Projektgruppe:
Process Landscaping

Seminar:

Petri-Netze
und
Petri-Netz Werkzeuge

Veranstalter:
Prof. Dr. V. Gruhn
U. Wellen

Lehrstuhl Software – Technologie
SS 2000

Referent:
Martin Otten

Version: 20.03.2000

1. Einleitung

Das erste Kapitel wird eine grundlegende Einführung in die Idee der Petri-Netze und des Modellbegriffs gegeben, ohne weiter auf genaue Definitionen einzugehen. Kapitel 2. wird sich noch nicht mit richtigen Petri-Netzen befassen, sondern mit der graphentheoretischen Grundlagen. Von Kapitel 3. bis 5. werden dann immer komplexere Petri-Netze eingeführt. Über rechnergestützte Werkzeuge für die Modellierung von Petri-Netzen wird in Kapitel 6. gesprochen. Der Zusammenhang zwischen Petri-Netzen und unserer Projektgruppe wird in Kapitel 7. hergestellt und Kapitel 8. bildete dann ein zusammenfassendes Fazit.

1.1 Was sind Petri-Netze ?

Die Geschichte der Petri-Netze begann im Jahr 1962 mit der Dissertation von Carl Adam Petri über die „Kommunikation mit Automaten“ im Institut für instrumentelle Mathematik in Bonn. Sie dienen zur Beschreibung nebenläufiger, kommunizierender Prozesse und sind ein *Systemmodell* für Vorgänge, Organisationen und Geräte, bei welchen geregelte Flüsse, insbesondere Nachrichtenflüsse, eine Rolle spielen. [Rei86]

Um Prozesse beschreiben zu können, müssen wir uns ihrer gemeinsamen Eigenschaften und Funktionen bewusst werden. Schauen wir uns Prozesse aus dem alltäglichen Leben an, wie Geschäftsvorgänge, betriebliche Organisationsstrukturen, Computer-Kommunikation, Brettspiele und Bauanleitungen. Man wird feststellen, dass sie folgende gemeinsame Merkmale haben:

- Sind enthalten viele einzelne, wohlunterscheidbare *Ereignisse* oder Aktivitäten, bei denen Objekte (Zahlen, Gegenstände, Informationen) erzeugt, benutzt, verändert oder verbraucht werden.

- *Bedingungen*, die für diese Ereignisse notwendig oder hinreichend sind, damit sie stattfinden. (Farbe einer Ampel, Grösse einer Zahl usw.)

Die Ereignisse können jeweils voneinander abhängig sein oder unabhängig nebeneinander stattfinden. Mache Ereignisse können unvorhersehbar eintreten und viele unterschiedliche Endergebnisse entstehen. Alle diese Eigenschaften können mit Petri-Netzen formal dargestellt werden. Kurz, Petri-Netze ermöglichen [Ger99]:

- die Beschreibung von Aufbau, Arbeitsweise und Eigenschaften nichtsequenzieller, verteilter Systeme
- die Darstellung kausaler Abhängigkeiten
- die Darstellung von Kontroll- und Datenfluss
- Nichtdeterminismus und Nebenläufigkeit
- der Nachweis von Systemeigenschaften und Korrektheitsbeweise
- dir Darstellung von verschiedenen Abstraktionsebenen mit einheitlicher Beschreibungssprache

Sämtliche folgende Definitionen und Begriffe bis einschließlich Kapitel 5 stammen aus [Bau96], soweit nicht anders angegeben.

1.2 Der Modellbegriff

Petri-Netze sind ein Systemmodell. Ein *Modell* kann aus zwei Sichten gesehen werden. Zum einen gibt es die Sicht der mathematischen Logik und Modelltheorie, in der ein Modell als „ein wirklicher Repräsentant einer abstrakten mathematischen Theorie“ bezeichnet wird. In diesem Zusammenhang spricht man von einem *Modell im Sinn der Logik.* Ein Beispiel: in der Mathematik existiert die Idee der Gruppenaxiome, mit Mengen, auf denen Operationen bestimmt sind. Sie enthalten gewisse ausgezeichnete Elemente (neutrales Element, inverses Element) oder Eigenschaften (zyklisch, abelsch). $(Z_{40},+)$ ist eine zyklische Gruppe mit der Addition als Operation. Sie ist ein Modell der Gruppenaxiome, sie ist eine *Interpretation* dieser Theorie. [Weg95]

Andersherum verhält es sich im Sprachgebrauch der abstrakten Spezifikation. Hier bezeichnet man ein Modell als eine formale oder mathematische Beschreibung einer außermathematischen Wirklichkeit. Man spricht von einem *Modell im Sinne der Spezifikation*. Gerade im Bereich der Datenverarbeitung hat man mit diesem Modellbegriff häufig zu arbeiten, insbesondere wenn man von endlichen Automaten als Abstraktion eines Prozessors oder von einer Turing-Maschine als Formalisierung von Algorithmen spricht. Zum Beispiel ist $(Z_{40},+)$ für einen Spezifizierer ein Modell für eine Glücksrad mit 40 Feldern, wie man es auf Jahrmärkten antrifft.

Welchem Modellbegriff entspricht denn nun das Systemmodell der Petri-Netze? Die Antwort lautet: beiden. Der theoretische Informatiker, der an neuen Varianten der Petri-Netze arbeitet, sieht Petri-Netze als Modell der Netzaxiome und Graphentheorie. Der praktische Informatiker, der reale System in Programmen nachbilden muss, sieht Petri-Netze als Spezifikationssprache, um formale Modelle zu erstellen. Der Zusammenhang wird in folgender Abbildung 1.1 visualisiert:

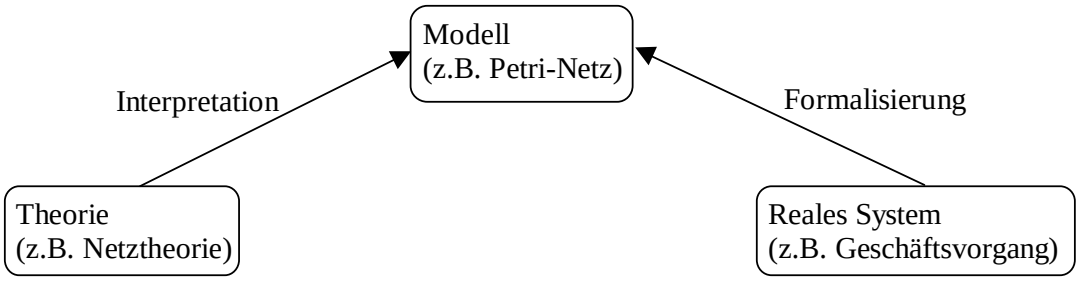

Abbildung 1.1

1.3 Was nützen formale Modelle ?

Wieso sollte man sich die Mühe machen, Systeme in formalen Modellen auszudrücken ? Immerhin schafft man es ja auch im alltäglichen Leben, alles mögliche in natürlicher Sprache zu erklären. Jedoch birgt die natürliche Sprache, die syntaktisch korrekt seien mag, viele semantische Uneindeutigkeiten und unnötige Redundanzen. So kann eine etwas komplexere Systembeschreibung auf rein umgangssprachlicher Basis von jedem Programmierer anders verstanden werden. Teilweise merken Kommunikationspartner gar nicht, dass sie „aneinander vorbeireden“. Man kann die Vorteile eines formalen Modells grob in vier Punkten zusammenfassen:

- Entwurfsaspekt: Da alle am Entwurf beteiligten Personen die gleiche(n) Spezifikationssprache(n) sprechen (sollten), ist die Kommunikation schnell, effizient und eindeutig. So werden Missverständnisse vermieden und eine möglichst genaue Beschreibung des realen Systems erreicht.

-Analyseaspekt: Umgangsprachlich formulierte Systembeschreibungen bieten kaum ernsthaften Möglichkeiten zu Analyse. Erst formale Modelle können korrekt analysiert werden. Die Analyse hat den Zweck der Überprüfung, ob das gedachte Modell das gleiche Verhalten wie das reale System aufweist. Stellen sich während der Analyse Widersprüche zum wirklichen System heraus, ist das Modell zu überarbeiten und erneut zu analysieren. So nähert sich das Modell schrittweise der Wirklichkeit an [Pet81]. Entspricht das gedachte Modell dem Realen, können auch bisher unentdeckte Eigenschaften des realen Systems erkannt werden. Dieser Sachverhalt ist hier dargestellt (Abbildung 1.2):

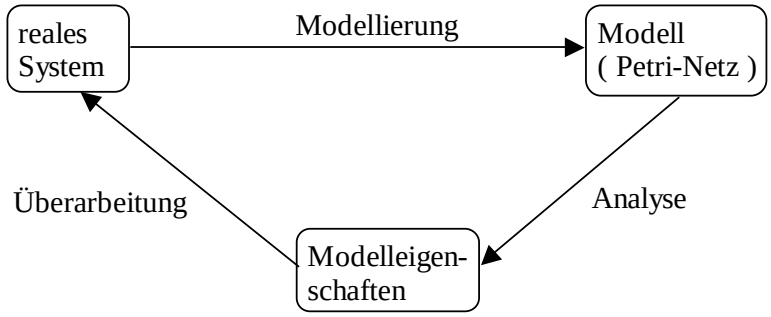

Abbildung 1.2

- Testaspekte: Auch eine besonders praxisbezogene Art der Analyse, die Simulation, ist mit formalen Modellen möglich. Somit könne reale Probebauten oder Probeläufe vermieden werden, und damit Zeit und Aufwand gespart werden. Natürlich wird man häufig vorkommende Standardsituationen simulieren oder auch kritische Grenzfälle testen. Erst wenn die Simulation erfolgreich war, kann man daran denken, erste reale Prototypen des Modells zu verwirklichen.

-Abstraktion: Viele Systeme sind so komplex, dass sie im Gesamten nicht überschaubar sind. Viele Modelle bieten Darstellungsmöglichkeiten der Systeme auf verschiedenen Abstraktionsebenen, auf denen irrelevante Aspekte weggelassen oder zum Komponenten zusammengefasst werden. Das veranschaulicht nicht nur das ganze Modell, sondern macht auch seine Analyse einfacher, da immer nur kleine Teilsysteme betrachtet werden müssen.

2. Netzgraphen

Netzgraphen sind die Grundlage für Petri-Netze aus dem Bereich der Netztheorie. Sie bestehen aus „ Rechteck und Kreisen sowie Pfeile zwischen Rechtecken und Kreisen “ [Bau96]. In dieser simplen Definition steck doch viel Wahres über Netzgraphen, insofern, dass Netzgraphen aus zwei verschiedene Kontenmengen und einer Relation zwischen diesen Mengen besteht.
Wir gehen hier noch nicht auf die dynamische Eigenschaft von Petri-Netzen ein, die erst später mit den Marken eingeführt wird. Zunächst werden wir Netzgraphen genau definieren und ihre grafische Darstellung zeigen. Anschließend werden Begriffe für verschiedene Netzänderungen beim Modellieren eingeführt.

2.1 Definition

Ein Netz (oder Netzgraph) ist ein gerichteter, bipartiter Graph, formal ausgedrückt ein Tripel N = (S,T,F) mit zwei Eigenschaften:

- $S \cap T = \varnothing$	2 Unterschiedliche Knotentypen
- $F \subseteq (S \times T) \cup (T \times S)$	keine Relation zwischen Knoten gleichen Typs

Die Elemente der Menge S heißen *Stellen*, die Elemente der Menge T heißen *Transitionen*. Elemente der *Flussrelation* F sind Tupel und bilden die Kanten in dem Netz. Grafisch werden die Stellen als Kreise, die Transitionen als Rechtecke dargestellt. Die Flussrelationen werden als Pfeile zwischen die Knoten gezeichnet. Ein Beispiel in Abbildung 2.1:

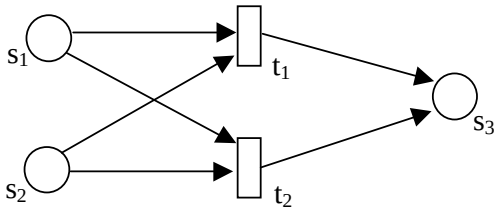

Abbildung 2.1

Abbildung 2.1 zeigt ein Netz N = (S,T,F) mit :

$S = \{s_1,s_2,s_3\}$
$T = \{t_1,t_2\}$
$F = \{ (s_1,t_1), (s_1,t_2), (s_2,t_1), (s_2,t_2), (t_1,s_3), (t_2,s_3) \}$

Gerade in älteren Büchern aus dem amerikanischen Raum werden Transitionen nicht als Rechtecke gezeichnet, sondern als Striche. Um Verwechslungen zu vermeiden wurde jedoch von der ISO ein Dokument [Iso97] verfasst, das die genaue Definition von Petri-Netzen und ihre Darstellung festlegt. Darin wird das Rechteck als Symbol für die Transition vorgegeben.

Um über wichtige Knotenmengen zu sprechen, müssen wir noch zwei Begriffe einführen. Beide Begriffe beziehen sich auf die Nachbarknoten eines Knotens. Die Menge aller Eingangs- oder Inputknoten eines Knotens x bezeichnen wir als *Vorbereich*:

$$\bullet x := \{ y \mid (y , x) \in F \}$$

Ebenso definieren wir alle Ausgangs- oder Outputknoten eines Knoten x als *Nachbereich*:

$$x\bullet := \{ y \mid (x , y) \in F \}$$

Falls $|x\bullet| > 1$, nennt man x *vorwärtsverzweigt,* entsprechend gilt bei $|\bullet x| > 1$, x ist *rückwärtsverzweigt*.

Ein Netz ist *schlicht,* wenn keine zwei Knoten denselben Vor- und denselben Nachbereich haben, also wenn:

$$\forall x, y : \bullet x = \bullet y \wedge x\bullet = y\bullet \Rightarrow x = y$$

Das Netz aus Abbildung 2.1 ist nicht schlicht, da sowohl $\bullet t_1 = \bullet t_2 = \{ s_1, s_2 \}$, als auch $t_1\bullet = t_2\bullet = \{ s_3 \}$ gilt. Falls ein Netz nicht schlicht ist, deutet dies auf eine unnötige Redundanz hin. So könnte z.B. in unserem Beispiel Netz aus Abbildung 2.1 die Transitionen t_1 und t_2 zusammengelegt werden, ohne dass die Grundstruktur des Netzes sich verändern würde.

Zwei Knoten s und t bilden eine *Schlinge,* falls $(s, t) \in F$ und $(t, s) \in F$. Anschaulich dargestellt sehen wir eine Schlinge in Abbildung 2.2 :
Ù

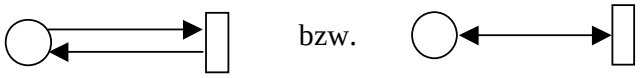

Abbildung 2.2

Die Wahl der Stellen- und Transitionsbezeichnungen ist nicht bestimmt und frei wählbar. Auch gibt es keine eindeutigen Vorschriften, wie ein Netzgraph zu zeichnen ist. So kann es kommen, dass zwei Netze, die komplett unterschiedlich aussehen, von der Struktur her gleich sind. Äquivalenz zwischen zwei Netzen N_1 und N_2 besteht genau dann, wenn eine bijektive, kanten-, stellen- und transitionserhaltende Abbildung (*Isomorphismus*) zwischen den Netzen existiert.
Die beiden Netze heißen dann *isomorph*. Der Nachweis über die Isomorphie zweier Netz ist jedoch keine schnell erledigte Aufgabe, da das Graphenisomorphieproblem wahrscheinlich in NP liegt. [Weg90]

2.2 Netztransformationen

Beim Modellieren von Netzen kommen immer wieder die gleichen Arbeitsschritte vor. Es werden Elemente hinzugefügt, entfernt, zusammengefasst oder weiter aufgesplittet. Die wichtigsten Vorgänge werden im folgenden begrifflich benannt und erklärt:

2.2.1 Vergröberung / Verfeinerung

Bei der *Vergröberung* handelt es sich um lokale Abstraktion. Hier werden mehrere Knoten zu einer Stelle oder Transition zusammengefasst, die diese Knotenmenge ersetzt. Wichtig ist bei

der Konstruktion, dass das neu entstandene Netz immer noch bipartit ist. So darf eine Menge von Knoten nur zusammengefasst werden, falls alle Knoten am Rand vom gleichen Typ sind. Knoten am Rand sind solche, die auch Relationen zu anderen Knoten haben, die nicht in dieser Menge liegen. Genau von diesem Typ muss dann der Knoten sein, der diese Menge ersetzt. Man spricht dann von einem *stellen-* oder *transitionsberandetem* Teilnetz.

Bei der *Verfeinerung* handelt es sich um das Gegenstück zur Vergröberung. Sie wird benutzt um die innere Struktur einer Transition oder einer Stelle feiner zu konkretisieren. Auch hier ist zu beachten, dass die bipartite Eigenschaft der Petri-Netze erhalten bleiben muss.

Ob und wann man Verfeinerung oder Vergröberung einsetzen muss, hängt jeweils vom gewünschten Abstraktionsgrad ab. Oft ist es sinnvoll, das System am Anfang möglichst grob zu modellieren. Dann kann es Schritt für Schritt solange verfeinert werden, bis das Gesamtmodell alle Aspekte des Systems beinhaltet. Das Grundverhalten eines Systems sollte sich nach Vergröberung oder Verfeinerung nicht ändern. Ein Beispiel für Vergröberung und Verfeinerung ist in Abbildung 2.3 gezeigt:

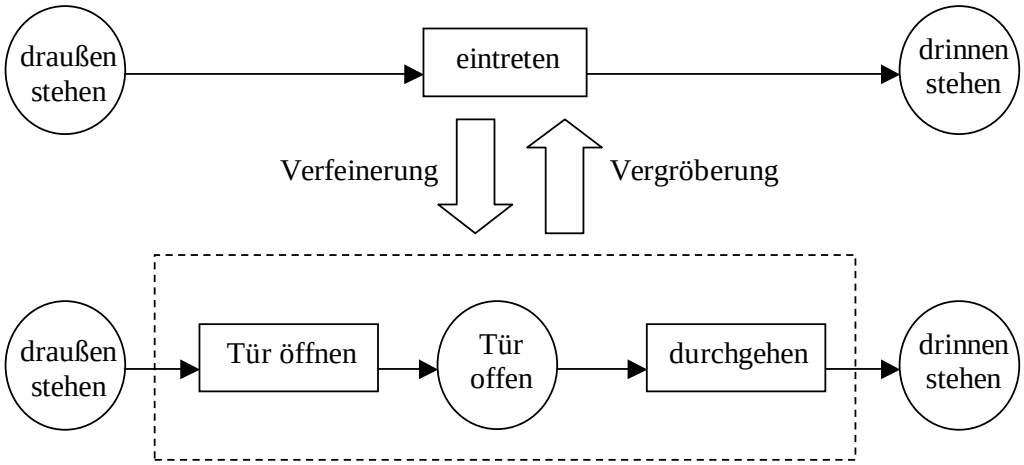

Abbildung 2.3

2.2.2 Einbettung / Restriktion

Wenn man sein Modell um weitere Aspekte und Systemteile erweitern will, wird man dem Netz neue Kanten und Knoten hinzufügen. Dieser Vorgang wird als *Einbettung* bezeichnet.
Im Gegensatz zur Verfeinerung werden keine Knoten durch andere ersetzt, sondern nur neue Knoten und Kanten hinzugefügt. Die *Restriktion* ist das Gegenstück zur Einbettung. Durch sie werden Knoten und Kanten aus einem Netz entfernt. Einbettung und Restriktion können wesentliche Auswirkungen auf das Verhalten des Modells haben.

2.2.3 Faltung / Entfaltung

Wenn ein Netz aus gleichstrukturierten Teilnetzen besteht, kann man das Netz in sofern vereinfachen, indem man diese gleichen Teilnetze einfach „aufeinander“ legt. Dabei müssen sämtliche Strukturen erhalten bleiben, insofern dürfen nur Knoten gleichen Typs übereinandergelegt werden, auch müssen sämtliche Relationsbeziehungen erhalten bleiben.

Die Teilnetze müssen isomorph sein, jedoch nicht am Netzrand Knoten gleichen Typs aufweisen (wie bei der Vergröberung). Diese global Abstraktion nennt man *Faltung*.
Entsprechend wird das Verdoppeln von Teilnetzen als *Entfaltung* bezeichnet wobei, zwei gleichstrukturiert Teilnetz entstehen. Abbildung 2.4 zeigt ein Beispiel für Faltung und Entfaltung. Die beiden Transitionen t_1 und t_2 werden zu einer Transition t_{12} zusammengelegt, entsprechen s_2 und s_3 zu s_{23}.

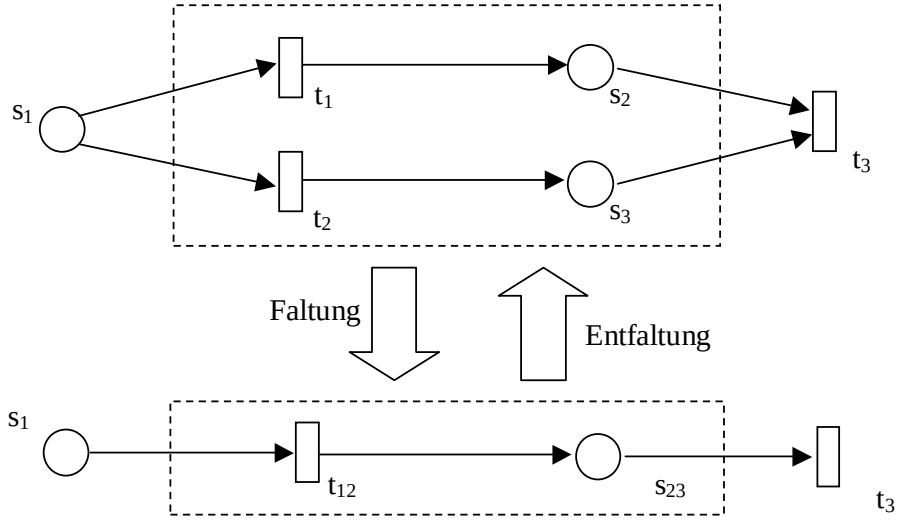

Abbildung 2.4

3. Systeme mit anonymen Marken

Bis jetzt haben wir nur statische Netze betrachtet, die noch keine ausreichende Möglichkeit der Prozessmodellierung bieten. Prozesse enthalten immer Dynamik und Veränderung. Um diese Möglichkeit der Veränderung in unsere Netzgraph einzubauen, müssen wir sie noch ein wenig erweitern . Die Hauptidee dabei ist, dass die Stellen verschiedene Zustände annehmen können, und sich diese Zustande ändern können, in Abhängigkeit der Zustände benachbarter Stellen und Transitionen. Grafisch werden diese Zustände als Marken in den Stellen versinnbildlicht.
In diesem Kapitel werden Stellen-Transitions-Systeme formal definiert und einige wichtige Eigenschaften und deren Begrifflichkeiten erläutert. Zum Schluss wird noch eine spezielle Version dieser Stellen-Transitions-Systeme vorgestellt, die Bedingungs-Ereignis-Systeme.

3.1 Definition

Ein 6-Tupel $Y = (S, T, F, K, W, M_0)$ heißt *Stellen-Transitions-System* bzw. S/T-System falls:

- (S, T, F) ein Netz ist,
- $K : S \rightarrow N_0 \cup \{ \infty \}$ Kapazitäten der Stellen, evtl. unbeschränkt
- $W : F \rightarrow N$ Kantengewichte
- $M_0 : S \rightarrow N_0 \cup \{ \infty \}$ Startmarkierung, wobei gilt $\forall s \in S : M_0(s) \leq K(s)$

Bei der grafischen Darstellung werden die Zustände der Stellen dadurch ausgedrückt, indem man die Anzahl der Marken einer Stelle M(s) als Punkte in den Stellenkreis zeichnet. Die Kantengewichte werden an die Kanten geschrieben. Sind Kanten nicht beschriftet, haben sie ein Gewicht von eins. Die Kapazitäten wird an die Stellen geschrieben, jedoch bedeutet hier keine Beschriftung eine Kapazität von unendlich. Die Notation wird an einem Beispiel verdeutlicht:

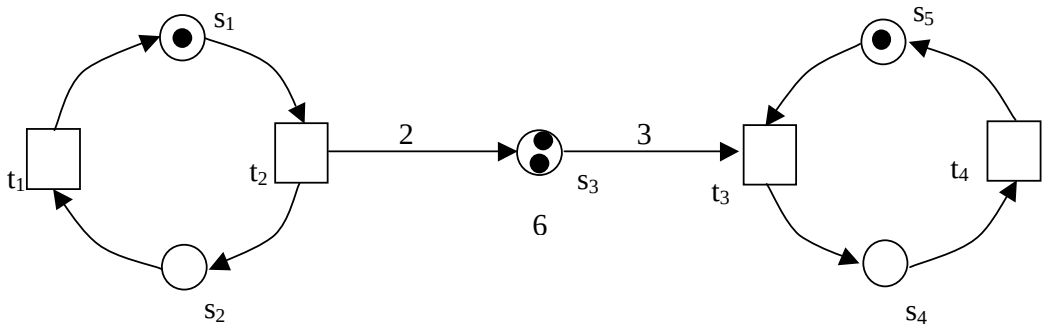

Abbildung 3.1

Das in Abbildung 3.1 gezeigte Stellen-Transitions-System lässt sich formal wie folgt beschreiben:

$S = \{ s_1, s_2, s_3, s_4, s_5 \}$
$T = \{ t_1, t_2, t_3, t_4 \}$
$F = \{ (t_1,s_1), (s_1,t_2), (t_2,s_2),(s_2,t_1), (t_2,s_3), (s_3,t_3), (t_3,s_4), (s_4,t_4), (t_4,s_5), (s_5,t_3) \}$
$K = \{ (s_1, \infty), (s_2, \infty), (s_3,6),(s_4, \infty),(s_5, \infty) \}$
$W = \{ ((t_1,s_1), 1), ((s_1,t_2), 1), ((t_2,s_2), 1), ((s_2,t_1),1), ((t_2,s_3)2) , ((s_3,t_3),3), ((t_3,s_4), 1),$
$((s_4,t_4), 1), ((t_4,s_5),1), ((s_5,t_3),1) \}$
$M_0 = \{ (s1,1), (s2,0), (s3,2), (s4,0), (s5,1) \}$

Dies Beispiel kann man gut als Erzeuger-Verbraucher-System mit begrenztem Zwischenspeicher interpretieren. In der Transition t_2 findet der eigentliche Produktionsschritt statt, wobei zwei Einheiten produziert werden. Diese wandern in den Zwischenspeicher s_3, der eine Kapazität von sechs Einheiten hat. Der Verbraucher benötigt für jeden Schritt in t_3 je 3 Einheiten aus dem Speicher. Die Startmarkierung gibt an, dass es einen Erzeuger und einen Verbraucher gibt, und dass schon zwei Einheiten im Speicher liegen.

3.2 Dynamik

Um Bewegung in das Ganze zu bringen, werden nun Schaltegeln definiert, so dass sich die Zustände ändern können, bzw. die Marken im Netz wandern (eigentlich sieht es nur so aus als würden die Marken wandern, vielmehr werden n Marken an einer Stelle entnommen und m Marken einer anderen Stelle hinzugefügt, nur falls n = m ist, entsteht der Eindruck des Wanderns).

3.2.1 Markierungen und Schaltregeln

Die *Markierung* des S/T-Systems ist eine Abbildung $M : S \rightarrow N_0$ mit $\forall s \in S : M(s) \leq K(s)$. In einer Markierung M können Transitionen aktiv oder inaktiv sein. Genauer, eine Transition $t \in T$ heißt *aktiviert unter M*, geschrieben M[t>, falls:

- $\forall s \in \bullet t : M(s) \geq W(s, t)$
  in allen Stellen im Vorbereich der Transition sind genügend Marken vorhanden

- $\forall s \in t\bullet : M(s) \leq K(s) - W(s, t)$
  in allen Stellen im Nachbereich der Transition ist ausreichend Platz für Marken

Ist eine Transition t aktiviert können aus dem Vorbereich die entsprechenden Marken entnommen werden und in den Nachbereich eingefügt werden. So einsteht aus einer Markierung M eine neue *Folgemarkierung* M'. Wir sagen dann, *t schaltet von M nach M'* und schreiben M[t>M' (oder M' = Mt). Diese Aktivierungsbedingungen und die Definition der Folgemarkierung wird als *Schaltregel* bezeichnet.

In dem Beispiel aus Abbildung 3.1 ist die Transition t_3 genau dann aktiviert, wenn im Speicher s_3 mindestens drei Marken sind, und die Stelle s_5 markiert ist. Da die Stelle s_4 im Nachbereich unendliche Kapazität hat, spielt ihr Markierung keine Rolle. Anders bei der Transition t_2, hier ist die Stelle s_3 im Nachbereich in ihrer Kapazität beschränkt, und t_2 kann nur schalten , falls vier oder weniger Marken in s_3 liegen. Grundlegende Konstellationen von aktivierten und nicht aktivierten Transitionen werden später in Kapitel 3.3 besprochen.

3.2.2 Schaltfolgen und Erreichbarkeitsgraphen

Je nachdem welche und wie viele Transitionen in einer Markierung aktiviert sind, kann es keine, eine oder mehrere mögliche Folgemarkierungen geben, die ihrerseits wiederum Folgemarkierungen haben können. Eine solche Kette von möglichen Folgemarkierungen nennen wir *Schaltfolge*, deren erste Markierung immer die Startmarkierung M_0 ist. Alle Markierungen einer Schaltfolge sind *erreichbare* Markierungen dieses Systems. Folglich ist die *Erreichbarkeitsmenge* eines Systems die Menge aller erreichbarer Markierungen aller möglichen Schaltfolgen. Die Erreichbarkeitsmenge kann, sobald unendliche Kapazitäten existieren, unendlich groß werden, muss aber nicht. Siehe Abbildung 3.2:

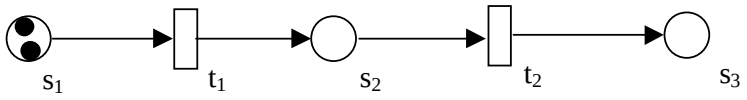

Abbildung 3.2

Wenn man bei der Startmarkierung $M_0 = \{ (s_1, 2), (s_2, 0), (s_3, 0) \}$ anfängt und alle weiteren möglichen Markierungen berechnet, ergibt sich eine Erreichbarkeitsmenge mit folgenden Markierungen:

$M_0 = \{ (s_1, 2), (s_2, 0), (s_3, 0) \}$ $\quad$ $M_1 = \{ (s_1, 1), (s_2, 1), (s_3, 0) \}$

$M_2 = \{ (s_1, 0), (s_2, 2), (s_3, 0) \}$ $\quad$ $M_3 = \{ (s_1, 1), (s_2, 0), (s_3, 1) \}$

$M_4 = \{ (s_1, 0), (s_2, 1), (s_3, 1) \}$ $\quad$ $M_5 = \{ (s_1, 0), (s_2, 0), (s_3, 2) \}$

Um den Zusammenhang zwischen den Markierungen zu verdeutlichen. benutzt man einen gerichteten *Erreichbarkeitsgraphen*, der als Knoten die Markierungen trägt und die schaltenden Transitionen als Kanten zwischen den Markierungen. Zu dem S/T-System aus Abbildung 3.2 würde folgender Erreichbarkeitsgraph gehören (Abbildung 3.3):

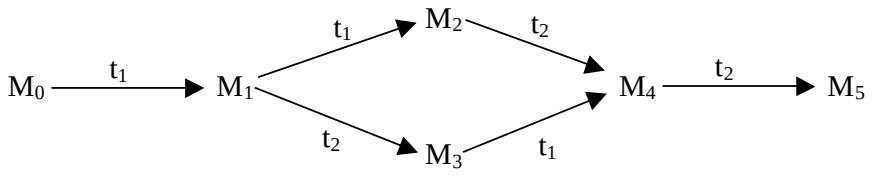

Abbildung 3.3

Ein Pfad im Erreichbarkeitsgraphen von M_0 aus entspricht einer Schaltfolge. Auf das Erstellen von Erreichbarkeitsmengen und deren Erreichbarkeitsgraphen wird später noch mal im Kapitel 4. eingegangen.

3.2.3 Inzidenzmatrix

Für spätere Analysewecke ist es wichtig von Stellen-Transitions-Systemen eine Inzidenzmatrix zu erstellen. Inzidenzmatrixen haben für jede Transition eine Spalte und für jede Stelle eine Zeile. Die Felder der Matrix werden wie folgt bestimmt:

$$C_{ij} = W(t_j, s_i) - W(s_i, t_j)$$

Sollte ein Kantengewicht nicht definiert sein, da die entsprechende Kante in dem jeweiligen S/T-System nicht existiert, wird ein Wert von 0 für diese Kantengewichte gewählt. Insgesamt drückt der Wert des Feldes C_{ij} die Anzahl der Marken aus, die beim Schalten der Transition t_j der Stelle s_i hinzugefügt (bei positiven Werten) oder abgezogen wird (bei negativen Werten).
Als Beispiel wird die Inzidenzmatrix aus dem Erzeuger-Verbraucher Beispiel aus Abbildung 3.1 berechnet:

$$C = \begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 2 & -3 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{pmatrix}$$

Die Inzidenzmatrix wird uns später bei der Berechnung der Invarianten in Kapitel 4.5 wieder begegnen.

3.3 Grundsituationen

Gewisse lokale Anordnungen von Transitionen und Stellen kommen immer wieder vor und haben auch bestimmt charakteristische, dynamische Eigenschaften. Diese Konstellationen werden wir als Grundsituationen im folgenden weiter besprechen.

3.3.1 Kausalität

Für das Eintreten von Ereignissen wird zwischen *notwendigen* und *hinreichenden* Voraussetzungen unterschieden.

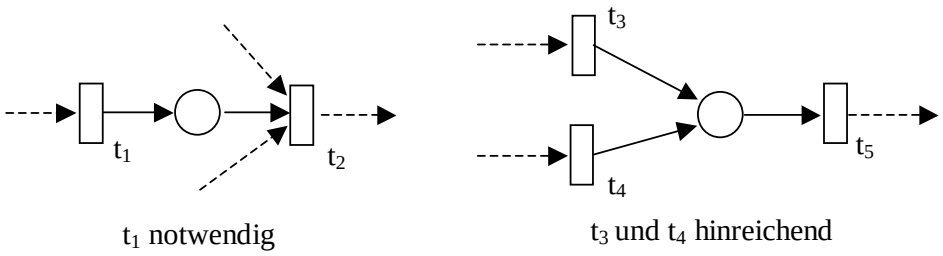

Abbildung 3.4

Im linken Beispiel von Abbildung 3.4 ist das Schalten der Transition t_1 notwendig für das Schalten der Transition t_2. Im Gegensatz dazu ist das Schalten im rechten Beispiel der Transitionen t_3 oder t_4 hinreichend für das Schalten von t_5.

3.3.2 Nebenläufigkeit

Wenn in einer Markierung mehrere Transitionen aktiviert sind, ist die Reihenfolge, wie sie schalten nicht vorgegeben. Beeinflussen, also deaktivieren oder aktivieren sich diese Transitionen sich durch ihr Schalten nicht, so ist das Ergebnis unabhängig der Schaltfolge immer gleich. Es besteht also weder eine notwendige, noch eine hinreichende Kausalität zwischen ihnen. Da die Schaltreihenfolge der Transitionen nicht von Bedeutung für das Endergebnis ist, kann man sie auch „gleichzeitig“ schalten. Sie sind dann zueinander *nebenläufig*.

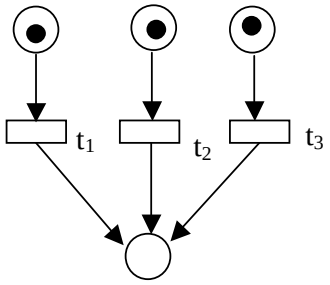

Abbildung 3.5

In Abbildung 3.5 sehen wir eine Beispiel für die Nebenläufigkeit. Die Reihenfolge, wie man t_1,t_2 oder t_3 hintereinander schaltet ist egal, sie führen alle zum gleichen Ergebnis. Diese Transitionen sind nebenläufig und könnten gleichzeitig geschaltet werden.

3.3.3 Konflikt

Es kann sein, dass zwei oder mehrere Transitionen aktiviert sind, sich ihre Vorbereiche aber überschneiden oder gar gleich sind. Sind in diesem Vorbereich aber nicht genug Marken vorhanden, um alle aktivierten Transitionen zu schalten, stehen diese im *Konflikt* zueinander.
Der Konflikt ist eine nicht-nebenläufige Aktiviertheit von Transitionen, es ist dann nicht festgelegt welche dieser Transitionen geschaltet wird. Hier liegt ein lokaler *Nichtdeterminismus* vor. Um das unvorhersehbare Ausgehen solcher Konflikte zu lösen, kann man *konfliktlösende Elemente* dem Netz hinzufügen, indem man die Vorbereiche der im Konflikt stehenden Transitionen so erweitert, dass nicht mehr alle beteiligten Transitionen aktiviert werden. Häufig wird der Konflikt auch *Alternative* genannt, um die Situation positiver zu bewerten.

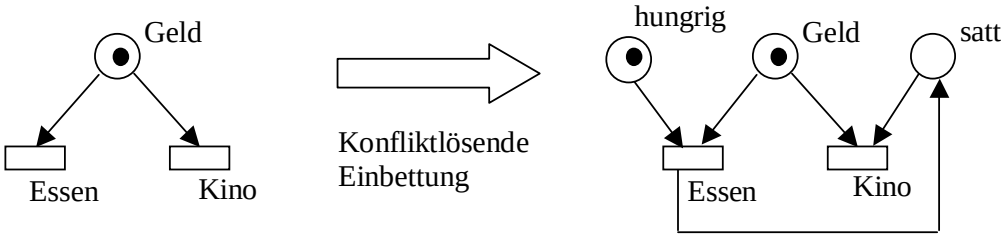

Abbildung 3.6

In Abbildung 3.6 sieht man links einen typischen Konflikt. Man hat Geld und kann sich jetzt entweder dafür entscheiden ins Kino oder Essen zu gehen. Diesen Konflikt kann man im Modell durch zusätzliche Bedingungen auflösen, so wie hier, indem man nur Essen geht, wenn man auch hungrig ist, ansonsten ins Kino.

3.3.4 Kontakt

Die Kontaktsituation ähnelt dem Konflikt: obwohl mehrere Transitionen aktiviert sind, können nicht alle schalten. Lag beim Konflikt die Ursache dafür im Vorbereich, ist es beim *Kontakt* der Nachbereich, der verhindert, dass andere Transitionen schalten können, sobald eine Transition geschaltet hat. Kontakt kommt immer dann vor, wenn im Nachbereich mehrer Transitionen gemeinsame, kapazitätsbeschränkte Stellen liegen. Durch diese Kapazitätsbeschränkung, kann es sein, dass nach dem Schalten einer Transition, für andere Transitionen nicht mehr genügend Aufnahmekapazität für Marken bereit steht. Da die Schaltreihenfolge von Transitionen nicht festgelegt ist, kommt es auch hier zu einem unvorhersehbarem Ergebnis. Systeme, in denen solche Situationen nicht auftauchen, nennt man *kontaktfrei*. Beim praktischen Entwurf hat sich die Eigenschaft der Kontaktfreiheit von Netze als sehr wichtig herausgestellt. Abbildung 3.7 zeigt eine typische Kontaktsituation zwischen zwei Transitionen:

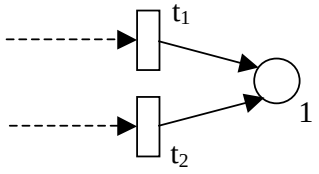

Abbildung 3.7

Hier stehen t_1 und t_2 im Kontakt, wenn sie beide aktiviert sind. Sobald eine der beiden schaltet, ist die maximale Kapazität in der Stelle erreicht.

3.3.5 Synchronisation

Synchronisation bedeutet „Ereignisströme“ aufeinander abzugleichen oder zeitgleich zu starten. Auf diese Weise lassen sich Konstrukte modellieren, die nebenläufige Prozesse starten oder vereinigen. Das Beispiel in Abbildung 3.8 kommt aus dem Sport:

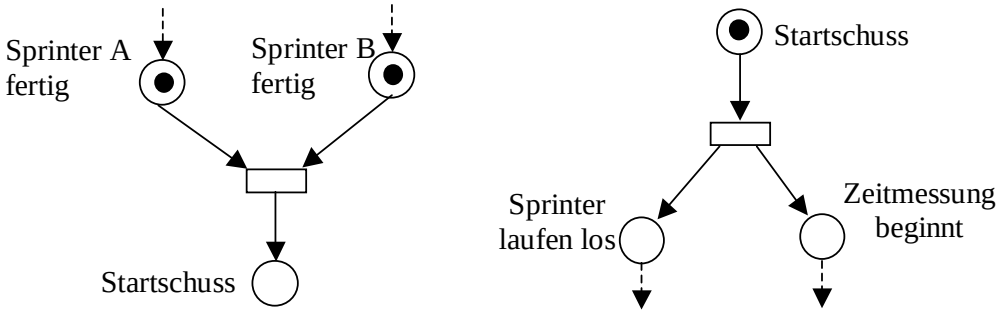

Abbildung 3.8

Egal wie lange zwei Sprinter sich auf ein Rennen vorbereiten, das Rennen kann erst beginnen wenn beide fertig sind. Diese Synchronisation wird auch „Join“ genannt. Das eigentliche Rennen wird durch den Startschuss eingeleitet, was zeitgleich 2 Prozesse in Gang setzt: das Rennen zwischen den Sprintern und die Zeitmessung. Diese Situation nennt man „Fork“.

3.4 Bedingungs-Ereignis-Systeme

In den Beispielen war es häufig so, dass jeweils nur eine Marke in den Stellen vorkam. Das hat zum einen damit zu tun, dass Beispiele einfach gehalten werden, zum anderen damit, das häufig nur interessiert, ob eine Stelle markiert ist oder nicht. Die Anzahl der Marke spielt dabei keine Rolle. In vielen Systemen müssen häufig nur zwei Zustände modelliert werden (z.B. Strom an/Strom aus). Bedingungen haben auch nur zwei Zustände, entweder sie gelten oder sie gelten nicht. Bedingungen könne Aussagen über beliebig komplexe Systeme sein. Sie können aufgrund von Ereignissen wahr werden oder ihre Gültigkeit verlieren. Ereignisse benötigen meist gewisse Vorraussetzungen, also erfüllte Vorbedingungen, und führen meist zu vorher nicht erfüllten Nachbedingungen. Diese Analogie führt zu einem neuen, eingeschränktem Stellen-Transitions-System, dem *Bedingungs-Ereignis-System*. Wir werde es formal aus den Stellen-Transitions-Systemen ableiten und noch eine wichtige Eigenschaft erläutern, die Widerspruchsfreiheit.

3.4.1 Definition

Ein Bedingungs-Ereignis-System (B/E-System) ist ein S/T-System $Y = (S, T, F, K, W, M_0)$ mit schlingenfreiem, schlichten Netz $N = (S, T, F)$, $K=1$ und $W = 1$. Eine Stelle kann also nur eine Marke aufnehmen und ist dann belegt. Stellen werden Bedingungen genannt,

Kantengewichte sind alle 1 (wobei das nicht zwingend notwendig ist) und die Transitionen entsprechen den Ereignissen. So reduziert sich das ganze auf ein 4-Tupel (B, E, F, M_0). In diesem Kontext wird eine Markierung auch als Fall bezeichnet.

Ein B/E-System ist wesentlich leichter zu verstehen als ein S/T-System, auch ist die Analyse einfacher, da wir nur endlich vielen Fällen zu betrachten müssen.

Historisch gesehen wurden die B/E-Systeme vor den S/T-Systemen entwickelt, werden hier jedoch logisch als Spezialfall der S/T-Systeme behandelt. Ein extra Beispiel sei hier nicht aufgeführt, da schon viele der vorherigen Beispiele B/E-Systeme waren (z.B. Abbildung 3.6).

3.4.1 Widerspruchsfreiheit

Da die Stellen nur zwei Zustände haben können, ist es einfach, für eine Bedingung in einem B/E-System ein *Komplement* (Gegenteil) zu definieren. Es gilt folgender Satz:

> eine Bedingung $b' \in B$ eines B/E-Systems nennen wir das Komplement ($\neg b$) der Bedingung $b \in B$, falls $\bullet b = b'\bullet$ und $\bullet b' = b\bullet$.

Der Vorbereich einer Bedingung ist jeweils der Nachbereich ihres Komplements und umgekehrt. Dieser Satz kann auch dafür verwendet werden um nicht vorhandene Komplemente in vorhandene B/E-Systeme einzufügen. Erfolgt diese Einbettung für alle Bedingungen, die noch kein Komplement besitzen, spricht man von *Vervollständigung*. Ein System heißt *widerspruchsfrei*, falls in allem möglichen Fällen für alle möglichen Paare b und $\neg b$ gilt, dass genau einer der beiden markiert ist. Widerspruchsfreiheit ist häufig ein Indiz dafür, dass das System richtig modelliert wurde.

Auch hier können wir auch ein altes Beispiel eingehen, und zwar das zur konfliktlösenden Einbettung aus Abbildung 3.6. Hier gibt es 2 Bedingungen „hungrig“ und „satt“, deren Namen intuitiv ihren gegensätzlichen Charakter versinnbildlichen. Würde man eine Pfeil von „Kino“ nach „hungrig“ ziehen, wären es sogar Komplemente im Sinne der Definition (was auch gar nicht so unabwegig ist, da ein langer Kinobesuch hungrig macht). Die Bedingung „Geld“ hat kein Komplement, würde man jedoch eine Bedingung „Kein Geld“ hinzufügen, wäre das System vollständig. Das vervollständigte, widerspruchsfreie Beispiel ist in Abbildung 3.9 zu sehen:

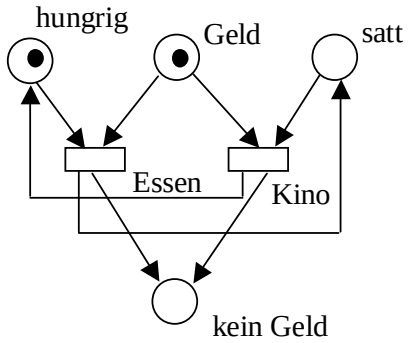

Abbildung 3.9

4. Analyse von Systemen

Bei der Analyse von Petri-Netzen werden verschiedene Verfahren angewandt, um wichtige Eigenschaften von Petri-Netzen zu zeigen oder zu widerlegen. Wir haben bereits vorher Analyse betrieben, indem wir Grundsituationen zwischen benachbarten Stellen und Transitionen untersucht haben. Jedoch waren diese Grundsituationen immer nur auf einen lokalen Raum beschränkt. Die Erstellung des Erreichbarkeitsgraphen entspricht schon eher einer Analyse über das ganze System. Bevor wir mit der Erklärung der wichtigsten Eigenschaften anfangen, sollte noch erwähnt werden, dass für alle zu analysierenden Netze folgende Grundvorrausetzungen bestehen: sie sind schlicht, in der Knotenzahl endlich und schwach zusammenhängend (also keine „Inseln“).

4.1 Erreichbarkeit

Bei S/T-System wird man sich viele Fragen stellen: wird dieser oder jener Fall eintreten, enden alle Schaltfolgen im gleichen Zustand, wird eine Stelle s je mehr als n Marken haben, läuft meine System endlos ?? All diese Fragen benötigen zur Beantwortung die Berechnung der Erreichbarkeitsmenge oder die Erstellung eines Erreichbarkeitsgraphen.
Ein möglicher Algorithmus zur Berechnung der Erreichbarkeitsmenge arbeitet nach dem Breitendurchlaufprinzip mit der Startmarkierung M_0 als Wurzel. Dann wird jede neu gefundene Markierung in die Erreichbarkeitsmenge aufgenommen. Das ganze geht so lange bis jede Folgemarkierung bereits in der Erreichbarkeitsmenge ist. Würde man im Tiefendurchlauf nach den Markierungen suchen, könnte es passieren das nicht alle Markierungen gefunden werden. Dieser iterative Ansatz hat natürlich den Harken, dass er bei unendlichen Erreichbarkeitsmenge endlos läuft. Dieses Halteproblem kann man jedoch in endlicher Zeit mittels eines *Überdeckungsbaumes* lösen.

Die Berechnung der Erreichbarkeitsmenge ist schon deshalb nicht effizient, da die Anzahl der Markierungen exponentiell wachsen kann. Die Größe der Erreichbarkeitsmenge hängt weniger von der Anzahl der Kanten und Knoten ab, als von den Stellenkapazitäten und Kantenbeschriftungen. Eine Erhöhung einer Stellenkapazität um eins, kann schon zu einer Verdopplung der möglichen Markierungen führen. Erreichbarkeitsmengen von B/E-System lassen sich naturgemäß wesentlich schneller berechnen, da ihre Kapazität und Kantengewichte gerade mal eins sind.

Zum Glück werden die meisten Petri-Netze von Menschen entworfen, so dass sich ihre Kanten-, Knoten- und Kapazitätszahlen für heutige Rechenleistungsverhältnisse in eher kleinen Bereichen bewegen. So ist es trotz ineffizienter Berechnung meist möglich, Erreichbarkeitsmengen und –graphen in akzeptabler Zeit zu erstellen.

4.2 Sicherheit

Sicherheit ist eine Art Garantie, dass nichts „Verbotenes“ passiert. Formal wird Sicherheit folgendermaßen definiert : sein $Y = (S,T,F,K,W,M_0)$ ein S/T-System und $B : S \rightarrow N_0 \cup \{\infty\}$ eine Abbildung, die jeder Stelle eine „*kritischen Markenzahl*“ zuordnet. Das System heißt dann *B-sicher*, falls

$$\forall M \in [M_0>, s \in S : M(s) \leq B(s)$$

Bei allen möglichen Markierungen muss die Markenzahl an den Stellen unter der kritischen Grenze bleiben. Gilt für alle Stellen dieselbe Grenze n, spricht man von n-sicher. Das System heißt *beschränkt*, falls es ein solches n überhaupt gibt, so dass es n-sicher ist.

Jetzt kommt natürlich die Frage auf, wieso eine kritischen Grenze? Wir haben doch die Kapazität an den Stellen, welche die Markenzahl begrenzt. Aber häufig ist es sinnvoll, eine Stelle, obwohl sie in der Realität eine gewisse Kapazität hat, trotzdem mit unendlicher Kapazität zu modellieren. Dann kann man die kritische Grenze gerade auf ihr reale Kapazität setzen und überprüfen, ob das System sicher ist. Ist es sicher, wurde der Vor- und Nachbereich der Stellen korrekt modelliert und die reale Kapazität wird nie überschritten.

Ein Bespiel: ein Verkehrsplaner will eine Kreuzung mit Ampeln entwerfen. Nun kann er Kreuzung selbst mit einer Kapazität eins versehen, da nur ein Auto gleichzeitig auf der Kreuzung fahren darf. Die Realität zeigt aber: es kommt immer wieder vor, dass zwei oder mehr Autos versuchen die Kreuzung gleichzeitig zu überqueren. Besser wäre es, die Kapazität der Kreuzung auf unendlich zu setzen und ihre kritische Grenze auf eins. Ist das System dann sicher, weiß der Verkehrsplaner, dass die Ampelschaltung so richtig funktioniert. Also ist die Stelle Kreuzung durch ihr umgebendes System (Ampelschaltung) auf die Markenzahl eins begrenzt, was gewünscht war.

Ob ein System sicher ist, wird mit Hilfe der Erreichbarkeitsmenge entschieden. Ist die Erreichbarkeitsmenge endlich, ist die Sicherheit auf jeden Fall in endlicher Zeit berechenbar. Dann braucht man nur für jede Markierung überprüfen, ob alle Stellen unterhalb der kritischen Grenze liegen. Bei unendlichen Erreichbarkeitsmengen wissen wir, dass sie nicht n-sicher seien können. Wäre die Markenzahl an allen Stellen immer kleiner n, kann es nur endlich viele Markierungen geben.

Diese Idee der Überprüfung von Stellen kann man auch auf Transitionen übertragen. Wenn man z.B. wissen will, ob eine Transition insgesamt nicht häufiger als n-mal schaltet, sollte man diese Transition auf ihre n-Sicherheit überprüfen. Die Sicherheit einer Transition kann man über eine extra eingefügte *Beobachtungsstelle* erreichen, die unendliche Kapazität hat und bei jedem Schalten der Transition eine Marke dazu erhält. Die Beobachtungsstelle wird die Funktionsweise des Systems nicht beeinflussen. In Abbildung 4.1 wird Transition t durch die zusätzliche Beobachtungsstelle s überprüft:

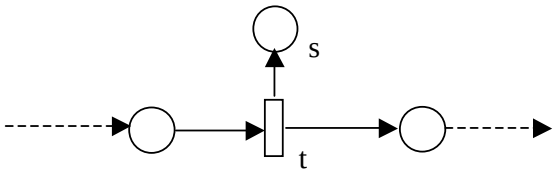

Abbildung 4.1

4.3 Lebendigkeit

Um über Lebendigkeit des ganzen System zu sprechen, müssen gewisse Bezeichnungen für lokale Lebendigkeit von Transitionen geklärt werden. Eine Transition t eines S/T-Systems heißt *tot*, falls sie unter keiner erreichbaren Markierungen aktiviert ist. Sie heißt *aktivierbar*, falls sie mindestens unter einer Markierung aktiviert ist, und schließlich ist sie *lebendig*, wenn

für jede Markierung gilt, dass diese Transition in mindestens einer Folgemarkierung wieder aktivierbar ist..

Diese Begriffe werden jetzt auch für das ganze S/T-System definiert. Ein S/T-System heißt *schwach lebendig* oder *deadlockfrei* falls unter jeder erreichbaren Markierung mindestens eine Transition aktiviert ist. Es heißt *lebendig* oder *stark lebendig*, wenn alle Transitionen lebendig sind, also alle Transitionen immer wieder aktiviert werden könnten. Das System ist *tot*, wenn keine Transition aktiviert ist.

Wenn das System in eine tote Markierung wechselt, nennt man das auch einen *Deadlock*. Tote Markierungen müssen nicht immer unerwünscht sein, gerade erwünschte Zielzustände sind häufig Deadlocks. Häufig ist es so, dass sich in nebenläufigen Systemen Prozesse gegenseitig blockieren , indem sie aufeinander warten. Klassische Beispiele sind dafür 4 Autos an einer Kreuzung ohne Ampel, oder der Betriebssystemdeadlock zweier Prozesse, die gegenseitig darauf warten, dass der andere gewünschte Betriebsmittel wieder freigibt.

Eigenschaften über Lebendigkeit eines Systems lassen sich besonders gut an Erreichbarkeitsgraphen zeigen. Ist der Erreichbarkeitsgraphen z.B. stark zusammenhängend und jede Transition mit mindestens 1 Kante vertreten, so ist das dazugehörige S/T-System stark lebendig. Gibt es einen Knoten (Markierung) von der aus keine Kanten abgehen, ist sie tot.

4.4 Synchronie

Synchronieaussagen enthalten Information darüber, wie oft Ereignisse in Abhängigkeit anderer Ereignisse eintreten. Man nimmt zum Beispiel zwei beliebige Transitionen t_1,t_2 und will zeigen wie abhängig sie voneinander sind. Nun kann man alle möglichen Schaltfolgen überprüfen und für jede Schaltfolge schauen, wie häufig wurde die jeweilige Transition geschaltet. Für jede Schaltfolge kann man dann die Differenz der beiden Werte bilden. Das Supremum aller Differenzen nennen wir *Synchronieabstand*. Diese Vorgehensweise kann man nicht nur zwischen zwei Transitionen, sondern auch auf ganze Transitionsmengen E_1,E_2 anwenden. Der Synchronieabstand wir dann geschrieben als $\sigma(E_1,E_2)$. Dieser Synchronieabstand sagt etwas darüber aus, wie stark das eine Ereignis dem anderen „vorauseilt“ oder das andere „hinterherhinkt“. Je kleiner dieser Abstand ist desto synchroner sind die Ereignisse. Zwei alternierende Ereignisse haben den Synchronabstand von 1. Ungefähr gleichzeitig eintretende Ereignisse haben den Abstand 2. Je grösser der Abstand, desto asynchroner sind die Ereignisse. Treten beide Ereignisse nie ein, haben sie einen Synchronieabstand von 0. Interessant ist, dass diese Definition des Synchronieabstands sogar die Eigenschaften einer Metrik erfüllt.

4.5 Invarianten

Wurde vorher die Netzanalyse hauptsächlich über die Erreichbarkeitsmenge und den Erreichbarkeitsgraphen betrieben, so kommt die Erstellung von Invarianten aus dem Bereich der linear-algebraischen Netzanalyse. Hier geht es um die Frage, existieren im Netz Teilmengen von Stellen, deren Gesamtsumme an Marken stets konstant bleibt ? Diese Frage ist zum Beispiel wichtig, wenn man ein System modellieren will, in dem nichts „verloren“ gehen darf.

Bei S/T-System, die als Kantengewichte nur Eins zulassen, bilden alle Stellen eins Zyklus eine solche Menge. Hier wird die Markenanzahl durch unterschiedliche Kantengewichte nicht

erhöht oder erniedrigt. Diese Stellenmengen nennen wir S-Invarianten. Ein Bespiel dafür ist die Stellenmenge $\{s_1, s_2, s_3\}$ in Abbildung 4.2 links:

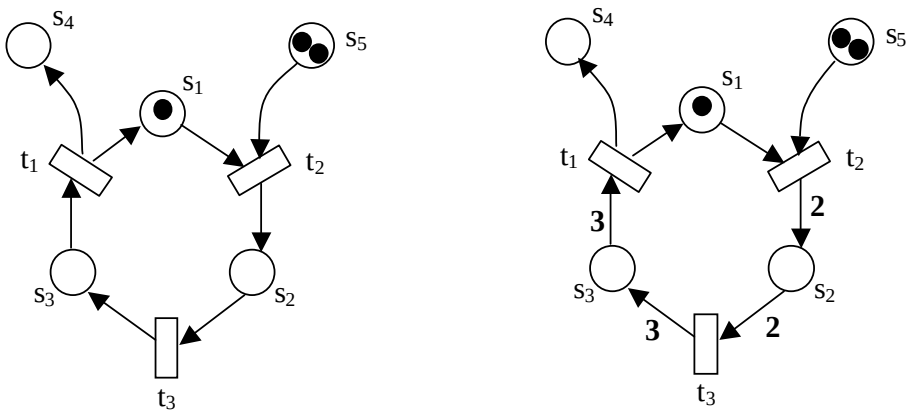

Abbildung 4.2

Bei S/T-Systemen mit gewichtetet Kanten ist ein Zyklus kein hinreichender Beweis für die Existenz einer Stellen-Invariante. Hier kann sich die Markenzahl auch innerhalb eines Zyklus durch unterschiedliche Kantengewichte erhöhen oder verringern. Man kann jedoch die einzelnen Stellen gewichten, also ihre Markenzahl in der Summerberechnung mit konstanten, positiven Faktoren multiplizieren. So kann man auch für die gewichtete Version des Netzes auf der rechten Seite von Abbildung 4.1 eine konstante Markenzahl berechnen:

$$6*M(s_1) + 3*M(s_2) + 2*M(s_3) = 6$$

Um Festzustellen, ob eine solche Gewichtung möglich, ist muss man eine lineares Gleichungssystem lösen. Sehen wir die Faktoren für die Stellen also n-Tupel x = (6,3,2,0,0), dann gilt folgender Zusammenhang mit der Inzidenzmatrix C:

$$C^T x = 0$$

Dieses Tupel $x \in Z^n$ wird auch als S-Invariante bezeichnet, da es eine Menge von Stellen bezeichnet, deren Gesamtmarkenanzahl immer innerhalb fester Grenzen bleibt. Diese Markenanzahl kann nie null oder unendlich groß werden. Die triviale Lösung (Nullvektor) ist keine gültige Invariante. Die Matrixschreibweise für das obige Beispiel:

$$\begin{pmatrix} 1 & 0 & -3 & 1 & 0 \\ -1 & 2 & 0 & 0 & -1 \\ 0 & -2 & 3 & 0 & 0 \end{pmatrix} * \begin{pmatrix} 6 \\ 3 \\ 2 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Entsprechend den S-Invarianten kann man auch T-Invarianten für Transitionen berechnen, nur wird hier die Inzidenzmatrix für das lineare Gleichungssystem nicht transponiert: Cy = 0.

5. Systeme mit individuellen Marken

Wie die Überschrift von Kapitel 3. „Systeme mit anonymen Marken“ schon erraten lässt, gibt es auch Netze mit individuellen, unterscheidbaren Marken. Diese Erweiterung unseres Systems stellt ein wesentlich mächtigeres Ausdrucksmittel dar. Durch die Veränderung der Markendefinition müssen natürlich auch die Kantengewichte und die Schaltregeln angepasst werden. Viele Netz- und Systemtypen mit individuellen Marken sind in der letzen Zeit definiert worden, die alle unter dem Oberbegriff „höhere Netze“ (High-Level Petri-Nets) zusammengefasst sind. Einige wichtige Vertreter dieser Gruppe sind:

- Prädikat/Transitions Netze
- Produktnetze
- Gefärbte Netze
- Relationale Netze
- FUNSOFT-Netze
- Numerische Netze

Folgende Abbildung 5.1 verdeutlich die historische Entwicklung der Petri-Netze (aus [Rah99]).

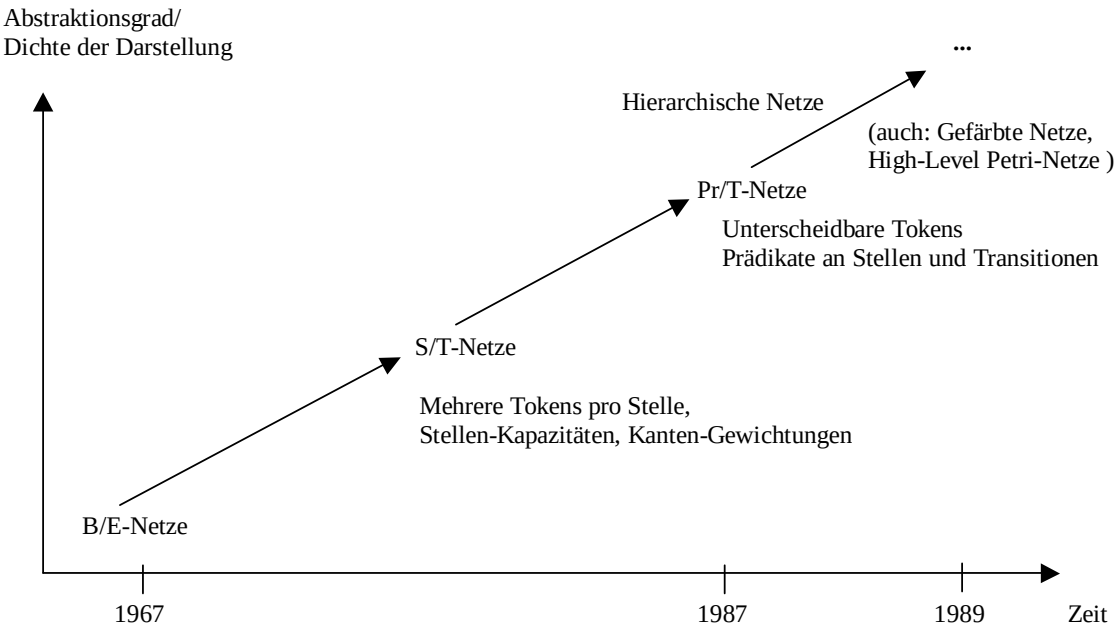

Abbildung 5.1

Zuerst werde wir ein Grundlageschema (IM-Systeme) kennen lernen, woraus dann häufig benutzte Varianten (Prädikat-Ereignis-Netz und die gefärbten Netze) entwickelt werden.

5.1 Grundlagen

Als Grundlage für High-Level Petri-Netze sind IM-Systeme definiert, wobei IM für „individuelle Marken“ stehen. Formal werden wir IM-Systeme hier nicht definieren, da es den gegebenen Rahmen Sprengen würde. Wir werden an einem Beispiel die wichtigsten Unterschiede kennen lernen. Dazu die Abbildung 5.2 eines IM-Systems:

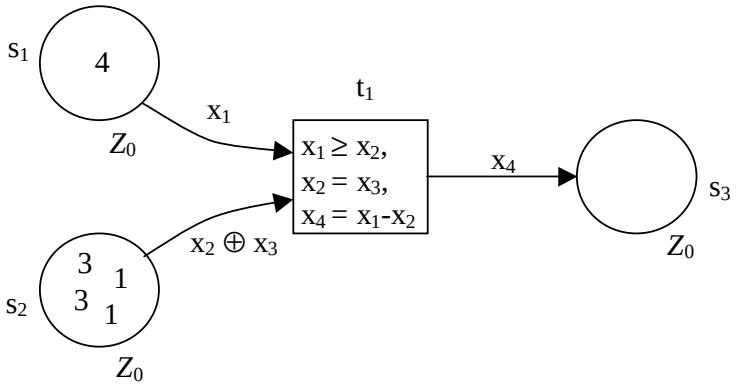

Abbildung 5.2

Wie man sieht, ist vieles hinzu gekommen. Die Punkte der Marken sind durch Zahlen ersetzt worden. Das ist aber nur ein diesem Beispiel so. Marken können Daten sämtliche Typs sein, hier ganze Zahlen, in anderen Fällen Strings, reelle Zahlen, Vektoren oder gar zusammengesetzte Datentypen. Zusammengesetzte Datentypen entsprechen einem kartesischem Produkt anderer Datentypmengen. Jede Stelle kann nur Marken des gleichen Typs aufnehmen. Da in dieses Beispiel nur ganze Zahlen enthält, steht an allen Stellen eine Z_0 als Datentyp dieser Stelle. Eine Stelle kann beliebig viele, auch identische Marken aufnehmen (also eine Mehrfachmenge) . Innerhalb eines Systems können viele unterschiedliche Datentypen existieren, sowie Datentypen, die aus anderen Datentypen zusammengesetzt sind. In den Transitionen befinden sich Prädikate, die entscheidend für die Schaltregel sind. Für diese Prädikate stehen die Inputvariabeln an den Kanten von den Stellen zu der Transition und Outputvariabeln an den Kanten von der Transition zu den Stellen. Wird mehr als eine Variabel aus einer Stelle benötigt, werden sie als formale Summen geschrieben ($x_2 \oplus x_3$).
Die Prädikate übernehmen zwei Funktionen. Zum einen müssen sie zum Schalten der Transition mit der konkreten Variabelbelegung (mit Marken aus den Inputstellen) erfüllt sein ($x_1 \geq x_2$, $x_2 = x_3$). Zum anderen definieren sie die Belegung der Ausgangsvariabeln ($x_4 = x_1-x_2$). Eine schaltende Variabelenbelegung wäre hier $x_1=4$, $x_2=3$ und $x_3=3$. Die Outputvariabel x_4 hätte dann den Wert 1. Schaltet eine Transition, werden die Inputmarken dem Vorbereich entnommen, entsprechend die Outputmarken dem Nachbereich hinzugefügt. Sind mehrere schaltende Belegungen möglich, ist nicht bestimmt welche Belegung als nächstes schaltet. Die Prädikate dürfen nur Operationen benutzen, die auf den Datentypen der entsprechenden Variabeln definiert sind. Anmerkung: formal sind Prädikate keine Sätze von Ausdrücken, sondere eine Menge aller schaltenden Variabelenbelegungen. Hier in diesem Beispiel wäre dann das Prädikat der Transition t_1 ein Menge von 4-Tupeln $= \{ x \in Z_0^4 \mid x_1 \geq x_2 \wedge x_2 = x_3 \wedge x_4=x_1-x_2 \}$. So umgeht man die komplizierte Definition für erlaubte Operationen auf den Datentypen.

5.2 Anwenderfreundliche Schreibweise

Um die grafische Darstellung der IM-Systeme überschaubarer zu gestalten wurden einige Vereinfachungen der Schreibweise eingeführt. Folgende Abkürzungen gelten für die *Identitätsrelation* (Abbildung 5.3) :

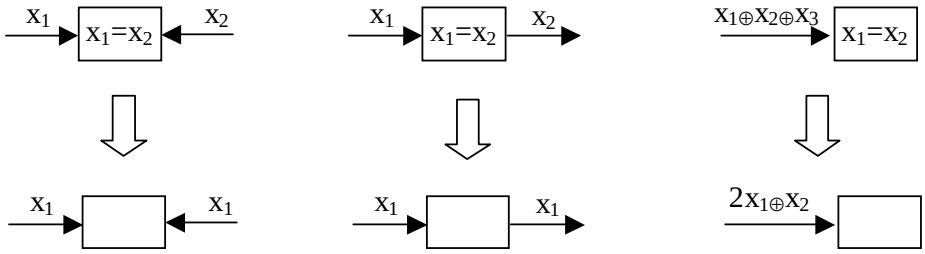

Abbildung 5.3

Wie man sieht werden Variabeln, die identischen sein müssen, einfach zusammengefasst und die entsprechende Identitätsrelation als Prädikat in den Transitionen weggelassen.
Weiter vereinfacht wird die Darstellung wenn man erlaubt, *Ausdrücke direkt an die Kanten* zu schreiben. Das wird gerne benutzt, wenn die Ausdrücke sehr einfach oder Konstanten sind. Beispiele sind man in Abbildung 5.4:

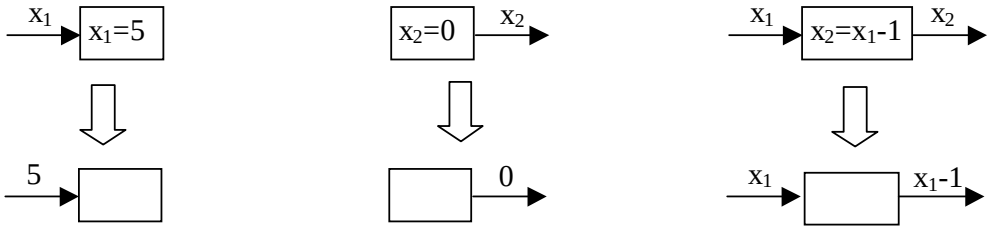

Abbildung 5.4

Eine weitere Möglichkeit ist die *Tupelschreibweise*. Bisher haben wird den Variabeln keine sprechenden Namen gegeben, zudem wurde bei mehreren Variabeln die formale Summe benutzt. Bei der Tupelschreibweise wird allen Variabeln aussagekräftige Namen gegeben und alle Variabeln einer Kante als Elemente eines Tupels übergeben. Die Tupel werden in < und > eingeklammert. So wird z.B. aus $x_2 \oplus x_3$ ein Tupel <Würfel1, Würfel2>.

5.3 Prädikt-Ereignis-Netz

Obwohl wir für die IM-Systeme Schreibabkürzungen benutzen können, ist für die alltägliche Modellierung von Systemen, dieses Modell viel zu umständlich. Theoretisch mag die Definition der Prädikate über Mengen von schaltenden Variabelenbelegungen möglich sein, praktisch ist sie jedoch nicht sehr handlich, da diese Mengen schnell unüberschaubar werden. Deshalb wird eine eingeschränkte Version eingeführt, die Prädikat-Ereignis-Netze. [Rei86]

Die erste Einschränkung zu IM-Systemen ist die, dass es nur eine Grundmenge D gibt, also keine unterschiedlichen Datentypen. Auf D wird dann eine Algebra $\underline{D} = (D, \phi)$ definiert, wobei ϕ eine Menge von Funktionen des Typs $f : D^n \rightarrow D$ ist. Diese Algebra wird benutzt um die Prädikate zu beschreiben. Die Prädikate werden nicht mehr in die Transitionen

geschrieben, sondern wie Ausdrücke an die Kanten. Deshalb gibt es eine Abbildung λ, die allen Kanten eine Menge von Termen zuweist. Für die Definition dieser Abbildung brauchen wir eine Variabelenmenge X (die Variabeln, die wir benutzen x,y,x_1,x_2 usw.) und bezeichnen die Menge $T(\underline{D}, X)$ als Terme über $\underline{D}$ und X. Diese Terme können Konstanten (n = 0) sein oder Funktionen aus $\underline{D}$, die wiederum mit Termen belegt sind:

$$t_1,...t_n \in T(\underline{D}, X) \wedge f : D^n \rightarrow D \in \phi \Rightarrow f(t_1,....,t_n) \in T(\underline{D}, X)$$

Die Menge $T(\underline{D}, X)$ enthält alle möglichen Terme, die aus Funktionen der Algebra $\underline{D}$ und der Variabelenmenge X zusammengesetzt sind. Eine Abbildung $\beta : X \rightarrow D$ nennen wird Variabelenbelegung. Die Variabeln können natürlich auch durch Terme ersetzt werden, also $\beta : T(\underline{D}, X) \rightarrow D$. Die Abbildung λ wird nun folgendermaßen definiert:

$$\lambda : F \rightarrow P(\ T(\underline{D}, X)) \setminus \{\varnothing\} \quad \text{und es gilt:} \quad \forall f \in F, \forall \beta : t_1 \neq t_2 \in \lambda(f) \Rightarrow \beta(t_1) \neq \beta(t_2)$$

An jeder Kante steht mindestens ein Term, wobei für alle Terme einer Kante gefordert wird, dass wenn sie unterschiedlich sind, sie auch immer unterschiedliche Werte berechnen. Die letzte Forderung kommt daher, dass bei Prädikat-Ereignis-Netzen in den Stellen auch keine Mehrfachmengen mehr erlaubt sind, sondern jede Marke (Element aus D) nur einmal pro Stelle erlaubt ist. Folglich ist M_0 eine Startmarkierung der Form $M_0 : S \rightarrow P(D)$. Formal ist eine Prädikat-Ereignis-Netz (P/E Netz) ein 6-Tupel N = (S, T, F, $\underline{D}$, λ, M_0) mit den oben beschriebenen Eigenschaften. Häufig wird N auch als (P, E, F, $\underline{D}$, λ, c) geschrieben. Zur Erläuterung ein Beispiel (Abbildung 5.5).

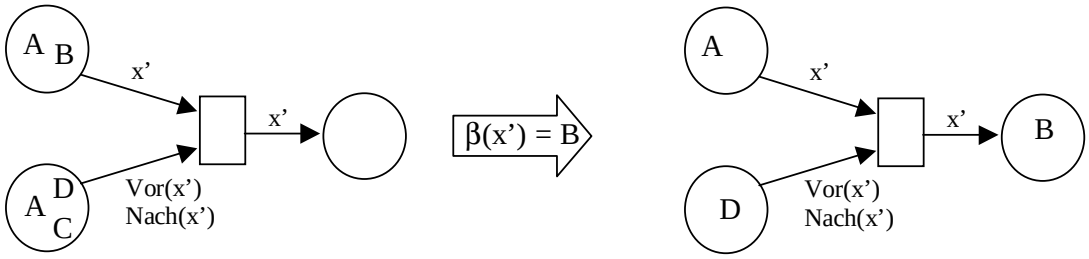

Abbildung 5.5

In diesem Beispiel ist die Grundmenge D die Menge der großen Buchstaben. Die Algebra $\underline{D}$ enthält zwei Funktionen Vor(x) und Nach(x), also der jeweilige Vorgänger oder Nachfolger des Buchstabens im Alphabet (bei Z und A zyklischer Umbruch). Markierungen werden hier wie bei B/E-Systemen Fälle genannt. Eine Transition oder Ereignis e ist genau dann aktiviert, falls es eine alle Inputprädikate enthaltende Variabelenbelegung aus dem Vorbereich gibt und alle Outputprädikate nicht im Nachbereich enthalten sind. Sei β eine mögliche Belegung für das Ereignis e, dann bezeichnet $\beta(\underline{p, e})$ mit $p \in \bullet e$ die Teilemenge der Terme, die zu den Kanten gehören, die von p nach e führen. Für $\beta(\underline{e, p})$ mit $p \in e\bullet$ gilt entsprechendes. Formal ist Ereignis $e \in E$ unter einem Fall c aktiviert durch Belegung β, falls gilt:

$$\forall p \in \bullet e : \beta(\underline{p, e}) \subseteq c(p) \qquad \text{und} \qquad \forall p \in e\bullet : \beta(\underline{e, p}) \subseteq D \setminus c(p)$$

Wurde ein Ereignis e, das unter dem Fall c durch Belegung β aktiviert ist, geschaltet, so werden die Inputelemente aus dem Vorbereich entnommen und die Outputelemente dem Nachbereich hinzugefügt. Formal wird der Folgefall c' so definiert:

$$c'(p) := \begin{cases} c(p) \setminus \beta(\underline{p, e}), & \text{falls } p \in {\bullet e} \setminus e{\bullet} \quad \text{p liegt nur im Vorbereich} \\ c(p) \cup \beta(\underline{e, p}), & \text{falls } p \in e{\bullet} \setminus {\bullet e} \quad \text{p liegt nur im Nachbereich} \\ c(p) \setminus \beta(\underline{p, e}) \cup \beta(\underline{e, p}), & \text{falls } p \in e{\bullet} \cap {\bullet e} \quad \text{p liegt im Vor- und Nachbereich} \\ c(p) & \text{sonst} \quad \text{p hat keine Kontakt zu e} \end{cases}$$

5.4 Gefärbte Netze

Würde man IM-Netze und gefärbte Petri-Netze mit Programmiersprachen vergleichen, wäre IM-Netze Assembler und gefärbte Netze eine mächtige Hochsprache. Die Einschränkungen sind nur minimal gegenüber IM-Netzen. Die größten Unterschiede bestehen in der Begrifflichkeit und der kompakten Darstellung der Prädikate. Es können mehrere Datentypen (Grundmengen) existieren, nur werden sie hier Color set genannt. Diese Grundmengen müssen jedoch endlich und nicht leer sein. Die einzelnen Element dieser Grundmengen werden Colors genannt. Σ bezeichnet die Menge der Color sets. Bedingungen und Funktionen werden über eine Erweiterung der funktionalen Programmiersprache SML (Standard Meta Language), genannt CPN ML definiert. Stellen sind Mehrfachmengen und sind vom Typ eines Color Sets, dessen Marken sie aufnehmen können. Eine Abbildung (color function) definiert für jede Stelle ihre color set, welches kursiv neben die Stelle geschrieben wird. Es gibt zwei Abbildungen auf Funktionen. Einmal von Kanten auf Funktionen (arc expression function) und von Transitionen auf Funktionen(guard function). Die Kantenfunktionen bestimmen Bedingungen für die Input- und Output-Marken. In der grafischen Darstellung werden sie direkt an die Kanten geschrieben. Die Transitionsfunktionen sind zusätzliche semantische Bedingungen , die für das Schalten notwendig sind. Sie werden in eckigen Klammern neben die Transitionen geschrieben. Für diese Funktionen werden typisierte Variabeln benutzt. Um besser über die Kanten zu sprechen, gibt es extra eine Menge von Kanten (arcs) und eine dazugehörige Abbildung auf $S \times T \cap T \times S$ (node function). Natürlich gibt es auch wieder eine Startmarkierung (initialization function). Markierungen werden nicht in die Stellen gezeichnet sondern daneben. Die Form ist n'm wobei n für die Anzahl und m für die Farbe steht. Startmarkierungen werden unterstrichen. Aktivierte Transitionen werden mit einem dickeren Rand gezeichnet.

Zusammen mit den Stellen (places) und Transitionen (transitions) ist eine gefärbtes Petri-Netz (Colored Petri Net) ein 9-Tupel CPN = (Σ, Places, Transitions, Arcs, Node function, Color function, Guard function, arc Expression function, Initialization function).

Weitere formale Definitionen wären viel zu umfangreich um sie hier sinnvoll zu besprechen. Dem interessiertem Leser ist hier [Jen90] zu empfehlen. Aber nach folgendem Beispiel dürften die grundlegenden Funktionen (Abbildung 5.6) klar sein :

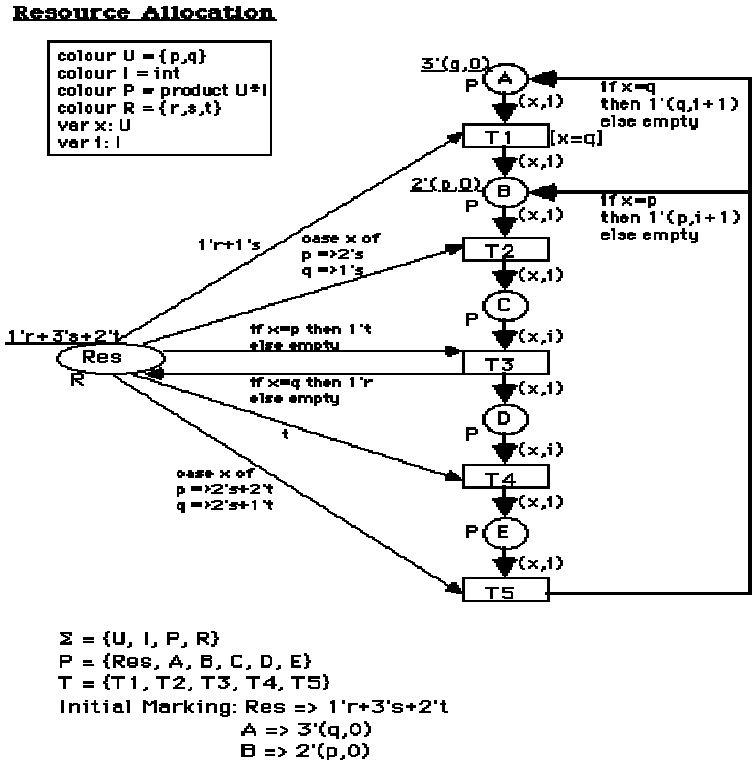

Abbildung 5.6

Diese Beispiel können wir uns als Produktion von zwei verschiedenen Produkten {p, q} vorstellen. Für diese Produktion werden verschiedene Rohstoffe benutzt {r, s, t}. Es gibt zwei Variabeln, die als Tupel (x, i) ein Produkt und seine Durchlaufanzahl angeben. Produkt p hat am Anfang eine Produktionsstelle mehr (A). Das diese Stelle nicht von Produkten des Typs q durchlaufen wird, stellt die Bedingung [x=q] an T1 sicher. An den Stellen T1 bis T5 werden je nach Produkt verschiedene Rohstoffmengen benötigt. Ein Produkt vom Typ q bei T3 gibt sogar wieder eine Rohstoff r and das Lager (Res) zurück. Oben links sieht man die Deklaration der Color sets und der Variablen. Die Deklaration als auch die Funktionen sind alle in CPN ML Notation. Die Stellen A bis E sind alle vom Color Set P, was einem zusammengesetztem Datentyp entspricht. „product“ steht hier für das kartesische Produkt.

5.4 FUNSOFT Netze

Für die Modellierung eines Softwareentwicklungsprozess wurde das FUNSOFT Netz entwickelt. Es nimmt besondere Rücksicht auf zwischenmenschliche Kommunikation und die Tatsache, dass auch in späteren Phasen der Entwicklung immer wieder rückwirkende Änderungen am Gesamtkonzept stattfinden. Function Nets waren die Vorlage bei der Entwicklung der FUNSOFT Netze, jedoch waren Function Nets formal keine Petri-Netze. FUNSOFT Netze ermöglichen die Modellierung sehr komplexer Entwicklungsprozesse, was sich darin wiederspiegelt, dass sie ein 12-Tupel sind und zur Klasse der Very High-Level Petr-Netze gehören. Transitionen werden Jobs zugeordnet, die beim Schalten ausgeführt werden. Diese Jobs werden wiederum Rollen zugewiesen. Marken können vordefinierte (real, String, int, bool usw.) oder zusammengesetzte Datentypen sein. Unter anderem wird auch der Datentyp Mensch (human) unterstützt. Auf die Marken in den Stellen kann in bestimmten

Reihenfolgen zugegriffen werden (LIFO, FIFO, RANDOM). Prädikate bilden semantische Bedingung für das Schalten von Transitionen und werden in der Sprache C ausgedrückt. Es sind sehr komplexe Schaltmöglichkeiten wählbar. Unter anderem kann man bei Konflikten automatisch oder manuell entscheiden. Kanten können Wahrscheinlichkeiten zugeordnet werden , die bestimmen, wie sich die Marken auf den Nachbereich verteilen sollen. Auch die beim Schalten beteiligte Markenanzahl kann vom Wert bestimmter Marken abhängen. [Gru91]

6. Werkzeuge für Petri-Netze

Gerade bei den High-Level Petri-Netzen ist deutlich geworden, dass Petri-Netze sehr komplex und schnell umfangreich werden können. Modellierung und Analyse von Hand wären langwierige und fehleranfällige Aufgaben, so dass schnell der Wunsch nach leistungsfähigen Werkzeugen aufkommt. Da heutige Computer über die notwendige grafische Anforderung und Rechenleistung verfügen, wurden eine Unzahl von Programmen entwickelt, die den Menschen bei der Modellierung und Analyse von Petri-Netzen unterstützen. Diese Programme unterscheiden sich teils stark in ihren Fähigkeiten und Eigenschaften.

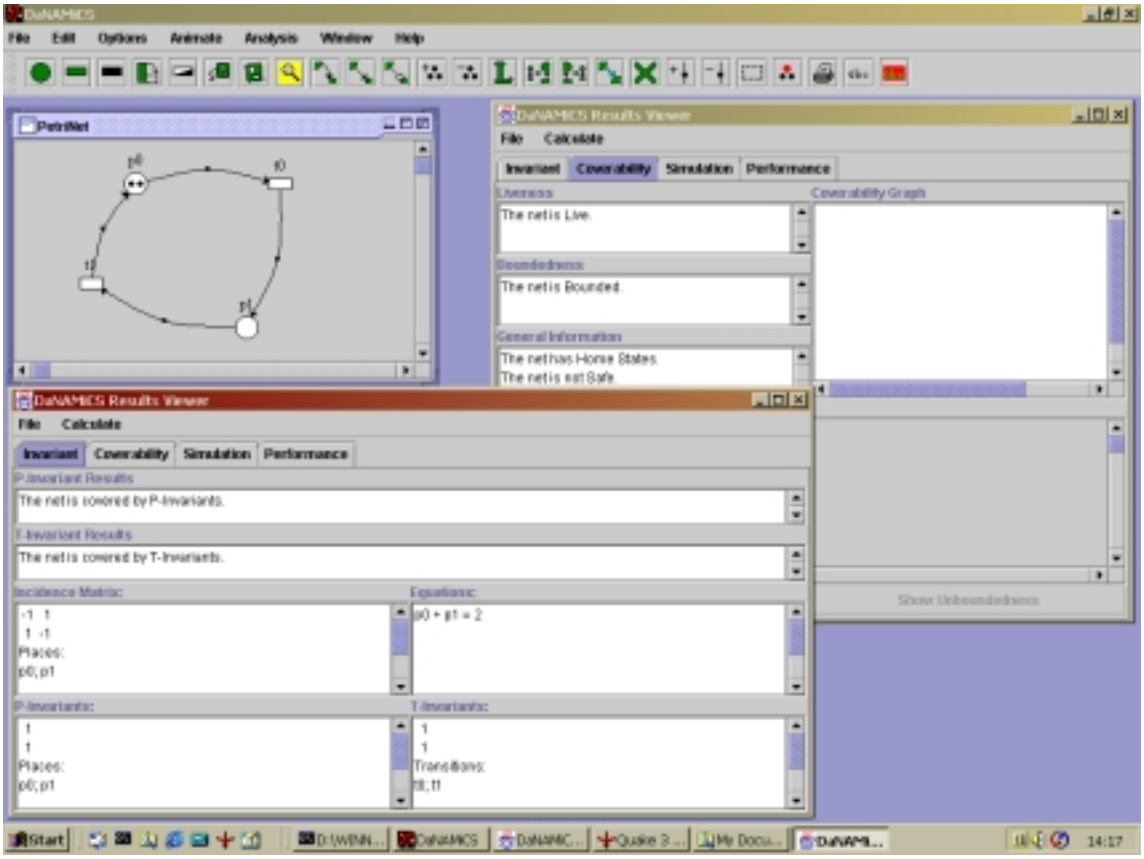

Abbildung 6.1

Abbildung 6.1 zeigt eine typische Oberfläche eines Petri-Netz Werkzeugs (hier DaNAMiCS [DaN00]), mit der Iconleiste des Editors, der grafischen Darstellung eines Netzes und Ausgabefenster verschiedener Analysen.

6.1 Anforderungen

An ein Petri-Netz Werkzeug werden je nach ursprünglicher Aufgabenstellung verschiedene Anforderungen gestellt. So wird wohl im akademischen Bereich mehr Wert auf eine detaillierte theoretische Analyse gelegt, wobei im industriellen Bereich eher auf Modellierung und praktische Simulation geachtet wird. Folgende Eigenschaften sind typische Vergleichspunkte, nach denen man sein gewünschtes Werkzeug wählt [Stö98]:

- Editor: die meisten Werkzeuge haben heute einen grafischen Editor, mit dem man seine Netze mit der Maus zeichnen kann. Hilfreich dabei sind Grids, freie Kommentartexte, skalierbare Darstellung und eine optimierte, möglichst kreuzungsfreie Kantendarstellung. Auch schön sind verschiedene Modellierungshilfen wie automatischen Vergröberung oder Entfaltung.

-Netzarten: Es sollte möglichst viele Petri-Netz Typen unterstütz werden, angefangen bei B/E-Netzen über S/T-Netze gehören zu den Standardanforderungen. Bessere Werkzeuge bieten auch Unterstützung für eine Vielzahl von High-Level Petri-Netzen, wie gefärbte Netze, stochastische Netze usw..

-Anaylse: je nach Netzart sollte man verschiedene Analysen durchführen können. Angefangen mit Strukturanalysen zur Erreichbarkeit (Erreichbarkeitsgraphen), Lebendigkeit, Fairness und Invarianten für einfachere Netze. Für High-Level Petri-Netze können z.B. folgende Analysen hinzukommen: Statistische- und Performanceanalysen.

  Die Simulation (als eine praktische Form der Analyse) sollte es möglich machen, das Netz interaktiv im Einzelschrittmodus zu „debuggen“. Wünschenswert ist die Visualisierung der Markenwanderung, um die Anschaulichkeit zu erhöhen. Hilfreich sind auch Breakpoints und Watches, wie von Debuggern höherer Programmiersprachen bekannt. Die Simulation sollte es ermöglichen, die Startwerte manuelle einzugeben oder aus einer externen Quelle zu beziehen. So könne Trace-Daten aus realen Systemen verwenden werden, um ein möglichst wirklichkeitsgetreue Simulation durchzuführen.

-Offenheit: Die Werkzeuge sollte möglichst offen und modular gestaltet sein, um das im- und exportieren von Netzen und Daten zu ermöglichen. Funktionen sollte nach außen verfügbar sein, um sie in eigenen Programmen zu verwenden. Anderseits sollt es möglich sein, eigen Code zu integrieren, um z.B. die Analysefunktionen an spezielle Wünsche anzupassen.

Generelle Anforderungen wie Stabilität, Benutzerfreundlichkeit und Geschwindigkeit werden vorausgesetzt.

6.3 Aktuelle Situation

Werkzeuge für Petri-Netze gibt es für jeden Bedarf. Es werde alle gängigen Plattformen unterstützt (Windows, Unix, Linux, Mac, Java, DOS). Geboten wird alles, vom kleinen

Freeware Programm bis zum großen Alleskönner. Bei einer Untersuchung des aktuellen Marktes [Stö98] wurden drei Hauptgruppen festgestellt.
Einmal der „akademische Einzelkämpfer“, meist Produkt einer Dissertation oder Promotion, das auf spezielle Einzelprobleme getrimmt wurde. Meist haben diese Programme eine einfache Oberfläche und sind nicht sehr modular. Fehlende Pflege und Dokumentation sind die Regel, jedoch sind es diese Programme, die originelle, neu Ideen aufgreifen. Dazu sind sie meist kostenlos.
Die kommerziellen Werkzeuge glänzen mit einer ausgereiften Oberflache, Stabilität und guter Dokumentation. Sie decken eine breite Palette von Funktionen ab und sind meist selbst erweiterbar. Analytische Funktionen sind eher auf praktische Wünsche abgestimmt, als auf theoretische Untersuchungen. Natürlich haben solche Produkte ihren Preis, häufig wird jedoch eine kostenlose Evaluationversion bereitgestellt.
Programme akademischer Gruppen sind häufig eine Mischung aus den beiden obigen Gruppen. Sie wurden meist an Universitäten von mehreren Wissenschaftlern und Studenten entwickelt. Sie sind häufig modular und erreichen beachtliche Qualität, auch was Pflege und Dokumentation angeht. Ihre Ausrichtung ist meist eher Wissenschaftlich, sie können jedoch im Einzelfall mit kommerziellen Produkten mithalten. Für akademische Zwecke sind sie meist kostenlos.

Bei der Suche nach dem richtigen Werkzeug wird man im Internet durch spezielle Suchmaschinen und Datenbanken unterstützt [Gar00][PNC00]. Sie erlauben es, nach Programmen mit gewünschten Eigenschaften zu suchen. Bis zu 100 und mehr Werkzeuge sind mit ausführlichen Beschreibungen aufgeführt. Auch gibt es schon detaillierte Untersuchungen über die wichtigsten Werkzeuge mit genauen Bewertungssystemen für die einzelnen Funktionen und Eigenschaften. [Stö98]

7. Petri-Netze und die Projektgruppe

Wie werden wir Petri-Netze in der Projektgruppe einsetzen ? Zum einen werden wir sie wie anderen PGs auch benutzen, um unseren eigenen Programmablauf zu modellieren. In unserem speziellen Fall werden wir sie aber auch für das Process landscaping einsetzen.

7.1 Als Spezifikationssprache

Wie bereits vielfach bewährt, werden auch wir bei der Organisation und Realisation unseres Projektes auf die rollenbasierte Softwareentwicklung [Gru98] zurückgreifen. Von der Analyse über die Spezifikation bis hin zur Implementierung müssen sich alle Beteiligten mehr oder weniger über die Funktion und Arbeitsweise unsers Programms einig sein. Um das möglichst effektiv zu gestalten sollten wir ein Modell entwerfen, das unser gedachtes Programm wiedergibt. Je nachdem aus welcher Sichtweise man das System betrachtet, ist die ein oder andere Spezifikationssprache besser geeignet. Aus Sicht der objektorientierten Programmierung bietet sich z.B. UML an. Liegt jedoch das Augenmerk auf der Darstellung dynamischer Prozesse, sind Petri-Netze zur Beschreibung die bessere Wahl. Besonders die Person in der Rolle des *Spezifizierers* sollte ausgeprägte Kenntnisse über Petri-Netze und deren Werkzeuge besitzen. Er muss aus der Anforderungsanalyse ein System entwerfen, das den geforderten Ansprüchen entspricht. Dabei hat er sicherzustellen, dass sein Modell fehlerfrei ist und in der Implementation keine Mehrdeutigkeit zu lässt. Hier spielen die umfangreichen Analyse und Simulationsmöglichkeiten von Petri-Netzen ihr volle Stärke aus.

Da die Spezifikation eng mit der Analyse und dem Design verzahnt ist, sollten alle daran beteiligten Rollen ein Grundverständnis für Petri-Netze haben. Einfache Petri-Netze können wegen ihrer schnellen, optischen Verständlichkeit schon beim Kunden eingesetzt werden um grundlegende Eigenschaften zu klären.

7.2 Als Model für die Softwareentwicklung

Wir werden am Problem des Process Landscapings arbeitet, in unserem Fall geht es um den Prozess der Software-Entwicklung. Dabei geht man davon aus, dass es für umfangreiche und langwierige Prozesse nicht vorteilhaft ist, sie in einem großen Prozessmodell zu erfassen. Das wäre zu unübersichtlich und zu starr gegenüber Veränderungen. Vielmehr ist es sinnvoll zusammenhängende Kernprozesse zu finden, diese dann einzeln zu modellieren und ihre Verbindung über Schnittstellen zu realisieren. Die Aufteilung in Kernprozesse und ihre Verbindungen untereinander ist dann die Prozesslandschaft. So können einzelne Kernprozesse intern verändert werden, ohne das sich an der eigentliche Prozesslandschaft etwas ändert. Gruppen, die für einzeln Prozesse verantwortlich sind, können so unabhängig ihre Prozesse modellieren, solange sie die Schnittstellenspezifikation einhalten. Um die Kernprozesse und ihre Unterprozesse zu Modellieren brauchen wir natürlich wieder eine Modellsprache, die in unserem Fall ein High-Level Petri-Netz seien wird (FUNSOFT Netze mit Unterstützung durch das Werkzeug LeuSmart). [GW99]

Insgesamt sprechen wir also mit Petri-Netzen über Petri-Netze.

8. Fazit

Petri-Netze sind eine mächtige Modellierungssprache, die für jede Ebene der Abstraktion eine passende Variante bereitstellt. Die einfachen Vertreter B/E- und S/T-System sind grafisch so einleuchtend, dass sie auch ohne größere Einarbeitung schnell verständlich sind. High-Level Petri-Netze bieten kompakte Darstellung komplexer Prozesse ohne dass Mehrdeutigkeiten bei der Interpretation aufkommen können. Die formalen Definitionen ermöglichen genaue mathematischen Analysen und Korrektheitsbeweise, was vielen anderen Modellsprachen fehlt. Die Möglichkeit der Schrittweisen Simulation erlauben es bestens, Vorgänge zu verstehen oder Denkfehler zu entdecken. Durch weitestgehende Standardisierung wurde eine Sprache geschaffen die international verstanden wird und zum Allgemeinwissen eines jeden Informatikers gehören sollte. Selbst wenn man noch nie was von Petri-Netzen gehört hat, hat man schon ähnlich Modelle intuitiv benutzt. Natürlich sind Petri-Netze nicht vollkommen, so ist bis heute kein Algorithmus bekannt, der Petri-Netz-Systeme automatisch optimieren kann [Rah99]. Auch können Petri-Netze ab gewissen Größen unüberschaubar werden, einheitliche Modularisierung und definierte Schnittstellen fehlen.

Aber gerade auf dem Gebiet soll Process Landscaping Abhilfe schaffen ;-)

9. Quellenangaben

9.1 Literaturangaben

[Bau96] Baumgarten, B. ,"Petri-Netze – Grundlagen und Anwendung",
Spektrum Akademischer Verlag, 1996

[Ger99] Gerber, S. „Petri-Netze", Skript zur Vorlesung,
Institut für Informatik, Universität Leipzig, 1999

[Gru91] Gruhn, V., „Validation and Verification of Software Process Models", Dissertation,
Fachbereich Informatik, Universität Dortmund, 1991

[Gru98] Gruhn, V., „Software-Technologie", Folien zur Vorlesung,
Fachbereich Informatik, Universität Dortmund, 1998

[GW99] Gruhn, V.,Wellen,U., "Process Landscaping: Modelling Complex Business Processes",
European Journal of Engineering for Information Society Applications, 1999,
http://www.nectar.org/journal/03/012.htm (20.3.2000)

[Iso97] Committee Draft ISO / IEC 15909, „High-level Petri Nets - Concepts, Definitions and
Graphical Notation", 1997,
http://www.daimi.au.dk/PetriNets/standard/docs/pnstd34.pdf (20.3.2000)

[Jen90] Jensen, K., "Coloured Petri Nets: A high-level Language for System Design and
Analysis", Springer Verlag 1990

[Pet81] Peterson, J.L., „Petri net theory and the modeling of systems",
Prentice-Hall Inc., 1981

[Rah99] Rahm, E., „Workflow-Managment-Syteme" , Skript zur Vorlesung,
Institut für Informatik, Universität Leipzig, 1999,
http://www.informatik.uni-leipzig.de/ifi/abteilungen/db/skripte/WMS/inhalt.html
(20.3.200)

[Rei86] Reisig, W., „Petrinetze – Eine Einführung", 2. Auflage,
Springer Verlag 1986

[Stö98] Störrle, H., „An Evaluation of High-End Tools for Petri-Nets",
Institut für Informatik, Ludwig-Maximilians-Universität München, 1998,
http://www.pst.informatik.uni-muenchen.de/personen/stoerrle/survey.ps.gz (20.3.200)

[Weg90] Wegener, I., „Kompexitätstheorie „ Skript zur Vorlesung,
Fachbereich Informatik, Universität Dortmund, 1990

[Weg95] Wegner, G., „Lineare Algebra für Informatiker", Skript zur Vorlesung
Fachbereich Mathematik, Universität Dortmund, 1995

9.2 WWW-Seiten im Internet

[DaN00] "DaNAMiCS - a Petri Net Editor",
Computer Science Department, University Cape Town, South Africa
http://www.cs.uct.ac.za/~idavies/DaNAMiCS.html (20.3.2000)

[Gar00] Garbe,W., "The Petri Net tool survey"
http://home.arcor-online.de/wolf.garbe/petrisoft.html (20.3.2000)

[PNC00] "Homepage of the International Petri Net Community",
Computer Science Department, University of Aarhus, Denmark,
http://www.daimi.au.dk/PetriNets (20.3.2000)