Zeitschrift: Bulletin des Schweizerischen Elektrotechnischen Vereins, des Verbandes Schweizerischer Elektrizitätsunternehmen = Bulletin de l'Association suisse des électriciens, de l'Association des entreprises électriques suisses

Herausgeber: Schweizerischer Elektrotechnischer Verein ; Verband Schweizerischer Elektrizitätsunternehmen

Band: 79 (1988)

Heft: 17

Artikel: Ein interaktives Simulations- und Programmgenerations-Werkzeug für erweiterte Petri-Netze

Autor: Dähler, J.

DOI: https://doi.org/10.5169/seals-904072

Ein interaktives Simulations- und Programmgenerations-Werkzeug für erweiterte Petri-Netze

J. Dähler

Petri-Netze eignen sich für die Beschreibung von diskreten, ereignisorientierten, verteilten Systemen. Sie können als Entwurfs- und Spezifikationsmodell sowie als Simulationswerkzeug für Leistungsuntersuchungen verwendet werden. In diesem Bericht wird ein Werkzeug beschrieben, das am Institut für Elektronik an der ETH Zürich entwickelt wurde. Es unterstützt das graphische Editieren, das Simulieren, und das automatische Erzeugen eines C-Programmes von erweiterten Petri-Netzen.

Les réseaux de Pétri se prêtent bien à la description de systèmes répartis, discrets et orientés événements. Ils peuvent s'utiliser comme modèle de projet et de spécification ainsi que comme outil de simulation. Dans ce rapport est décrit un outil qui a été développé à l'EPF Zurich. Il assiste l'édition graphique, la simulation, la génération automatique de programmes C de réseaux de Pétri élargis.

Adresse des Autors
Jacques Dähler, Dipl. El.-Ing. ETH, Institut für Elektronik, 8092 Zürich.

Das in diesem Bericht beschriebene Werkzeug unterstützt das graphische Editieren, das Simulieren, und das automatische Erzeugen eines C-Programmes von erweiterten Petri-Netzen. Die Netzelemente werden mit der Programmiersprache Smalltalk-80 [1] beschriftet. Smalltalk ist eine objektorientierte Programmiersprache, für die eine mächtige Entwicklungsumgebung existiert. Diese stellt Hilfsmittel für die Konstruktion window-orientierter, interaktiver Applikationen zur Verfügung. Leistungsfähige Implementationen dieser Sprache sind nun für praktisch alle modernen Arbeitsstationen erhältlich. Das hier beschriebene Werkzeug kann auf diese Rechner mit wenig Aufwand portiert werden.

Die Petri-Netze eignen sich für die Beschreibung von diskreten, ereignisorientierten, verteilten Systemen. Sie können als Entwurfs- und Spezifikationsmodell sowie als Simulationswerkzeug für Leistungsuntersuchungen verwendet werden. Im ersten Fall kann für die in Software zu realisierenden Teile des Systems ein Programm automatisch erzeugt werden, im zweiten Fall können die interessierenden Systemgrössen mit Hilfe der Statistikfunktionen untersucht werden. In diesem Bericht wird an Hand eines einfachen Simulationsbeispiels eine Übersicht über das Werkzeug und einige seiner Funktionen gegeben.

Erweiterte Petri-Netze

Die hier verwendeten erweiterten Petri-Netze bestehen wie die einfachen Petri-Netze[1] [2] aus S- und T-Elementen (Stellen und Transitionen), die als Kreise und Quadrate dargestellt werden und mit Pfeilen verbunden sind. Die Stellen enthalten durch schwarze Punkte dargestellte Marken, die beim Feuern der Transitionen über die Pfeile fliessen. Eine Transition entfernt (konsumiert) beim Feuern von allen Inputstellen eine Marke und legt auf allen Outputstellen eine neue ab.

Hierarchie

Um komplexe Systeme übersichtlich zu modellieren, können die erweiterten Petri-Netze hierarchisch strukturiert werden. Dies geschieht durch Verfeinern (Differenzieren) der T-Elemente. Verfeinerte T-Elemente – man nennt sie Module – bestehen aus einem Netz von weiteren S- und T-Elementen. Die Figur 1 zeigt ein einfaches Beispiel einer Verfeinerung.

Smalltalk-Objekte als Marken-Attribute

Die Marken können mit *Attributen* behaftet sein. Jedes Attribut ist ein Smalltalk-Objekt. Die Programmiersprache Smalltalk arbeitet mit *Objekten* als Einheiten eines ablaufenden Programmes. Ein Objekt besteht aus einer Datenstruktur und Zugriffsprozeduren. Diese gehören zur Definition der Klasse des Objekts und erscheinen im Modell nicht. Durch Senden einer Meldung an das Objekt können seine Daten mit Hilfe der Prozeduren verändert und abgefragt werden. Jedes Objekt gehört zu einer *Klasse*, man nennt es auch eine *Instanz* seiner Klasse. Die Klasse definiert, wie sich ein Objekt beim Empfangen einer Meldung verhält.

Die *Beschriftungen* der Netz-Elemente bestehen aus Smalltalk-Statements. Diejenigen der Stellen erzeugen die Anfangsmarkierung. Diejenigen der Pfeile bestimmen, welche Marken über die Pfeile fliessen können. Die

[1] Die einfachen Petri-Netze werden auch im Beitrag von *H.P. Gisiger, A. Kündig* auf den Seiten 1026...1034 dieser Nummer beschrieben.

Beschriftungen der Transitionen können neue Attribute generieren, bestehende verändern, und die Transitionsaktivierung von den Markenattributen abhängig machen.

Inhibitoren

Inhibitoren sind spezielle Inputpfeile der Transitionen, die das Feuern verhindern, falls die Stelle eine entsprechende Marke enthält. Über diese Verbindungen fliesst nie eine Marke. Sie werden vom Werkzeug mit einem kleinen Kreis an der Spitze des Pfeiles gekennzeichnet.

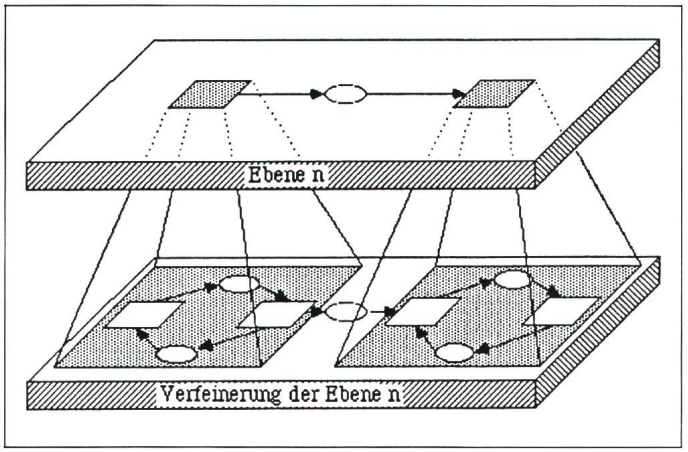
Bedeutung und Modellierung der Zeit

Unter dem Begriff *Zeit* versteht man im allgemeinen einen Wert, der in einem System global bekannt ist. In räumlich verteilten Systemen kann eine solche Grösse grundsätzlich nicht existieren. In den ursprünglichen Petri-Netzen gibt es deshalb keine Zeit. Trotzdem beobachten wir auch in verteilten Systemen, dass die Häufigkeiten voneinander unabhängiger Ereignisse in einem bestimmten Verhältnis zueinander stehen. Der Zweck von Simulationen ist oft, gerade solche Häufigkeitsverhältnisse, d.h. Leistungsmerkmale eines Systems zu untersuchen. Das Petri-Netz-Werkzeug bietet deshalb dem Benützer die Möglichkeit, dem Simulator zusätzliche Information über die gewünschte Feuerungshäufigkeit der Transitionen einzubringen, indem man ihnen Aktivierungszeiten zuweist. Dies geschieht dadurch, dass in der Transitionsinschrift der dafür reservierten Variablen *Delay* ein Wert zugewiesen wird. Die Feuerung durch eine aktivierende (d.h. eine speziell dafür reservierte Marke) wird dann um diesen Delay verzögert. Andere Marken können trotzdem gleichzeitig die Transition aktivieren bzw. feuern. Für andere Transitionen sind die Marken nicht reserviert, sie können von ihnen konsumiert werden.

Funktionen des Werkzeugs

Die Benutzeroberfläche des Petri-Netz-Werkzeugs ist window-orientiert, wobei verschiedene Typen von Windows existieren. Die wichtigsten sind die Editier-, Simulations-, Beschriftungs- und Statistik-Windows. Die Windows enthalten graphisch oder alphanumerisch dargestellte Elemente, die mit Hilfe der Maus selektiert werden können. Das Werkzeug

Figur 1
Hierarchische Strukturierung von Petri-Netzen



präsentiert auf Wunsch ein Pop-Up-Menü mit den Funktionen, die bei der aktuellen Selektion ausgeführt werden können. Es können eine beliebige Anzahl Windows gleichzeitig auf dem Bildschirm geöffnet sein. Für jedes Modul des Netzes wird ein separates Window verwendet. Darin sind die Stellen speziell gekennzeichnet (Rasterung), welche mit dem Modul auf der nächst höheren Ebene verbunden sind; sie bilden die Schnittstelle des Moduls gegen aussen.

Editieren

Der Editor stellt Funktionen zum Erzeugen, Verbinden, Verschieben, Beschriften, Verfeinern, Vergröbern, Löschen und Speichern der Netzelemente zur Verfügung. Beim Ausführen jeder Funktion wird geprüft, ob sie das Modell in korrekter Weise verändert. Diese Tests beziehen sich sowohl auf die graphische Struktur des Modells, wie auch auf die Syntax der Elementbeschriftungen.

Die Hierarchie eines Netzes kann nachträglich verändert werden. Wenn ein Modul zu komplex wird, kann aus einem Teil davon ein neues gebildet und als Unterknoten in den Modul-Baum eingefügt werden. Im aktiven Modul erscheint dieses neue Modul als T-Element, zu dem die Schnittstelle automatisch generiert wird. Ebenso existiert die Umkehrfunktion, die ein Modul aus dem Baum entfernt und es durch die Elemente seiner Verfeinerung ersetzt. Durch diese Funktionen wird der Top-down- und Bottom-up-Entwurf eines Modells unterstützt. Ein T-Element kann mit seinem gesamten Unterbaum und seinen Input- und Outputstellen auf einem File abgespeichert werden. Dieser Baum kann in jedem Modul wieder eingefügt werden, er ist eine Art Bibliotheksmodul. Für jedes Netzelement und für einen Teil der Pfeile kann ein Text-Window geöffnet werden. Darin kontrolliert ein Texteditor die Benutzer-Interaktionen. Bei einem Teil der Elemente hat die Beschriftung eine formale Bedeutung, bei anderen dient sie nur als Kommentar oder als Name.

Simulation und Animation

Für jedes Modul kann ein Simulations-Window geöffnet werden. Darin werden auch die Marken als schwarze Punkte auf den Stellen gezeigt. Ihre Attribute werden unter der Stellen-Beschriftung beschrieben. Im aktiven Window wird der Markenfluss animiert, d.h. es werden die Marken mit ihren Beschriftungen (Attributen) kontinuierlich über die Pfeile verschoben. Somit ist es möglich, das dynamische Verhalten des Systems für eine beliebige Auswahl von Modulen zu beobachten. Das aktive Modul ist dabei immer ganz sichtbar. Die Simulation kann schrittweise ausgeführt werden, wobei die zu feuernde Transition vom Benützer bestimmt oder zufällig ausgewählt wird. Zur Unterstützung der Fehlersuche kann auch rückwärts simuliert werden.

Die aktuelle Markierung kann jederzeit durch Hinzufügen und Entfernen von Marken, oder durch Modifikation der Markenattribute verändert werden. Auch gewisse Modelländerungen sind während der Ausführung möglich. Bei tiefergreifenden Modifikationen des Modells wird hingegen die Simulation automatisch neu gestartet. Es kann beliebig zwischen Editieren und Ausführen abgewechselt werden, wobei jedes aktive Window immer den neuesten Stand des Modelles zeigt.

Für alle Elemente des Systems (Stellen, Transitionen, Module, Pfeile)

kann ein Statistik-Window geöffnet werden. Darin wird eine von diesem Element abhängige Grösse laufend graphisch in einem Histogramm dargestellt. Diese Grösse wird durch eine vom Benützer bestimmbare Funktion berechnet, die bei jeder Behandlung des Elementes während der Simulation ausgewertet wird.

Code-Generation

Das Werkzeug bietet die Möglichkeit, aus einem Modell ein Programm in der Programmiersprache C zu generieren. Durch Erzeugung von Testsequenzen kann verifiziert werden, ob dieses sich gleich wie das Modell verhält.

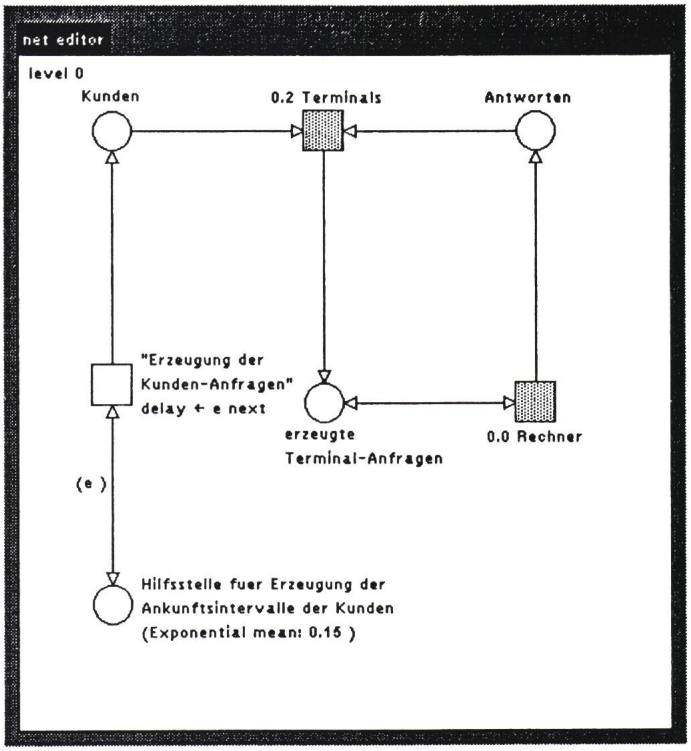
Einige Smalltalk-Klassen (z.B. Integer, Symbol, Array) wurden in C nachgebildet. Eine vollständig automatische Codegeneration ist nur dann möglich, wenn für die verwendeten Markenattribute und Meldungen eine C-Implementation existiert. In den getesteten Beispielen konnte ein Effizienzgewinn des generierten C-Codes gegenüber der Simulation von bis zu einem Faktor 40 erreicht werden. Eine mögliche Anwendung der Codegeneration ist zum Beispiel die automatische Implementation eines modellierten Kommunikations-Protokolls, das zusammen mit seiner Umgebung zu simulieren ist. Eine spezifische Von-Hand-Implementation wäre natürlich effizienter. Für viele Anwendungen aber dürfte die Effizienz des automatisch erzeugten Codes hingegen ausreichen. Die Vorteile sind die garantierte Übereinstimmung mit dem Modell und der kleinere Entwicklungsaufwand.

Anwendungsbeispiel

Das folgende Simulationsbeispiel wurde in [3] vorgestellt. Es handelt sich um ein Rechnersystem, das aus einem zentralen Rechner und einer Anzahl Terminals besteht. Diese Terminals werden von den Benützern für Informationsabfragen verwendet. Die folgenden Zahlen stammen aus der oben genannten Quelle:

Die Kunden-Anfragen kommen mit exponentiell verteilten Intervallen mit einem Mittel von 0,15 Minuten. Sie warten auf ein freies Terminal, wobei die Anzahl wartender Kunden nicht begrenzt ist. Die Generierung einer Anfrage auf einem der Terminals dauert 0,3 bis 0,5 Minuten, gleichverteilt. Die Anfrage wartet dann auf die Bearbeitung durch den Rechner. Dieser ist mit einem rotierenden Scanner ausgerüstet, welcher nach einer Round-Robin-Strategie der Reihe nach alle Terminals testet. Eine Rotation dauert 0,0027 Minuten, die gleiche Zeit wird für den Test benötigt. Wenn der Scanner bei einem Terminal eine Anfrage sieht, kopiert er diese in einen Puffer, was 0,0117 Minuten dauert. Der Puffer hat drei Plätze. Wenn er voll ist, muss der Scanner warten. Die Bearbeitung einer Anfrage dauert eine konstante Zeit von 0,0397 Minuten plus eine gleichverteilte Zeit zwischen 0,05 und 0,1 Minuten.

Figur 2
Die oberste Ebene des Rechnersystems



Transition

Module

Stelle

Beschriftung der Transitionen: generiert und beeinflusst die Attribute

Beschriftung der Module: Modulbezeichnung, Stellung in der Hierarchie

Beschriftung der Pfeile: Liste der Attribute der darüberfliessenden Marken (in Klammern)

Beschriftung der Stellen: Bezeichnung und Smalltalk-Code für Anfangsmarkierung (in Klammern)

Petri-Netz-Modell des Rechnersystems

Um das Modell für diese Einführung übersichtlich zu gestalten, wurde es relativ stark strukturiert. Es besteht aus drei hierarchischen Ebenen, welche vier verschiedene Netze enthalten.

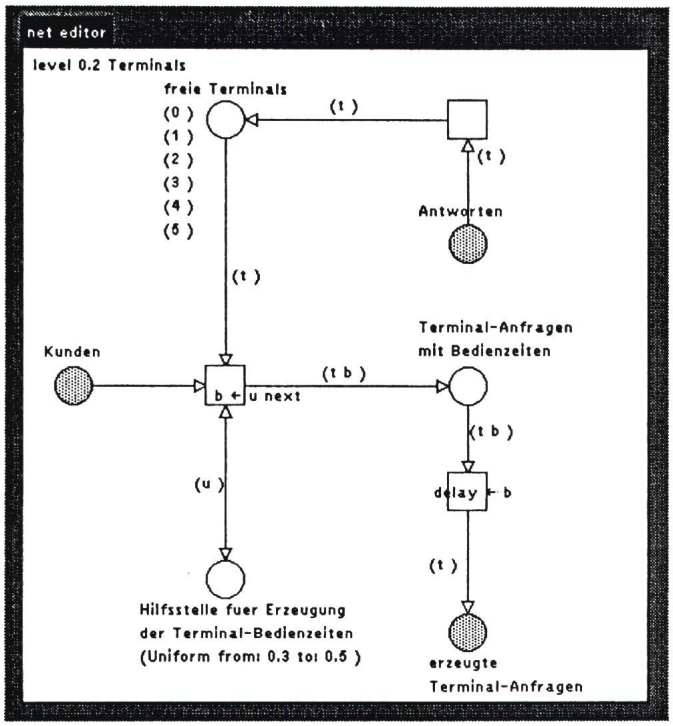
Oberstes Modul

Die Figur 2 zeigt eine Editor-Sicht der obersten Ebene (Level 0). Sie wird von einem Netz gebildet, welches eine Transition (weisses Quadrat), zwei Module (graue Quadrate) und vier Stellen (Kreise) enthält. Die Doppelpfeile sind eine Abkürzung für einen hin- und einen zurückführenden Pfeil. Sie bedeuten, dass die Marke beim Feuern der Transition von der Stelle entfernt und gerade wieder darauf zurückgelegt wird. Die Beschriftungen der Pfeile bestehen aus einer Liste der Attribute der darüberfliessenden Marken (in runden Klammern). Sehr oft sind es *Variable*, die auch in der Transitionsinschrift verwendet werden können und als Wert das Markenattribut enthalten.

Die Transition des obersten Moduls (Fig. 2, weisses Viereck) erzeugt die Marken, welche die Kunden repräsentieren. Der Pfeil zur Stelle, welche die wartenden Kunden enthält, hat keine Beschriftung. Dies bedeutet, dass die generierten Marken keine Attribute haben. Die Transition ist zudem mittels eines Doppelpfeils mit der Hilfsstelle für die Erzeugung der Ankunftsintervalle verbunden. Der Doppelpfeil hat nur eine Beschriftung, die Transition legt darum die Marke unverändert wieder auf der Stelle ab. Die Variable *e* der Pfeilbeschriftung erhält beim Feuern als Wert das Attribut der Marke der Hilfsstelle, welches eine Instanz der Klasse *Exponential* ist (Def. s. Smalltalk-Objekte als Markenattribute). In der Beschriftung der Stelle zeigt der Editor den Smalltalk-Code (in runden Klammern), der die Attribute der Anfangsmarken generiert. Dieser Code wird nur einmal, beim Start der Simulation, ausgeführt. Der Code der Hilfsstelle besagt, dass der Klasse *Exponential* die Meldung

mean: mit dem Parameter 0,15 geschickt wird, wodurch eine Instanz dieser Klasse erzeugt wird, die dann das (einzige) Attribut der Anfangsmarke hervorbringt. Beim Feuern der Transition wird dieser Instanz, die durch die Variable e repräsentiert wird, jeweils die Meldung *next* geschickt, worauf diese die nächste exponentiell verteilte Zufallszahl (e next) mit dem Mittelwert 0,15 erzeugt. Diese wird der reservierten Variablen *Delay* zugewiesen, so dass die Transition mit einer entsprechenden Zeitverzögerung feuert. Das Modul *Terminals* konsumiert die Kunden-Marken, schickt dem Rechner Anfragen und konsumiert dessen Antworten. Die Pfeile von Modulen haben nie eine Beschriftung; man sieht ihnen also die Art und Anzahl Attribute der darüberfliessenden Marken nicht an. Der Rechner konsumiert die Terminal-Anfragen und schickt die Antworten zurück.

Figur 3
Editor-Sicht des Moduls Terminals



Modul Terminals

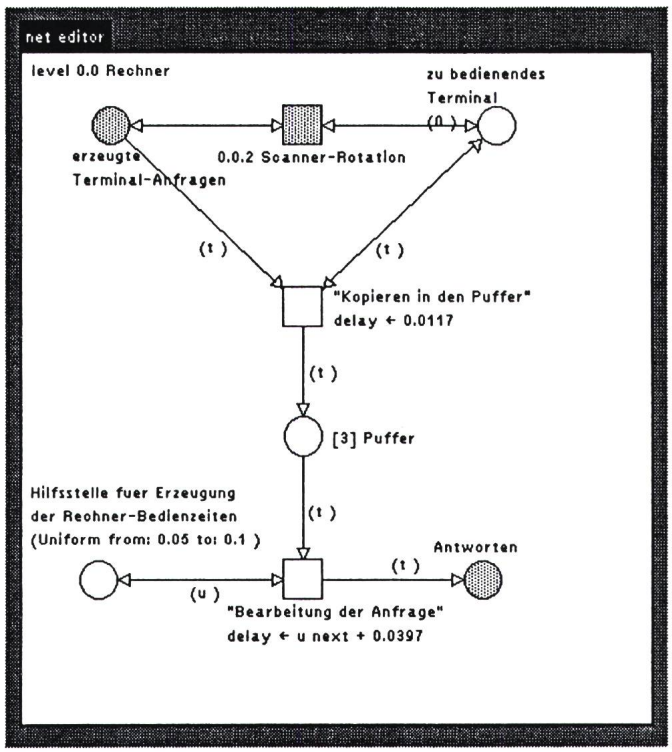
Die Figur 3 zeigt die Verfeinerung des Moduls *Terminals*. Die grau gerasterten Stellen gehören nicht zum Modul selber, sondern sind Input- oder Outputstellen des Moduls auf der nächst höheren Ebene. Sie können aber trotzdem in diesem Modul mit Transitionen verbunden werden, allerdings nur in diejenige Richtung, in der auch auf der oberen Ebene eine Verbindung existiert. Stellen können auf diese Art in einer beliebigen Anzahl Ebenen erscheinen.

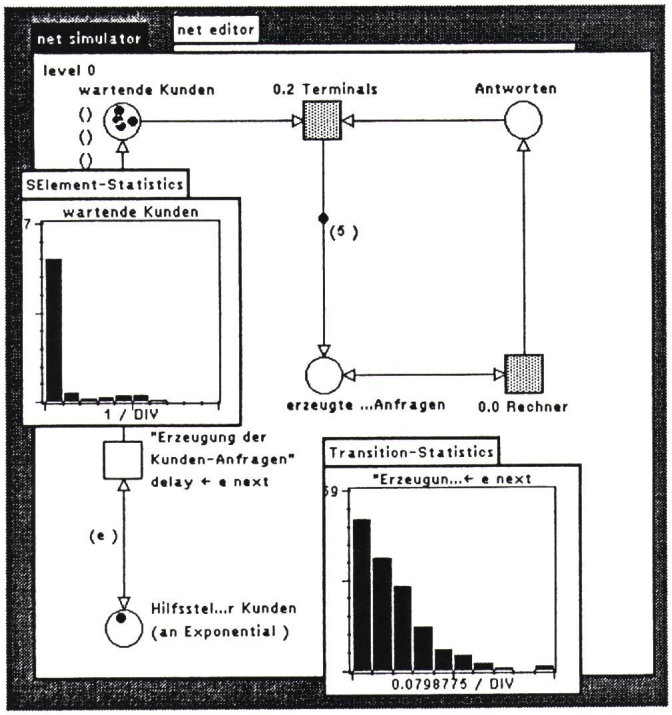
Die Transition (in der Mitte) generiert ohne Verzögerung eine Anfrage, wenn mindestens ein Kunde wartet und mindestens ein Terminal frei ist. Die freien Terminals werden durch Marken auf der Stelle *freie Terminals* repräsentiert, am Anfang sind alle Terminals frei. Diese Marken haben als Attribut einen Integer, der das Terminal kennzeichnet. Die entsprechende Variable t (für Integerzahl 0 bis 5) kommt sowohl an einem Input- wie auch an einem Output-Pfeil vor. Der Integer-Wert des Terminals wird dadurch als erstes von zwei Attributen der Outputmarke (t b) mitgegeben. Die Hilfsstelle für die Erzeugung der Terminal-Bedienzeiten enthält immer eine Marke mit einer Instanz der Klasse *Uniform* als Attribut, sie ist bereits in der Anfangsmarkierung vorhanden. Schickt man einer solchen Instanz die Meldung *next*, was beim Feuern der Transition gemacht wird, antwortet sie mit einer gleichverteilten Zufallszahl aus dem Intervall, das bei der Erzeugung der Instanz angegeben wurde (hier 0,3 bis 0,5). Diese Zufallszahl (u next) wird als zweites Attribut *b* der Outputmarke (t b) zugewiesen. Die nächste Transition wird um den Wert dieses Markenattributes verzögert. Nach Ablauf des Delays erzeugt sie eine Marke, welche nur noch die Terminal-Nummer als Attribut enthält. Man beachte, dass mehrere Marken gleichzeitig auf den Ablauf ihres Delays warten können. Die Transition oben rechts kopiert die Antworten ohne Verzögerung auf die Stelle *freie Terminals*.

Modul Rechner

Die Figur 4 zeigt das Modul *Rechner*

Figur 4
Das Modul Rechner





Figur 5
Simulatorsicht der obersten Ebene mit Statistik der wartenden Kunden und der Ankunftsintervalle

Simulation des Modells

Die Figur 5 zeigt eine Simulator-Sicht des obersten Moduls. Es wird gerade das Feuern einer Transition im Modul *Terminals* animiert, welche die Marke mit dem Attribut 5 auf die Stelle erzeugte Terminal-Anfragen legt. Das linke Statistik-Window zeigt in x-Richtung die Anzahl wartender Kunden, und in y-Richtung die Dauer, während welcher diese Anzahl registriert wurde. Der erste Balken gibt an, über welche Zeitdauer diese Anzahl grösser oder gleich null und kleiner als eins – weil in diesem Fall nur ganze Zahlen möglich sind –, also null war. Man sieht, dass sehr selten ein Kunde warten musste, dass aber trotzdem für sehr kurze Zeit einmal eine Schlange von 6 Kunden vorhanden war. Das zweite Statistik-Window enthält die Intervalle zwischen den ankommenden Kunden. Es zeigt die erwartete Exponentialverteilung.

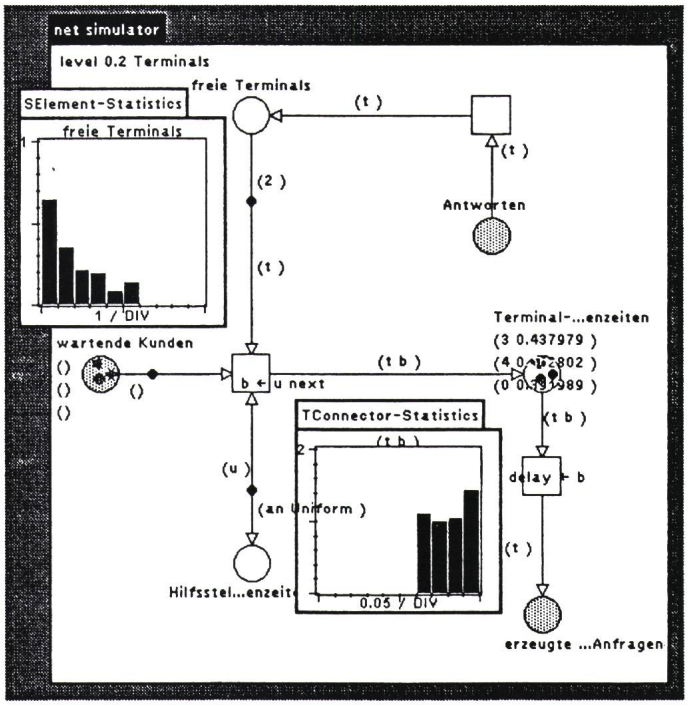
Die Figur 6 ist eine Simulator-Sicht des Moduls *Terminals*. Man sieht, wie die Inputmarken über die Pfeile einer Transition verschoben werden. Die Auslastung der Terminals (linke Statistik) und die Verteilung der Bedienzeiten wurden statistisch erfasst.

Nun wollen wir betrachten, wie sich das Rechnersystem verhält, wenn es mit nur 3 anstatt 6 Terminals ausgerü-

der zweiten Ebene. Es enthält ein weiteres Modul *Scanner-Rotation*. Dieses liest die Marken der Stelle *erzeugte Terminal-Anfragen* und verändert nur diejenigen der Stelle *zu bedienendes Terminal*, obwohl das auf dieser Ebene nicht ersichtlich ist. Man sieht lediglich, dass im Modul *Scanner-Rotation* von beiden Stellen Marken entfernt und darauf abgelegt werden können. Die Transition in der Mitte ist nur dann feuerbar, wenn eine Terminal-Anfrage vorhanden ist, deren Nummer gleich dem Attribut der Marke auf der Stelle *zu bedienendes Terminal* ist. Diese Anforderung muss deshalb erfüllt sein, weil beide Input-Pfeile der Transition mit der gleichen Variablen beschriftet sind. Eine weitere Bedingung für die Feuerbarkeit ist, dass auf der Stelle *Puffer* weniger als 3 Marken liegen. Die Kapazität dieser Stelle ist auf 3 beschränkt. Die Transition wird um eine konstante Zeit verzögert. Die untere Transition modelliert schliesslich die Bearbeitung der Anfrage. Sie wird um eine konstante Zeit plus eine gleichverteilte Auswahl aus dem Intervall 0.05 bis 0.1 verzögert. Beide Transitionen in diesem Modul können nur von einer Anfrage gleichzeitig aktiviert sein, weil sie beide eine Inputstelle haben, die immer nur eine Marke enthält. Das System hat ja auch nur einen Scanner und einen Rechner.

Modul Scanner-Rotation

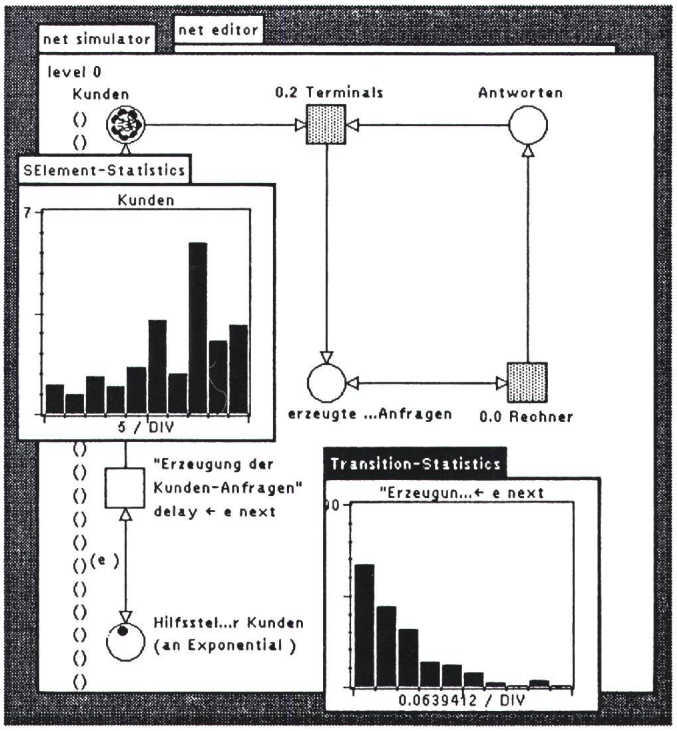
Dieses Modul sorgt dafür, dass das Attribut der Marke auf der Stelle *zu bedienendes Terminal* zyklisch ändert. Es wird hier nicht gezeigt.



Figur 6
Simulator-Window für das Modul Terminals mit Statistik der Terminalbelegung und Bedienzeiten

stet ist. Die Figur 7 zeigt die resultierende Statistik der wartenden Kunden. Meistens lag deren Anzahl zwischen 35 und 40, und zeitweise mussten bis zu 49 Kunden warten. Mit 3 Terminals ist das System also überlastet, während es mit 5 und sogar mit 4 noch akzeptabel arbeitete.

Figur 7
Statistik der wartenden Kunden bei 3 Terminals



Das hier vorgestellte Werkzeug ist an der Swissdata an den Ständen 212.255 und 211.315 zu sehen. Interessenten können sich auch direkt an den Autor wenden.

Bei komplexen Systemen können solche Leistungsaussagen nur mit Hilfe von Simulationen gemacht werden. Das Petri-Netz-Werkzeug erlaubt ein interaktives Verändern des Systems, wobei die Resultate sofort auf dem Bildschirm sichtbar werden.

Modellierungsaufwand

Wenn man mit den erweiterten Petri-Netzen und der Bedienung des Werkzeuges einmal vertraut ist, kann ein System sehr schnell modelliert werden. Eine typische Anwendung eines solchen Tools geschieht aber oft in einer Phase, wo das zu untersuchende System noch gar nicht genau bekannt ist. Das heisst, man lernt es gerade durch die Modellierung und Optimierung kennen und spezifiziert es damit exakt. Als Resultat erhält man neben der exakten Spezifikation auch Leistungsaussagen und eventuell bereits das Programm, wenn das System oder ein Teil davon in Software realisiert werden soll. Wichtig ist dabei, dass durch die Bedienung des Werkzeuges nicht erheblicher zusätzlicher Aufwand entsteht, was hier dank der Benützerfreundlichkeit der Smalltalk-Umgebung der Fall ist.

Literatur

[1] *A. Goldberg* and *D. Robson:* Smalltalk-80. The language and its implementation. Reading/Massachusetts, Addison-Wesley, 1985.
[2] *W. Reisig:* Systementwurf mit Netzen. Berlin u.a., Springer-Verlag, 1985.
[3] *G.M. Birtwistle:* A system for discrete event modelling on simula. London and Basingstoke, MacMillan, 1979.