Ein Verfahren zur einheitlichen Bearbeitung von Analyse- und Entwicklungsvorhaben

Bernd Eichenauer, **IBE**

1. Vorbemerkung

Im folgenden wird ein Verfahren zur Analyse und Entwicklung von diskreten Ablaufsystemen beschrieben, dessen Grundlagen schon seit den 60-iger Jahren bekannt sind [1], das aber bis vor wenigen Jahren wegen seiner Unhandlichkeit in der Praxis kaum genutzt wurde. Mit der Verfügbarkeit von schnellen und bequem handhabbaren Rechenanlagen an fast jedem Arbeitsplatz sind heute die Voraussetzungen für eine effiziente Nutzung des zwischenzeitlich von zahlreichen Autoren erweiterten Verfahrens gegeben.

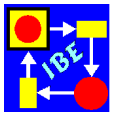
In letzter Zeit wurde das Verfahren praxisnah implementiert und an die heute üblichen Benutzer-Oberflächen angepaßt (z.B. [2]). Entstanden sind dabei bequem handhabbare Entwicklungs-Tools, die praktisch überall dort einsetzbar sind, wo technische oder kommerzielle Abläufe zu planen und zu realisieren sind.

Dieser Artikel wurde geschrieben, um zu zeigen, daß die früheren Vorbehalte gegen die Methode der Petri-Netze heute gegenstandslos geworden sind und um die enormen Vorteile dieser Methode bei der Bearbeitung von Analyse- und Entwicklungsvorhaben aufzuzeigen.

2. Das Drei-Phasen-Vorgehensmodell bei der Bearbeitung von Analyse- und Entwicklungsvorhaben

Würde man einem Systementwickler die Aufgabe stellen, ein Produkt, z.B. ein Rechnerprogramm oder ein Fahrzeug, zu entwickeln und ohne weiteren Test auszuliefern, so würde er das mit Recht ablehnen. Wir wissen, daß wir bei der Systementwicklung Fehler machen und daß diese durch nachfolgende Tests möglichst weitgehend zu beseitigen sind. Es ist heute unbestritten, daß ein Produkt erst nach der Durchführung umfangreicher Qualität-Sicherungs-Maßnahmen ausgeliefert werden sollte [3]. Normalerweise kann man davon ausgehen, daß man für den Komponenten- und den Systemtest bei technischen Produkten je nach Anwendungsgebiet zwischen 30% und 50% der gesamten Entwicklungskosten aufwenden muß.

Während Prüfverfahren bei der Entwicklung von technischen Produkten problemlos angewandt werden können, weil hier Prototypen der Produkte für das Austesten zur Verfügung stehen, gab es beim Entwurf oder der Veränderung betrieblicher Abläufe, abgesehen von den selten inszenierten kostspieligen Rollenspielen, bis vor kurzem kein Äquivalent dazu. Die Folge davon ist, daß die Modellierung häufig schon nach



der Spezifikationsphase abgeschlossen wurde und man Erfahrungen mit dem dynamischen Verhalten des spezifizierten Modells sozusagen 'am lebenden Objekt' selbst sammeln mußte.

Die Behebung der bei dieser Vorgehensweise unvermeidlichen Planungsfehler und Schwachstellen verursacht nicht selten enorme Folgekosten, die bei Verfügbarkeit geeigneter Planungshilfen zumindest teilweise vermeidbar gewesen wären. Sicher sind die zahlreichen Firmen-Reorganisationen, von denen man allenthalben hört, zum Teil auch auf frühere Planungsfehler zurückzuführen.

In den letzten Jahren sind nun durch den raschen Fortschritt der Rechnertechnik Entwicklungssysteme möglich geworden, welche die kostengünstige Erstellung realistischer Prototypen von allgemeinen technischen und/oder kommerziellen Systemen vor deren Implementierung ermöglichen. Damit kann man schon sehr früh Erfahrungen über das dynamische Verhalten eines geplanten Systems sammeln und das zu implementierende reale System am Modell studieren und optimieren.

Voraussetzung dafür ist eine geeignete Spezifikationsmethode für den Modellentwurf, mit der sowohl die statischen als auch die dynamischen Anteile des zu entwerfenden Systems einheitlich beschreibbar sind. Wir verlangen also eine Beschreibungssprache, mit der sich alle Aspekte eines zu entwickelnden Modells vollständig beschreiben lassen und in der die von zahlreichen CASE-Tools her bekannten Brüche in den Beschreibungsmitteln fehlen. Einheitlichkeit in der Beschreibungsmethodik bedeutet bekanntlich auch Effizienz bei der Entwicklung und Änderung, bessere Übersicht und weniger Fehler.

Die beiden ersten Phasen eines Vorhabens sind also die Spezifikations- und die Simulationsphase. Für viele Anwendungen, die sich auf Analysen beschränken und insbesondere im kommerziellen Bereich vorkommen, sind nur diese beiden ersten Phasen von Bedeutung. In einer dritten Phase, die bei technischen Anwendungen immer interessanter wird, kann das Modell an die Umwelt angeschlossen werden und dann, wo anwendbar, direkt als Automatisierungsprogramm eingesetzt werden. Mit der zunehmenden Verflechtung von betrieblichen und technischen Abläufen (z.B. bei dezentral organisierten Fertigungssystemen) und mit der immer schneller ablaufenden Hardware gewinnt diese dritte Phase zunehmend an Bedeutung.

3. Die Spezifikationsphase

Für die Realisierung des Drei-Phasen-Modells reichen die häufig eingesetzten mehr informellen Beschreibungsverfahren (Flußdiagramme, Hierarchiediagramme, usw.) nicht aus. Es wird vielmehr eine exakte Systembeschreibung benötigt, ohne die eine aussagekräftige Simulation und erst recht die Programmausführung nicht möglich ist. Gleichzeitig soll das Modellierungsverfahren aber so einfach und praktisch sein, daß es auch von in der Programmierung weniger erfahrenen Anwendern ingenieurmäßig eingesetzt werden kann.

3.1 Erweiterte Petri-Netze

Eine Methodik, mit der sich sowohl die Forderung nach Exaktheit als auch die nach einfacher Formulierbarkeit erfüllen läßt, wurde schon zu Beginn der 60-iger Jahre

von Petri [1] vorgeschlagen. Die nach ihm benannten sog. Petri-Netze (z.B. [4,5,6]) ermöglichen zwar eine exakte graphische Modellspezifikation, waren aber seinerzeit in der Handhabung so unpraktisch, daß sie zunächst nur von wissenschaftlichem Interesse waren. Durch die Möglichkeiten der modernen Rechnertechnik hat die Methode mit zahlreichen Erweiterungen in den letzten Jahren sehr an Attraktivität gewonnen und ist derzeit eine der wenigen praktisch einsetzbaren Methoden, die gleichermaßen für die Spezifikation wie für die Simulation diskreter Ablaufsysteme geeignet sind.

Ein Petri-Netz sieht, oberflächlich betrachtet, einem Flußdiagramm ähnlich. Es basiert auch auf Graphen und stellt den jeweiligen Systemzustand graphisch dar. Im Gegensatz zu Flußdiagrammen sind Petri-Netze jedoch streng strukturiert. Sie weisen eine exakt definierte graphische Syntax und eine klare Semantik auf.

Petri-Netze sind ereignisorientiert und lassen sich aus wenigen einfachen graphischen Elementen aufbauen. Sie bestehen aus zwei verschiedenen Knoten-Typen (Stellen und Transitionen), aus Konnektoren und aus Marken (siehe Abb. 1).

Marken repräsentieren die zu bearbeitenden Objekte wie Information, Material, Rechnungen, Werkstücke, Lagergut, Transportwagen, usw. Zustandsänderungen des Systems spiegeln sich in der Bewegung der Marken von Stellen zu Transitionen und wieder zu Stellen wider.

Stellen sind die passiven Elemente eines Petri-Netzes. Sie dienen als Speicher für Informationen und Marken. Die Belegung der Stellen eines Netzes mit Marken repräsentiert den jeweiligen Zustand eines Netzes.

Transitionen sind die aktiven Elemente eines Petri-Netzes. Sie führen Aktionen aus und verarbeiten Information. Eine Transition zieht Marken von Eingangsstellen ab, verändert ihre Eigenschaften nach vom Anwender vorgegebenen Verfahren und legt neue Marken auf Ausgangsstellen ab.

Konnektoren verbinden Stellen und Transitionen.

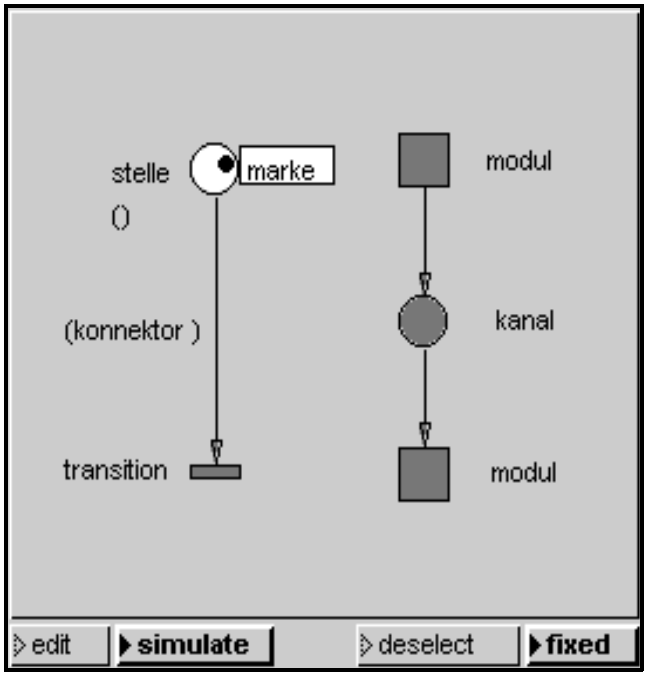


Abb. 1: Netz-Elemente verallgemeinerter Petri-Netze.

Man sagt, daß eine Transitions 'feuert', wenn die Bedingungen für das Einziehen von Marken von ihren Eingangs-Stellen und für das darauf folgende Ausgeben von anderen Marken auf ihre Ausgangs-Stellen erfüllt sind. Bedingungen für das Feuern sind beispielsweise, daß auf allen Eingabe-Stellen der Transition eine Marke liegt, oder das beim Feuern die Kapazität einer Ausgabe-Stelle nicht überschritten wird.

Wie bei jeder technischen Entwicklung wurden auch an den ursprünglichen Petri-Netzen im Laufe der Jahre zahlreiche praxisgerechte Verbes-

serungen und Erweiterungen vorgenommen. Das ursprüngliche Petri-Netz-Konzept wurde dabei stark erweitert, um die Akzeptanz durch die Anwender zu erhöhen. Ziel war die Bereitstellung von effizienten Arbeitswerkzeugen, mit denen sich schnell und intuitiv komplexe Systeme entwerfen lassen. Stellvertretend für die zahlreichen Erweiterungen wird in den folgenden Abschnitten eine Auswahl von Erweiterungen beschrieben, die wir bei der Modellierung einsetzen.

3.2 Objekt-orientierter Modellentwurf

Petri-Netze eignen sich hervorragend für den objektorientierten Entwurf von Anwendungen. In der Tat abstrahieren ja die verschiedenen Elemente von Petri-Netzen allgemeine Bausteine von ereignisorientierten, parallelen, diskreten Systemen. Der dynamische Objektfluß wird unabhängig davon formuliert, was die verwendeten Bausteine des Petri-Netzes in der realen Welt bedeuten. Deshalb lassen sich mit Petri-Netzen die unterschiedlichsten Anwendungen entwerfen.

Welche Bedeutung die einzelnen Elemente eines Petri-Netzes im jeweiligen Anwendungsgebiet haben, legt der Anwender von Fall zu Fall selbst fest. So können, wie oben schon erwähnt, die dynamischen Objekte, die sich innerhalb eines Petri-Netzes bewegen (Marken), im Modell eines Kommunikations-Systems eine Botschaft bedeuten, während sie im Modell eines Produktions-Systems irgendein Produkt oder Material auf einem Förderband darstellen. Da Marken in Stellen gespeichert werden, dienen Stellen dabei zur Modellierung der Sender und Empfänger von Botschaften oder zur Modellierung des Förderbands. Transitionen schließlich geben an, wie Botschaften übertragen werden und welche Verarbeitungsmaßnahmen dabei stattfinden sollen oder modellieren die Bearbeitungsmaschinen für das Material auf dem Förderband.

Wie flexibel und einfach die Zuordnung zwischen den Bausteinen eines Petri-Netzes und realen Objekten ist, sieht man auch an dem in Abb. 6 dargestellten Kreuzungsmodell. Hier bedeuten die Marken die verschiedenen Fahrzeuge, die Stellen modellieren die sich kreuzenden Straßen und Transitionen werden u.a. zur Darstellung von Ampeln eingesetzt.

3.3 Beschriftung und Attributierung von Netzelementen

Für die Beschreibung der Verarbeitung innerhalb von Transitionen und für die Attributierung von Marken und Konnektoren können prinzipiell beliebige Programmiersprachen verwendet werden. Wir stellen aber an eine Programmiersprache, die für die Bearbeitung von Netzen verwendet werden soll, folgende Forderungen:

- Da die verschiedenen Netz-Elemente, wie oben schon erwähnt, als Objekte anzusehen sind, denen der Anwender bei der Modellierung reale Objekte zuordnet, sollte die zu verwendende Programmiersprache ebenfalls objekt-orientiert sein. Sind die einzelnen Objekte des Petri-Netzes Instanzen geeignet festzulegender Klassen, so kann ein Modell einheitlich objekt-orientiert formuliert werden.

- Die Sprache soll es ermöglichen, während der Entwicklung schnell zwischen Editor und Simulator hin- und herzuschalten. Damit kann der Entwicklungs-

prozeß beschleunigt werden; z.B. können Teilnetze problemlos ausprobiert werden und Fehler lassen sich während der Simulation schnell beseitigen.

- Die Sprache soll für alle gängigen Rechner und Betriebssysteme verfügbar sein. Die Übertragung von Netzen von einem Rechnertyp auf einen anderen soll ohne Aufwand möglich sein.

- Die Sprache soll einfach und intuitiv verständlich sein und auch von dem in der Programmierung wenig erfahrenen Anwender schnell erlernt und eingesetzt werden können.

Eine der Programmiersprachen, die alle genannten Forderungen gut erfüllt, ist Smalltalk-80 in der Implementierung von ParcPlace Systems [7,8]. Smalltalk-80 bietet einerseits zusammen mit der Klassenbibliothek einen mächtigen objekt-orientierten Sprachumfang und kann andererseits ohne große Programmier-Erfahrung eingesetzt werden. In den späteren Beispielen wurde überall Smalltalk-80-Code eingeblendet.

Die meisten Elemente in unseren erweiterten Petri-Netzen können mit Text beschriftet werden. Während Kommentar als freier Fließtext abgefasst werden kann, sind die eigentlichen Inskriptionen Smalltalk-Anweisungen und müssen daher den Smalltalk-Konventionen genügen.

Das Verhalten von Transitionen beim Feuern von Marken wird über Inskriptionen festgelegt. Die Interaktion findet statt, sobald Botschaften mit den einlaufenden Marken ausgetauscht werden. Botschaften können die Attribute von Marken verändern oder ihren Wert testen, wenn beispielsweise das Ausführen des Transitions-Codes vom aktuellen Zustand des Netzes (dem Wert eines Marken-Attributs) abhängt.

Jeder Initial-Marke können beliebig viele Attribute zugeordnet werden. Jedes Attribut wird dabei über Smalltalk-Anweisungen festgelegt, die bei der Erzeugung der Marke ausgeführt werden. Der Wert der letzten Anweisung liefert den Attribut-Wert. Der inskribierte Smalltalk-Code kann aus einem ganz einfachen Objekt, beispielsweise einer Ganzzahl oder einem anderen Literal bestehen; er kann aber auch ein komplexes Programm sein. Abb. 2 zeigt ein Fenster, in dem die Initial-Marken für die Stelle 'stelle1' definiert werden. 'stelle1' besitzt vier Initial-Marken. Die dritte Marke besitzt zwei Attribute. Das zweite Attribut ist die Ganzzahl 22.

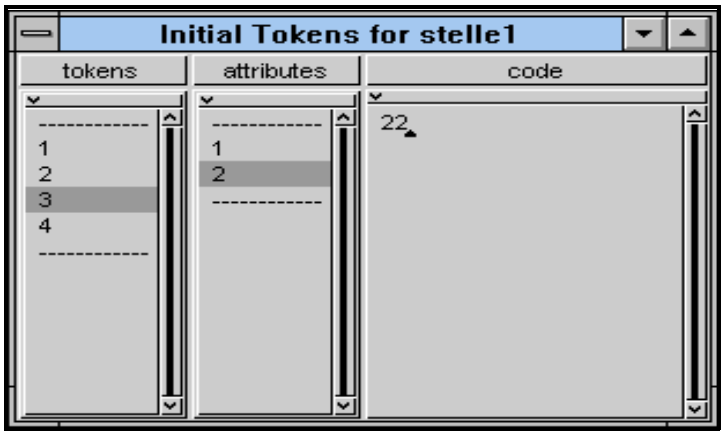


Abb. 2: Vereinbarung von Initial-Marken und deren Attributen

Ähnlich wie bei Marken sind auch Konnektoren Attribute zuordenbar. Über attributierte Konnektoren dürfen nur Marken fließen, deren Attribute mit den Konnektor-Attributen verträglich sind. Damit kann u.a. die Auswahl eines bestimmten Konnektors für das Abfließen einer Marke gesteuert werden.

Die Inskriptionen von Transitionen werden in die drei Bestandteile Bedingungscode,

Verzögerungscode und Aktionscode unterteilt.

Mit dem Bedingungscode, der einen der boolschen Werte 'true' oder 'false' liefern muß, kann das Aktivieren einer Transition von den Werten der Input-Marken abhängig gemacht werden. Ergibt sich bei der Abarbeitung des Bedingungscodes der Wert 'false', so kann die Transition nicht feuern.

Klassische Petri-Netze arbeiten ohne Zeitbezug. Alle Transitionen werden auf einmal abgearbeitet, wenn die dafür erforderlichen Bedingungen erfüllt sind. Dadurch wird die “Gleichzeitigkeit” von parallelen Abläufen modelliert. Das ist natürlich für die adäquate Modellierung realer Systeme und für das prozeßgebundene Echtzeit-Verhalten auf einem Zielsystem zu wenig. Deshalb kann in manchen Petri-Netz-Erweiterungen jeder Transition wahlweise ein Zeitverhalten zugeordnet werden, indem das Feuern der Transition verzögert wird. Über den Verzögerungscode, der eine nichtnegative Zahl liefern muß, kann das Feuern einer Transition um so viele Simulator-Zeiteinheiten, wie die Zahl angibt, verzögert werden. Wir kommen später noch auf den Begriff der Simulator-Zeiteinheit zurück.

Der Aktionscode schließlich, wird abgearbeitet, sobald die Transition feuert. Aktionscode wird in der Regel eingesetzt, um Variablen Werte zuzuweisen, die an anderer Stelle des Netzes ausgewertet werden, um Attribute zu verändern oder um Seiteneffekte zu bewirken. Beispielsweise kann über den Aktionscode eine Liste, die in einem Fenster angezeigt wird, auf den neuesten Stand gebracht oder ein Wert von einem Prozeß-Eingabegerät eingelesen werden.

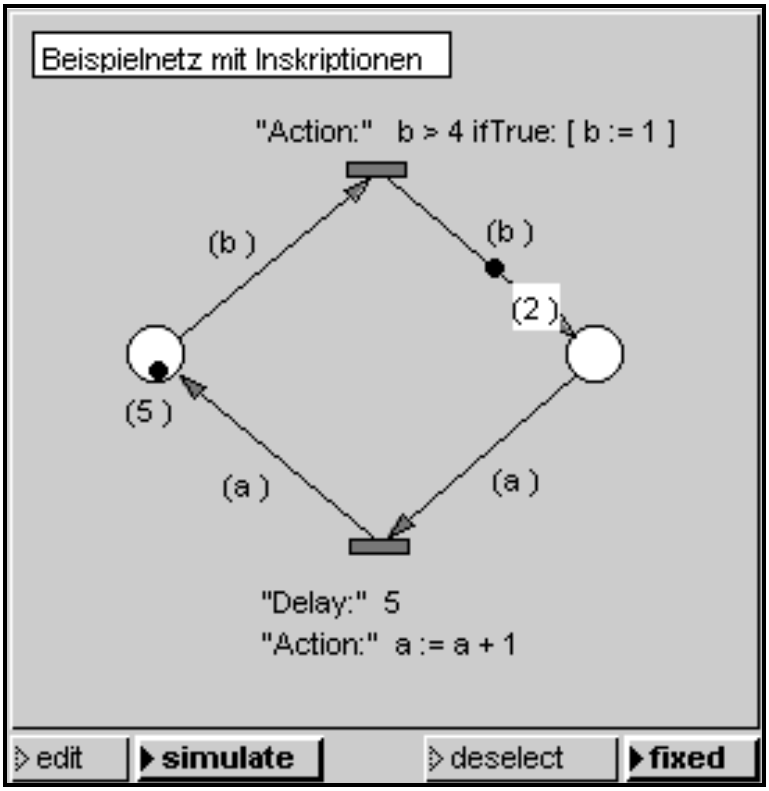


Abb. 3: Einfaches Beispielnetz mit Smalltalk-Inskriptionen und Marken

In Abb. 3 ist ein einfaches Netz dargestellt, in dem Smalltalk-Inskriptionen verwendet werden. Die Marken führen als Attribute Werte mit sich, die in den Aktionscodes erhöht bzw. bei Überschreiten einer Grenze wieder zurückgesetzt werden. Eine Marke mit dem Attributwert 2 läuft gerade von der oberen Transition zur rechten Stelle hin. Eine zweite Marke mit dem Attributwert 5 wartet in der linken Stelle.

3.4 Anfangs- und Ende-Code

Jedem Netz kann ein Anfangs- und ein Ende-Code zugeordnet werden. Der Anfangscode dient häufig zur Initialisierung globaler Variablen und wird beim Start des Netzes ausgeführt. Mit dem Ende-Code können z.B. externe Geräte, die bei der Simulation bzw. Ausführung des Netzes verwendet wer-

den, zurückgesetzt werden.

3.5 Hierarchische Strukturierung

Der Übersichtlichkeit halber werden komplexe Netze hierarchisch strukturiert, d.h. in über- und untergeordnete Teilnetze zerlegt. Die in den einzelnen Ebenen anzusiedelnden Teilnetze können jederzeit durch Auswahl von Netzknoten darüberliegender Ebenen frei bestimmt werden. Die Zahl der Hierarchie-Ebenen darf nicht beschränkt sein. Damit ist es möglich, ein System Top-down oder Bottom-up zu entwickeln. Gelegentlich wird auch das Middle-Out-Verfahren eingesetzt.

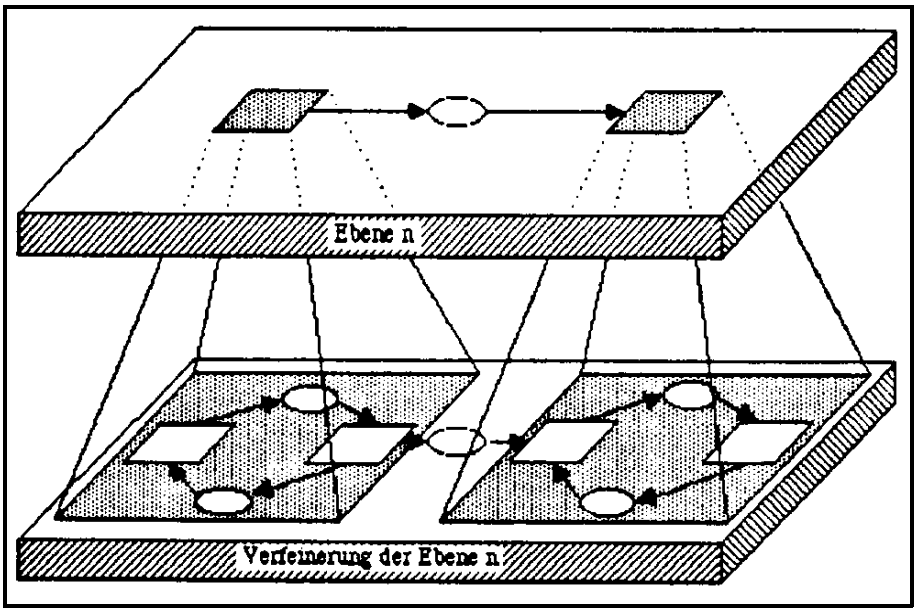


Abb. 4: Hierarchisierung von Petri-Netzen

Es gibt im wesentlichen zwei Typen von Teilnetzen, nämlich 'Module' und 'Kanäle' (siehe Abb.1).

Module sind aktive Elemente, in denen alle Arten von Netz-Elementen vorkommen dürfen. Jeder Modul kann einzeln gespeichert und auch wieder geladen bzw. in ein vorhandenes Netz eingefügt werden. Aus Transparenzgründen sollten die Schnittstellen zu Moduln im Obernetz auch in dem verfeinerten Unternetz dargestellt werden, damit die jeweilige Einbettung des Moduls auch im Unternetz erkennbar ist.

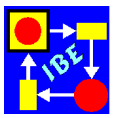
Kanäle sind verfeinerte passive Netz-Elemente, die selbst wieder aus Stellen und Kanälen bestehen. Auch hier sollten die Schnittstellen zu den aktiven Elementen im Obernetz der besseren Übersicht halber im Unternetz kenntlich sein.

Während die Bedeutung von Moduln beim Erstellen von Netzen unmittelbar einleuchtet, ist die von Kanälen, die bei der Erstellung komplexer Netze vorteilhaft eingesetzt werden, nicht ohne weiteres erkennbar. Deshalb soll im folgenden das Kanalkonzept anhand einer Analogie verdeutlicht werden:

> Stellen verhalten sich wie einzelne Leitungen zwischen Hardware-Moduln. Kanäle verhalten sich wie Röhren, in denen Leitungsbündel zusammengefaßt sind, und verbinden Module.

3.6 Mehrere Marken pro Stelle

In den ursprünglichen Petri-Netzen durfte eine Stelle höchstens eine Marke speichern. In der praktischen Anwendung zeigt sich jedoch sehr bald, daß Netze, die nur



eine Marke pro Stelle zulassen, dazu neigen, sich selbst bei kleinen Modellen stark aufzublähen. Um die Übersichtlichkeit zu steigern und um Modelle möglichst realitätsnah formulieren zu können, wurden in vielen Erweiterungen von Petri-Netzen Stellen mit beliebig vielen Marken vorgesehen. Die Maximalzahl von Marken einer Stelle kann in der Regel vom Anwender vorgegeben werden.

Darüberhinaus kann häufig eine deterministische oder zufallsgesteuerte Abarbeitungs-Reihenfolge für Marken festgelegt werden, wenn eine Transition mehrere Marken gleichzeitig feuern soll.

3.7 Statistik-Funktionen

In der Regel enthalten Petri-Netz-Tools Online-Statistikfunktionen in Form von frei definierbaren Linien- und Balkendiagrammen. Diagramme mit statistischen Informationen können häufig mit wenigen Mausklicks an die diversen Netzelemente angeschlossen werden. Wir kommen darauf in Abschnitt 4 noch zurück.

Mit anderen Statistik-Programmpaketen kann normalerweise über Files oder über dynamischen Datenaustausch (DDE) kommuniziert werden.

3.8 Frei definierbare Bilder für Netz-Elemente

Die bisher beschriebene Benutzer-Schnittstelle hat die gleichen Vor- und Nachteile, wie man sie auch bei Eingabe-Schnittstellen zahlreicher anderer Entwicklungstools findet. Die Nutzer-Schnittstelle ist einerseits wegen der geringen Zahl von Elementen leicht erlernbar. Andererseits werden selbst kleine und erst recht große Spezifikationen, auch wenn sie hierarchisch organisiert werden, sehr schnell unübersichtlich. Wer einmal eine größere graphische Spezifikation, die z.B. gemäß der Methode "Structured Analysis" abgefaßt wurde, durchgesehen oder verfaßt hat, weiß, wovon hier die Rede ist.

Da unsere Entwurfsmethode objekt-orientiert ist, eröffnet sich jedoch die Möglichkeit, den einzelnen Netz-Elementen graphische Repräsentationen zuzuordnen. In der Tat bildet ja der Anwender, wie früher erläutert, bei Verwendung von erweiterten Petri-Netzen seine Objekte auf die verschiedenen Netz-Elemente ab. Es liegt deshalb nahe, die früher beschriebenen Standard-Ikonen durch Bilder zu ersetzen, aus denen die Zuordnung eines Netz-Elementes zur physikalischen Realität unmittelbar hervorgeht.

Einige Petri-Netz-Tools bieten deshalb die Möglichkeit, die in Abb. 1 dargestellten Standard-Ikonen durch Ikonen oder Bitmaps zu ersetzen. Diese Möglichkeit ist von sehr großer Bedeutung, weil bei der Anwendung und Begutachtung von Modellen häufig nicht die letzten Implementierungsdetails interessieren, sondern die für die Bedienung bereitgestellte Nutzer-Oberfläche. Die Situation ist durchaus vergleichbar mit der bei konventionell entwickelten Programmsystemen; auch hier interessiert den Anwender meist nur die Funktionalität und die Bedienoberfläche. Das Implementierungsverfahren und der zugehörige Source-Code wird nicht benötigt. Zumindest die obersten Hierarchie-Ebenen von Modellen, die für die tägliche Arbeit verwendet werden, sollten ähnlich wie die Oberfläche anderer Anwender-Programmsysteme intuitiv verständlich und einsetzbar sein.

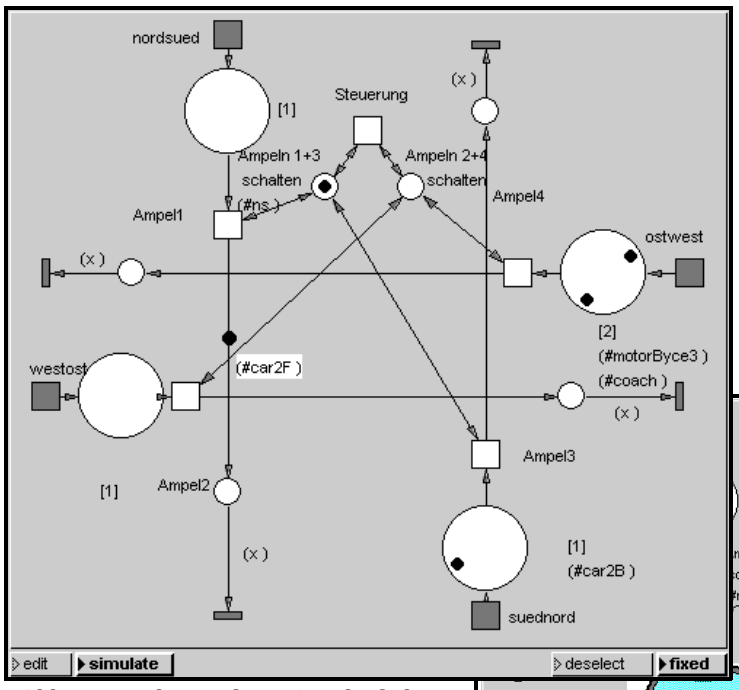


Abb. 5: Netzbeispiel mit Standard-Ikonen

Wie dramatisch man den Einblick in ein Netz durch Ikonisierung der Netz-Elemente steigern kann, erkennt man am Vergleich des in Abb. 5 und Abb. 6 dargestellten Beispiels. Es handelt sich in beiden Fällen um das gleiche Netz. Obwohl es sich um ein recht einfaches Netz handelt, muß man bei den in Abb. 5 verwendeten Standard-Ikonen schon genauer hinschauen, bis man die dargestellte Anwendung erkennt. Bei Abb. 6 ist das auf den ersten Blick möglich..

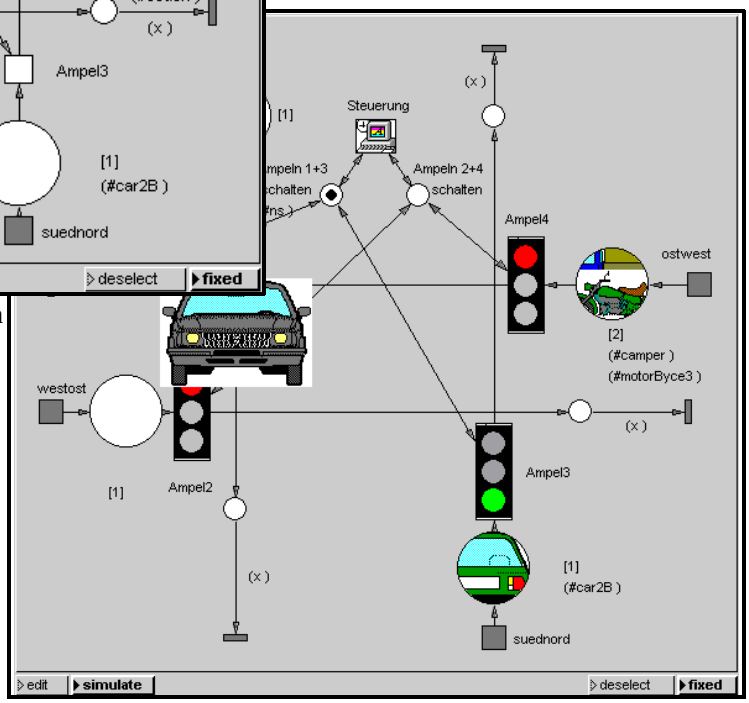


Abb. 6: Netzbeispiel aus Abb. 3 mit Anwender-Ikonen

Für die Erstellung von Benutzer-Ikonen kann eines der zahlreichen Malprogramme oder ein Ikon-Editor eingesetzt werden. Letztere sind normalerweise schon mit riesigen Ikonen-Bibliotheken ausgestattet, so daß die Ikonisierung von Netzen leicht bewerkstelligt werden kann, sofern das jeweils eingesetzte Entwicklungstool diese Möglichkeit vorsieht.

3.9 Ein vollständiges Beispiel

Wir betrachten das in Abb. 7 dargestellte einfache Förderproblem. Es besteht aus zwei Handhabungs-Robotern und einem Drehtisch mit drei Positionen. Ist der Behälter in Position1 leer, so kann der Roboter1 ein Teil hineinlegen. Ist der Behälter in Position3 gefüllt, so kann das Teil vom Roboter2 entnommen werden. Der Drehtisch benötigt eine Zeiteinheit, um die Behälter von einer Position in die nächste zu drehen.

Die Herkunft der handzu habenden Teile (Modul 'TeileQuelle') und ihre weitere Bearbeitung (Modul 'TeileBearbeitung') wird hier aus Platzgründen nicht weiter verfolgt. Wir nehmen an, daß die Teile statistisch verteilt ankommen und abgerufen werden.

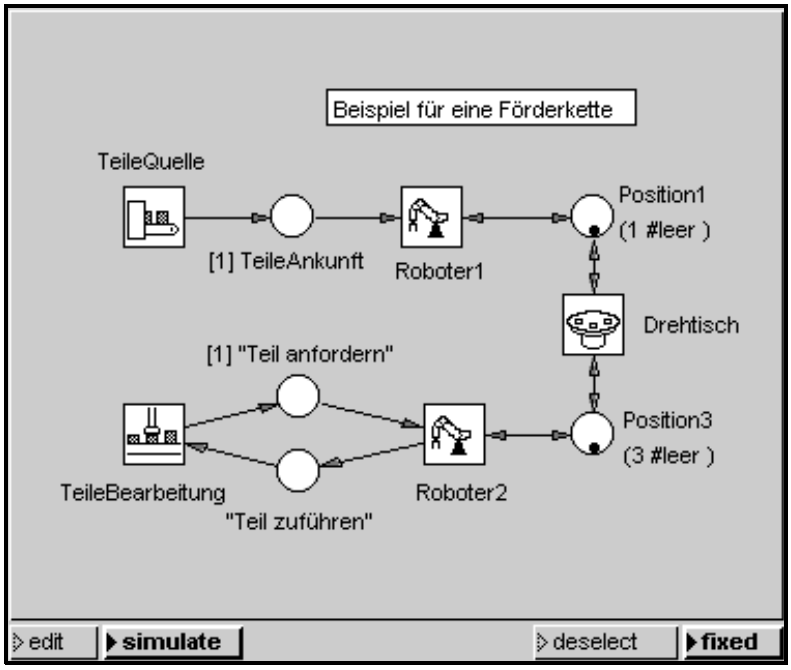


Abb. 7: Oberste Hierarchie-Ebene eines einfachen Fördermodells.

In Abb. 8 und Abb. 9 sind die zwei Module 'TeileQuelle' und 'TeileBearbeitung' dargestellt. Ersichtlich werden die Zeitpunkte, zu denen jeweils ein Teil angeliefert oder verbraucht wird, als zufallsverteilt angenommen. Die Zuführung von Teilen geschieht dabei wie folgt (Abb. 8): Die in der Stelle befindliche Marke wird solange verzögert, wie der zufällige Wert der Zeitverzögerung angibt. Dann feuert die Transition und liefert an jede der beiden mit ihr verbundenen Stellen jeweils eine Marke ab. Die eine wird erneut für die Einplanung der nächsten Teile-Zuführung verwendet. Die Marke, welche zur Stelle 'TeileAnkunft' hinläuft, stellt das von außen zugeführte Teil dar.

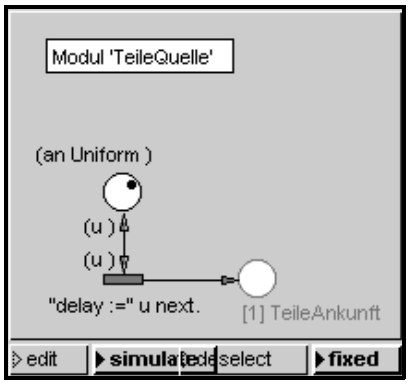


Abb. 8: Erzeugung von Teilen

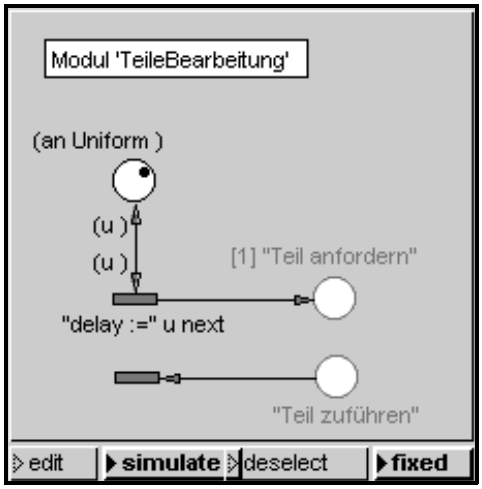


Abb. 9: Verbrauchen von Teilen

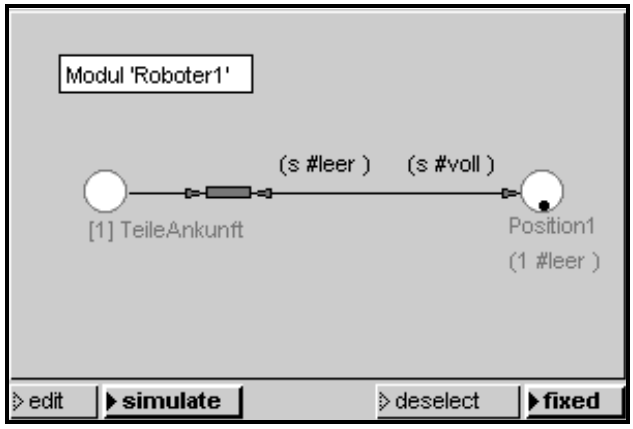


Abb. 10: Ablegung eines Teils in einen Behälter

In Abb. 10 und Abb. 11 ist die Arbeit der Handhabungs-Roboter modelliert. Wir betrachten wieder nur die Zuführung von Teilen. Im Behälter an 'Position1' liegt eine Marke, deren Attributwerte angeben, welcher der drei Behälter des Drehtisches gerade positioniert ist und daß dieser Behälter leer ist. Die Transition 'Roboter1' kann nur feuern, wenn außer dieser Marke auch eine Marke (ein Teil) in der Stelle 'Teileankunft' vorliegt. Feuert die Transition, so

zieht sie beide Marken ein und legt eine Marke mit dem Attributwert 'voll' (ein Teil) im Behälter an 'Position1' ab.

Nicht viel komplizierter ist das Modell des in Abb. 12 dargestellten Drehtisches. Wir sehen hier zunächst etwas schwächer gezeichnet die Ein-/Ausgabe-Schnittstellen 'Position1' und 'Position3' aus der darüberliegenden Ebene (Abb. 7) mit ihrer Anfangsbelegung und die Behälter-'Position2', die den beiden Robotern nicht zugänglich ist. Das dargestellte Teilnetz rotiert die aktuell vorliegenden Attribute bzgl. der 3 angegebenen Positionen.

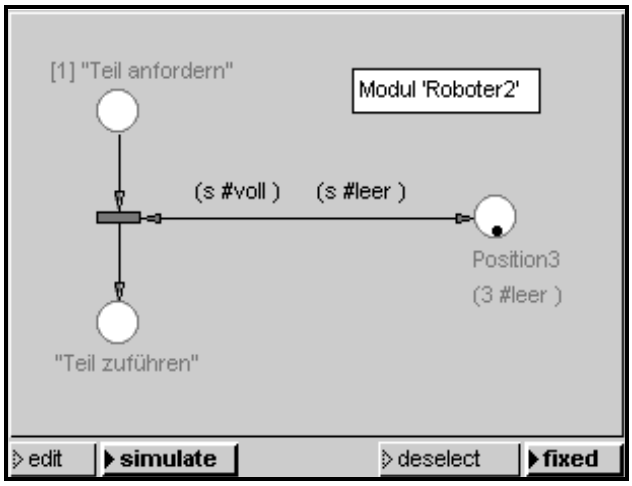


Abb. 11: Herausnehmen eines Teils aus einem Behälter.

Da der Drehtisch die Positionen in einer Zeiteinheit weiterschaltet, feuert die Transition tr4 einmal pro Zeiteinheit und legt auf den Stellen st1 bis st4 jeweils eine Marke ab. Die in st1 abgelegte Marke plant das Feuern zum nächsten Zeittakt ein. Bis dahin feuern die drei Transitionen tr1 bis tr3, da in allen Eingangsstellen dieser Transitionen jeweils eine Marke liegt, und transportieren die in den drei Positionen liegenden Marken in die jeweils nächste Position.

Das vorliegende Beispiel zeigt einerseits, wie einfach und durchsichtig sich ereignisorientierte, parallele, diskrete Prozesse mit Petri-Netzen darstellen lassen. Andererseits läßt sich unschwer erkennen, daß die verwendeten Modellierungsmittel allgemein einsetzbar sind.

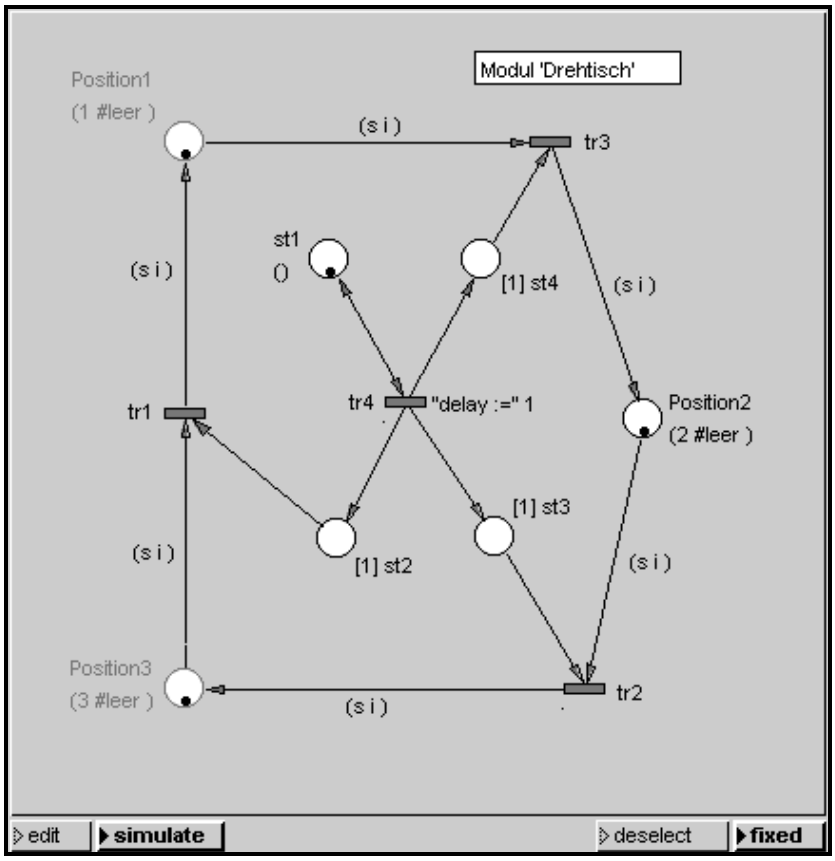
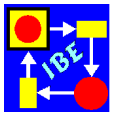


Abb. 12: Modell eines Drehtisches mit drei Behältern



4. Die Simulationsphase

In der Simulationsphase kann ein korrektes Netz, auch wenn es noch unvollständig ist, animiert und dabei sein dynamisches Verhalten analysiert werden. Unter 'korrekt' verstehen wir hierbei, daß alle statisch möglichen Eingabeprüfungen (Prüfung auf zulässige Netz-Konstruktionen, syntaktische und semantische Prüfungen des inskribierten Codes) erfolgreich durchgeführt wurden.

4.1 Simulations-Modi und Debugging-Hilfen

Die gängigen Petri-Netz-Tools bieten in der Regel zwei Modi zur Ausführung von Netzen.

Animations-Modus

Das Austesten von Netzen kann effizient gestaltet werden, wenn man den Ablauf des Netzes mit einstellbarer Geschwindigkeit beobachten, bei Bedarf anhalten, ändern und danach wieder fortsetzen kann. Bei der Animation sieht man wie die Marken bzw. vom Benutzer definierte anwendungs-orientierte Ikonen von Stelle zu Stelle fließen. Zwischen den einzelnen, in verschiedenen Fenstern dargestellten Teilnetzen wird dabei automatisch umgeschaltet.

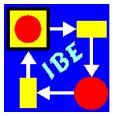
Für die Analyse kritischer Netzteile, z.B. Hemmungen in Teilprozessen, die den Gesamtprozeß verzögern, kann bei vielen Tools auf schrittweise Abarbeitung umgeschaltet werden. Dabei wird nach jedem Simulationsschritt angehalten und man kann sich den aktuellen Zustand des Netzes (Belegung der Stellen, Werte von Attributen und Variablen, usw.) ansehen. Manche Entwicklungstools bieten auch die Möglichkeit, die Simulation schrittweise rückwärts laufen zu lassen.

Hintergrund-Modus

Hat man die prinzipielle Funktionsweise eines Netzes verifiziert, so kann man es durch Variieren von Parametern optimieren und dabei ggf. in seiner Struktur verändern. Dazu ist es nützlich, wenn man das Netz im sog. Hintergrund-Modus mit optimaler Geschwindigkeit ohne Animation ablaufen lässt und dabei statistische Daten sammelt. Diese werden dem Anwender in tabellarischer und/oder graphischer Form aufbereitet, so daß er erkennen kann, ob mit dem entwickelten Modell seine Anforderungen erfüllt werden können.

In beiden Simulations-Modi können Unterbrechungspunkte definiert werden. Die meisten Tools ermöglichen die Definition von Knoten-Unterbrechungspunkten. Wird während der Simulation ein Knoten mit einem aktiven Unterbrechungspunkt erreicht, so wird der Simulationslauf angehalten. Das Fenster, in dem der unterbrechende Knoten liegt, wird geöffnet und aktiviert.

Bei Petri-Netz-Tools mit Zeitmodellierung sind auch zeitliche Unterbrechungspunkte möglich. Dabei wird die Simulation unterbrochen, sobald ein vorgegebener Simulations-Zeitpunkt erreicht ist. Zeitliche Unterbrechungspunkte werden z.B. zur Beurteilung von Netzen eingesetzt, die innerhalb einer bestimmten Zeitspanne abgearbeitet werden müssen.



4.2 Wichtige Grundbegriffe

Auch wenn der Anwender nicht viele Details über den Aufbau von Petri-Netz-Simulatoren kennen muß, so ist ein bestimmtes Hintergrundwissen für das Verständnis der Verhaltensweise von Modellen während der Simulation nützlich. Wir beschreiben deshalb kurz einige wichtige Begriffe und was sich dahinter verbirgt.

Simulations-Zeit

Die ursprüngliche Definition der Petri-Netze schreibt das zeitlose Schalten von Transitionen vor. Sobald also die Bedingungen für das Feuern einer Transition erfüllt sind, soll der Markierungswechsel zeitlos stattfinden. Die zeitliche Abfolge von Ereignissen spielt bei klassischen Petri-Netzen nur insofern eine Rolle, als Ereignisse in einer bestimmten Reihenfolge stattfinden und das Kausalitätsprinzip gilt.

Es gibt zahlreiche Arbeiten über Verfahren zur Einführung des Zeitbegriffs in Petri-Netze (siehe z.B. [4] und die dort angegebene Literatur). Sehr häufig wird das Modell der sog. Aktivierungszeit verwendet. Jeder Transition kann eine Aktivierungszeit zugeordnet werden, die angibt, wieviele Simulator-Zeiteinheiten zwischen dem Aktivieren einer Transition und ihrem frühesten Schalten verstreichen sollen. Bei dem Modell in Abschnitt 3.9 wurden Aktivierungszeiten mehrfach eingesetzt.

Dabei spielt es bei der Simulation normalerweise keine Rolle, wie sich die Simulator-Zeiteinheit zu physikalischen Zeiteinheiten verhält, da nur das relative Verhalten der Abläufe interessiert. Deshalb lassen sich auf diese Weise auch sehr schnelle oder sehr langsamer Prozesse simulieren.

Ereignisse

Ein Ereignis tritt ein, wenn die Bedingungen für das Feuern einer Transition erfüllt sind. Es besteht aus einer Transition, deren Input-Marken und ggf. den aktuellen Werten von weiteren Simulator-Daten, die beim Feuern der Transition zu berücksichtigen sind. Der Simulator speichert alle Ereignisse, die entweder bereits vorüber oder zu einem späteren Zeitpunkt eingeplant sind, in einer Ereignis-Liste. Diese Ereignis-Liste ist eine zeitlich sortierte Auflistung aller Ereignisse.

Jeder Eintrag in die Ereignis-Liste besteht aus einer Zeitangabe und einer Menge von Ereignissen, die zu diesem Zeitpunkt stattfinden sollen.

Simulations-Algorithmus

Die oben beschriebene Unabhängigkeit von der realen Zeit ist auch der Grund, warum sich der Zeitbedarf für die Durchführung einer Simulation in vernünftigen Grenzen hält. Der Anwender kann zwar einer Simulator-Zeiteinheit gedanklich eine dem modellierten System angemessene physikalische Zeiteinheit zuordnen, doch hat das natürlich keinen Einfluß auf die zeitliche Durchführung der Simulation.

Alle Transitionen, die zu einem bestimmten Zeitpunkt feuerbereit sind, feuern parallel. Der Simulator arbeitet die gleichzeitig aktiven Feuerungen sequentiell ab. Ist keine feuerbare Transition mehr vorhanden, so wird die Simulationszeit auf den nächsten Eintrag in der Ereignis-Liste gesetzt.

Der Simulationslauf ist beendet, sobald keine Transition mehr gefeuert werden kann, d.h. sobald die Ereignis-Liste leer ist.

Simulations-Protokoll

Die zuletzt gefeuerten Transitionen werden in einer Liste notiert, die beim Rückwärtslauf des Simulators ausgewertet wird. Die Länge der Liste und damit auch die Zahl der möglichen Rückwärts-Schritte kann bei vielen Tools vom Anwender festgelegt werden.

4.3 Diagramme.

Diagramme sind nur in Netzen mit Zeitmodellierung sinnvoll. Sie ermöglichen die graphische Darstellung von während der Simulation anfallenden Daten. In der Regel werden Balken- und Linien-Diagramme geboten, die an Stellen und Konnektoren gebunden werden (siehe Abb. 13).

Die Funktion, welche die anzuzeigenden Daten errechnet, wird jedesmal aufgerufen, bevor sich der Zustand des zugeordneten Netzelements ändert. Das Element selbst und die Zeitspanne, die seit dem letzten Aufruf vergangen ist, liefern die auszuwertenden Input-Parameter für die Berechnung der Diagrammwerte.

Der Anwender kann eigene Funktionen einbringen, um die sich während der Simulation ändernden Merkmale darzustellen. Standardmäßig wird als Merkmal die Anzahl der Marken oder der Wert eines Marken-Attributs verwendet.

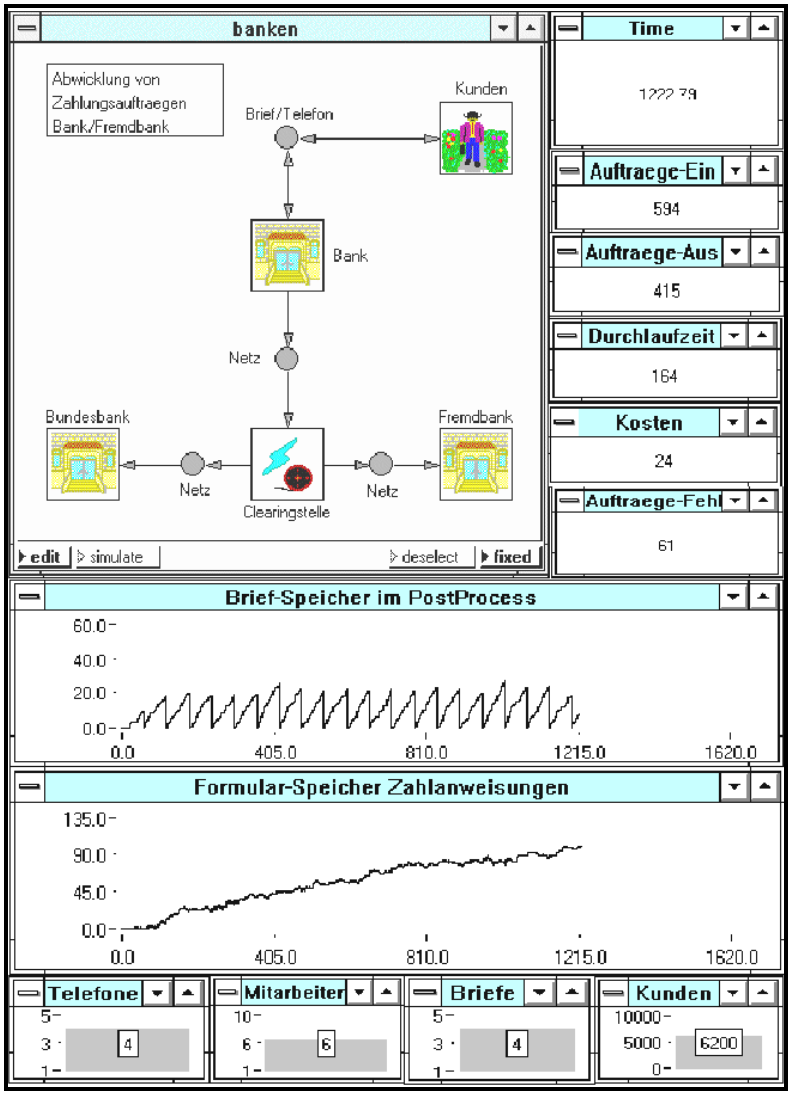


Abb. 13: Oberste Ebene eines Banken-Modells mit Linien-Diagrammen und Balken-Schiebereglern.

Balken-Diagramme (Histogramme)

Bei Balken-Diagrammen wird zu jedem Intervall der X-Achse die Dauer in Y-Richtung angezeigt, mit der die durch das X-Intervall definierte Anzahl von Marken auf dem Netz-Element vorkam.

Bei Stellen wird vor der Zustandänderung des entsprechenden Netz-Elements berechnet, wie lange sich eine bestimmte Anzahl von Marken auf der Stelle befunden hat. In X-Richtung wird dabei die Anzahl auf der Stelle befindlicher Marken, in Y-

Richtung die Dauer angezeigt, während der diese Anzahl von Marken in der Stelle gespeichert wurde. Im Fall eines Warteschlangen-Modells kann damit z.B. festgestellt werden, welche Warteschlangenlängen vorkommen und wie ihr gegenseitiges Verhältnis ist (siehe Abb.17).

Im Fall eines Konnektors kann zu jedem Intervall der X-Achse angezeigt werden, wie oft eine Marke, deren erster Attributwert im X-Intervall liegt, über den Konnektor geflossen ist.

Linien-Diagramme

Im Fall von Linien-Diagrammen wird ein vom Anwender festzulegendes Merkmal als Funktion der Zeit aufgetragen und durch gerade Linien miteinander verbunden.

Im Fall einer Stelle wird als Merkmal häufig die Gesamtzahl der Marken, die sich auf der Stelle befinden, verwendet. Als Default-Merkmal bei Konnektoren, dient wieder der Wert des ersten Attributs der durch den Konnektor fließenden Marken.

4.4 Daten-Ein/Ausgabe während der Simulation

Die konventionelle Daten-Ein/Ausgabe über Ein-/Ausgabe-Fenster kann über Inskriptionen relativ einfach bewerkstelligt werden. Wird Smalltalk-80 von ParkPlace Systems verwendet, so kann die Ein/Ausgabe mit vorgefertigten Klassen und Methoden, die für alle Rechner-Plattformen funktionell identisch zur Verfügung stehen, bequem und effizient bewerkstelligt werden.. Als Beispiel betrachten wir eine Ja-Nein-Abfrage über ein Eingabefenster. Mit der Anweisung:

```
returnValue := DialogView
                    confirm: 'Wollen Sie die Simulation fortsetzen?'
                    initialAnswer: true.
```

wird das in Abb. 14 dargestellte Eingabefenster angezeigt. Die Frage wird durch Anklicken eines der Knöpfe beantwortet und weist der Variablen returnValue das gewählte boolsche Ergebnis zu.

Abb. 14: Beispiel eines Eingabefensters.

Neben der konventionellen Dateneingabe wird gelegentlich auch die graphische Dateneingabe über Balken-Schieberegler (Bar Gauge) angeboten. In dem in Abb. 13 dargestellten Netz, das die oberste Ebene eines Bankenmodells darstellt, befinden sich unterhalb der Liniendiagramme vier Balken-Schieberegler, mit denen die Simulations-Parameter vom Anwender mit einem Zeigegerät (z.B. Maus) variiert werden können.

4.5 Simulations-Beispiele

Wir betrachten zunächst das in Abschnitt 3.9 dargestellte Beispiel und stellen die Frage, ob die 'TeileBearbeitung' unter den modellierten Voraussetzungen ausreichend mit Teilen versorgt wird. Dazu können wir z.B. ein Histogramm an die Stelle 'Teil anfordern' koppeln und danach den Simulationslauf starten. Das Ergebnis sehen wir in Abb. 15. In der Stelle 'Teile anfordern' ist die meiste Zeit keine Anforderung (Marke) gespeichert.; die 'Teile-Bearbeitung' wird also ausreichend mit Teilen versorgt.

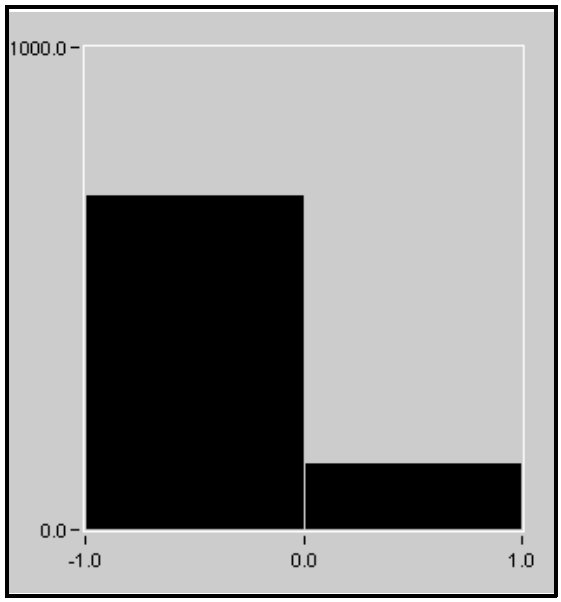


Abb. 15: Histogramm für 'Teil anfordern'

In Abb. 16 ist ein einfaches Warteschlangenproblem mit einem Bedienplatz dargestellt. Nehmen wir an, daß eine Simulator-Zeiteinheit einer Minute entspricht, so soll im Mittel alle 5 Minuten ein Kunde eintreffen. Als mittlere Bearbeitungsdauer wurde 3.33 Minuten angenommen; der Bediener ist also im Mittel zu 66% ausgelastet. Die Ankunfts- und Bearbeitungszeiten seien exponentiell verteilt.

Wir interessieren uns dafür, ob im Normalfall zumutbare Warteschlangenlängen auftreten und schließen deshalb an die Warteschlange ein Histogramm an. Das Ergebnis sehen wir in Abb. 17. Die Ordinate zeigt die Anteile der Gesamtzeit an, in der die einzelnen Warteschlangenlängen auftreten.

5. Die Ausführungsphase

An die Erstellung und das Austesten einer Spezifikation schließt sich normalerweise die Implementierung an. Diese besteht bei kommerziellen Anwendungen in der Anpassung der Abläufe (Workflows) und sonstiger organisatorischer Maßnahmen an das entwickelte Modell. Das Entwicklungssystem kann hierbei nur in den Fällen helfen, in denen sich herausstellt, daß bestimmte Modell-Eigenschaften nicht realisierbar oder durchsetzbar sind und man deshalb Abweichungen vom ursprünglichen Konzept studieren muß. Solche Änderungen der Spezifikation sind aber Iterationen bei der Modellentwicklung gleichzusetzen und deshalb den beiden schon beschriebenen Phasen zuzuordnen.

Bei technischen Anwendungen besteht die Implementierung in der Erstellung bzw. Änderung von technischen Anlagen und den dazugehörigen Automatisierungsprogrammen. Auch hier können Modell-Iterationen aus Machbarkeitsgründen erforderlich sein. Die Netz-Spezifikation selbst kann aber zusätzlich zur automatischen Generierung der Automatisierungsprogramme eingesetzt werden.

Da mit dem Petri-Netz nur die diskreten Schritte innerhalb der Abläufe modelliert und

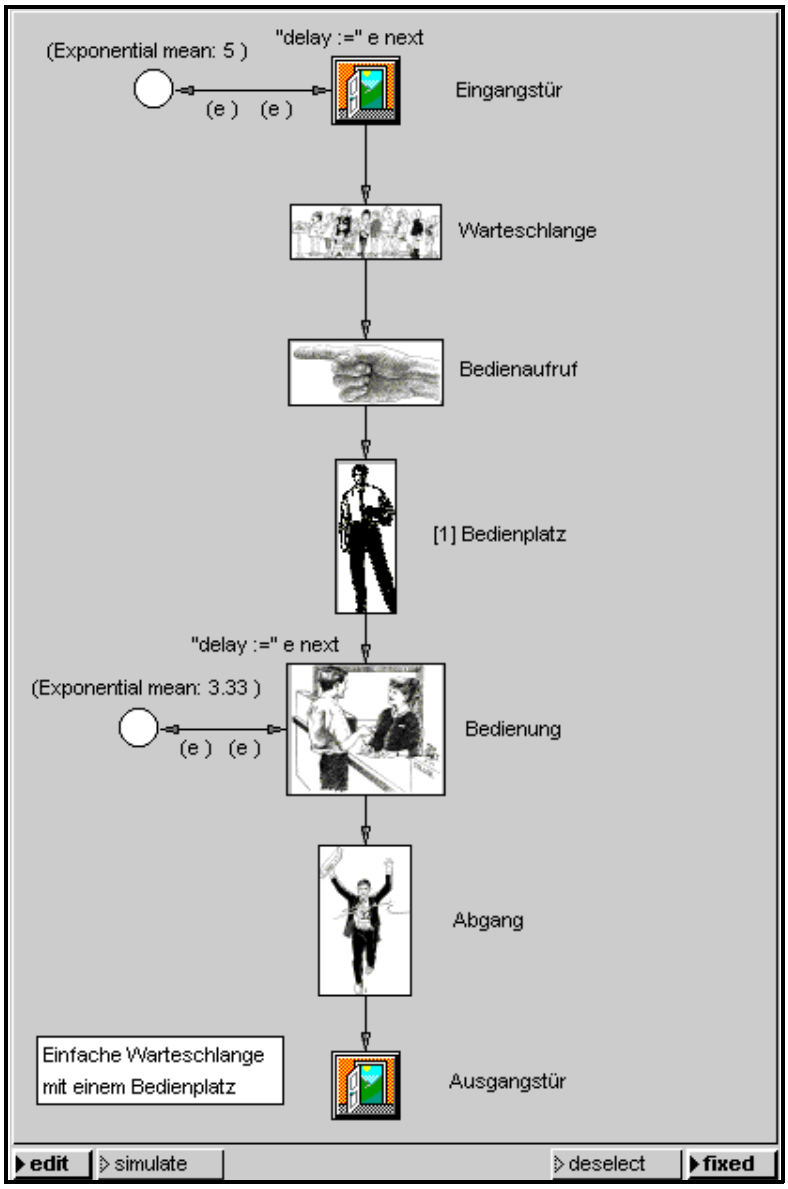


Abb. 16: Ein einfaches Warteschlangenmodell

dabei die physikalischen Vorgänge nur funktionell eingekoppelt werden, ist die Spezifikation um Programmteile zur Kommunikation mit der physikalischen Umwelt (z.B. I/O-Handler und Treiber für die Prozeß-Peripherie) zu vervollständigen. Bestimmte Teile der Spezifikation, die zur Simulation des Umwelt-Verhaltens erforderlich waren (z.B. die Verwendung von Verteilungsfunktionen zur Erzeugung stochastischer Ereignisse) sind durch Anweisungen zu ersetzen, welche den Eintritt eines Ereignisses im technischen Prozeß (z.B. die Ankunft eines Bauteils bzw. das Auftreten eines bestimmten Interrupts) feststellen.

Es gibt im wesentlichen drei Methoden, um von der Netz-Darstellung zu einem ausführbaren Programm zu kommen:

Generierung von Programmen in Hochsprachen

Das ist die Methode, die am häufigsten eingesetzt wird. Aus der Netz-Darstellung werden Programme in einer Hochspache (meist C) erzeugt. Sie sind bei manchen Entwicklungstools noch manuell nachzubearbeiten und zumindest um die externen Funktionen für die Prozeß-Ein/Ausgabe zu ergänzen.

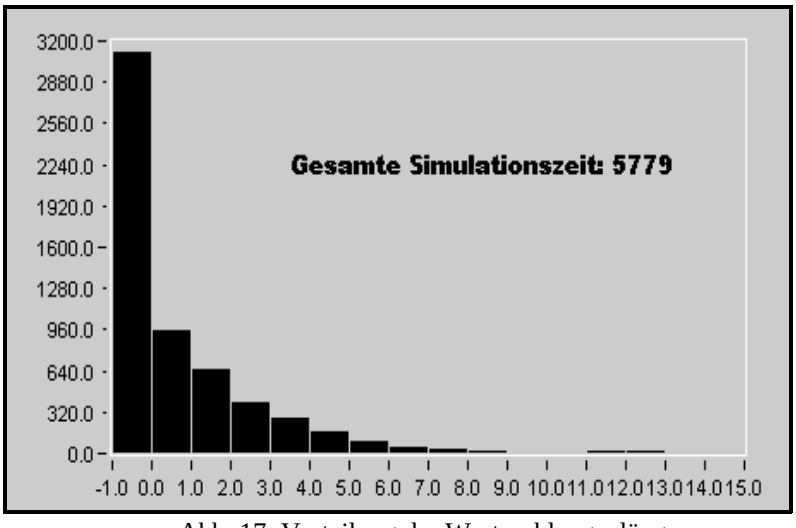
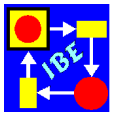


Abb. 17: Verteilung der Warteschlangenlängen

Die Methode hat den Vorteil, daß sie für die meisten Rechner anwendbar ist. Sie hat aber auch erwähnenswerte Nachteile. Falls das Programm



nicht vollständig aus dem Entwicklungs-Tool erzeugt werden kann oder wenn bei vollständiger Generierung direkt mit dem generierten Programm weitergearbeitet wird, so zeigt die Erfahrung mit anderen CASE-Tools, daß sich Spezifikation und Programm im Lauf der Zeit auseinander leben und später kostspielige Reverse-Engineering-Arbeiten zur Aktualisierung der Spezifikation notwendig werden. Außerdem muß man damit rechnen, daß weiterer Adaptionsbedarf erforderlich wird, weil die Kompiliersysteme für die unterschiedlichen Ziel-Plattformen unterschiedliche Dialekte und Bibliotheken implementieren.

Verwendung einer Petri-Netz-Maschine

Die zu favorisierende Methode ist ebenfalls für die meisten Plattformen anwendbar. Sie geht von einer virtuellen Petri-Netz-Maschine PNM aus. Programme für die PNM, die für jede Ziel-Plattform implementiert werden muß, werden direkt und vollständig aus dem Netz erzeugt. In unserem Fall liegt die PNM in ANSI-C vor und kann deshalb leicht auf unterschiedlichen Plattformen bereitgestellt werden.

Damit die Methode funktioniert, muß das Entwicklungs-Tool Eigenschaften aufweisen, mit denen externe Funktionen (wie z.B. Bearbeitungsprogramme für Prozeßgeräte) identifiziert und während der Simulation durch Simulations-Routinen ersetzt werden können. Damit das Programm auf der Ziel-Plattform läuft, ist die virtuelle Maschine um diese externen Funktionen zu erweitern.

Das Verfahren hat gegenüber der direkten Übersetzung in Quellcode keine Effizienz-Nachteile, weil auch bei der direkten Generierung große Teile des generierten Codes Daten sind, die interpretativ abgearbeitet werden. Es wird bevorzugt bei der Entwicklung von eingebetteten Systemen und von Controllern eingesetzt.

Ausführung eines Netzes mit dem Petri-Netz-Tool

Die beste und eleganteste Möglichkeit, das Netz auszuführen, wäre gegeben, wenn das Petri-Netz-Tool mit gewissen Änderungen und Erweiterungen (wie Anbindung an die physikalische Zeit) auch für die Programm-Ausführung verwendet werden könnte. Die Inskriptionen, in denen Eigenschaften der physikalischen Umwelt parametriert sind, wären, ähnlich wie im Fall der PNM, durch einen zweiten Satz von Inskriptionen zu ersetzen, in denen z.B. über direkte Aufrufe externer Prozeduren die Verbindungen zur Umwelt in das Netz eingeklinkt werden.

Diese Möglichkeit wurde in der Vergangenheit aus Effizienzgründen verworfen. Mit dem Erscheinen immer schnellerer Prozessoren kann man heute daran denken, das Entwicklungs-Tool auch für die dritte Entwicklungsphase zu verwenden. Damit wäre das Ziel einer einheitlichen Beschreibung und Vorgehensweise bei der Systementwicklung vollständig erreicht.

6. Einsatz des Verfahrens in verschiedenen Anwendungsgebieten

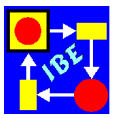
Unter Verwendung des Petri-Netz-Entwicklungssystems PACE [2] ist das beschriebene Verfahren in den vergangenen Jahren in vielen Bereichen der Industrie, in Behörden, in der Forschung und in der Lehre erfolgreich eingesetzt worden. Im folgenden wird eine kleine Auswahl der bisher durchgeführten Anwendungen aufgezählt, die das weite Anwendungsspektrum des Verfahrens verdeutlichen sollen.

Industrieanwendungen

- Spezifikation, Simulation und Optimierung von Automatisierungsprojekten bei Zulieferanten der Auto- und Papier-Industrie
- Spezifikation, Simulation und Optimierung von Montagestraßen
- Simulation von Kommunikationsnetzen
- Simulation von Automaten
- Industrielle Workflow-Analysis
- Modellierung und Simulation von Fabrikbetrieben
- Programmierung von Steuerungen
- Analyse und Optimierung von Produktionsstraßen für Relais und Optosensoren
- Engineering und Beratung gemäß DIN/ISO 9000
- Spezifikation und Simulation des CIM-Produktionskonzepts
- Analyse und Optimierung bereits erstellter Transportanlagen
- Logistik-Simulation für Militärflughäfen
- Simulation von Gruppen- und Linienfertigung
- Simulation von Lackieranlagen
- Spezifikation von CIMOSA-Business-Strukturen
- Planung flexibler Fertigungsanlagen in der Automobilindustrie
- SPS-Programmierung bei Fräsmaschinen
- Programmierung von Fahrkarten-Automaten
- Minimierung des Ausschusses bei der Produktion von Getrieben
- Optimierung einer Prozessorstruktur für die Online-Bildverarbeitung
- Simulation einer Aluminium-Fabrik mit Entwicklung eines Automatisierungsprogramms unter Verwendung des PACE-Simulators
- Modellierung, Simulation und Optimierung einer Fertigungsstraße für Sonnenkollektoren.

Verwaltung

- Konzept-, Überprüfung und Verfeinerung der Gepäckverteilung in einem großen europäischen Flughafen (Zürich Kloten)
- Analyse und Optimierung des Material- und Informationsflusses durch verschiedene Abteilungen eines großen Unternehmens
- Vorgangsbearbeitung und Ablaufsimulation



Kommerzielle Anwendungen

- Entwurf und Programmierung von Prototypen für die automatische Geldausgabe bei Banken
- Erstellung von Netzwerkprotokollen
- Modellierung, Simulation und Resultatsanalyse der Abwicklung verschiedener Aufträge bei Banken, Versicherungen und öffentlichen Diensten.

Forschung, Entwicklung und Ausbildung

Zahlreiche Hochschulinstitute, Fachhochschulen, Forschungseinrichtungen und Entwicklungsabteilungen namhafter Unternehmen verwenden das Verfahren im Unterricht und für die unterschiedlichsten Forschungsprogramme, wie z.B.

- Untersuchung von Mensch-Maschine-Schnittstellen
- Modellierung von Fertigungszellen
- Analyse bestehender und Entwicklung neuer Produktionsmethoden

7. Zusammenfassung und Ausblick

Wie die vorangegangenen Abschnitte deutlich gemacht haben, kann heute unter Verwendung von Petri-Netz-Entwicklungssystemen die Entwicklung von diskreten Ablaufsystemen einheitlich von der Spezifikation über die Simulation bis zum fertigen Automatisierungsprogramm ausgeführt werden. Die Funktionen, die in den ersten beiden Entwicklungsphasen benötigt werden, sind sachgerecht und effizient verfügbar und bedürfen kaum einer Nachbearbeitung. Wie die Erfahrung zeigt, decken sie die Anforderungen ab, die normalerweise bei der Analyse und Optimierung von verwaltungstechnischen und kommerziellen Aufgaben auftreten.

Auch die dritte Phase ist, verglichen mit anderen heute üblichen Verfahren, schon sehr weit fortgeschritten. Andere Verfahren setzen in der Regel nur statisch geprüfte Spezifikationen in Source-Code von Hochsprachen um. Die generierten Programme sind noch manuell nachzubearbeiten, weil häufig nur die Ablaufstruktur generiert wird und weil der dynamische Test erst nach Installation der Programme erfolgen kann.

Die von uns verwendete Petri-Netz-Maschine PNM ist auch in Zukunft erforderlich, wenn entweder sehr schnelle Automatisierungsprogramme benötigt werden oder wenn Entwicklungs- und Ziel-Plattform verschieden sind und auf dem Zielsystem kein Smalltalk-System verfügbar ist. Sie hat den Vorteil, daß die Installation der generierten Programme einfach und mit geringem Aufwand möglich ist. Es bietet sich an, zu untersuchen, ob die PNM als Teil eines Prozessorkerns realisiert werden kann.

Dagegen ist die direkte Programmausführung unter Verwendung eines Petri-Netz-Tools meines Wissens bisher noch nirgendwo realisiert worden.

Literaturangaben

[1] Carl Adam Petri: "Kommunikation mit Automaten", Dissertation, erschienen in:

Schriften des Rheinisch-Westfälischen Instituts für instrumentelle Mathematik an der Universität Bonn, Bonn, 1962.

[2] PACE Benutzer-Handbuch, Version 2.3, 1996
IBE, Postfach 1142, D-85623 Glonn

[3] DIN EN ISO 9001 - 9003 : 1994, DIN Deutsches Institut für Normung e.V., Beuth Verlag GmbH, Berlin

[4] Bernd Baumgarten: "Petri-Netze: Grundlagen und Anwendungen", BI-Wiss-Verlag, Mannheim Wien Zürich, ISBN 3-411-14291-X, 1990

[5] Dirk Abel: "Petri-Netze für Ingenieure, Modellbildung und Analyse diskret gesteuerter Systeme", Springer Verlag, Berlin Heidelberg New York, ISBN 3-540-51814-2, 1990

[6] Wolfgang Reisig: "Petrinetze - Eine Einführung", Springer Verlag, Berlin Heidelberg New York, 2. Auflage, ISBN 3-540-16622-X, 1991

[7] VisualWorks, User's Guide, Part Number: DS 10005002, 1994
ParkPlace Systems, Inc., 999 E. Arques Avenue, Sunnyvale, CA 94086-4593

[8] Matthias C. Bücker, Joachim Geidel, Matthias Lachmann: "Programmieren in Smalltalk mit VisualWorks, Smalltalk - nicht nur für Anfänger", Springer Verlag, Berlin Heidelberg New York, 2. Auflage, ISBN 3-540-58813-2, 1995