Implementierung von Petri-Netz-Modellen für die Prozeßsteuerung - ein Überblick

Herwig Unger, Bernd Däne, Wolfgang Fengler

30. November 1994

Zusammenfassung

Petri-Netze haben sich in den vergangenen Jahren als eines der besten Beschreibungs- und Analysemittel für technische und technologische Prozesse mit einer hohen Anzahl von Nebenläufigkeiten etabliert. Ungeachtet dessen gibt es erst eine kleine Anzahl von Verfahren, die Petri-Netz-Modelle direkt in eine lauffähige Software überführen können. Der vorliegende Artikel gibt einen Überblick über den Stand der Technik auf diesem Sektor und leitet Forderungen für eine Entwicklungsumgebung ab, die eine Evolution der Petri-Netze in eine graphische Programmiersprache ermöglichen soll.

1 Einführung

Nachdem Mehrprozessorarchitekturen kostengünstig selbst für kleine und mittlere Automatisierungsaufgaben eingesetzt werden können, ist die Entwicklung von Parallelprogrammen zu einer Kernaufgabe des Softwareentwurfs geworden. Die dabei neu auftretenden Probleme können im wesentlichen den folgenden drei Klassen zugeordnet werden (vgl. [UNG94]):

- **Softwareprobleme**, die aus der Problematik der Beschreibung einer Lösungsstrategie in Form von mehreren parallel effektiver abarbeitbaren und miteinander kommunizierenden Prozessen entstehen, wobei die eingeschränkte Möglichkeit des Menschen zum 'parallelen Denken' ein wesentliches Hindernis darstellt.
- **Hardwareprobleme**, die die Verteilung, Kommunikation und die Bestimmung von Abarbeitungsrestriktionen von Prozessen in Wechselwirkung mit der Konfektionierung der Hardware umfassen sowie ferner Fragen der Eignung und Spezifikation spezieller Hardwarearchitekturen für die Bearbeitung einer speziellen Aufgabe betreffen.

- **Standardisierungsprobleme**, die aus dem unterschiedlichen Verhalten derselben Software auf verschiedenen Rechner resultieren bzw. sich direkt in Fragen der Kompatibilität und Portabilitiät paralleler Software äußern.

Mit Hilfe heute vorhandener programmiertechnischer Hilfsmittel erfolgt der parallele Softwareentwurf im wesentlichen auf einem der folgenden zwei Wege:

Parallelisierung vorhandener Standardsoftware.
Den Ausgangspunkt hierfür bildet eine Reihe komplexer, anwendungsspezifischer Programme, die für den Gebrauch auf bisher vorhandenen seriellen Rechnern entworfen wurden. Mit Hilfe heute existierender Parallelisierungstools können diese in Parallelsoftware überführt werden. Da kein eigentlich neuer Algorithmus entsteht, bleiben viele Möglichkeiten zur Geschwindigkeitssteigerung ungenutzt. Der speed up wird hier im wesentlichen durch die beschleunigte Abarbeitung spezieller Programmelemente (Vektoren, Matrizen, Schleifen) durch mehrere Prozessoren bewirkt. Durch die hohe Anzahl existierender serieller Programme rechtfertigt sich diese Vorgehensweise, da der Entwurf völlig neuer Software oft zu ineffizient werden würde. Jedoch treten Programmelemente der genannten Art in Softwarepaketen der Automatisierungstechnik relativ selten auf.

Manuelle Parallelisierung von Problemlösungen. Eine ganze Reihe technischer und technologischer Probleme lassen bei näherer Betrachtung eine klare Struktur zwischen parallel ausführbaren Aktionen und deren notwendiger Verbindung bzw. Wechselwirkung (Kommunikation) erkennen. Diese Struktur kann dann unmittelbar in ein Parallelprogramm überführt werden, wobei Petri-Netze und auch andere formale Beschreibungsmittel (z.B. Automatennetze) sehr gut zu deren Beschreibung und Analyse herangezogen werden können. Die Wahl der parallel ausführbaren Blöcke bleibt dabei jedoch dem Anwender überlassen. Außerdem muß festgestellt werden, daß die große Komplexität der Problemlösungen eine effiziente Bearbeitung relevanter Prozesse häufig unmöglich macht.

Es wird ersichtlich, daß keines der genannten Verfahren eine wirklich geeignete Methode zum parallelen Softwareentwurf für die Zukunft darstellt. Diese Aufgabe erfordert zwei neue Features in der Arbeit: Zum einen eine kompakte Darstellung von Daten- und Kontrollflußaspekten in ihren zwei Dimensionen, d.h. der Serialität der Abarbeitung einzelner Aktionen sowie die gleichzeitige, nebenläufige Ausführbarkeit einzelner Aktionen zu verschiedenen Zeitpunkten. Zum anderen muß gewährleistet werden, daß diese kompakte Darstellung durch formale Verfahren in Prozesse und Kommunikationselemente eines parallelen Programms überführt werden kann. Diese Anforderungen sind z.Zt. durch eine Spezifikation mit herkömmlichen Programmiersprachen kaum zu erfüllen, so daß Anwendungen i.a. auf gut parallelisierbare, spezielle Probleme beschränkt bleiben.

2 Parallelprogrammierung mit Petri-Netzen

2.1 Modellierung und Analyse

Petri-Netze stellen nach [FEN91] und [ROZ87] für die Beschreibung und Analyse von nebenläufigen Prozessen eine der am besten entwickelten formalen graphischen Grundlagen dar. Zur Unterstützung des Spezifikations- und Verifikationsprozesses existiert ferner eine Reihe von Petri-Netz-basierten Tools, die ein graphisches Interface und einen Netzinterpreter enthalten. Außerdem wird i.a. entweder eine Analysekomponente oder ein Interface für eine solche enthalten sein, um die für den Softwareentwurf interessanten formalen Netzeigenschaften wie z.B. Lebendigkeit und Sicherheit testen zu können. [1] Der Einsatz dieser Tools im Rahmen der Modellspezifikation und -verifikation hat dabei bereits einen sehr hohen Standard erreicht, wie sich auch am Stand zahlreicher durch die DFG geförderter Projekte zeigt. Erwähnenswert sind dabei auch erste Bemühungen zu einer Standardisierung von Schnittstellen und Datenformaten.
Eine wesentliche Rolle in diesen Arbeiten hat dabei die Weiterentwicklung der Petri-Netz-Klassen gespielt. Ausgehend von den Platz-Transitions-Netzen, die im wesentlichen eine Berücksichtigung von Kontrollflußaspekten erlauben, haben sich durch die gefärbten und algebraischen Netze die Möglichkeiten zur Repräsentation komplexer abstrakter Datenstrukturen immer weiter verbessert. Durch die Einführung von Hierarchien in Form von Platz- und Transitionsunternetzen sind auch die Voraussetzungen für einen modularen Top-Down-Entwurf gegeben. Aus diesem Grund kann heute davon ausgegangen werden, daß die Darstellungsmächtigkeit von Petri-Netz-Modellen der von HLL-Programmiersprachen im nötigen Umfang entspricht.

2.2 Implementierung

Lediglich eine kleine Anzahl von Petri-Netz-Tools enthält parallelisierende Implementierungskomponenten, die eine direkte Umsetzung der in der Petri-Netz-Beschreibung formulierten Problemlösung in ein Parallelprogramm auf formalem Wege erlauben.

Die dabei verwendeten Verfahren können grob in zwei Kategorien unterteilt werden (vgl. Abb. 1):

- **Direkte bzw. Interpretative Verfahren**
  Hierbei erfolgt in unmittelbarer Art und Weise eine verteilte Implementie-

[1] so z.B. INA, PROD, PENECA, DESIGN/CPN, PACE, POSES; vgl. auch Unterlagen zum Workshop 'Concurrency, Specification and Programming', Informatik-Berichte Nr. 36 der HUB, Berlin 1994, und Tagungsmaterialien des 39. IWK der TU Ilmenau, 1994

Abbildung 1: Klassifikation bekannter Implementierungsmethoden

rung der jeweiligen Schaltregel des Petri-Netzes, so daß meist jeder Transition direkt ein Prozeß zugeordnet wird (vgl. auch [HAR87], [HAU87] und [THO91]). Eine Verbesserung dieser Vorgehensweise wird i.a. nur durch Methoden zur Beschränkung der Anzahl der zu testenden Transitionen erreicht (mit Hilfe von Vorplätzen ist dies in [BOE94] zu finden) oder aber einer günstigen Selektion von zu testenden Transitionsgruppen (wie z.B. in [COL86] auf der Basis von Konfliktklassen).
Der Vorteil der direkten Verfahren besteht in ihrer Anwendbarkeit für alle Netzklassen, der Nachteil in der oft zu hohen Zahl von Prozessen und einem damit verbundenen extrem hohen Kommunikationsaufwand.

- **Indirekte bzw. Compilative Verfahren**
  Das Wesen der indirekten Verfahren besteht darin, ein gegebenes Petri-Netz entweder mit Zustandsmaschinen zu überdecken oder aber eine Transformation in der Art zu finden, daß eine solche Überdeckung leicht zu erreichen ist. Da die Überdeckbarkeit mit Zustandsmaschinen nicht in jedem Fall gegeben ist (vgl. [REI79]) und entsprechende Algorithmen eine hohe Komplexität aufweisen, ist die erste Vorgehensart auf spezielle Applikationen beschränkt.
  In beiden genannten Fällen erfolgt in einem zweiten Schritt eine Umsetzung der Zustandsmaschinen in je einen Prozeß des Parallelprogramms sowie die Bestimmung der verbleibenden Netzkomponenten als Kommunikationsstrukturen (s. [BRE91], [UNG92], [UNG94]).
  Somit kann i.a. ein effektiverer Programmkode als bei den direkten Verfahren erzeugt werden ([UNG94]). Nachteilig wirkt sich dabei aber aus, daß die indirekten Verfahren i.a. nur für ganz bestimmte Petri-Netz-Klassen effektiv anwendbar sind; ferner erfordern sie meist einen hohen rechentechnischen Verarbeitungsaufwand. Abb.1 zeigt einen Überblick über heute bekannte Verfahren.

Favorisiert man die Klasse der indirekten Verfahren, so stellt man fest, daß die Einschränkungen der Netzklasse ihren Ursprung insbesondere in Restriktionen bezüglich der Alternativ- bzw. Konfliktstrukturen im Ausgangsnetz finden. Dies äußert sich insbesondere in der Forderung nach der Persistenz der Netze oder aber in einer konkreten Beschränkung der behandelbaren Konfliktsituationen (meist wird die free choice Eigenschaft gefordert). Teilweise kann nur mit Nebeneffekten bei der Implementierung (z.B. exklusive Nutzbarkeit von Ressourcen oder Datenkonzentration in einem gemeinsamen Objekt bzw. Server-Prozeß) eine Lösung dieses Problems gefunden werden.
Weitere Einschränkungen ergeben sich aus der Kopplung der Teilprozesse: Systeme gekoppelter Zustandsmaschinen erfordern im Gegensatz zu Systemen kooperierender Zustandsmaschinen (vgl. Abb. 2) oft die Nutzung eines *shared memory*, der jedoch bei automatisierungstechnischen Mehrrechnersystemen meist nicht verfügbar ist. Die Simulation eines solchen in Server-Prozessen schafft dabei meist keine leistungsmäßig zufriedenstellende Lösung (s. [BRE91], [UNG94]).

Abbildung 2: Möglichkeiten zur Kopplung von Zustandsmaschinen

Abschließend muß zu dieser Thematik festgestellt werden, daß bei den angeführten Compilervarianten eine ganze Reihe von Fragen offen bleiben. Dies erfordert neben den erwähnten Schwierigkeiten insbesondere die Bearbeitung folgender Aufgabenstellungen:

- die Auswirkungen einer Reihe von zunächst frei wählbaren Optionen z.B. im Rahmen von Alternativen in Wegsystemen;
- die gezielte Beeinflussung des Schedulings bzw. der gegebenen Hardwarearchitektur;
- die Abschätzung des Echtzeitverhaltens der implementierten Parallelsoftware;
- die Beurteilung der Zuverlässigkeit des Systems bezüglich Systemfehlern und -ausfällen.

2.3 Hierarchisierung

Hierarchische Strukturen spielen im Entwurfsprozeß, insbesondere bei Anwendung einer Top-Down-Entwurfsstrategie, eine wesentliche Rolle. Dies erfordert nach Ansicht der Autoren einen weiteren Ausbau der Möglichkeiten einer hierarchischen bzw. modularen Strukturierbarkeit von Petri-Netz-Modellen, die folgende drei Varianten umfassen sollten:

Unternetze, wie sie heute bereits in den verschiedenen Tools berücksichtigt werden können. Platz- bzw. Transitionsunternetze schaffen dabei die Möglickeit, einzelne Netzkomponenten im Sinne eines Top-Down-Entwurfs systematisch zu verfeinern. Aus der Sicht des Softwareentwurfs stellen sie jedoch lediglich eine Methode zur Verbesserung der Modellstruktur und -übersichtlichkeit dar, d.h. sie berücksichtigen im wesentlichen zeichnerische Aspekte.

Funktionsteilnetze, die Funktionen herkömmlicher Programmiersprachen entsprechen, beschreiben von verschiedenen Stellen des Programms aufrufbare Teilnetze. Beim Aufruf erfolgt dabei ausgehend von einer bestimmten Markierung die Abarbeitung des Teilnetzes, der Abarbeitung der Funktion entspricht eine von dieser aus erreichbare zweite Markierung, die im nächsten Schritt in die Initialmarkierung überführt wird und den Rücksprung in das rufende Netz auslöst. Wesentlichster Unterschied zu den Unternetzen ist dabei die Tatsache, daß Funktionsteilnetze von verschiedenen Stellen aufgerufen werden können, eine Parameterübergabe möglich ist und erst nach Abarbeitung das Originalnetz ab der rufenden Markierung unter Berücksichtigung der Rückgabeparameter weiter bearbeitet wird. Dabei sind im Rahmen der Parallelprogrammierung insbesondere Fragen der Wiedereintrittsfähigkeit eines Funktionsteilnetzes zu klären.

Rekursionen stellen einen Spezialfall von Funktionsteilnetzen dar, wobei ein Funktionsteilnetz immer wieder bis zum Erreichen eines Abbruchkriteriums von sich selbst aufgerufen wird. Hierbei sind insbesondere Fragen der Parameterübergabe und Beschränktheit des entsprechenden Teilnetzes zu klären, was jedoch wegen der vielseitigen Verwendbarkeit rekursiver Beschreibungen in der Technik durchaus gerechtfertigt ist.

Insbesondere die Diskussion von rekursiven Darstellungen und ihre Integration in den Netzsyntax läßt die Frage nach der sinnvollen Darstellbarkeit und dem Verhältnis von funktioneller und struktureller Sicht unter einem neuen Blickwinkel erscheinen.

3 Grundkonzeption einer Nutzeroberfläche

Kern einer zu entwickelnden Nutzeroberfläche bleibt - wie auch bei bisher bekannten Tools - ein Petri-Netz-Editor. Dieser muß die Bediengewohnheiten des Nutzers ausgehend von der genutzten Systemoberfläche in weitem Umfang unterstützen.
Es ist anzustreben, dem Nutzer für alle Arbeitsaufgaben eine einheitliche Arbeitsumgebung zu schaffen; d.h. auch für die Erledigung von Bedienaufgaben auf Betriebssystemebene. Deshalb ist es berechtigt, die Eignung eines Petri-Netz-basierten Interfaces für diesen Zweck zu untersuchen.

Abbildung 3: Schalenmodell einer Petri-Netz-Maschine

Die Favorisierung von algebraischen Netzen legt dabei nahe, strukturelle Komponenten (Schleifen, Alternativen etc.) mit Mitteln der Petri-Netz-Topologie darzustellen und Datenmanipulationen mit Hilfe von funktionalen, mit den Transitionen verbundenen Notationen zu beschreiben. Somit kommt den Plätzen des Petri-Netzes die Aufgabe von Variablen zu, wobei durch eine geeignete Zoomfunktion (z.B. platzgebundene Windows) eine Anzeige und Repräsentation der aktuellen Belegung der Plätze im System möglich sein muß. Dabei ist auch die Frage zu klären, inwiefern bestimmte Daten, die das Netz enthält (z.B. Rücksprungadressen von Funktionen oder Rekursionstiefen), in der Anzeige verborgen werden sollten bzw. gesondert darzustellen sind. Abb. 3 zeigt ein Schalenmodell für eine Petri-Netz-basierte Programmierumgebung.

Ein weiterer Schwerpunkt bei der Entwicklung geeigneter Nutzerumgebungen liegt in der Lösung des Geschwindigkeitsproblems. Von herkömmlichen Tools bekannte Petri-Netz-Interpreter, die üblicherweise ein direktes Implementierungsverfahren eines variablen Netzes auf einer Einprozessormaschine realisieren, eignen sich lediglich für die Simulation ausgewählter Teilkomponenten in ausgewählten Situationen. Für einen Echtzeitbetrieb der entworfenen Netzkomponenten bzw. eine schnelle Simulation des kompletten System sind Interpreter keine geeigneten Werkzeuge mehr; hier muß man auf compilative Verfahren zurückgreifen. Dabei steht die Aufgabe, die eingegebenen Netze möglichst im Hintergrund in einen ausführbaren Code umzusetzen, der dann im Rahmen einer schnellen Simulation zur Abarbeitung der entworfenen Netzkomponenten

genutzt werden kann.
Diese Vorgehensweise unterstützt gleichzeitig eine der eingangs beschriebenen Methoden zum Design von paralleler bzw. verteilter Software: einzelne Komponenten für die Abarbeitung auf jeweils einem Prozessor werden entworfen, nach dem Entwurf auf dem jeweilgen Verarbeitungsknoten gestartet und liefern dann in Echtzeit die für den Entwurf bzw. die Simulation der anderen Komponenten notwendigen Daten, Nachrichten oder Signale. Wichtig für den Entwerfer ist dabei, daß auch diese Komponenten als Netzmodell vorliegen und ggf. noch beeinflußt und modifiziert werden können. Das Hauptproblem bei der Umsetzung liegt hierbei in der gleichzeitigen Existenz von Netzkomponenten, die in Echtzeit abgearbeitet werden, und solchen, die sich im Zustand des Entwurfs befinden.

Abschließend soll noch kurz auf ein Problem eingegangen werden, das selbst für kommerzielle, serielle Softwarekomponenten kaum als gelöst angesehen werden kann: die Fragen der Archivierung und Wiederverwendung einzelner Blöcke. Nach Ansicht der Autoren bietet die Möglichkeit zu einer strukturellen Analyse der Petri-Netz-Modelle einen entscheidenden Vorteil gegenüber den nur in textueller Notation vorliegenden herkömmlichen Programmen. Dies könnte eine Klassifikation und Archivierung wesentlich vereinfachen, falls es in Zukunft gelingt, hinreichende Kriterien und praktisch relevante Anwendungen zu finden.

4 Zusammenfassung

Im vorliegenden Beitrag werden Möglichkeiten und noch zu lösende Aufgaben aufgezeigt, um Petri-Netze als Grundlage für eine graphische Programmiersprache verwenden zu können. Dabei wird herausgearbeitet, daß neben einer Reihe von noch fehlenden Analyseverfahren die Frage der Nutzerakzeptanz insbesondere von der Verfügbarkeit geeigneter Programmierumgebungen abhängt, die die Bearbeitung der genannten Entwurfsprobleme für praktisch relevante Probleme mit einem einheitlichen Beschreibungsmittel unterstützen.

Literatur

[BOE94] Böhme, T.; Unger, H.: *Some Aspects to the Automatic Generation of Parallel Programs.*- in: Parallel Computing: Trends and Applications, Elsevier Science B.V., (1994)

[BRE91] Breant, F.: *Rapid Prototyping From Petri Net on a loosly Coupled Parallel Architecture.*- Applications on Transputers 3, S. 644-649, IOS-Press, (1991)

[COL86] Colom, J.M.; Silva, M.; Villaroel, J.L.: On software implementation of Petri nets and coloured Petri nets using high level concurrent languages.- 7. European Workshop on Application and Theory of Petri Nets, p. 207-241, Oxford, (1986)

[FEN91] Fengler, W., Philippow, I.: *Entwurf industrieller Mikrocomputersysteme.*- Carl-Hanser-Verlag, München, (1991)

[HAR87] Hartung, C.A.: *Programmierung einer Klasse von Multiprozessorsystemen mit höheren Petrinetzen.*- Hüthig-Verlag, (1987)

[HAU87] Hauschildt, D.: *A Petri Net Implementation.*- FBI-HH-M-145/87, FB Informatik, Universität Hamburg, (1987)

[REI79] Reisig, W.: *On a Class of Co-operating Sequential Processes.*- 1st. European Conference on parallel and distributed processing, Toulouse, (1979)

[ROZ87] Brauer, W.; Reisig,W.; Rozenberg, G.:
Petri Nets: Applications and Relationships to Other Models of Concurrency.-
LNCS 255, Springer-Verlag, Berlin-Heidelberg-New York, (1987)

[THO91] Thomas, G. S.: *Parallel Simulation of Petri-Nets.*- University of Washington, Technical Report 91-05-05, (1991)

[UNG92] Unger, H.; Ben Achour, K.: *A Method of the Design of Parallel Programs for a Multiprocessor System.*- in: Lecture Notes in Computer Science 634, Springer-Verlag, (1992)

[UNG93] Unger, H.: *Spezifikation und Implementierung paralleler Programme mit SATURN.*- Tagungsmaterialien der epp'93, Wien, (1993)

[UNG94] Unger, H.: *Untersuchungen zur Implementierung von Petri-Netz-Modellen auf Mehrprozessorsystemen.*- TU Ilmenau, FG Rechnerarchitektur, Dissertation A, (1994)

Keywords:
CASE, Multiprocessor Systems, Parallel Programs, Petri Net, Software Engineering, Programming Concepts

Kontaktadressen:
Dr.-Ing. Herwig Unger
Fachbereich Informatik
Universität Rostock
D-18051 Rostock

Prof. Dr.-Ing. habil. Wolfgang Fengler
Dr.-Ing. Bernd Däne
Institut für Theoretische und Technische Informatik
Technische Universität Ilmenau
D-98684 Ilmenau

E-mail:
hunger@informatik.uni-rostock.de
fengler@theoinf.tu-ilmenau.de
daene@theoinf.tu-ilmenau.de