b]: H.-M. Hanisch: *Petri- Netze in der Verfahrenstechnik*, Oldenbourg, 1992

[Haupt 89]: R. Haupt: *A survey of priority rule-based scheduling*, OR Spektrum 11, S. 3 - 16

[He et al. 96]: D.W. He, B. Strege, H. Tolle, A. Kusiak: *Generation of hierarchical Petri nets for manufacturing control*, Anfang 1996 eingereicht bei der Zeitschrift IEEE Transactions on Robotics and Automation

[Hoitomt 90]: D. J. Hoitomt, P. B. Luh, K. R. Pattipati: *A Lagrangian Relaxation approach to job shop scheduling problems*, IEEE Conference on Robotics and Automation, 1990

[Hörmann 92]: A. Hörmann: *Begleitende Montageablaufplanung für ein sensorgestütztes Zweiarm-Manipulator-System*, Dissertation, Universität Karlsruhe, Fakultät für Informatik, Institut für Prozeßrechentechnik und Robotik, 1992

[Huang und Zang 94]: Biqing Huang, Bo Zhang: *A new scheduling model based on Extended Petri Net - TREM Net*, IEEE Conference on Robotics and Automation, 1994

[Jeng 92]: Mu Der Jeng: *Theory and applications of resource control Petri Nets for automated manufacturing systems*, PhD Thesis, Rensselaers Polytechnic Institute, Troy, New York, 1992

[Jörns et al. 95]: C. Jörns, L. Litz, S. Bergold: *Automatische Erzeugung von SPS-Programmen auf der Basis von Petri-Netzen*, atp 37, 3/95, S. 10 - 14

[Kegel 90]: G. Kegel: *Erhöhung der Autonomie von Robotersystemen durch multisensorielle Informationen und Nutzung einer Wissensbasis*, Darmstädter Dissertationen D 17, 1990

[Kirkpatrick et al. 83]: S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi: *Optimization by Simulated Annealing*, Science, Vol. 220, No. 4598, 13 May 1983, S. 671 - 680

[Kluwe et al. 95]: M. Kluwe, V. Krebs, J. Lunze, H. Richter: *Rekonstruktion qualitativer Prozeßzustände durch ereignisdiskrete Beobachter*, at 43, 6/95

[Koh 91]: I. Koh: *A transformation theory for Petri Nets and their applications to manufacturing automation*, PhD Thesis, Rensselaers Polytechnic Institute, Troy, New York, 1991

[Kraft und Krüger 94]: H.-U. Kraft, U. Krüger: *Ausgezeichnete Logistik - Lagerverwaltung und -steuerung mit SICOMP*, Zeitschrift SICOMP Report, 1/94

[Kusiak 87]: A. Kusiak, A. Villa: *Architectures of expert systems for scheduling flexible manufacturing systems*, IEEE Int. Conference on Robotics and Automation, 1987

[Kusiak 90]: A. Kusiak: *Intelligent Manufacturing Systems*, Prentice Hall, 1990

[Lee 94]: Doo Yong Lee: *Scheduling Flexible Manufacturing Systems Using Petri Nets and Heuristic Search*, IEEE Transactions on Robotics and Automation, Vol. 10, No. 2, April 1994

[Lee und DiCesare 93]: Doo Yong Lee, F. DiCesare: *Integrated models for scheduling flexible manufacturing systems*, Int. Conference on Robotics and Automation, 1993

[Lin und Lee 94]: J. T. Lin, Chia-Chu Lee: *Modular modelling for performance evaluation of robot-centred manufacturing cells using timed Petri Nets*, Journal of Advanced Manufacturing Technology, 9/94, S. 271 - 280, Springer-Verlag

[Lunze 92]: J. Lunze: *A Petri-Net approach to qualitative modelling of continuous dynamical systems*, SAMS, Vol. 9, 1992, S. 89-111

[Lutton und Philippart 96]: J.-L. Lutton, E. Philippart: *A simulated annealing algorithm for the computation of marginal costs of telecommunication links*, In: I. Osman, P. Kelly: *Meta-Heuristics*, Kluwer, 1996, S. 265 - 276

[Mayer et al. 92]: R.J. Mayer, T.P. Cullinane, P.S. de Witte, B. Knappenberger, B. Perakath, M.S. Wells: *Information integration for concurrent engineering (IICE) IDEF3 Process Description Capture Method report*, Armstrong Laboratory, Wright-Patterson AFB, Ohio, 45433, AL-TR-1992-0057, 1992

[Matthiesen 95]: J. Matthiesen: *Informationsverarbeitungsstrukturen zur Erhöhung der Roboterautonomie*, Dissertation, TH Darmstadt, 1995

[Moser 93]: M. Moser: *Regelung der Maschinenbelegung in der flexiblen Fertigung*, VDI Fortschrittsberichte, Reihe 20, Nr. 96, 1993

[Moßig und Rehkopf 96]: K. Moßig, A. Rehkopf: *Einführung in die „Max-Plus"-Algebra zur Beschreibung ereignisdiskreter dynamischer Prozesse*, at 44, 1/1996

[Muth und Thompson 63]: J.F. Muth, G. Thompson: *Industrial Scheduling*, Prentice-Hall, 1963

[Neumann und Morlock 93]: K. Neumann, M. Morlock: *Operations Research*, Carl Hanser, 1993

[Panwalker und Iskander 77]: S.S. Panwalker, W. Iskander: *A survey of scheduling rules*, Operations Research, Vol. 25, No. 1, 1977, S. 45 - 61

[Pitschelsrieder 93]: K. Pitschelsrieder: *PetRis - Steuerung autonomer mobiler Roboter in einer Fertigungsumgebung*, In: VDI-Berichte 1094: Intelligente Steuerung und Regelung von Robotern, Tagung in Langen, 9.-10. November 1993

[Quäck 88]: L. Quäck: *Petri-Netze in der Steuerungs- und Digitaltechnik*, Oldenbourg, 1988

[Ramamoorthy und Ho 80]: C. Ramamoorthy, G. Ho: *Performance evaluation of asynchronous concurrent systems using Petri nets*, IEEE Transactions on Software Engineering, Vol. 6, No. 5, 1980, S. 440 - 449

[Rehkopf und Krebs 92]: A. Rehkopf, V. Krebs: *Modellierung und Steuerung Flexibler Fertigungssysteme mit Hilfe hierarchischer Petrinetzstrukturen*, Fachtagung Automatisierung, 20., 21.02.1992, Dresden, Band 4

[Rehkopf 92]: A. Rehkopf: *Steuerung des ereignisdiskreten Fertigungsablaufs in einer CIM-Zelle unter Einbeziehung heuristischer und analytischer Optimierungsstrategien*, Dissertation, Universität Karlsruhe, 1992

[Reuter 95]: B. Reuter: *Automatisierungssysteme im Unternehmenskontext*, 4. Fachtagung Entwurf komplexer Automatisierungssysteme, Braunschweig, Juni 1995

[Rutten 93]: W.G.M.M. Rutten: *Hierarchical mathematical programming for operational planning in a process industry*, European Journal of Operational Research 64, 1993, S. 363 - 369

[Sadeh und Thangiah 96]: N. Sadeh, S. Thangiah: *Learning to recognize (Un) promising simulated annealing runs: Efficiant search procedures for job shop scheduling and vehicle routing*, In: I. Osman, P. Kelly: *Meta-Heuristics*, Kluwer, 1996, S. 277 - 298

[Schnieder 92]: E. Schnieder: *Petrinetze in der Automatisierungstechnik*, Oldenbourg, 1992

[Shen et al. 92]: L.Shen, Q.Chen, J.Y.S.Luh: *Truncation of Petri net models for simplifying computation of optimum scheduling problems*, Computers in Industry 20, 1992, S. 25 - 43

[Shih und Sekiguchi 91]: Heloisa Shih, Takashi Sekiguchi: *ATimed Petri Net Beam Search based on-line FMS scheduling system with routing flexibility*, IEEE International Conference on Robotics and Automation, April 1991, S. 2548 - 2553

[Simon 91]: W. Simon: *Untersuchungen zu einer intelligenten und lernfähigen Rotersteuerung für die Montageautomatisierung*, VDI Fortschrittsberichte, Reihe 20, Nr. 47, 1991

[Starke 90]: P. Starke: *Analyse von Petri-Netz-Modellen*, Teubner, 1990

[Strege et al. 96]: B. Strege, A. Weigl, A. Gros: *Scheduling of disassembly cells based on Timed Extended Controlled Petri Nets*, im Januar 1996 als Veröffentlichung akzeptiert für eine zukünftige Ausgabe der Zeitschrift *Flexible Automation and Intelligent Manufacturing*

[Strege und Pham 95]: B. Strege, Pham Quang Bac: *Scheduling Based on Timed Extended Controlled Petri Nets*, Third European Control Conference ECC, Rom, 1995

[Strege und Tolle 93]: B. Strege, H. Tolle: *Autonomous reaction on faults in FMCs by modification of Petri Nets through meta rules*, Int. Conference CARS & FOF, 1993

[Stulle 95]: M. Stulle: *Die ereignisdiskrete Beobachtungsaufgabe im Kontext des Koordinationssteuerungsentwurfs für flexible Fertigungssysteme*, 4. Fachtagung Entwurf komplexer Automatisierungssysteme, 7.-9- Juni 1995, Braunschweig, S. 585 - 596

[Tempelmeier und Kuhn 93]: Horst Tempelmeier, Heinrich Kuhn: *Flexible Fertigungssysteme*, Springer, 1993

[U.S. Airforce 81]: *Integrated Computer Aided Manufacturing (ICAM) architecture part II, Volume IV - functional modeling manual (IDEF0)*, Air Force materials laboratory, Wright-Patterson AFB, Ohio 45433, AL-TR-81-81-4023, 1981

[Valette 79]: R. Valette: *Analysis of Petri Nets by stepwise refinements*, Journal of Computer Systems Science, Vol. 18, No. 1, 1979, S. 35 - 46

[Viswanadham et al. 90]: N. Viswanadham et al.: *Deadlock prevention and deadlock avoidance in Flexible Manufacturing Systems using Petri net modells*, IEEE Transactions on Robotics and Automation, Vol. 6, No. 6, 1990, S. 713 - 723

[Wagner 69]: H. M. Wagner: *Principles of Operations Research*, Prentice-Hall, 1969

[Zheng und Behnam 91]: Zheng-Ping Lo, Behnam Bavarian: *Job scheduling on parallel machines using simulated annealing*, IEEE Conference on Systems, Man and Cybernetics, 1991

[Zhou et al. 89]: MengChu Zhou, F. DiCesare, A. Desrochers: *A top-down approach to systematic synthesis of Petri Net models for manufacturing systems*, IEEE Conference on Robotics and Automation, 1989

[Zhou und DiCesare 90]: MengChu Zhou, F. DiCesare: *Modeling buffers in automated manufacturing systems using petri nets*, Rensselaers Int. Conference on CIM, 1990

[Zhou und DiCesare 91]: MengChu Zhou, F. DiCesare: *Parallel and sequential mutual exclusions for Petri Net modeling of manufacturing systems with shared resources*, IEEE Transactions on Robotics and Automation, Vol. 7, No. 4, August 1991, S. 515 - 527

[Zhou und DiCesare 96]: MengChu Zhou, Frank DiCesare, *Petri net modeling of buffers in Automated Manufacturing Systems*, IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics, Vol. 26, No. 1, Februar 1996, S. 157 - 164

Während der Arbeit entstandene Veröffentlichungen im Überblick:

B. Strege: *Einsatz wissensbasierter Methoden zur Koordinierung und Steuerung von Sonderbetriebsarten in Flexiblen Fertigungssystemen*, Workshop Modellierung für wissensbasierte Systeme in technischen Anwendungen, 15.-17.02.1993, Emmelndorf, Hamburg, 1993

B. Strege, K. D. Bettenhausen: *Intelligente Mensch-Maschine-Systeme in der Automatisierungstechnik*, DGLR-Fachausschußsitzung *Optimaler Automatisierungsgrad von Mensch-Maschine-Systemen*, 1993, Berlin

B. Strege, H. Tolle: *Autonomous reaction on faults in FMCs by modification of Petri Nets through meta rules*, Int. Conference CARS & FOF, 1993

B. Strege, Pham Quang Bac: *Autonomous treatment of special system states in flexible manufacturing cells by integration of knowledge based reaction planning, scheduling and Petri Net modifications*, Int. Conference on CAD/CAM, Robotics ans Factories of the Future, Ottawa, 1994

B. Strege, A. Gücker: *Scheduling Deadlock-kritischer Systeme auf der Basis von Timed Extended Controlled Petri Nets*, 4. Fachtagung Entwurf komplexer Automatisierungssysteme, Braunschweig, 1995

B. Strege, Pham Quang Bac: *Scheduling Based on Timed Extended Controlled Petri Nets*, Third European Control Conference ECC, Rom, 1995

B. Strege, H. Loydl: *Automatic transformation of simple user commands to hierarchical Petri nets*, IEEE Conference on Systems, Man and Cybernetics, Vancouver, 1995

B. Strege, A. Weigl, A. Gros: *Scheduling of disassembly cells based on Timed Extended Controlled Petri Nets*, im Januar 1996 als Veröffentlichung akzeptiert für eine zukünftige Ausgabe der Zeitschrift *Flexible Automation and Intelligent Manufacturing*

Anhang

Anhang A: Petrinetzbasismodule

Hier sind die Petrinetzbasismodule zu den folgenden Operationstypen abgebildet:
Werkzeug montieren/demontieren/wechseln, Werkstück bearbeiten, Objekt demontieren, FIFO-Puffern, LIFO-Puffern.

Petrinetzmodul *Werkzeug montieren/demontieren/wechseln:*

Flexible Fertigungssysteme sind häufig mit Werkzeugwechselsystemen ausgestattet, die eine Bestückung der Roboter mit verschiedenen Endeffektoren erlauben. Der Vorgang der Werkzeugmontage bzw. -demontage wird durch die Petrinetzmodule in Bild 35 bzw. Bild 36 dargestellt. Die Erläuterungen beschränken sich auf die Werkzeugmontage. Zum Starten der Operation *Werkzeug montieren* (im Beispielmodul aus Bild 35 wird ein Greifer montiert) muß der Roboter bereit (*Roboter.1*) und der Greifer im Magazin vorhanden sein (*Greifer.1* markiert).

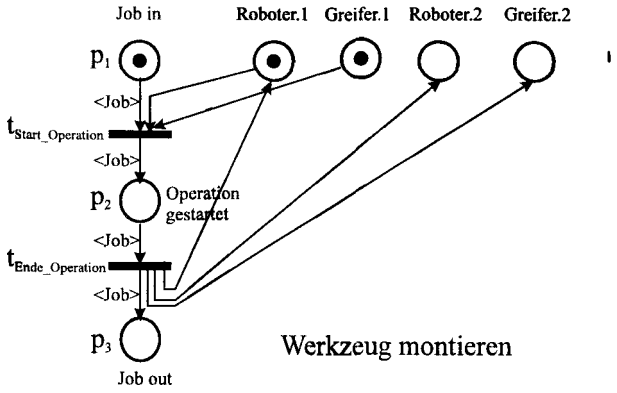
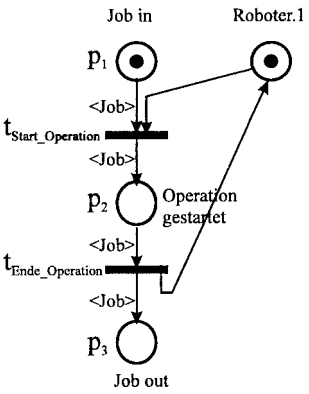


Bild 35: Petrinetzbasismodul *Werkzeug montieren.*

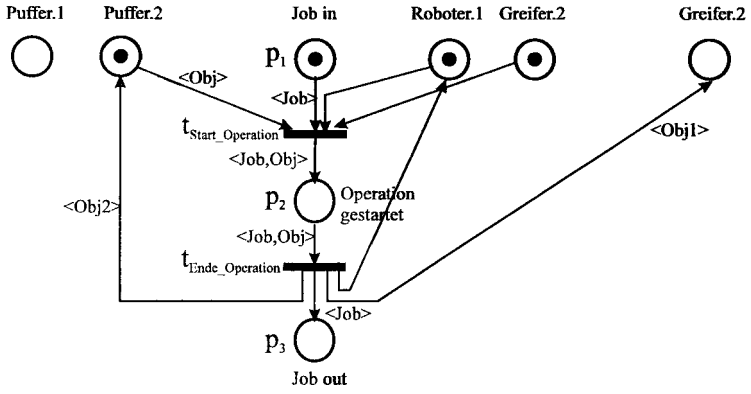


Bearbeitung durch Roboter

Bild 38: Petrinetzbasismodul zur Werkstückbearbeitung durch einen Werkzeug-bestückten Roboter.

Petrinetzmodul *Objekt demontieren:*

Die Demontage eines Objektes ist in Bild 39 zu sehen. Ein Beispiel hierfür ist die Demontage der Rückwand eines in einer Halterung (Puffer) fixierten Fernsehers durch einen Roboter. Der Roboter greift die bereits vorher vom Rahmen getrennte Rückwand und hält sie im Greifer.



Objekt demontieren

Bild 39: Demontage einer Komponente eines zusammengesetzten Objektes. Aus einem auf dem Puffer liegenden Objekt wird eine Komponente entfernt. Die Komponenten-Marke wandert über die Kante <Obj1>, der Rest des Objektes über die mit <Obj2> beschriftete Kante. Beim Feuern der Endetransition werden also Marken erzeugt. Die Bezeichnung der Marken kann aus einem Produktmodell[19] entnommen werden und durch den Bediener als Parameter bei der Operationenvorgabe angegeben werden.

[19] in der prototypischen Systemimplementierung nicht vorhanden.

Petrinetzmodul *FIFO-Puffern:*

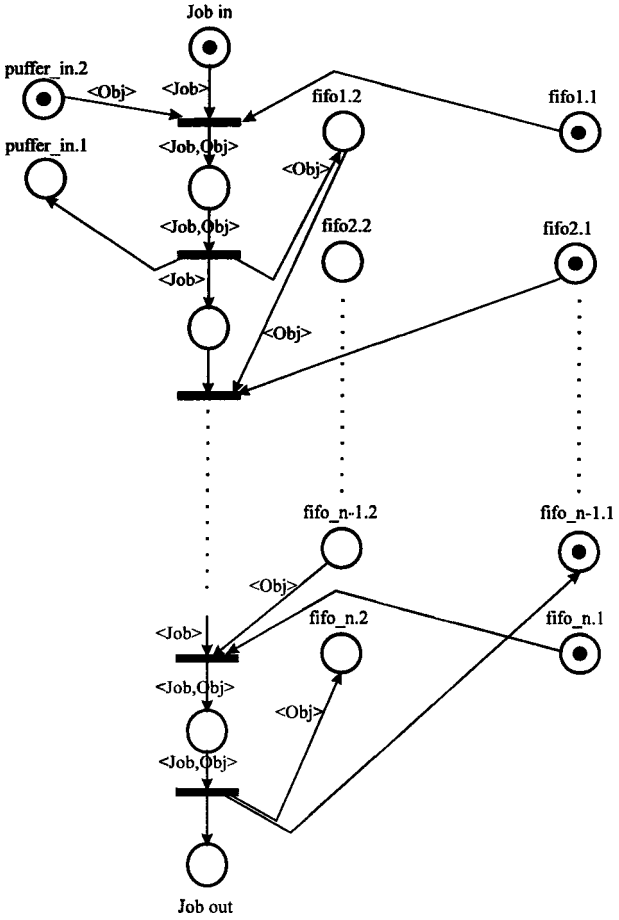


Bild 40: FIFO-Puffern-Petrinetzmodul für einen Puffer mit der Kapazität n. Dieses Modul bildet den aktiven Vorgang des FIFO-Pufferns ab. Ein solcher Vorgang kann mit einem Transport verbunden sein (z.B. Transport auf einem FIFO-Förderband). Aber auch ein Puffern ohne Transportvorgang in einem dafür vorgesehenen Werkstückspeicher ist hiermit modellierbar.

Petrinetzmodul *LIFO-Puffern:*

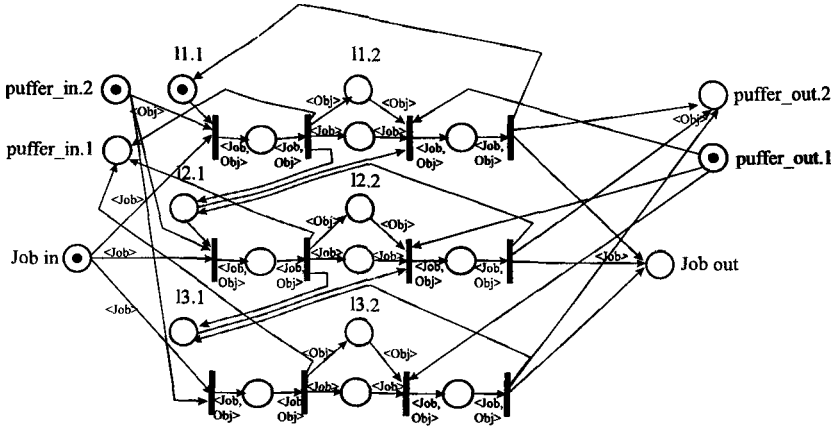
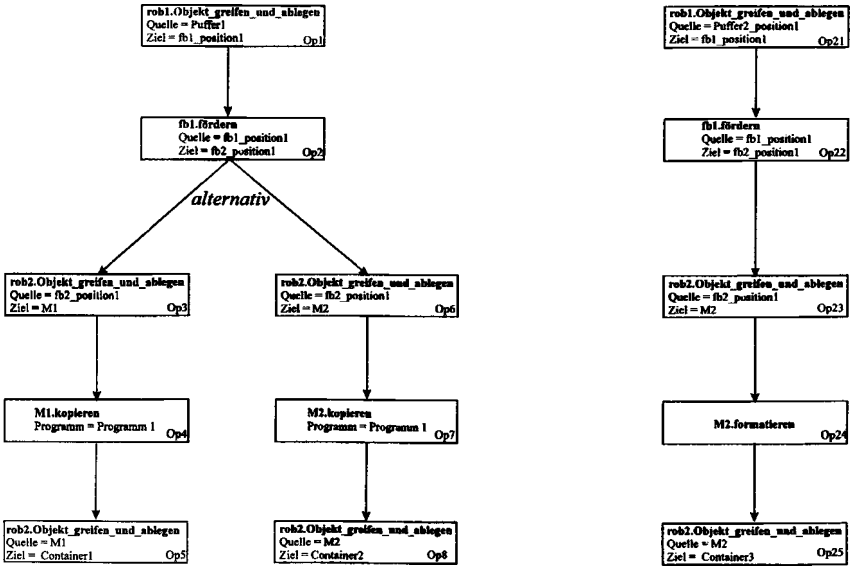


Bild 41: LIFO-Puffern-Petrinetzmodul für einen LIFO-Puffer mit der Kapazität 3.

puffer_in: Eingangspuffer des LIFO-Puffers (=Ausgangspuffer des vorgeschalteten Puffers). *puffer_out*: Ausgangspuffer des LIFO-Puffers (=Eingangspuffer des nachgeschalteten Puffers). *l1, l2, l3*: Die drei Pufferplätze des LIFO-Puffers.

Anhang B: Petrinetze: Zwei Beispiele

Beispiel 1: 2 Aufträge



Auftragsgraph zum Auftrag 1

Auftragsgraph zum Auftrag 2

Bild 42: Ein Beispiel: Zwei Aufträge. Die Container 1, 2 und 3 gehören zur Klasse der indifferenten Random-Puffer und besitzen unbegrenzte Kapazität.

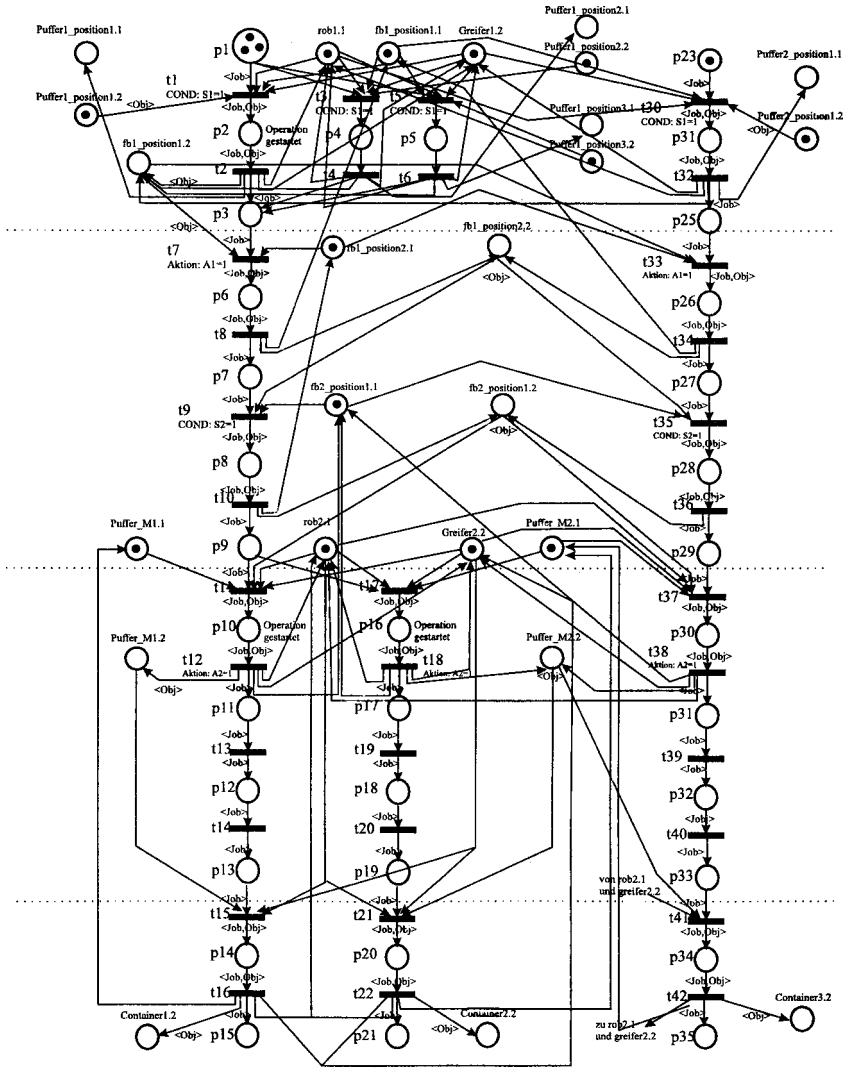


Bild 43: Das synthetisierte Petrinetz (Beispiel 1).

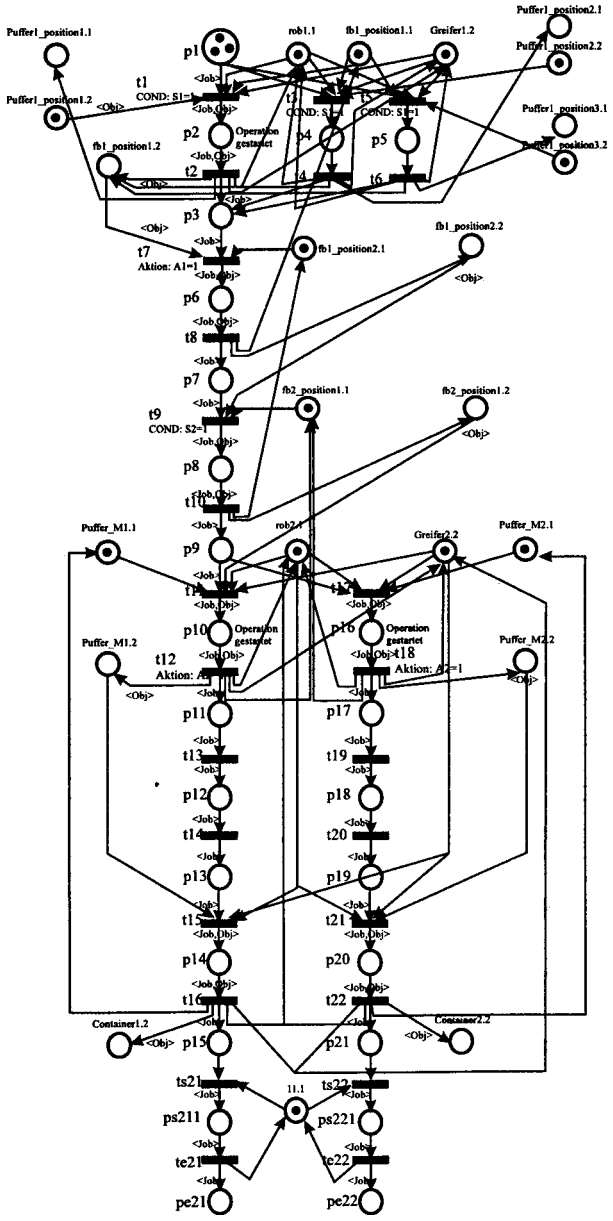


Bild 44: **Beispiel 2 (1 Auftrag):** Petrinetz zum Auftragsgraphen entsprechend Bild 24 (vgl. Anhang D zur Bedeutung der Stellen und Transitionen).

Anhang C: Sequentieller Gruppierungsalgorithmus

Es werden im folgenden die einzelnen Schritte einer Zerlegungsiteration mit Hilfe des sequentiellen Gruppierungsalgorithmus am Beispiel des IDEF-Modells aus Bild 22 erläutert. Das Resultat dieser Zerlegungsiteration wurde in Abschnitt 3 bereits gezeigt (Bild 24).

Schritt 0: G'=G=∅, k=1.

Matrix $A^{(1)}$:

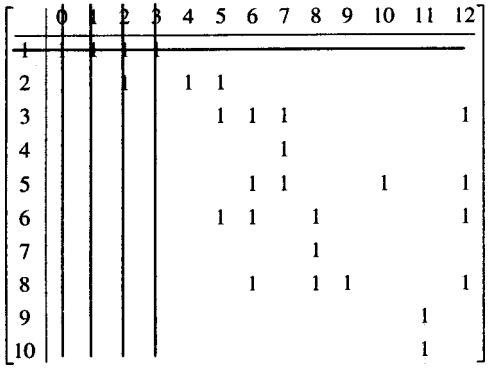
	0	1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1									
2			1		1	1							
3						1	1	1					1
4								1					
5							1	1			1		1
6						1	1		1				1
7									1				
8							1		1	1			1
9												1	
10												1	

Matrix $A^{(1)}$ nach Schritt 1:

	0	1	2	3	4	5	6	7	8	9	10	11	12
~~1~~	~~1~~	~~1~~	~~1~~	~~1~~									
2			1		1	1							
3						1	1	1					1
4								1					
5							1	1			1		1
6						1	1		1				1
7									1				
8							1		1	1			1
9												1	
10												1	

G'={1}

Matrix $A^{(1)}$ nach Schritt 2:

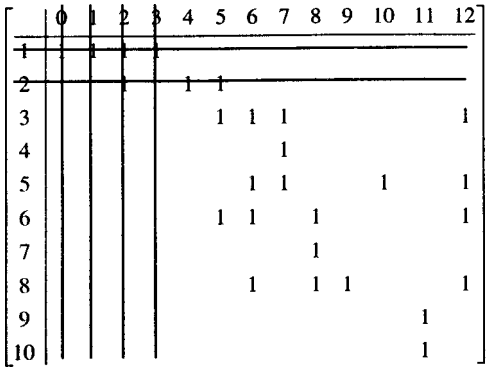


	0	1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1									
2			1		1	1							
3						1	1	1					1
4								1					
5							1	1			1		1
6						1	1		1				1
7									1				
8							1		1	1			1
9												1	
10												1	

G'={1}.

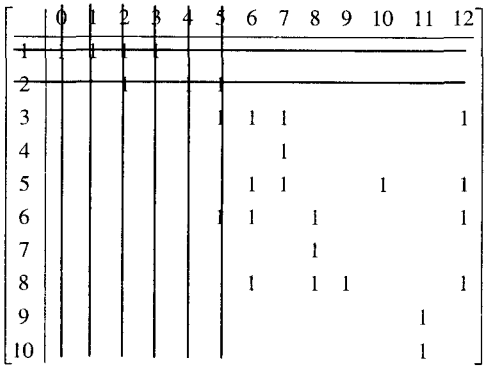
Nach Schritt 3:

G'=G={1, 2}. $A^{(1)}$:



	0	1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1									
2			1		1	1							
3						1	1	1					1
4								1					
5							1	1			1		1
6						1	1		1				1
7									1				
8							1		1	1			1
9												1	
10												1	

Nach Schritt 4:

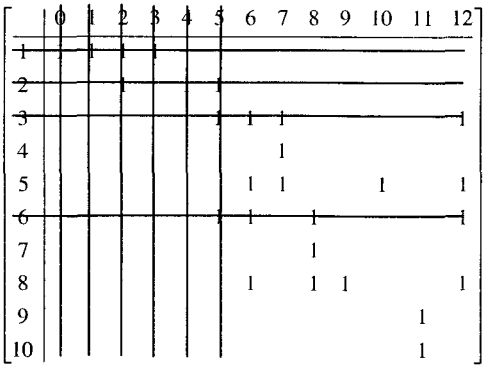


	0	1	2	3	4	5	6	7	8	9	10	11	12
~~1~~	~~1~~	~~1~~	~~1~~	~~1~~									
~~2~~			~~1~~			~~1~~							
3						1	1	1					1
4								1					
5							1	1			1		1
6						1	1		1				1
7									1				
8							1		1	1			1
9												1	
10												1	

Nach Schritt 3:

G'=G={1, 2, 3, 6}. Op3 und Op6 erfüllen die *Sequenzenbedingung* gemäß Regel 1.

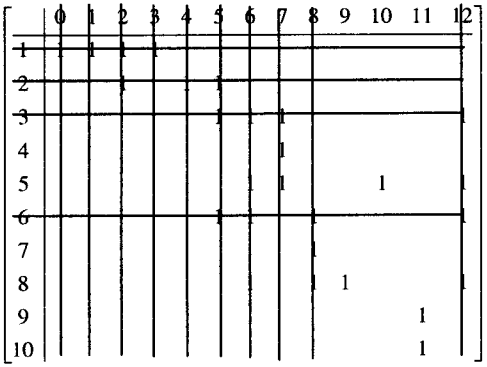
$A^{(1)}$:



	0	1	2	3	4	5	6	7	8	9	10	11	12
~~1~~	~~1~~	~~1~~	~~1~~	~~1~~									
~~2~~			~~1~~			~~1~~							
~~3~~						~~1~~	~~1~~	~~1~~					~~1~~
4								1					
5							1	1			1		1
~~6~~						~~1~~	~~1~~		~~1~~				~~1~~
7									1				
8							1		1	1			1
9												1	
10												1	

Nach Schritt 4:

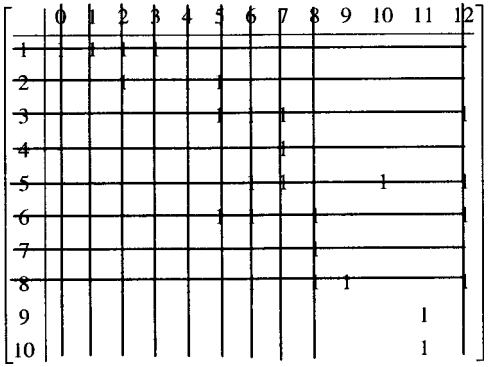
$A^{(1)}$:



Nach Schritt 3:

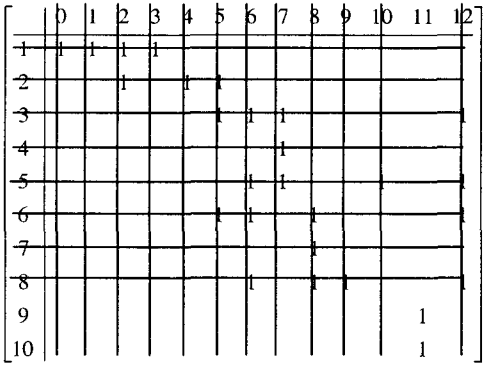
G'=G={1, 2, 3, 4, 5, 6, 7, 8}. Op4, Op5, Op7 und Op8 erfüllen die *Sequenzenbedingung* gemäß Regel 1.

$A^{(1)}$:



Nach Schritt 4:

$A^{(1)}$:



	0	1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1									
2			1		1	1							
3						1	1	1					1
4								1					
5							1	1				1	1
6						1	1		1				1
7									1				
8							1		1	1			1
9												1	
10												1	

Nach Schritt 5:

Der erste Teilgraph TG1 besteht aus den Operationen 1, 2, 3, 4, 5, 6, 7 und 8.

Nach Schritt 6:

$$A^{(2)} = \begin{array}{|c|c|} \hline & 11 \\ \hline 9 & 1 \\ 10 & 1 \\ \hline \end{array}$$

Nach Schritt 7:

k=2; G'=G=∅.

Nach den weiteren Schritten 1, 2 und 3 ist die zweite Operationengruppe (Teilgraph TG2) bestehend aus den Operationen 9 und 10 ermittelt und damit die erste Zerlegungsiteration beendet.

Anhang D: Hierarchisches Petrinetz zur Testzelle (Beispiel 2)

Es gelten: s_{max}=3 für die *sequentielle Ressourcennutzungsregel*

n_{max}=0.6 für die *häufigste Ressourcennutzungsregel*

Mindestgröße eines Teilgraphen als Bedingung für eine weitere Zerlegungsiteration: 2 Operationen.

Zerlegungsiteration 1:

(in Anhang C Einzelschritten dargelegt)

Resultat: Teilgraphen: TG1 = Op 1, 2, 3, 4, 5, 6, 7, 8; TG2 = Op 9, 10 (siehe Bild 24).

Zerlegungsiteration 2:

Anwendung des SGA auf den Teilgraphen TG1:

Die Ressourcen 7 und 8 werden zunächst aus der Operationen-Ressourcen-Matrix von Teilgraph TG1 gestrichen (sequentielle Ressourcennutzungsregel: Ressourcen 7 bzw. 8 werden von den drei aufeinanderfolgenden Operationen 3, 4 und 5 bzw. 6, 7, und 8 benötigt).

Resultat: Teilgraphen: TG11 = Op 1, 2, 3, 6; TG12 = Op 4, 5, 7, 8.

Anm.: Die Operationen 5 und 8 bilden zusammen mit den Operationen 4 und 7 den Teilgraphen TG12 und werden nicht einzeln jeweils einem Teilgraphen der Größe 1 zugeordnet.

Zerlegungsiteration 3:

3.1 Anwendung des SGA auf den Teilgraphen TG11:

Resultat: Teilgraphen: TG111 = Op 1, 2; TG112 = Op 3, 6.

3.2 Anwendung des SGA auf den Teilgraphen TG12:

Resultat: keine weitere Zerlegung mehr möglich.

Die **Zerlegungsiteration 4** entfällt, weil als Mindestgröße der Teilgraphen für die Anwendung einer weiteren Iteration eine Größe von 2 Operationen definiert wurde.

Bild 45 enthält das resultierende hierarchische Petrinetz[20]. Auf der obersten Ebene 0 besteht das Petrinetz aus den Modulen TG1 und TG2, die über die Transitionen tk1 und tk2 gekoppelt sind. Auf der Ebene 1 ist rechts das Teilnetz zum Modul TG2 abgebildet, dem keine weiteren Hierarchiestufen untergeordnet sind. Zur besseren Übersichtlichkeit wurden im Bild 45 alle Kantenanschriften weggelassen. Auf der Ebene 1 unterteilt sich das Modul TG1 in zwei weitere Module TG11 und TG12, die untereinander durch die automatisch ergänzten Transitionen tk3 und tk4 verbunden sind. Die Randstellen der Module (Petrinetzstellen ps11, p15, p21, ps21, ps22, pe21, pe22, p11, p17, ps121, ps122, p15, p21, ps1121 und ps1122) existieren nur einmal, werden aber in den Modulsymbolen als Randstellen und auf den unterlagerten Ebenen mehrfach angegeben (dies entspricht der in dieser Arbeit definierten Notation für Petrinetzmodule).
Die Ressourcenstellen 6.1, 7.1, 7.2, 8.1, 8.2, und 12.2 werden der Ebene 1 zugeordnet, weil sie mit Transitionen aus dem Teilnetz TG12 auf der untergeordneten Ebene 2 und mit Transitionen aus Teilnetz TG112 auf der Ebene 3 verbunden sind. In einem Software-Werkzeug, das die hierarchischen Netze dem Bediener anzeigen soll, könnten auf der Ebene 1 spezielle Linien zwischen diesen Ressourcenstellen und den Modulsymbolen TG11 und TG12 angezeigt werden. Damit wäre unmittelbar für den Betrachter ersichtlich, daß diese Stellen mit den Modulen TG11 und TG12 auf den unterlagerten Ebenen verknüpft sind und TG11 und TG12 über Ressourcenstellen verkoppelt sind.
Die Ressourcenstellenstellen 0.2, 1_1.1, 1_1.2, 1_2.1, 1_2.2, 1_3.1, 1_3.2, 2.1, 2.2, 3.1, 4.1 und 4.2 werden direkt in das Teilnetz TG111 aufgenommen, weil sie nur mit Transitionen aus diesem Teilnetz verknüpft sind. Das gleiche gilt sinngemäß für die Ressourcenstellen 9.2 und 10.2 bei Teilnetz TG12. Die Ressourcenstellen 5.1 und 5.2, die mit Transitionen aus den Teilnetzen TG111 und TG112 auf der Ebene 3 verbunden sind, sind der nächst höheren Ebene 2 zugeordnet.

[20] Ein Programm zur grafischen Darstellung automatisch synthetisierter hierarchischer Petrinetze durch den Rechner wurde nicht implementiert. Es handelt sich hierbei, wie auch bei allen anderen abgebildeten Petrinetzen in dieser Arbeit, um eine Zeichnung. Für nicht-hierarchische Petrinetze wurde in den System-Prototypen allerdings ein Visualisierungsprogramm integriert.

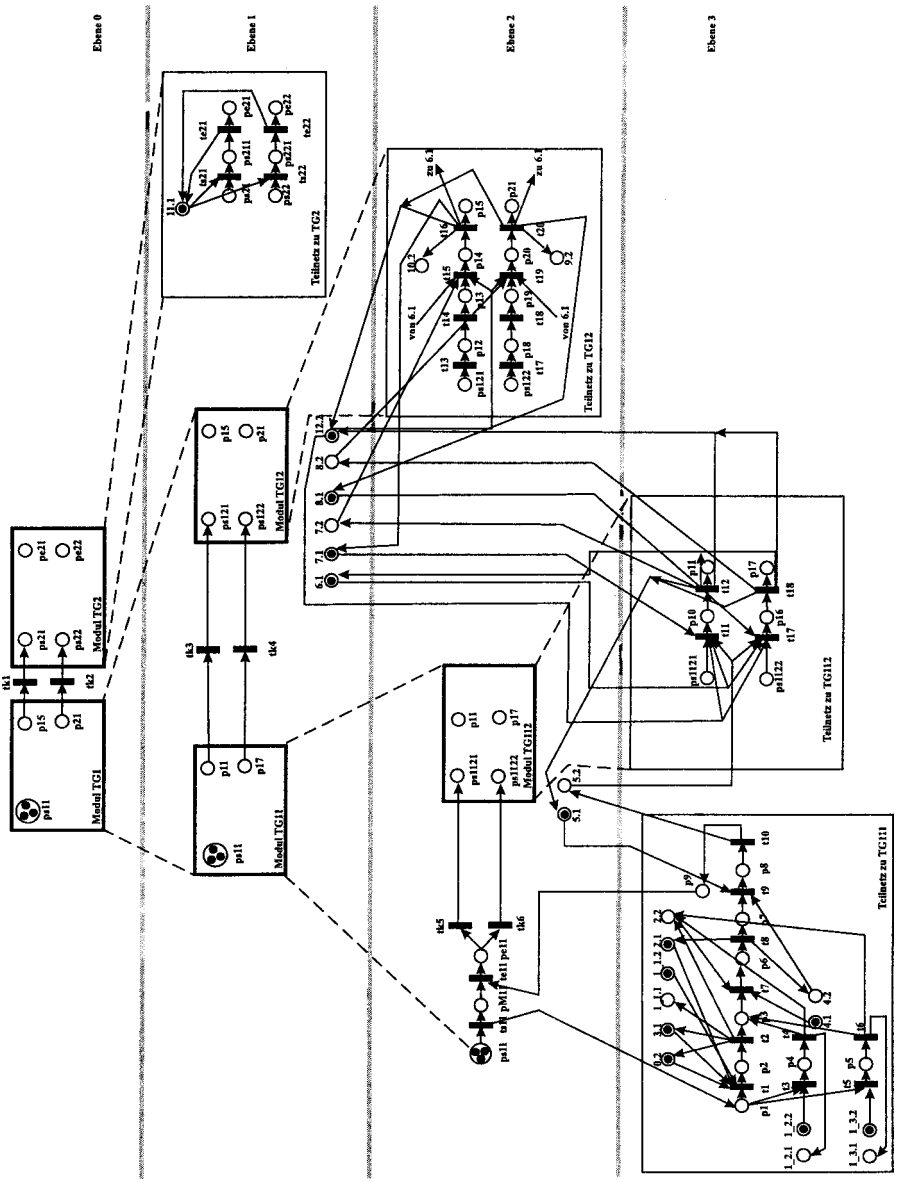


Bild 45: Hierarchisches Petrinetz für die Testzelle und den Auftragsgraphen aus Bild 22. Das korrespondierende nicht hierarchische Petrinetz ist in Anhang B abgebildet (Beispiel 1).

Bedeutung der in Bild 45 abgekürzt gekennzeichneten Ressourcenstellen:

0.2: Greifer1.2

1_1.1: Puffer1_position1.1

1_1.2: Puffer1_position1.2

1_2.1: Puffer1_position2.1

1_2.2: Puffer1_position2.2

1_3.1: Puffer1_position3.1

1_3.2: Puffer1_position3.2

2.1: fb1_position1.1

2.2: fb1_position1.2

3.1: rob1.1

4.1: fb1_position2.1

4.2: fb1_position2.2

5.1: fb2_position1.1

5.2: fb2_position1.2

6.1: rob2.1

7.1: Puffer_M1.1

7.2: Puffer_M1.2

8.1: Puffer_M2.1

8.2: Puffer_M2.2

9.2: Container2.2

10.2: Container1.2

11.1: Ressource, die bei den Operationen 9 und 10 (Bild 24) benötigt wird (z.B. eine Einrichtung zur Positionierung der Disketten)

12.2: Greifer2.2

Bedeutung der Jobstellen:

ps11: Startstelle von Modul TG1

pM11: Merkerstelle zu TG111

pe11: TG111 beendet

p1: bereit für Op1

p2: Op1 gestartet

p4: Op1 gestartet

p5: Op1 gestartet

p3: Op1 beendet

p6: Diskette auf fb1_position_1

p7: Diskette auf fb1_position_2

p8: Diskette auf fb2_position_1

p9: Diskette auf fb2_position_2

ps1121: Startstelle von Modul TG112: bereit für Op3

ps1122: Startstelle von Modul TG112: bereit für Op6

p10: Op3 gestartet

p16: Op6 gestartet

p11: Op3 beendet

p17: Op6 beendet

ps121: Startstelle von Modul TG12: bereit für Op4

ps122: Startstelle von Modul TG12: bereit für Op7

p12: Op4 gestartet

p18: Op7 gestartet

p13: Op4 beendet

p19: Op7 beendet

p14: Op5 gestartet

p20: Op8 gestartet

p15: Op5 beendet

p21: Op8 beendet

ps21: Startstelle von Modul TG2: bereit für Op9

ps22: Startstelle von Modul TG2: bereit für Op10

ps211: Op9 gestartet

ps221: Op10 gestartet

pe21: Op9 beendet

pe22: Op10 beendet

Bedeutung der Transitionen:

tk1 bis tk6: Koppeltransitionen

ts11: Starttransition von Teilnetz TG111

te11: Endetransition von Teilnetz TG111

t1: Start von Op1

t3: Start von Op1

t5: Start von Op1

t2: Ende von Op1

t4: Ende von Op1

t6: Ende von Op1

t7, t8, t9, t10: Transitionen des FIFO-Förderbandes fb1

t11: Start von Op3

t17: Start von Op6

t12: Ende von Op3

t18: Ende von Op6

t13: Start von Op4

t17: Start von Op7

t14: Ende von Op4

t18: Ende von Op7

t15: Start von Op5

t19: Start von Op8

t16: Ende von Op5

t20: Ende von Op8

t21: Start von Op9

t22: Start von Op10

te21: Ende von Op9

te22: Ende von Op10

Anhang E: Fischer/Thompson-Benchmarks

In den nachfolgenden Tabellen sind die Scheduling-Ergebnisse für das 6x6-, 10x10- und 20x5-Problem aus [Fischer und Thompson 63] dokumentiert.

Es bedeuten:

MC: Monte Carlo; LS: Lokale Suche; LS-SA: Lokale Suche mit simulierter Abkühlung; FIFO: First In First Out; LIFO: Last In First Out; KO: Kürzeste Operationszeit-Regel; MinvZ: Minimale verbleibende Zeit-Regel; MaxvZ: Maximale verbleibende Zeit-Regel; KQ: Kritischer Quotient-Regel;

Tabelle 1: 6x6-Problem (6 Aufträge mit jeweils 6 Operationen auf jeweils 6 verschiedenen Maschinen). Makespan der optimalen Lösung: 55.

	MC	LS	LS-SA*)	FIFO	LIFO	KO	MinvZ	MaxvZ	KQ
Makespan	59	59	56	78	138	88	68	70	70
Anzahl der Iterationen	100	8	160	1	1	1	1	1	1

*): bestes Ergebnis bei $T'_{Start}=50$, $\gamma=0.9$, $T'_{End}=10$, $L_{max}=10$

Tabelle 2: 10x10-Problem (10 Aufträge mit jeweils 10 Operationen auf jeweils 10 verschiedenen Maschinen). Makespan der optimalen Lösung: 930.

	MC	LS	LS-SA*)	FIFO	LIFO	KO	MinvZ	MaxvZ	KQ
Makespan	1117	1102	983	2400	2300	1074	1262	1281	1300
Anzahl der Iterationen	20	8	450	1	1	1	1	1	1

*): Bestes Ergebnis für T'_{Start}=100, γ=0.95, T'_{End}=10, L_{max}=10; Rechenzeit auf PC 486/33 Mhz: ca. 5 Minuten.

In den folgenden Tabellen sind die Scheduling-Ergebnisse für verschiedene Parameter der lokalen Suche mit simulierter Abkühlung enthalten:

Tabelle 3

	γ = 0.95, L_{max} = 10, T'_{End} = 10			γ = 0.9, L_{max} = 10, T'_{End} = 10		
Makespan	T'_{Start}=100	T'_{Start}=50	T'_{Start}=20	T'_{Start}=100	T'_{Start}=50	T'_{Start}=20
	983	1065	1117	1102	1102	1102
Iterationen	450	320	140	220	160	70

Tabelle 4

	$\gamma = 0.8$, $L_{max} = 10$, $T'_{End} = 10$			$\gamma = 0.5$, $L_{max} = 10$, $T'_{End} = 10$		
Makespan	T'_{Start}=100	T'_{Start}=50	T'_{Start}=20	T'_{Start}=100	T'_{Start}=50	T'_{Start}=20
	1030	1062	1117	1062	1088	1117
Iterationen	110	80	40	40	30	10

Tabelle 5

	$\gamma = 0.95$, $L_{max} = 5$, $T'_{End} = 10$			$\gamma = 0.5$, $L_{max} = 5$, $T'_{End} = 10$		
Makespan	T'_{Start}=100	T'_{Start}=50	T'_{Start}=20	T'_{Start}=100	T'_{Start}=50	T'_{Start}=20
	1018	1117	1117	1108	1117	1117
Iterationen	225	160	70	20	15	5

Tabelle 6

	$\gamma = 0.95$, $L_{max} = 2$, $T'_{End} = 10$			$\gamma = 0.95$, $L_{max} = 10$, $T'_{Start} = 100$		
Makespan	$T'_{Start}=100$ 1056	$T'_{Start}=50$ 1035	$T'_{Start}=20$ 1117	$T'_{End}= 10$ 983	$T'_{End}= 20$ 1015	$T'_{End}= 40$ 1018
Iterationen	90	64	28	450	320	180

Tabelle 7: 3. 20x5-Problem (20 Aufträge mit jeweils 5 Operationen auf jeweils 5 verschiedenen Maschinen). Makespan der optimalen Lösung: 1165.

	MC	LS	LS-SA*)	FIFO	LIFO	KO	MinvZ	MaxvZ	KQ
Makespan	1371	1371	1220	1607	3103	1272	1511	1600	1607
Anzahl der Iterationen	100	10	160	1	1	1	1	1	1

*): T'Start=50, γ=0.9, T'_{End}=10, L_{max}=10

Anhang F: Scheduling verklemmungskritischer Systeme

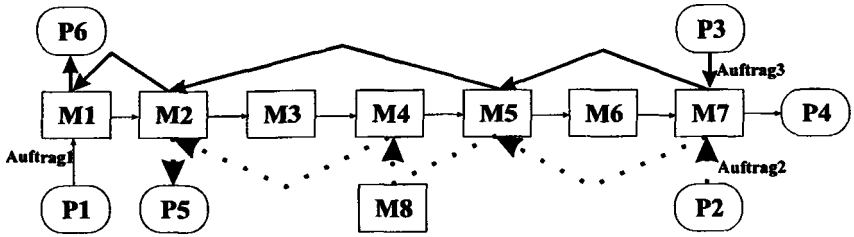


Bild 46: Beispiel 1 für ein verklemmungskritisches Job-Shop-Fertigungssystem bestehend aus 8 Maschinen mit der Kapazität 1 und 6 Puffern unbeschränkter Kapazität, in dem 3 Aufträge variabler Losgröße gefertigt werden sollen. Auf das zugehörige Petrinetz, das stark verklemmungsbehaftet ist, wurden die Monte Carlo-Simulation und heuristische Prioritätsregelverfahren erstens in Kombination mit den Rücksetz-Ausweich-Strategien und zweitens in Kombination mit den Verklemmungsverhinderungsregeln angewendet.

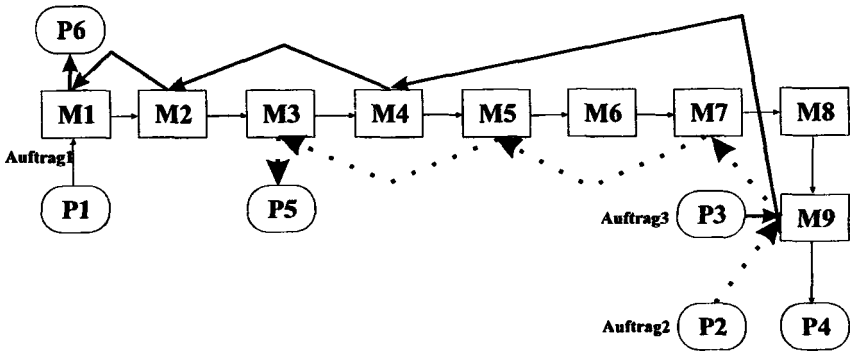


Bild 47: Beispiel 2: Job-Shop-Fertigungssystem bestehend aus 9 Maschinen mit der Kapazität 1 und 6 Puffern unbeschränkter Kapazität, in dem 3 Aufträge variabler Losgröße gefertigt werden sollen.

Tabelle 8: Simulationsergebnisse für Beispiel 1: Kombination verschiedener Scheduling-Verfahren mit der Rücksetz-Ausweich-Strategie. Rechenzeiten auf PC 486/33 MHz: Monte Carlo kleiner 10 Sekunden bei Losgröße 30, Prioritätsregeln kleiner 3 Sekunden.

Losgröße	MC	FIFO	LIFO	KO	MinvZ	MaxvZ	KQ
1	44	46	66	62	60	62	60
5	--	--	194	--	--	190	--
10	--	--	354	--	--	350	--
30	--	--	994	--	--	990	--

--: RA-Strategien scheitern, d.h. es kann kein verklemmungsfreier Schedule gefunden werden.

Tabelle 9: Simulationsergebnisse für Beispiel 1: Kombination verschiedener Scheduling-Verfahren mit den Verklemmungsverhinderungsregeln. Rechenzeiten auf PC 486/33 Mhz: Prioritätsregeln kleiner 10 Sekunden, Monte Carlo 113 Sekunden bei Losgröße 30, 3,5 Sekunden bei Losgröße 1.

Losgröße	MC	FIFO	LIFO	KO	MinvZ	MaxvZ	KQ
1	44	62	62	62	62	62	62
5	198	204	228	228	216	230	230
10	394	412	462	462	424	446	446
30	1178	1202	1346	1346	1208	1330	1346

Tabelle 10: Simulationsergebnisse für Beispiel 2: Kombination verschiedener Scheduling-Verfahren mit den RA-Strategien. Rechenzeiten auf PC 486/33 Mhz: kleiner als 4 Sekunden.

Losgröße	MC	FIFO	LIFO	KO	MinvZ	MaxvZ	KQ
1	48	44	67	67	53	67	53
5	--	199	199	--	--	199	--
10	--	--	364	--	--	364	--
30	--	--	1024	--	--	1024	--

--: RA-Strategien scheitern, d.h. es kann kein verklemmungsfreier Schedule gefunden werden.

Tabelle 11: Simulationsergebnisse für Beispiel 2: Kombination verschiedener Scheduling-Verfahren mit den Verklemmungsverhinderungsregeln. Rechenzeiten auf PC 486/33 Mhz: Prioritätsregeln kleiner als 20 Sekunden, Monte Carlo 150 Sekunden bei Losgröße 30, 10 Sekunden bei Losgröße 1.

Losgröße	MC	FIFO	LIFO	KO	MinvZ	MaxvZ	KQ
1	36	40	63	63	56	40	63
5	200	183	265	265	224	186	265
10	401	355	491	491	417	355	491
30	1202	1084	1369	1369	1234	1088	1369

Anhang G: Hierarchisches Scheduling

Bild 48 enthält ein umfangreicheres IDEF-Modell (ein Auftrag). Die Operationen sind durch Kästchen dargestellt, in denen oben die Operationennummern und unten die Nummern der benötigten Ressourcen angegeben sind.

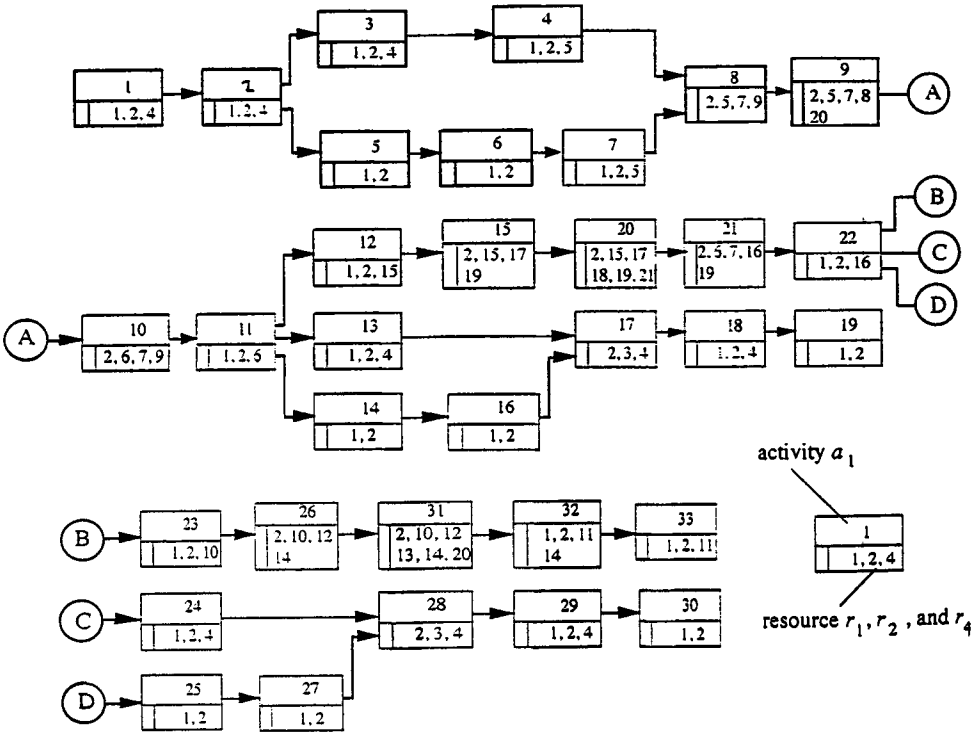


Bild 48: IDEF-Modell für ein als Beispiel erfundenes, umfangreicheres System. Die Bedeutung der Operationen spielt keine Rolle, da hier nur die Anwendung auf ein komplexeres Beispiel im Vordergrund steht.

Das IDEF-Modell wurde mit Hilfe des sequentiellen Gruppierungsalgorithmus in folgende Teilgraphen zerlegt:

TG1 = {Op1, Op2, Op3}; TG2 = {Op4, Op7, Op8, Op9, Op10, Op11}; TG3 = {Op5, Op6};

TG4 = {Op12, Op15, Op20, Op21, Op22}; TG5 = {Op13, Op17, Op18};

TG6 = {Op14, Op16}; TG7 = {Op19}; TG8 = {Op23, Op26, Op31, Op32, Op33};
TG9 = {Op24, Op28, Op29}; TG10 = {Op25, Op27}; TG11 = {Op30}.

Zu dem IDEF-Modell aus Bild 48 wurden zunächst zwei Petrinetze generiert: Ein nicht hierarchisches und ein hierarchisches. Für das nicht hierarchische Petrinetz wurden durch Anwendung der Scheduling-Verfahren (Kapitel 4) optimierte Schaltfolgen ermittelt. Für das hierarchische Netz (bestehend aus 2 Ebenen) wurde ein optimierter Schedule in zwei Phasen erzeugt:

Phase 1: Für jedes der Teil-Petrinetze wurde mit Hilfe der heuristischen Prioritätsregeln ein lokaler Schedule π_{TGi} ermittelt. Dabei wurde jedes Teil-Petrinetz separat betrachtet.

Phase 2: In einer Aggregierungsphase wurden diese lokalen Schedules π_{TGi} zu einem aggregierten Schedule zusammengesetzt. Dies erfolgte durch die Simulation des Gesamtnetzes unter Berücksichtigung der lokalen Schedules π_{TGi}. Die zu den nicht kritischen Ressourcen (Ressourcen, die nur mit Transitionen eines Teilnetzes verbunden sind) während der ersten Phase ermittelten Folgen von Transitionen π_r bleiben in der Aggregierungsphase unverändert. Die Transitionenfolgen der kritischen Ressourcen wurden aus den verschiedenen Transitionenfolgen zusammengesetzt: Wenn eine kritische Ressource mit Transitionen aus n verschiedenen Teilnetzen verknüpft ist, werden in Phase 1 n verschiedene Transitionenfolgen ermittelt, deren konsistente Verknüpfung in der Aggregierungsphase erfolgt. Bei der zur Aggregierung durchgeführten Simulation des Gesamtnetzes werden Konflikte entsprechend der Schaltregel eines determinierten Petrinetzes gelöst. Bei der Aggregierung der Teil-Schedules tritt die Situation auf, daß Konflikte bezüglich der kritischen Ressourcen zu lösen sind. Diese Konfliktlösung wurde durch Prioritätsregeln realisiert. Dazu wurden in das für die Aggregierung implementierte Programm die Regeln FIFO und LIFO integriert (als gewissermaßen „globale" Regeln, die zur Lösung der in Verbindung mit den kritischen Ressourcen auftretenden Konflikten angewendet werden).

Anhand der Tabellen 12 und 13 ist ein Vergleich der Ergebnisse möglich. Im oberen Teil ist das beste Scheduling-Ergebnis aufgeführt, welches durch das Scheduling des gesamten Netzes erzielt wurde (mit lokaler Suche und simulierter Abkühlung). Zusätzlich zum Makespan wurden auch die Ressourcennutzungsquotienten ermittelt (=Summe der Nutzungszeit einer Ressource -:- Makespan). Im unteren Teil sind die Ergebnisse des aggregierten Schedules zu sehen (erzielt mit der FIFO-Regel, angewendet sowohl zur Ermittlung der Schedules der Teilnetze als auch zur Lösung der Konflikte um die kritischen Ressourcen bei der Aggregie-

rung). Die Tabelle verdeutlicht exemplarisch, daß eine Aggregierung der lokalen Schedules ohne eine nennenswerte Reduktion der Güte der Ergebnisse möglich ist.

Diese Beobachtung wurde bei einer Reihe anderer Beispiele gemacht. Dabei wurden IDEF-Modelle mit einer Anzahl von Operationen zwischen 171 und 122 untersucht, wobei die Anzahl der Marken auf den Startplätzen (die Losgröße) zwischen 10 und 120 variierte. Die Ergebnisse zeigten, daß der Makespan der aggregierten Schedules bei vielen Petrinetzen nicht wesentlich schlechter war als derjenige, der durch Scheduling des Gesamtnetzes ermittelt wurde. Dies gilt aber keineswegs generell, sondern hängt vom konkreten Petrinetz ab (d.h. es ist möglich, daß der Makespan bei hierarchischem Scheduling erheblich schlechter ist als bei nicht hierarchischem Scheduling). Die Rechenzeiten lagen bis zu 90% niedriger (der Rechenzeitvergleich bezieht sich auf die Anwendung der heuristischen Prioritätsregeln beim nicht hierarchischen Scheduling).

Tabelle 12: Scheduling-Ergebnisse bei Anwendung von lokaler Suche und simulierter Abkühlung auf das zum IDEF-Graphen aus Bild 48 korrespondierende Gesamt-Petrinetz.

Makespan	Ressourcennutzung (%)											
	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12
	78,5	81,0	7,1	31,2	15,9	5,9	14,8	2,4	4,1	13,6	4,1	11,2
1685	r13	r14	r15	r16	r17	r18	r19	r20	r21	-	-	-
	2,4	13,0	13,6	4,2	11,2	2,4	13,0	2,4	4,7	-	-	-

Tabelle 13: Scheduling-Ergebnisse bei hierarchischem Scheduling in zwei Phasen:
Phase 1: Ermittlung von Teil-Schedules für die Teil-Petrinetze mit dem FIFO-Scheduling-Verfahren.
Phase 2: Aggregierung der Teil-Schedules zu einem Gesamtschedule (Lösung von Konflikten um kritische Ressourcen durch FIFO-Regel).

Makespan	Ressourcennutzung (%)											
	r1	r2	r3	r4	r5	r6	r7	r8	r9	r10	r11	r12
	67,1	87,2	7,1	31,2	16,0	5,9	14,8	2,4	4,1	13,6	4,1	11,3
1691	r13	r14	r15	r16	r17	r18	r19	r20	r21	-	-	-
	2,4	13,1	13,7	4,2	11,3	2,4	13,1	2,4	4,7	-	-	-

Anhang H: Struktur der Software-Implementierung

Die folgenden Bilder dokumentieren die Struktur der prototypischen Implementierung des Petrinetzgenerierungs- und Scheduling-Systems.

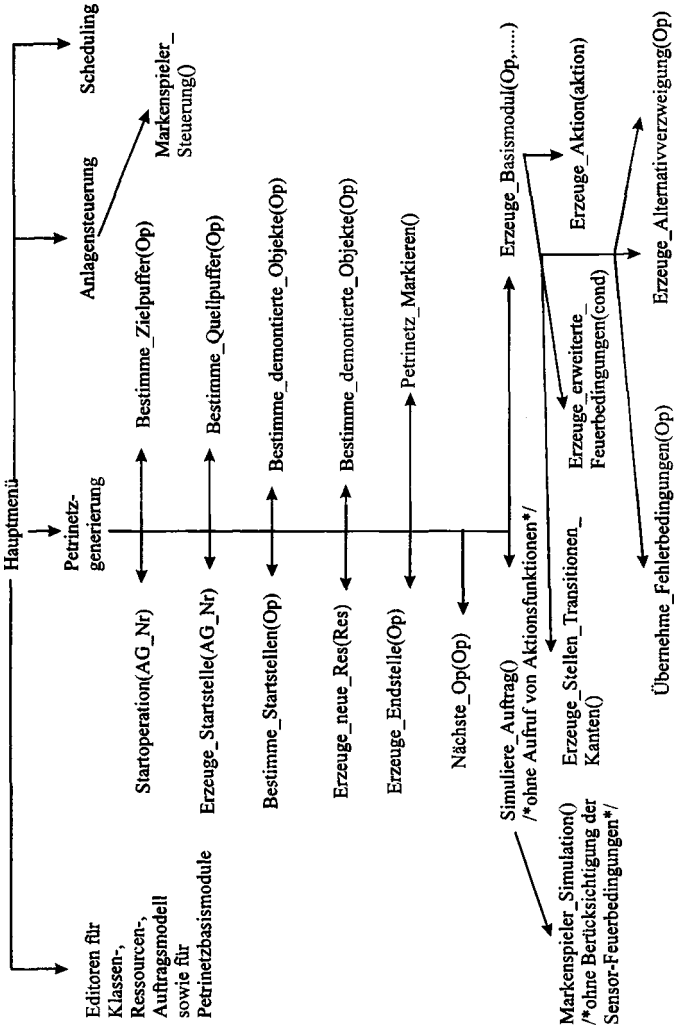


Bild 49: Struktur des Petrinetzgenerierungssystems. Die Struktur des Scheduling-Systems ist im folgenden Bild gezeigt. Die Pfeile symbolisieren die Aufrufhierarchie.

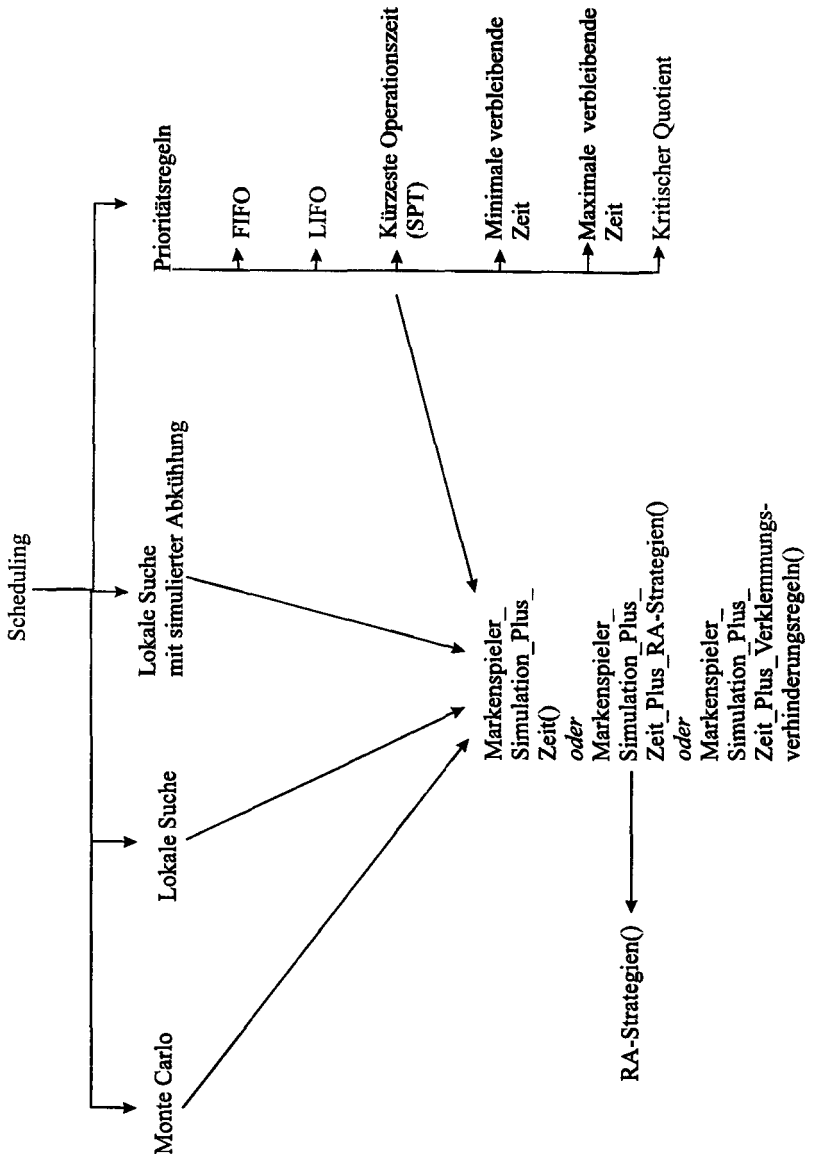


Bild 50: Struktur des Scheduling-Systems. Bei den verschiedenen Scheduling-Verfahren kann jeweils auf verschiedene Markenspieler-Versionen zugegriffen werden, je nachdem, ob Verklemmungen gar nicht beachtet, mit den RA-Strategien aufgelöst (vgl. Abschnitte 4.6.1 und 4.6.2) oder mit den Verhinderungsregeln aus Abschnitt 4.6.3 vermieden werden sollen.