Dynamische Generierung von parallelen Ablaufstrukturen

Holger Herzog
Siemens AG
Zentrale Forschung und Entwicklung
ZTI SOF 41
Otto-Hahn-Ring 6, 8000 München 83

Verteilte Systeme (VS) haben eine weite Verbreitung in Realzeitanwendungen gefunden. Gegenüber zentral organisierten Rechnerarchitekturen bieten sie eine Vielzahl von Vorteilen, die sich aus den in VS möglichen Transparenzaspekten wie z.B. der Leistungs-, der Fehler- und der Ortstransparenz herleiten. Zur Nutzung einer Reihe dieser Transparenzaspekte ist es notwendig Aktionen parallel auszuführen. Deshalb wird für VS und damit implizit auch für Realzeitanwendungen eine Entwicklungsunterstützung von parallelen Ablaufstrukturen benötigt. In Kapitel 2 dieser Arbeit werden Ansätze zur Entwurfsunterstützung von Softwaresystemen auf der Basis von Graphen kurz vorgestellt. Diese Ansätze werden in Kapitel 3 mit der Theorie der Graph-Grammatiken zu einem Konzept zur Entwicklungsunterstützung von parallelen Ablaufstrukturen verknüpft. An einem Anwendungsbeispiel wird demonstriert, daß dadurch eine nahezu durchgängige Entwicklungsunterstützung möglich wird.

1. Motivation

Parallele Ablaufstrukturen sind im Vergleich zu sequentiellen Ablaufstrukturen weitaus schwerer zu verstehen, zu verifizieren, zu testen und zu warten. Obwohl eine Reihe von Werkzeugen zur Entwurfsunterstützung /Dä 87/, zur Testunterstützung /Ga 85/ und zur Untersuchung der Korrektheit und Lebendigkeit von parallelen Systemen /Rö 86/ vorgeschlagen wurden, ist der erreichte Status unzureichend /Ka 87/. Die Komplexität des Entwicklungsprozesses paralleler Ablaufstrukturen verlangt nach einer möglichst durchgängigen Unterstützung, wobei aufgrund des hohen Schwierigkeitsgrades die Systementwicklung auf formalen Konzepten und Methoden aufsetzen sollte /Sch 86/.
Graphbasierte Modelle haben zur Darstellung von parallelen Abläufen eine weite Verbreitung gefunden. Insbesondere aufbauend auf Petri-Netzen wurden darüber hinaus Werkzeuge zur Entwurfsunterstützung entwickelt. Kapitel 2 stellt kurz einige Ansätze vor. In Kapitel 3 wird ein Konzept zur Unterstützung der Entwicklung von parallelen Ablaufstrukturen beschrieben, das die Ansätze aus Kapitel 2 mit der Theorie der Graph-Grammatiken verbindet. Die Umsetzung dieses Konzeptes in Werkzeuge demonstriert Abschnitt 3.3.

2. Systementwicklungsunterstützung durch Graphen

Die Entwicklung der Theorie der Parallelverarbeitung wurde begleitet durch die Entwicklung von graphbasierten Modellen zur Repräsentation von asynchronen nebenläufigen Aktionen. Die Knoten der Graphen stellen dabei einen Zustand, ein Ereignis oder die Ausführbarkeit einer Operation dar. Die Kanten modellieren Präzedenz- oder andere Abhängigkeitsbeziehungen zwischen den Knoten, d.h. sie können u.a. die Abarbeitungsreihenfolge in einem System aufzeigen. Der Gedanke - Graphen zur Entwicklungsunterstützung einzusetzen - lag deshalb nahe.
Es sind zwei Einsatzarten von Graphen zu unterscheiden:

- Graphen als direkte Benutzerschnittstelle zur visiuellen Darstellung komplexer Abläufe und Strukturen und
- Graphen als Datenstruktur zur internen Repräsentation komplexer Abläufe und Strukturen.

Die ersten Ansätze der Anwendung von Graphen beschränkten sich auf die Unterstützung eines einzigen Spezialgebietes (Entwurf, Beschreibung, Simulation) und hatten kaum eine formale Fundierung /Go 83/. Insofern war beim Systementwicklungsprozeß ein Wechsel zwischen den Modellierungskonzepten notwendig, und es

konnte kaum eine Gültigkeitsprüfung durchgeführt werden. Beide Faktoren stellen mögliche Fehlerquellen dar. Neuere Konzepte und Werkzeuge bauen auf Petri–Netzen /Dä 87//Rö 86//Re 83/ oder der allg. Netztheorie /Wi 86/ auf, um so die durch die formalen Grundlagen gegebenen Analysemöglichkeiten ausnutzen zu können. Auch sie sind allerdings häufig nur für einen Teil des Softwareentwicklungsprozesses konzipiert.

Die Methode ISAC /Wi 86/ unterstützt die Beschreibung und Analyse von informationsverarbeitenden Systemen. Ihr primäres Einsatzgebiet liegt somit in der ersten Phase der Softwareentwicklung – der Problemanalyse. Entsprechend einem top–down–Vorgehen beschreibt der Entwickler für jede Abstraktionsebene die relevanten Aspekte durch zweisortige Netze. Die Netze bestehen aus Aktivitäten und Informationen, die durch gerichtete Kanten verbunden werden. Beide Netzelemente können verfeinert werden, so daß sich ein gesamtes ISAC–Dokument als hierachische Struktur von zweisortigen Netzen darstellt. Insofern weist ISAC eine gewisse Ähnlichkeit mit der in /Re 83/ entwicklelten Methode zur Anforderungsdefinition und zum Systementwurf durch Petri–Netze auf.

Funktionsnetze /Go 83/ versuchen beide Konzepte miteinander zu verbinden, um so die Vorteile beider Modelle – die Modellierung dynamischer Abläufe bei Petri–Netzen und die Möglichkeit auf einer Menge von Abstraktionsebenen zu arbeiten bei ISAC – miteinander zu verbinden. Funktionsnetze erweitern die Kanal/Instanzen–Interpretation im Rahmen von Petri–Netzen u.a. indem Instanzen, Kanäle und Systembeziehungen in Klassen eingeteilt werden, denen je eine spezielle Bedeutung zugeordnet wird. Beispielsweise wird neben dem in Petri–Netzen gegebenen 'und'–Schaltverhalten ein partielles Schalten eingeführt. Im Gegensatz zu den o.a. Methoden wurden aufbauend auf Funktionsnetze auch Werkzeuge vorgestellt, die die Softwareentwicklungsphasen Implementierung und Test unterstützen /GT 86/.

3. Entwicklungsunterstützung von parallelen Ablaufstrukturen

3.1 Parallele Ablaufstrukturen

Bei der Implementierung der Steuerung paralleler Abläufe sind prinzipiell drei Möglichkeiten zu unterscheiden:

- die Realisierung als ein kommunizierendes Mehrprozeßsystem
- die Realisierung als prozeßinterne Aktivitätsverwaltung, wie sie aus dem Transaktionssystem CICS von IBM und dem Datenbanksystem SESAM von Siemens bekannt ist, und

- die Realisierung als Mehrprozeßsystem, bei dem jeder Prozeß intern eine Menge von Aktivitäten verwaltet.

Die Entscheidung zugunsten einer der Realisierungsmöglichkeiten wird durch eine Reihe von Faktoren beeinflußt. Je nach Betriebssystem gibt es eine unterschiedliche Obergrenze für gleichzeitig aktive Prozesse. Ist der Grad der möglichen Nebenläufigkeit größer als diese Obergrenze, so ist entweder die zweite oder dritte Alternative zu wählen oder der parallel mögliche Ablauf muß sequentialisiert werden, wobei auch eine prozeßinterne Aktivitätsverwaltung auf einem Einprozessorsystem eine Sequentialisierung darstellt. In Abschnitt 3.3 wird an einem Beispiel gezeigt, daß dies für ein Client/Server–System nicht in gleichem Sinne gelten muß. Ein weiterer durch das zugrundeliegende Betriebssystem bestimmter Faktor sind die Kosten für einen Prozeßwechsel, die äquivalent der Ausführungszeit von 5000 Befehlen bei Großrechenanlagen wie IBM/370 und mit 10–20 Befehlen beim System Embos von ELXSI /Ol 85/ angegeben werden. Ein dritter Faktor ist die Komplexität der Fehlerbehandlung, d.h. die Maßnahmen eine aktuelle Bearbeitung abzubrechen und auf einem definierten Sicherungszustand zurückzukehren. Insbesondere bei Mehrprozeßsystemen bedarf es zusätzlicher Strukturierungskonzepte, um eine adäquate Fehlerbehandlung zu erreichen /CR 86/.

3.2 Entwicklungsunterstützung durch Graph–Grammatiken

Die Entwurfsunterstützung der parallelen Ablaufstrukturen durch graphbasierte Werkzeuge liegt aufgrund von Kapitel 2 nahe. Im Gegensatz zu den sehr allgemein gehaltenen Konzepten aus Kap. 2 weisen parallele Ablaufstrukturen innerhalb festgelegter Anwendungsklassen wiederkehrende Strukturen auf, d.h. daß sich die Abhängigkeitsbeziehungen zwischen den parallel ausführbaren Aktivitäten durch eine Menge von Regeln beschreiben lassen.
Mit Hilfe von Graph–Grammatiken /Na 78/ ist es möglich, Strukturen in einem Graphen aufzubauen und dann im weiteren diese Strukturen als ein Objekt zu modifizieren. Die einzelnen Graphproduktionen modellieren dabei je ein bestimmtes Strukturmerkmal, das durch die die Abhängigkeitsbeziehungen darstellenden Kanten und durch eine mit einer Bedeutung versehenen Knotenmarkierung modelliert wird.

In dieser Arbeit werden Graphen als zentrale Datenstruktur zur Repräsentation der internen Ablaufstruktur bei einer prozeßinternen Aktivitätsverwaltung und als Basis einer automatischen Generierung eines kooperierenden Prozeßsystems vorgeschlagen. Dadurch wird es möglich durch Spezifikation einer Menge von Graphproduktionen eine Datenstruktur zu generieren, aus der parallele Ablaufstrukturen

abgeleitet werden können. Eine solche Eigenschaftsspezifikation paralleler Strukturen abstrahiert von der algorithmischen Beschreibung der Ablaufstruktur und von den benötigten Synchronisationsmechanismen. Dem Benutzer kann – entsprechend den Ansätzen in Kap. 2 – zur Unterstützung des interaktiven Entwurfs eine graphische Benutzeroberfläche angeboten werden. In Abschnitt 3.3 wird gezeigt, daß es Anwendungsfälle gibt, für die eine sprachliche Spezifikation geeigneter ist. Beide Beschreibungsformen können in das Konzept integriert werden.
Die Graphproduktionen werden aus den Abhängigkeitsbeziehungen der parallelen Operationen der speziellen Anwendung – also informell formulierten Regeln – abgeleitet. Bei den Knotenmarkierungen sind prinzipiell folgende Bedeutungen zu unterscheiden:

- Knotenmarkierungen, die auf die statische Struktur des Graphen Einfluß nehmen, indem von ihnen die Anwendbarkeit einer Graphproduktion abhängt
- Knotenmarkierungen, die den dynamischen Ablauf beeinflußen (Beispiele sind die Instanzen in Funktionsnetzen, die ein partielles Schalten ermöglichen, sowie die in Abschnitt 3.3 vorgestellten Knotenmarkierungen), und
- Knotenmarkierungen, denen eine auszuführende Aktion zugeordnet wird.

Durch Definition von Zuständen auf Knotenmarkierungen wird es möglich durch eine Folge von Zustandswechseln den dynamischen Ablauf zu beschreiben. Zusammen mit den durch die Generierung mittels Graphproduktionen festgelegten statischen Struktureigenschaften erreicht man ein deterministisches Schaltverhalten. Diese Eigenschaft läßt sich aufgrund der theoretischen Fundierung formal beweisen /Her 88c/. Dadurch kann eine prozeßinterne Aktivitätsverwaltung ohne Verklemmungsprobleme erzeugt werden. Wie am Beispiel in Abschnitt 3.3 erläutert wird, ist dies auch für kooperierende Prozeßsysteme möglich.
Die Graphstruktur (und damit implizit die Ablaufstruktur) kann durch Einfügen und Löschen von Graphproduktionen auf einfache Weise geändert werden, so daß die Wartung von Ablaufstrukturen sehr vereinfacht wird.

3.3 Ein Anwendungsbeispiel

In /Her 88a/ wird die automatische Parallelisierung von Aktivitäten in einem auf dem Client/Server–Konzept aufbauenden Datenbanksystem für nicht–konventionelle Anwendungen (NDBS) vorgestellt. Die Serverschnittstelle ist dabei als eine mengenorientierte Operationsschnittstelle realisiert. Der Client sendet eine Menge von Operationen an den Server, die dieser unabhängig von der Reihenfolge der Ankunft

der Aufträge bearbeiten kann. Auf Fertigmeldungen vom Server für einzelne Operationen reagiert der Client mit dem Senden von nun entsprechend der Abhängigkeitsbeziehungen ausführbaren Operationen.

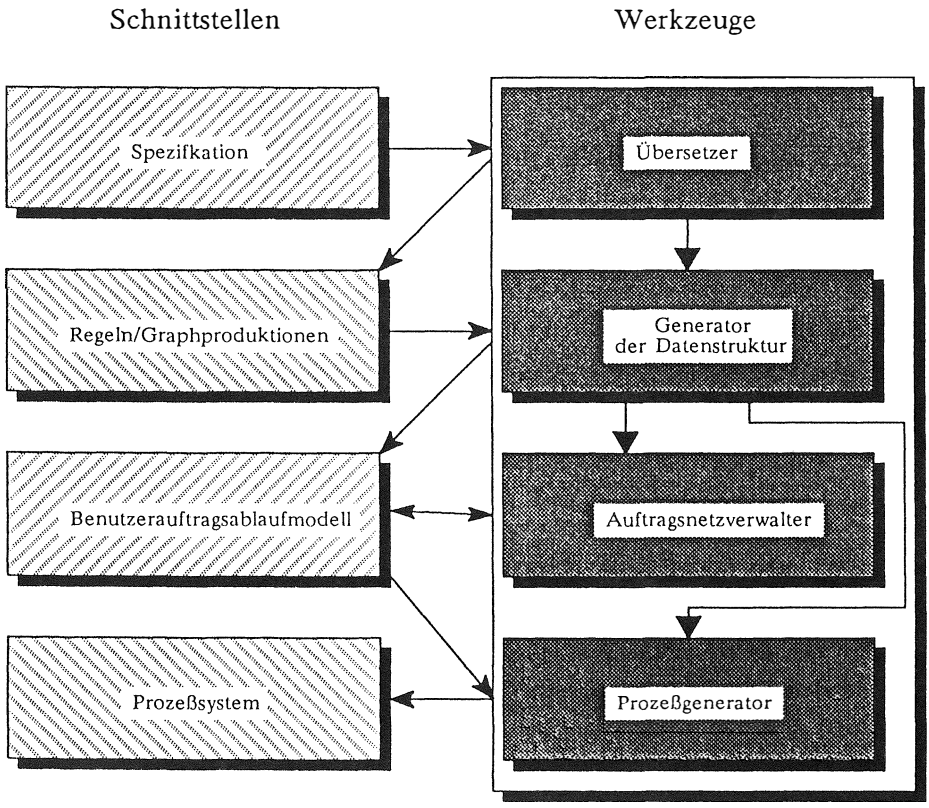

Bild 1: Entwicklungsunterstützung von Ablaufstrukturen

Der Entwerfer der zur Synchronisation der parallel zu bearbeitenden Serveraufträge notwendigen Ablaufstruktur wird durch vier aufeinander aufbauende Werkzeuge unterstützt:

- einem Übersetzer, der die Spezifikation der parallel ausführbaren Serveraufträge in eine Menge von Regeln überführt,
- einem regelbasierten Graphgenerator zur Erzeugung der zentralen Datenstruktur

- einem Generator eines entsprechend den spezifischen Abhängigkeitsbeziehungen kommunizierenden Prozeßsystems und
- alternativ zum Prozeßsystem der sog. Auftragsnetzverwalter, der eine prozeßinterne Aktivitätsverwaltung realisiert.

Durch anwendungsbezogene Sprachkonstrukte spezifiziert der Entwerfer, ob zwischen den Operationen Abhängigkeitsbeziehungen bestehen. Ist beim Entwurf schon sicher, daß solche Beziehungen nicht existieren (d.h. daß die Operationen parallel ausgeführt werden können), so wird dies durch die Schlüsselwörter (**OL ... OL**) und (**DL ... DL**) gekennzeichnet. Soll die Abhängigkeitsstruktur vom Übersetzer aus der Spezifikation abgeleitet werden, so ist dies durch (**OPAR .. OPAR**) zu spezifizieren. Der Übersetzer nutzt dazu Informationen über Eingangs- und Ausgangsgrößen der Serveroperationen und der spezifizierten Prozeduren aus. Das Prozedurkonzept erlaubt es, Abhängigkeitsbeziehungen zwischen Teilablaufstrukturen zu beschreiben. In Bild 2 ist ein fiktiver Rahmen eines Quellprogramms des Übersetzers dargestellt.

```
PROC MAIN ( in .... out ....)
          (OPAR     OpCALL  Suche ( in ... out ...),
                    OpCALL  Suche ( in ... out ...),
                    OpCALL  Prüfe  ( in ... out ...),
                    OpCALL  Lösche ( in ... out ...)
          OPAR)
ENDPROC

OpPROC Suche ( in ... out ...)
          (DL       RSC Suche_X1 ( in... out...)
                    RSC Suche_X2 ( in ... out ...)
          DL)
ENDPROC

OpPROC Prüfe ( in ... out ... )
          (OSEQ     RSC Prüfe_X1 ( in ... out ...),
                    RSC Prüfe_X2 ( in ... out ...)
          OSEQ)
ENDPROC

OpPROC Lösche ( in ... out ... )
          (OL       RSC Lösche_X ( in ... out ...),
                    RSC Lösche_Y ( in ... out ...)
          OL)
ENDPROC
```

Bild 2: Rahmen eines Quellprogramms des Übersetzers

Der Übersetzer bildet die Spezifikation auf eine Menge von Graphproduktionen ab. Durch Anwendung des Graphgenerators entsteht aus den Graphproduktionen eine Datenstruktur, die auf dem Benutzerauftragsablaufmodell (BA2M) aufbaut. Das BA2M wurde in /Her 88c/ zur Darstellung der Abhängigkeitsbeziehungen zwischen parallelen Aktivitäten in einem NDBS entwickelt. Ein Graph heißt BA2M, wenn er durch Anwendung der Produktionen der Graph-Grammatik für BA2M sequentiell aus dem Startgraphen abgeleitet werden kann und nur Knoten mit Markierungen aus dem terminalen Knotenmarkierungsalphabet enthält (Bild 3). Bis auf die terminale Knotenmarkierung der auszuführenden Serveroperation haben die Markierungen sowohl Einfluß auf die statische Struktur als auch – im Zusammenhang mit den weiter unten beschriebenen Auftragsnetzen – auf den dynamischen Ablauf. Bild 4 zeigt Beispiele für Graphproduktionen des BA2M, die die Erzeugung einer bestimmten Teilstruktur (1), einer bestimmten Form von Parallelität (2) und die die Generierung einer zusätzlichen Abhängigkeitsbeziehung (3) ermöglichen. Bild 5 zeigt ein vollständiges BA2M, das die im VLSI-Entwurf zur Darstellung und zur interaktiven Manipulation von Layouts benötigte Fensteroperation in einer möglichen Realisierungsform graphisch repräsentiert.

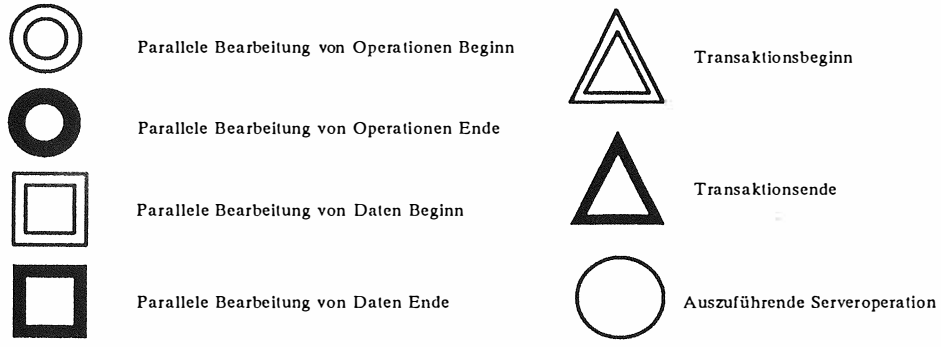

Bild 3: Darstellung des terminalen Knotenmarkierungsalphabets

Durch den Prozeßsystemgenerator wird ein BA2M auf zwei Arten von Prozessen abgebildet:

- Prozesse, die Operationsausführungen durch den Server kontrollieren und die nur eine aktive Phase haben, und
- Prozesse, die den parallelen Ablauf einzelner Teile des gesamten Auftrages initieren und kontrollieren.

Die Prozesse erster Art entsprechen im BA2M den Knoten mit der Markierung der auszuführenden Serveroperation. Jede 'Beginn'-'Ende'-Klammer in einem BA2M wird auf einen Prozeß zweiter Art abgebildet. Der Prozeßgenerator übersetzt die interne Darstellung eines BA2M in ein PEARL-Modul sowie in Kommandodateien zum Übersetzen und Laden dieses Moduls. Aus dem speziellen Sprachumfang von PEARL werden die Semaphoren als Synchronisationsmechanismus sowie Sprachkonstrukte zur Prozeßdeklaration und -verwaltung benutzt.

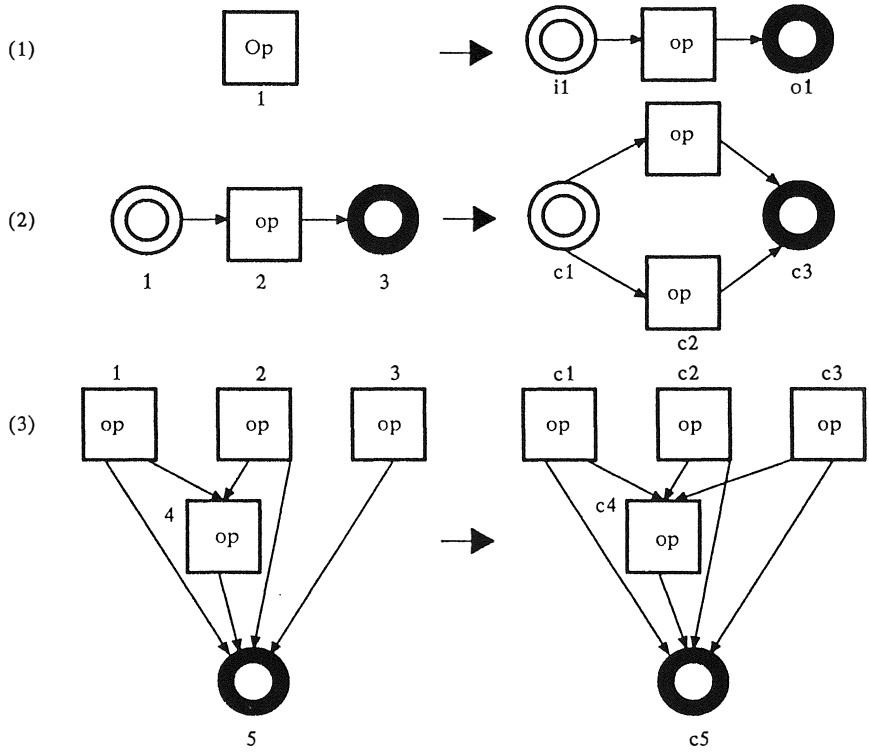

Bild 4: Beispiele für Graphproduktionen

Die Realisierung der Steuerung paralleler Abläufe durch eine prozeßinterne Aktivitätsverwaltung wird durch das Konzept der Auftragsnetze ermöglicht. Dazu werden auf den Knoten eines BA2M Zustände definiert. Die in Abhängigkeit von den Knotenmarkierungen festgelegten Zustandswechsel leisten die Synchronisation des Verarbeitungsvorgangs, wobei man sich prinzipiell ein Schaltverhalten wie bei Petri-Netzen vorstellen kann. Nimmt ein mit der Markierung der auszuführenden Serveroperation versehener Knoten den Zustand **gesendet** ein, so wird dem Server ein

Auftrag zur Ausführung übergeben. Anschließend sucht der Auftragsnetzverwalter nach weiteren Knoten, bei denen ein Zustandswechsel erfolgen kann. Durch eine Auftragsfertigmeldung vom Server wird in jedem Fall ein Fortschalten der Zustände ausgelöst. Auftragsnetze haben sich als geeignetes Konzept zur Erreichung einer adäquaten Fehlerbehandlung, die u.a. den Abbruch und die anschließende Wiederholung einer semantisch verbunden Menge von Operationen ermöglicht, erwiesen.

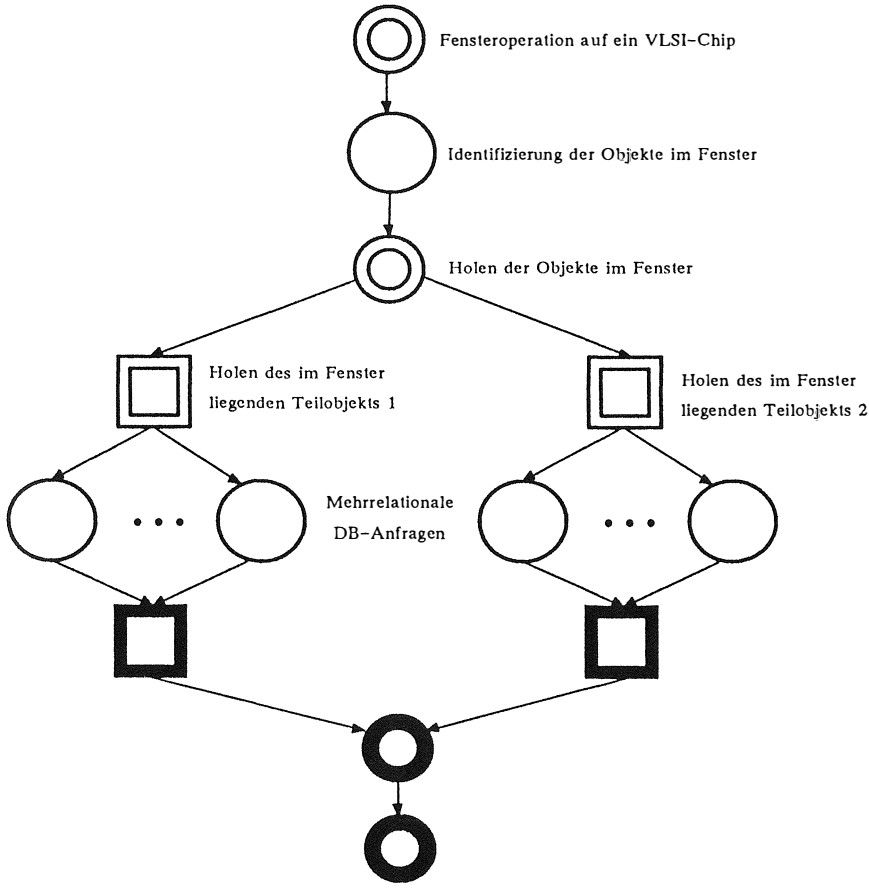

Bild 5: Beispiel für ein Benutzerauftragsablaufmodell

Die dritte Realisierungsform von parallelen Ablaufstrukturen kann durch eine Mischung zwischen Prozeßsystem und Auftragsnetzen erreicht werden. Es werden lediglich noch Prozesse der o.a. zweiten Art generiert, wobei die Prozeßanzahl

flexibel von der Schachtelungstiefe der Daten- und Operationsparallelität abhängig gemacht werden kann. Innerhalb dieser Prozesse verwalten Auftragsnetze den parallelen Verarbeitungsfluß.

3.4. Dynamische Generierung von parallelen Ablaufstrukturen

Der Softwareentwicklungsprozeß wird in eine Reihe von Phasen zerlegt:

Problemanalyse -> Entwurf -> Implementierung -> Test -> Installation -> Wartung

Das in dieser Arbeit vorgestellte Konzept zur Entwicklungsunterstützung von parallelen Ablaufstrukturen auf der Basis von Graphen als zentrale Datenstruktur sowie einer anwendungsbezogenen Graph-Grammatik zur Generierung dieser Datenstruktur unterstützt vier der genannten Phasen:

- *den Entwurf:* durch die Möglichkeit der Spezifikation auf der Basis von Eigenschaften
- *die Implementierung:* durch die Möglichkeit der automatischen Generierung von Ablaufstrukturen
- *den Test:* durch das durch die formale Fundierung und durch die automatische Generierung der Ablaufstrukturen garantierte "Schaltverhalten" und
- *die Wartung:* durch einfache Änderbarkeit der Graphstruktur und damit impliziert durch die vorgestellten Werkzeuge auch der Ablaufstruktur.

Somit bietet das vorgestellte Konzept eine gute Basis für die in Kap. 1 geforderte durchgängige Entwicklungsunterstützung. Da alle der in Abschnitt 3.1 dargestellten Möglichkeiten zur Implementierung der Steuerung von parallelen Abläufen unterstützt werden, ergibt sich ein gewisses Maß an Unabhängigkeit von den Einflußfaktoren bei der Entscheidung bzgl. der Implementierungsart.

4. Literatur

/CR 86/ Campbell, R.H.; Randell, B.: *"Error Recovery in synchronous Systems"*, IEEE Transactions on Software Engineering, Vol. SE-12, No. 8, August, 1986

/Dä 87/ Dähler, J. et al.: *"A Graphical Tool for the Design and Prototyping of Distributed Systems"*, ACM SIGSOFT, Software Engineering Notes, Vol. 12, No. 3, July, 1987

/Ga 85/ Gait, J.: *"A Debugger for Concurrent Programs"*, Software – Practice and Experience, Vol. 15, No. 6, June, 1985

/Go 83/ Godbersen, H.P.: *"Funktionsnetze – Eine Modellierungkonzeption zur Entwurfs- und Entscheidungsunterstützung"*, Ladewig Verlag, Birkbach, 1983

/GT 86/ Godbersen, H.P.; Trümner, H.: "*Ein graphisch orientierter Ansatz zur Bildung operationaler Modelle*", 16. GI–Jahrestagung, IFB 127, Springer–Verlag, Berlin, 1986

/Her 88a/ Herzog, H.: "*Automatische Parallelisierung von Aktivitäten in einem verteilten System*", Proc. 18.GI–Jahrestagung, Hamburg, Springer–Verlag, Berlin, 1988

/Her 88b/ Herzog, H.; Hildebrandt, F.: "*Überlegungen zur Entwicklung von Datenbankmaschinen für nicht konventionelle Anwendungen*", Informatik–Bericht 88–08, TU Braunschweig, 1988

/Her 88c/ Herzog, H.: "*Synchronisation kooperierender Aktivitäten in Nicht–Standard–Datenbanksystemen*", Dissertation, TU Braunschweig, 1988

/Ka 87/ Karp, A.H.: "*Programming for Parallelism*", IEEE Computer, Vol.20, No. 5, May, 1987

/Na 78/ Nagl, M.: "*Graph–Grammatiken. Theorie. Anwendungen. Implementierung*", Vieweg, Braunschweig, 1978

/Ol 85/ Olson, R.: "*Parallel Processing in a Message–Based Operating System*", IEEE Software, Vol.2, No. 4, July, 1985

/Re 83/ Reisig, W.: "*System Design using Petri Nets*", in: Hommel,G.; Krönig, D. (Hrsg.): "Rrequirements Engineering", Arbeitstagung der GI, IFB 74, Springer–Verlag, Berlin, 1983

/Rö 86/ Röhrich, J.: "*Parallele Systeme*", Informatik–Fachbericht 117, Springer–Verlag, Berlin, 1986

/Sch 86/ Schneider, H.–J.: "*Formale Gestaltungsaspekte in der Systementwicklung*", Handbuch der modernen Datenverarbeitung, Heft 130, 1986

/Wi 86/ Winter, D. et al.: "*ISAC – ein System zur Unterstützung der Systembeschreibung und Systemanalyse*", Handbuch der modernen Datenverarbeitung, Heft 130, 1986