TECHNISCHE
UNIVERSITÄT
MÜNCHEN

INSTITUT FÜR INFORMATIK

Sonderforschungsbereich 342:
Methoden und Werkzeuge für die Nutzung
paralleler Rechnerarchitekturen

Anleitung zur Spezifikation von mobilen, dynamischen Focus-Netzen

Ursula Hinkel und Katharina Spies

TUM-I9639
SFB-Bericht Nr.342/16/96 A
November 1996

TUM–INFO–11-96-I39-200/1.–FI

Anforderungen an: Prof. Dr. A. Bode
Sprecher SFB 342
Institut für Informatik
Technische Universität München
D-80290 München, Germany

Druck: Fakultät für Informatik der
Technischen Universität München

Anleitung zur Spezifikation von mobilen, dynamischen Focus-Netzen*

Ursula Hinkel und Katharina Spies
Institut für Informatik, Technische Universität München
Postfach 20 24 20, D-80333 München
e-mail: (hinkel|spiesk)@informatik.tu-muenchen.de

November 1996

Zusammenfassung

Focus steht für eine Methodik zur Entwicklung und Spezifikation verteilter Systeme. Mit dem klassischen Focus-Ansatz, siehe [BDD+93], konnten bisher ausschließlich statische Systemstrukturen beschrieben werden. Statische Systeme sind verteilte Systeme, deren Kanalverbindungen und Anzahl vorhandener Komponenten nicht variieren. Im vergangenen Jahr wurde der semantische Ansatz um die Theorie der mobilen stromverarbeitenden Funktionen erweitert, mit der es nun möglich ist, auch dynamische Systemstrukturen zu beschreiben, siehe [GS96b], [GS96c], [GS96a] und [Gro96b]. In Fallstudien ist die Anwendbarkeit dieser Erweiterung zu zeigen. Zusätzlich sind Anleitungen für den Umgang mit den vorgestellten Techniken wünschenswert, die es dem Anwender erleichtern, mit dem neuen Ansatz auch praktisch zu arbeiten.

Im vorliegenden Papier wird eine Anleitung zur Spezifikation von dynamischen, verteilten Systemen anhand der mobilen stromverarbeitenden Funktionen aus der Erweiterung von Focus gegeben. Das Papier wendet sich an Leser, die bereits mit den Grundlagen von Focus vertraut sind und sich in die Spezifikation mobiler, dynamischer Netze anhand eines Beispiels einarbeiten möchten. Zunächst erläutern wir den semantischen Ansatz und stellen Leitfäden für das Erzeugen und Löschen von Verbindungskanälen und für das Kreieren neuer Komponenten vor. Anhand eines abstrakten Batch-Systems mit sehr einfachen funktionalen Eigenschaften wird die Anwendung dieser Leitfäden demonstriert. Wir spezifizieren das Batch-System konstruktiv und verwenden hierfür Funktionsgleichungen und Tabellen, anhand derer der Leser in der Lage sein sollte, die vorgestellten Spezifikationen möglichst direkt nachzuvollziehen.

*Diese Arbeit wurde unterstützt vom Sonderforschungsbereich 342 „Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen“ und von Siemens-ZFE im Rahmen des SysLab-Projekts.

Inhaltsverzeichnis

1 Einleitung

Bei der Spezifikation und Beschreibung komplexer, verteilter Systeme und deren Verhalten ist es oft notwendig, das Verhalten des Systems zu erweitern und dieses durch neue Systemkomponenten zu modellieren. Neue Systemkomponenten sollen auf eine systematische, kontrollierte und nachvollziehbare Art und Weise in die bestehende Systemstruktur eingefügt werden können. Diese Erzeugung muß formal beschreibbar sein und sich problemlos in die bestehende Formalisierung des Systems einfügen lassen. FOCUS steht für eine Methodik zur Entwicklung und Spezifikation verteilter Systeme. In FOCUS werden verteilte Systeme durch Netzwerke von Komponenten beschrieben, die über gerichtete Kanäle miteinander verbunden sind und durch Nachrichtenaustausch miteinander kooperieren. Das semantische Modell wird über Mengen stromverarbeitender Funktionen definiert. Um mit dieser Modellierung auch dynamisches Verhalten und mobile Systemstrukturen beschreiben zu können, muß die Möglichkeit zur Veränderung der Systemstruktur also im Rahmen einer Erweiterung dieses semantischen Modells definiert werden. Ein bekannter Ansatz zur Beschreibung von Mobilität ist der π-Kalkül, siehe [MPW92a] und [MPW92b]. Angeregt dadurch wurde der klassische Ansatz von FOCUS, siehe [BDD+93], um Möglichkeiten zur Variation der Struktur eines Komponentennetzwerks und zur Modellierung dynamischen Verhaltens erweitert. Hierzu zählen das Erzeugen bzw. Löschen von Kommunikationsverbindungen sowie das Erzeugen bzw. Löschen von Komponenten. Die Erweiterung des semantischen FOCUS-Modells wird detailliert in [GS96b], [GS96a], [GS96c] und [Gro96b] beschrieben.

FOCUS-Kanäle werden durch eindeutige Bezeichner mit entsprechender Typinformation identifiziert. Neue Kommunikationsverbindungen können im Modell der mobilen, dynamischen FOCUS-Netze durch das Versenden von Kanalbezeichnern versehen mit einem Lese- oder Schreibrecht – sogenannten Ports – erzeugt werden. Komponenten können Ports auch weiterleiten; da eine Komponente einen weitergeleiteten Port sofort vergißt, können Verbindungen zwischen Komponenten hergestellt werden, die vorher nicht direkt miteinander verbunden waren. Es ist daneben auch möglich, bestehende Kommunikationsverbindungen zu löschen, indem eine Komponente ein bestehendes Lese- oder Schreibrecht wieder an den Verbindungspartner zurückschickt. Bei der Erzeugung neuer Kommunikationsverbindungen ist zu beachten, daß eine neue Verbindung zwischen Komponenten nur erzeugt werden kann, wenn zwischen diesen bereits ein Pfad, bestehend aus bereits existierenden Kommunikationsverbindungen, vorhanden ist. Diese Modellierung ist jedoch durchaus adäquat, da in praktischen Anwendungen nicht erwartet wird, daß Information dort fließen kann, wo keinerlei Kommunikationsverbindungen bestehen.

Komponenten werden in Kombination mit einem Verfeinerungsschritt (strukturelle Verfeinerung) erzeugt. Eine Komponente erzeugt eine Kindkomponente, indem sie zu einem Netzwerk, bestehend aus der erzeugenden und der neu erzeugten Komponente, verfeinert wird. Dieser Verfeinerungsschritt ist mit den für das semantische Modell definierten Verfeinerungsbeziehungen in Einklang zu bringen. Das Löschen einer Komponente kann modelliert werden, indem die Schnittstelle der Komponente, die sich aus den Ein- und Ausgabekanälen zusammensetzt, durch das Zurücksenden der entsprechenden Ports vollständig gelöscht wird. Da sich das Verhalten einer FOCUS-Komponente aus der Relation zwischen Ein- und Ausgabenachrichten ergibt, kann das Verhalten einer Komponente ohne Schnittstelle von außerhalb der Komponente nicht mehr beobachtet werden. Die Komponente ist nicht mehr in die Systemstruktur eingebunden und liefert keinen sichtbaren Beitrag zum Systemverhalten.

Im vorliegenden Bericht geben wir nach einer Einführung in das semantische Modell der mobilen, dynamischen Systeme eine Anleitung, wie die formale Spezifikation derartiger Systeme systematisch erfolgen kann. Dazu stellen wir schematisch anwendbare Leitfäden für das Erzeugen und Löschen von Kommunikationsverbindungen sowie für das Erzeugen und Löschen von Komponenten vor. Ein typisches Beispiel im Bereich der mobilen Systeme stellen die zellularen Telefone dar. Um zu zeigen, daß mobile, dynamische Systemeigenschaften auch in anderen Anwendungsbereichen von Interesse sind, demonstrieren wir die von uns vorgeschlagene Vorgehensweise anhand der formalen Spezifikation eines Batch-Systems mit einfachen funktionalen Eigenschaften. Die Spezifikationen werden konstruktiv erstellt und zum Teil tabellarisch dargestellt, so daß der Leser in der Lage sein sollte, die vorgestellten Spezifikationen direkt nachzuvollziehen. Weitere Beispiele für eine Anwendung der Leitfäden und Spezifikationsstile finden sich unter anderem auch in [Spi] oder [Hin].

Der vorliegende Bericht wendet sich an Leser, die bereits über Grundkenntnisse zu FOCUS verfügen und sich in die Spezifikation mobiler, dynamischer Netze einarbeiten und diese anhand eines Beispiels nachvollziehen möchten . Für eine Einführung in die grundlegenden Konzepte von FOCUS, wie zum Beispiel Ströme und stromverarbeitende Funktionen, sowie in die methodische Entwicklung verteilter Systeme mit FOCUS verweisen wir auf [BDD+93] und [BS].

Der Bericht ist wie folgt aufgebaut:

Kapitel 2 stellt die Erweiterung von FOCUS vor, die die Modellierung mobiler, dynamischer Netze ermöglicht und damit die semantische Basis für die Erzeugung von Kanälen und Komponenten bildet. Wir geben eine Einführung in die wichtigsten Konzepte des semantischen Modells.

In Kapitel 3 erläutern wir die semantischen Konzepte an einem einführenden Beispiel. Darüberhinaus entwickeln wir Leitfäden, die das Vorgehen bei der Erzeugung neuer Kommunikationsverbindungen und Komponenten aufzeigen, und somit den Anwender bei der Spezifikation unterstützen. Außerdem beschreiben wir, wie sich das Löschen von Kommunikationsverbindungen bzw. von Komponenten modellieren läßt.

In Kapitel 4 wird die Spezifikation eines Batch-Systems mit einfachen funktionalen Eigenschaften entwickelt. Wir spezifizieren ein System, das anfangs nur aus einer Komponente besteht. Während des Systemablaufs wächst die Anzahl der Komponenten nach einem festen Strukturierungsschema. Wir beschreiben das System zunächst informell und geben für die einzelnen Phasen des Systemablaufs Beschreibungen in Form von Tabellen an. Anschließend folgt die formale Spezifikation des Systems, wobei wir wieder auf eine tabellarische Darstellung zurückgreifen.

Der Bericht schließt mit einer kurzen Zusammenfassung der Ergebnisse der vorliegenden Arbeit und einem Vergleich zu anderen Ansätzen zur Spezifikation dynamischer Systeme.

2 Wichtige semantische Konzepte und Begriffe

Die klassische Version von FOCUS ([BDD+93]) ist eine Entwurfs- und Spezifikationsmethodik für statische, verteilte Systeme. Darunter verstehen wir Systeme, deren Komponenten feste Schnittstellen haben, deren Netzwerkstruktur während des Systemablaufs nicht verändert wird und in denen während des Systemablaufs keine Komponenten erzeugt bzw. gelöscht werden können. Grundlegende mathematische Konzepte in FOCUS sind Ströme, die die Kanalgeschichten modellieren, und stromverarbeitende Funktionen, die das Verhalten von Komponenten spezifizieren. Unter einer Kanalgeschichte verstehen wir die Folge von Nachrichten, die über einen Kanal gesendet werden.

In [GS96b] und [GS96c] wird als Erweiterung von FOCUS eine semantische Basis für die Modellierung mobiler Netze definiert, die in [Gro96b] um die dynamische Erzeugung von Komponenten erweitert wird. Dieser Ansatz ermöglicht es nun, Systeme zu spezifizieren, deren Kommunikationsstruktur sich während des Systemablaufs verändert und in denen neue Komponenten dynamisch kreiert und in die bisherige Systemstruktur integriert werden können. Die Semantik des neuen Ansatzes basiert nicht wie beim klassischen FOCUS auf der ordnungstheoretischen Bereichstheorie, sondern auf der topologischen Bereichstheorie.

In den folgenden Abschnitten stellen wir die wesentlichen Konzepte des semantischen Modells der mobilen, dynamischen Systeme vor. Wir erläutern diese Konzepte überwiegend informell, so daß auch ein mit mathematischen Formeln wenig vertrauter Leser in der Lage sein sollte, die später vorgestellten Spezifikationen nachzuvollziehen. Für eine detailliertere Einführung verweisen wir auf die bereits oben angegebene Literatur.

2.1 Mobile Netze

Mobile Systeme sind dadurch gekennzeichnet, daß die Komponenten ihre Kommunikationspartner aufgrund von Interaktionen und Berechnungen wechseln können. In [GS96c] wird ein denotationelles Modell für gezeitete, nichtdeterministische, mobile Systeme eingeführt. In diesem Modell wird ein System als Menge von Komponenten definiert, die über gerichtete Kanäle asynchron miteinander kommunizieren. Im vorliegenden Bericht spezifizieren wir Systeme mit **point-to-point Kommunikation**. Diese Eigenschaft stellt sicher, daß ein Kanal von genau zwei Komponenten verwendet wird: eine Komponente schreibt auf den Kanal, die andere liest von ihm. Eine Erweiterung dieser Kommunikationsform ist die

many-to-many Kommunikation, die erlaubt, daß mehrere Komponenten gleichzeitig lesend oder schreibend auf einen Kanal zugreifen dürfen. Mobile Netze mit many-to-many Kommunikation werden in [GS96a] behandelt.

Vollständige Kanalgeschichten beschreiben den Signalfluß über einen Kanal und werden durch unendliche Ströme über endlichen Nachrichtensequenzen repräsentiert. Wir betrachten verteilte Systeme mit einem globalen Zeitkonzept. Wir gehen davon aus, daß eine globale Uhr existiert, die die Zeitachse in diskrete Zeitintervalle unterteilt. Die Länge der Zeitintervalle kann dabei abhängig von dem zu spezifizierenden System beliebig fein gewählt werden. Jede endliche Nachrichtensequenz stellt die Kommunikationsgeschichte innerhalb eines Zeitintervalls dar. Komponenten und Netzwerke werden durch Mengen stromverarbeitender Funktionen beschrieben.

Die Komponenten in einem Netzwerk sind statisch durch eine Menge gerichteter Kanäle fest verbunden, wobei jede Komponente jeden Kanal als Ein- bzw. Ausgabekanal verwenden kann. Auf welche Kanäle eine Komponente tatsächlich zugreifen kann, wird durch die Vergabe von Rechten auf den Kanälen geregelt, wodurch Kommunikationsverbindungen zwischen Komponenten eingerichtet werden. Im Gegensatz zu den Kanälen sind diese Verbindungen nicht statisch und können während des Systemablaufs verändert werden; hierdurch wird **Mobilität** modelliert. Diese Eigenschaft stellt den wesentlichen Unterschied zwischen dem klassischen und dem erweiterten FOCUS-Ansatz dar.

Die Vergabe der Rechte auf den Kanälen wird modelliert, indem die Komponenten sogenannte Ports versenden können. Ein **Port** ist ein Kanalname, der zusätzlich mit einem Zugriffsrecht auf den Kanal (Lesen („?“) oder Schreiben („!“)) versehen ist. Zwischen zwei Komponenten kann eine neue Kanalverbindung erzeugt werden, wenn die beiden Komponenten bereits direkt über einen Kanal verbunden sind oder - falls keine direkte Verbindung vorliegt - eine indirekte Kanalverbindung – **Pfad** – über andere Komponenten existiert.

Die Menge der Nachrichten, die die Komponenten austauschen, wird um Ports erweitert. Sei D die Menge aller Nachrichten, M die Menge aller verfügbaren Kanalnamen. $?M = \{?m \mid m \in M\}$ ist die zu M gehörige Menge aller Leseports, $!M = \{!m \mid m \in M\}$ die zu M gehörige Menge aller Schreibports. Damit besteht die Menge der Nachrichten, die eine Komponente auf ihren Eingabeports erhalten kann, aus $D \cup !M \cup ?M$. Für $!M \cup ?M$ schreiben wir auch $?!M$; mit $?!m$ kürzen wir $\{?m\} \cup \{!m\}$ ab.

Zu Beginn verfügt eine Komponente über initiale Kanalverbindungen, genannt **initial wiring**, die aus einer Menge I von Eingabekanälen und einer Menge O von Ausgabekanälen besteht. Jeder Komponente steht eine Menge von privaten Kanalnamen P mit $(I \cup O) \cap P = \emptyset$ zur Verfügung, die sie für das Erzeugen neuer Kanalverbindungen verwenden kann. Um sicherzustellen, daß die Komponenten Kanalverbindungen mit unterschiedlichen Namen erzeugen, gilt, daß die Mengen der privaten Kanalnamen aller Komponenten disjunkt sind. Diese Eigenschaft ist eine wesentliche Voraussetzung für die point-to-point Kommunikation zwischen den Komponenten.

Während des Systemablaufs wächst bzw. verringert sich die Menge der verfügbaren Ein-und Ausgabekanäle der Komponente. Wenn die Komponente eine Nachricht $!j$ verschickt, wobei sie den Kanal j selbst erzeugt hat ($j \in P$), so darf sie später den Inhalt des Kanals j lesen. Schickt sie eine Nachricht $?p$, wobei sie den Kanal p selbst erzeugt hat ($p \in P$), so darf sie Nachrichten auf diesem Kanal senden. Erhält die Komponente den Port $?q$, so darf sie Nachrichten vom Kanal q lesen; erhält die Komponente den Port $!q$, so darf sie auf den Kanal q schreiben. Unter dem Weiterleiten („**Forwarden**“) von Ports verstehen wir, daß eine Komponente Ports, die sie nicht selbst erzeugt, sondern empfangen hat, an andere Komponenten weiterleitet. Um sicherzustellen, daß auf einen Kanal nur eine Komponente schreibt, wird gefordert, daß sich die Komponenten **privacy preserving** verhalten: die Komponente greift nur auf Ports zu und sendet nur Ports, die ihr zur Verfügung stehen. Somit hängt das Verhalten der Komponente nur von den Ports ab, die ihr zur Verfügung stehen. Sobald die Komponente einen Port gesendet hat, steht er ihr nicht mehr zur Verfügung. Damit ist das sogenannte **Channel Sharing** möglich: mehrere Komponenten können nacheinander das Schreibrecht auf dem gleichen Kanal erhalten; Interferenz, also das gleichzeitige Schreiben auf einen Kanal, ist hingegen ausgeschlossen. Es handelt sich demnach ausschließlich um eine Erweiterung der point-to-point Kommunikation. Mit der Weitergabe von Ports und dem Channel Sharing können bestehende Kanalverbindungen umgelenkt werden. Wir verwenden im vorliegenden Bericht die Variante der mobilen, dynamischen Netze, die point-to-point Kommunikation mit Channel Sharing ermöglicht, siehe [GS96b].

Um die dynamische Veränderung der Schnittstelle einer mobilen Komponente zu erfassen und zu kontrollieren, auf welche Kanäle eine Komponente aktuell zugreifen darf, werden die Mengen pp und ep im semantischen Modell eingeführt. ep enthält sämtliche Ports, auf die die Komponente aktuell lesend oder schreibend zugreifen kann. Diese Ports werden auch öffentliche (external) Ports genannt. pp enthält dagegen die Menge aller Ports, die nur der Komponente selbst bekannt sind; diese Ports werden als private Ports bezeichnet. Die Mengen pp und ep verändern sich, sobald die Komponente Ports empfängt oder versendet. Da mobile Komponenten durch mobile Funktionen modelliert werden, siehe folgender Abschnitt, wird die Aktualisierung von pp und ep durch die Eigenschaft privacy preservation im semantischen Modell durch deren inkrementelle Definition sichergestellt (siehe [GS96b]) . Zu Beginn enthält die Menge ep die Lese- bzw. Schreibrechte auf die statischen Ein- bzw. Ausgabekanäle der Komponente. Die Menge pp enthält die Schreib- und Leserechte für sämtliche Kanäle, die in der Menge der privaten Kanalnamen vorhanden sind.

Ein öffentlicher Port wird wieder privat, wenn eine Komponente einen Port empfängt, dessen Komplement sie als Ein- bzw. Ausgabeport besitzt. Damit besitzt die Komponente sowohl Lese- als auch Schreibrecht zu dem dazugehörigen Kanal. Das Komplement des empfangenen Ports wird aus der Menge ep gelöscht, und sowohl Lese- als auch Schreibport werden in die Menge pp eingefügt. Mit diesem Vorgehen kann das Löschen eines Kanals aus dem Netzwerk modelliert werden.

2.2 Grundlagen des semantischen Modells

In diesem Abschnitt erklären wir die wesentlichen mathematischen Grundlagen des semantischen Modells. Für detailliertere Definitionen verweisen wir auf die vorher erwähnten Literaturreferenzen.

Sei N die Menge aller verfügbaren Kanalnamen. Dann ist $?!N$ die Menge aller daraus ableitbaren Ports. Sei D die Menge von Nachrichten, die keine Ports darstellen. Die Menge aller Nachrichten M ist somit $?!N \cup D$.

$[M^*]$ bildet die Menge aller vollständigen Kanalgeschichten über M, d.h. die Menge aller unendlichen Ströme über den endlichen Nachrichtensequenzen über M. Wie bereits im vorhergehenden Abschnitt erwähnt, liegt dem semantischen Modell ein globaler Zeitbegriff zugrunde. Verzichten wir auf die Zeitinformation, die in vollständigen Kanalgeschichten gegeben ist, so erhalten wir ungezeitete Ströme M^ω über M. Sei $m \in [M^*]$ eine vollständige Kanalgeschichte. Dann bezeichnet $\overline{m}$ einen Strom aus M^ω, bei dem alle endlichen Sequenzen aus m zu einem einzigen Strom zusammengefaßt sind. Es handelt sich demnach bei $\overline{m}$ um einen endlichen oder unendlichen Strom, wohingegen vollständige Kanalgeschichten unendliche Ströme über endlichen Sequenzen darstellen.

Sei N die Menge der Kanalnamen. Die Abbildung $N \to [M^*]$ ordnet jedem Kanalnamen eine vollständige Kanalgeschichte über der Menge M zu. Wir sprechen dabei von benannten Kanalgeschichten. Sind den einzelnen Kanalnamen unterschiedliche Nachrichtenmengen zugeordnet, so verwenden wir folgende Schreibweise: Das kartesische Produkt $\prod\limits_{k \in N} [S_k^*]$ ordnet jedem Kanalnamen k aus der Menge aller im System vorhandenen Kanalnamen N eine vollständige Kanalgeschichte über der Nachrichtenmenge S_k zu, wobei S_k die für den Kanal k definierten Nachrichten enthält. Mit dem kartesischen Produkt werden somit Tupel von Kanalgeschichten modelliert.

Eine stromverarbeitende Funktion $f \in (N \to [M^*]) \to (N \to [M^*])$ bildet die vollständigen benannten Kanalgeschichten der Eingabekanäle auf vollständige benannte Kanalgeschichten der Ausgabekanäle ab. Die Funktion heißt **stark pulsgetrieben**, wenn die Eingabe bis zum Zeitintervall i die Ausgabe bis zum Zeitintervall $i+1$ festlegt. Damit hängt die Ausgabe in jedem Zeitintervall j von der Eingabe bis zum Zeitintervall $j-1$ ab; eine pulsgetriebene Funktion verarbeitet ihre Eingabe somit inkrementell – die Ausgabe hängt nicht von zukünftigen Eingaben ab. Stark pulsgetriebene Funktionen werden mit dem Symbol $\stackrel{s}{\to}$ gekennzeichnet. Da jede Komponente im Systemablauf theoretisch auf alle vorhandenen Kanäle zugreifen kann, werden sämtliche möglichen Kanäle in die Definition der Funktionalität mitaufgenommen. Aufgrund der privacy preservation greift die Funktion jedoch zu jedem Zeitpunkt nur auf diejenigen Kanäle zu, die ihr aktuell zur Verfügung stehen, deren Ports also in ihrer aktuellen Menge *ep* liegen.

Eine Funktion $f \in (N \to [M^*]) \to (N \to [M^*])$ wird als **mobil** bezüglich (I, O, P) bezeichnet (wobei $I \cap O = P \cap (I \cup O) = \emptyset$), wenn sie stark pulsgetrieben ist und die Eigenschaften der privacy preservation erfüllt. Ihr Verhalten hängt also nur von Ports ab, die sie bereits kennt, und Ports werden vergessen, sobald sie weitergegeben worden sind. Für mobile Funktionen wird das Symbol $\xrightarrow{I,O,P}$ eingeführt.

Das Verhalten einer mobilen Komponente wird durch eine Menge mobiler Funktionen definiert[1] (siehe folgender Abschnitt). Für die Modellierung von Netzwerken mobiler Komponenten wird der Kompositionsoperator $\otimes$ verwendet. Wenn mobile Komponenten auf diese Weise komponiert werden, besitzt die so gewonnene Komponente ebenfalls die Mobilitätseigenschaft.

2.3 Spezifikationen mobiler Komponenten

Das Verhalten einer mobilen Komponente wird durch eine Menge mobiler Funktionen spezifiziert. Diese Menge wird durch ein Prädikat definiert. Mit $[\![S]\!] = \{f \mid P_S.f\}$ bezeichnen wir die Semantik der Spezifikation S. Diese umfaßt die Menge aller mobilen Funktionen f, die die Spezifikation S erfüllen. P_S ist ein Prädikat, mit dem das Verhalten der spezifizierten Komponente beschrieben wird.

Wie vorher bereits erläutert, betrachten wir Systeme, die auf der Basis eines globalen Zeitkonzepts modelliert werden. Da die Modellierung expliziten Zeitverhaltens in vielen Anwendungen nicht unbedingt erforderlich ist, werden für die Erstellung von Spezifikationen zwei Formate angeboten: zeitunabhängige und zeitabhängige Spezifikationen.

In **zeitunabhängigen Spezifikationen** wird von der Zeitinformation in den Strömen abstrahiert; es werden ungezeitete Ströme sowie Funktionen auf ungezeiteten Strömen verwendet. Eine so spezifizierte Komponente verhält sich bezüglich der Zeit beliebig. Die Semantik einer zeitunabhängigen Spezifikation ist die Menge aller mobilen Funktionen, die die Spezifikation erfüllen, wenn von der Zeitinformation in den Strömen abstrahiert wird.

Wir spezifizieren das in Kapitel 4 modellierte Batch-System ausschließlich zeitunabhängig. In den Prädikaten verwenden wir einen konstruktiven Spezifikationsstil auf der Basis von rekursiven Funktionsgleichungen. Eine Funktionsgleichung genügt im wesentlichen folgendem Schema:

$$\begin{aligned} &f\,(state)\,(\{in_1 \to m_1,\, in_2 \to m_2,\, \ldots,\, in_n \to m_n\}\,\&\,s) \;=\\ &\qquad\qquad \{out_1 \to n_1,\, out_2 \to n_2,\, \ldots,\, out_m \to n_m\}\,\&\,f\,(state')\,(s) \end{aligned}$$

Die Funktion f erhält auf ihren Eingabekanälen $in_1, \ldots, in_n$ die Nachrichten $m_1, \ldots, m_n$, reagiert darauf mit der Ausgabe von $n_1, \ldots, n_m$ auf ihre Ausgabekanäle $out_1, \ldots, out_m$ und arbeitet mit dem Reststrom s weiter. Zusätzlich kann die Funktion über einen Zustandsparameter $state$ verfügen, der durch die Verarbeitung der Eingabenachrichten in $state'$ übergeht.

[1] Zusätzlich gilt, daß diese Menge geschlossen ist. Da diese Eigenschaft hier nicht von Bedeutung ist, verweisen wir für eine nähere Erklärung auf [GS96c].

Mit diesem Kapitel haben wir nun die wesentlichen Konzepte des semantischen Modells der mobilen, dynamischen Systeme vorgestellt. Für eine detailliertere Einführung verweisen wir nochmals auf die angegebene Literatur.

3 Ein einführendes Beispiel und Leitfäden

In diesem Abschnitt geben wir zur Veranschaulichung der in Kapitel 2 vorgestellten semantischen Konzepte und Begriffe ein einführendes Beispiel, in dem die Kanalverbindungen eines Systems dynamisch geändert werden. Wir zeigen, wie Kanäle erzeugt, gelöscht und weitergeleitet werden können. Darüberhinaus entwickeln wir Leitfäden zur Nutzung der Konzepte der mobilen, dynamischen Netze, die den Leser bei der Spezifikation des Erzeugens und Löschens von Kanälen bzw. Komponenten unterstützen sollen.

3.1 Beispiel zur Änderung der Vernetzung

Anhand des nachfolgenden Beispiels (siehe Abbildung 1) erläutern wir, wie Kanalverbindungen erzeugt und gelöscht werden und welchen Einfluß dies auf die Mengen *pp* und *ep* der einzelnen Komponenten hat. Wir stellen mehrere Schnappschüsse auf das System vor; das Verhalten der einzelnen Komponenten wird nicht näher spezifiziert.

Wir können für ein mobiles, dynamisches System **Phasen** angeben, in denen sich weder die Vernetzung der Komponenten untereinander durch Kanäle noch die Anzahl der im System existierenden Komponenten verändert. Das System verhält sich in diesen Phasen also im wesentlichen wie ein statisches System (siehe auch Abschnitt 4.3.2). Ein Schnappschuß zeigt die Struktur des Systems in einer bestimmten Phase.

Gegeben ist ein System S mit drei Komponenten (siehe Abb. 1). Phase 1 zeigt die initialen Kanalverbindungen zwischen K_1, K_2 und K_3. Jede dieser Komponenten verfügt über eine Menge von privaten Kanalnamen. In der nachfolgenden Tabelle sind für Phase 1 die Ein- und Ausgabekanäle, die privaten Kanalnamen sowie die Mengen *ep* und *pp* jeder Komponente angegeben:

	I	O	P	ep	pp
K_1	in, h_2	h_1	$\{a_i \mid i \in I\!N\}$[2]	$?in, ?h_2, !h_1$	$\{?!a_i \mid i \in I\!N\}$
K_2	h_1, h_4	h_2, h_3	$\{b_i \mid i \in I\!N\}$	$?h_1, ?h_4, !h_2, !h_3$	$\{?!b_i \mid i \in I\!N\}$
K_3	h_3	h_4, out	$\{c_i \mid i \in I\!N\}$	$?h_3, !h_4, !out$	$\{?!c_i \mid i \in I\!N\}$

Wir möchten folgende Änderungen an der Netzwerkstruktur vornehmen:

- Eine neue direkte Verbindung zwischen K_1 und K_3 erzeugen,
- den Eingabekanal in an K_2 anbinden und

[2]Mit $I\!N$ bezeichnen wir die Menge der natürlichen Zahlen ohne 0.

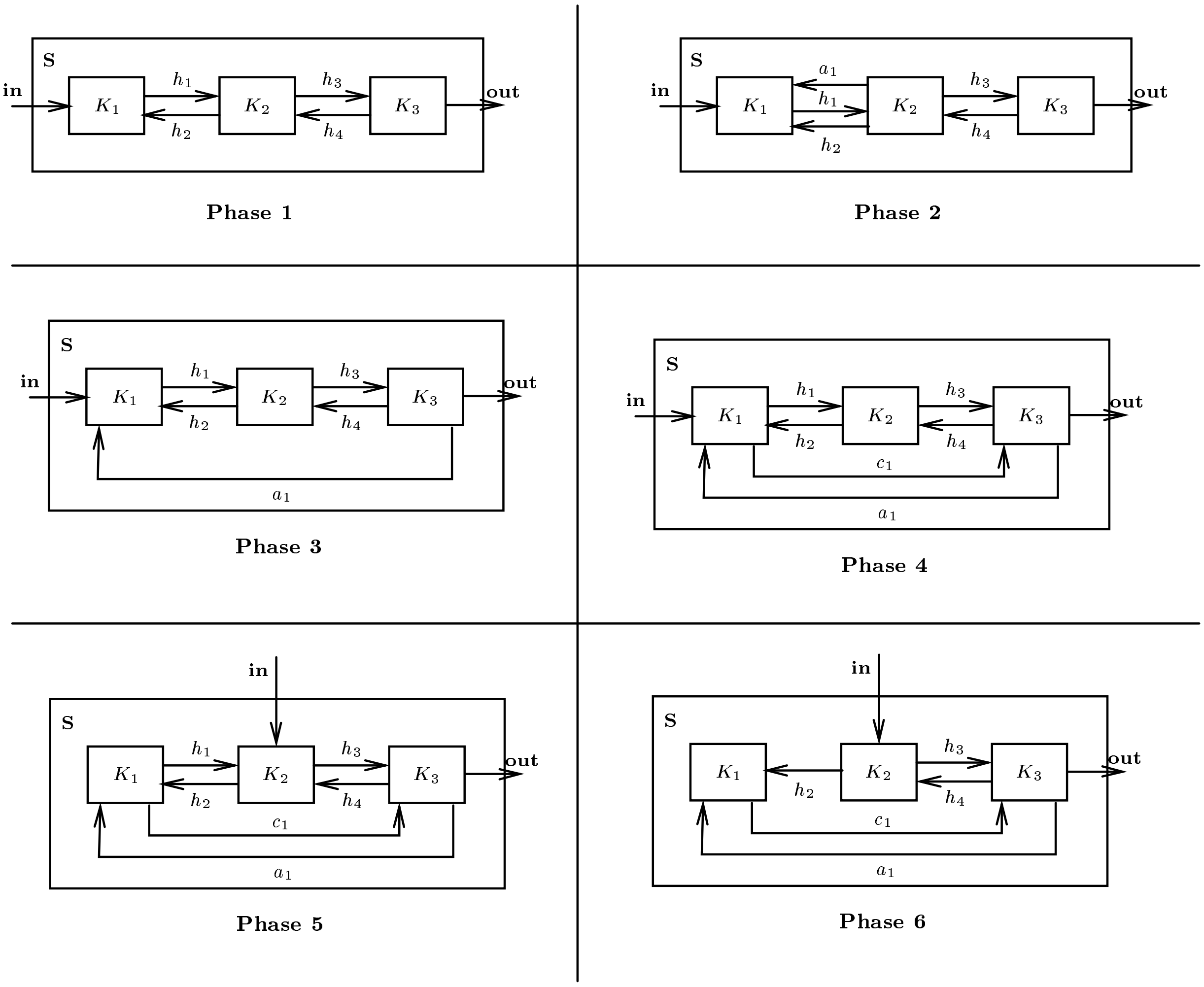

Abbildung 1: Systemkonfigurationen eines mobilen Netzes

- die interne Kanalverbindung h_1 zwischen K_1 und K_2 löschen.

Da die Mengen I, O und P die initiale Schnittstelle einer Komponente bilden, die Veränderungen an den Schnittstellen jedoch anhand der Mengen *ep* und *pp* zu beobachten sind, halten wir im folgenden nur die Veränderungen der Mengen *pp* und *ep* fest.

Im folgenden geben wir die Schritte an, die die Übergänge zwischen den einzelnen Phasen beschreiben.

Schritt 1: Komponente K_1 sendet den Port $!a_1$ an Komponente K_2. Damit wurde eine Kanalverbindung zwischen K_1 und K_2 erzeugt, von der K_1 lesen und auf die K_2 schreiben kann (siehe Phase 2).

	ep	pp
K_1	$?in, ?h_2, ?a_1, !h_1$	$\{?!a_i \mid i \in I\!N, i \geq 2\}$
K_2	$?h_1, ?h_4, !a_1, !h_2, !h_3$	$\{?!b_i \mid i \in I\!N\}$
K_3	$?h_3, !h_4, !out$	$\{?!c_i \mid i \in I\!N\}$

Schritt 2: Komponente K_2 leitet den Port $!a_1$ an Komponente K_3 weiter. Dabei vergißt sie ihn. $!a_1$ wird aus ihrer Menge *ep* gelöscht. Somit ist nun eine Kanalverbindung zwischen K_1 und K_3 vorhanden, auf die K_3 schreiben und von der K_1 lesen kann (siehe Phase 3).

	ep	pp
K_1	$?in, ?h_2, !h_1, ?a_1$	$\{?!a_i \mid i \in I\!N, i \geq 2\}$
K_2	$?h_1, ?h_4, !h_2, !h_3$	$\{?!b_i \mid i \in I\!N\}$
K_3	$?h_3, !h_4, !out, !a_1$	$\{?!c_i \mid i \in I\!N\}$

Schritt 3: Komponente K_3 sendet auf Kanal a_1 den Port $!c_1$ an Komponente K_1. Damit ist ein weiterer Kanal zwischen K_1 und K_3 erzeugt, für den K_1 das Schreibrecht und K_3 das Leserecht besitzt (siehe Phase 4).

	ep	pp
K_1	$?in, ?h_2, !h_1, ?a_1, !c_1$	$\{?!a_i \mid i \in I\!N, i \geq 2\}$
K_2	$?h_1, ?h_4, !h_2, !h_3$	$\{?!b_i \mid i \in I\!N\}$
K_3	$?h_3, !h_4, !out, !a_1, ?c_1$	$\{?!c_i \mid i \in I\!N, i \geq 2\}$

Schritt 4: Komponente K_1 sendet ihr Leserecht für den Eingabekanal *in* an Komponente K_2. Somit erhält nun K_2 die Eingaben aus der Systemumgebung (siehe Phase 5).

	ep	pp
K_1	$?h_2, !h_1, ?a_1, !c_1$	$\{?!a_i \mid i \in I\!N, i \geq 2\}$
K_2	$?h_1, ?h_4, !h_2, !h_3, ?in$	$\{?!b_i \mid i \in I\!N\}$
K_3	$?h_3, !h_4, !out, !a_1, ?c_1$	$\{?!c_i \mid i \in I\!N, i \geq 2\}$

Schritt 5: Um den Kanal h_1 zu löschen, sendet die Komponente K_2 der Komponente K_1 über den Kanal h_2 ihr Leserecht für Kanal h_1. Damit besitzt K_1 sowohl $?h_1$ als auch $!h_1$, so daß aus dem öffentlichen Kanal ein privater geworden ist (siehe Phase 6), was dem Löschen des Kanals entspricht (siehe auch Abschnitt 3.2).

	ep	pp
K_1	$?h_2, ?a_1, !c_1$	$?!h_1, \{?!a_i \mid i \in I\!N, i \geq 2\}$
K_2	$??h_4, !h_2, !h_3, ?in$	$\{?!b_i \mid i \in I\!N\}$
K_3	$?h_3, !h_4, !out, !a_1, ?c_1$	$\{?!c_i \mid i \in I\!N, i \geq 2\}$

3.2 Leitfäden für den Umgang mit mobilen, dynamischen Netzen

In diesem Abschnitt geben wir Leitfäden für die schrittweise und systematische Anwendung der bisher erläuterten Konzepte zur Spezifikation der mobilen, dynamischen FOCUS-Netze. Mit diesen Hilfestellungen und deren Demonstration an dem Beispiel eines Batch-Systems in Kapitel 4 sollte es für den Leser möglich sein, das Modell der mobilen, dynamischen FOCUS-Netze für die Spezifikation eigener Anwendungen zu verwenden.

3.2.1 Erzeugen eines neuen Kanals

Im folgenden geben wir einen **Leitfaden für das Erzeugen eines neuen Kanals** an. Ziel ist es, eine neue Kanalverbindung zwischen zwei existierenden Komponenten A und B zu erzeugen. Das zeitliche Verhalten der Komponenten wird nicht näher beschrieben. Wir können davon ausgehen, daß sämtliche Komponenten Nachrichten ohne Verzögerung empfangen und weiterleiten. Um eine neue Verbindung aufbauen zu können, muß bereits eine Verbindung zwischen A und B existieren. Hier sind folgende Fälle zu unterscheiden:

1. Zwischen A und B existiert eine direkte Kanalverbindung. Sei A diejenige der beiden Komponenten, die das Schreibrecht auf diesem Kanal besitzt. Dann übernimmt A die Rolle des Initiators für den Verbindungsaufbau. Falls mehrere Kanäle existieren und sowohl A als auch B Schreibrechte besitzen, kann auf beliebige Weise eine der beiden Komponenten gewählt werden.

2. Zwischen A und B existiert keine direkte Kanalverbindung. Es existiert jedoch eine Verbindung zwischen A und B (bzw. B und A), die über weitere Komponenten führt. Aufgrund dieser indirekten Verbindung („*Pfad*") ist es möglich, Nachrichten von A an B (B an A) zu senden. Dann übernimmt A bzw. B die Rolle des Initiators. Das Verhalten der Komponenten, die auf dem Pfad zwischen A und B liegen, ist so zu spezifizieren, daß sie den Port, der von A an B (bzw. B an A) zu senden ist, sofort nach dem Empfang direkt an die nächste Komponente, die auf dem Pfad liegt, weiterleiten.

3. Es liegt keine der beiden eben genannten Möglichkeiten vor. Dann kann eine Kanalverbindung zwischen A und B nur aufgebaut werden, wenn eine dritte Komponente existiert, die an A und B Nachrichten senden kann und die Initiative für den Verbindungsaufbau übernimmt.

4. Existiert keine Verbindung zwischen A und B, die einer der gerade drei beschriebenen Fälle entspricht, kann keine neue Kanalverbindung zwischen A und B erzeugt werden.

Wir gehen im folgenden davon aus, daß die Komponente A die Rolle des Initiators übernommen hat. Die Kanalerzeugung wird dann anhand des folgenden Schemas vorgenommen:

1. A wählt Lese- und Schreibrecht $?!k$ für einen Kanal k aus der Menge ihrer privaten Ports pp.

2. Soll A das Schreibrecht auf dem neu erzeugten Kanal behalten, so sendet sie den Port $?k$ an B. Soll A hingegen das Leserecht auf dem neu erzeugten Kanal behalten, so sendet sie den Port $!k$ an B. Das entsprechend komplementäre Zugriffsrecht auf den Kanal wird in die Menge ep aufgenommen. Aus der Menge pp werden beide Zugriffsrechte auf Kanal k entfernt.

3. Die Veränderungen der Mengen pp und ep werden in der Spezifikation der Komponente nicht explizit angegeben, sondern durch die Eigenschaft der Mobilität auf der

semantischen Ebene automatisch vorgenommen.

4. Falls der oben beschriebene dritte Fall vorliegt, muß folgerichtig eine Komponente C existieren, die sowohl an A als auch an B Nachrichten senden kann; ob direkt oder über weitere Komponenten spielt hierbei keine Rolle. Falls A auf den neu erzeugten Kanal schreibend und B lesend zugreifen sollen, entnimmt Komponente C Lese- und Schreibrecht $?!l$ aus der Menge ihrer privaten Ports und sendet das Schreibrecht für l an A und das Leserecht für l an B. Die Komponente C selbst hat somit kein Zugriffsrecht auf Kanal l. Verhält es sich umgekehrt, so sendet C das Schreibrecht an B und das Leserecht an A.

Wird eine Komponente mit einer leeren Menge privater Portnamen deklariert, so ist sie nicht in der Lage, selbständig neue Kanalverbindungen zu kreieren. Die einzige Möglichkeit, die Struktur der bestehenden Kanalverbindungen zu ändern, besteht darin, Ports von Kanälen, die die eigene aktuelle Schnittstelle bilden, weiterzuleiten.

3.2.2 Löschen eines Kanals

Es ist neben dem Erzeugen von neuen Kanalverbindungen auch möglich, bestehende Kanalverbindungen zu löschen, indem eine Komponente ein bestehendes Lese- oder Schreibrecht wieder an den Verbindungspartner zurückschickt.

Sei k ein Kanal, auf den die Komponente A lesend und die Komponente B schreibend Zugriff haben. Die Komponente A ergreift die Initiative für das Löschen des Kanals und sendet ihr Leserecht auf k, also $?k$, an die Komponente B. Somit ist $?k$ nicht länger in der Menge ep von A enthalten. Der Verbindungspartner B besitzt nun sowohl das Schreib- als auch das Leserecht zu Kanal k. Aus dem öffentlichen Kanal k ist somit ein privater Kanal geworden, auf den nur noch B zugreifen kann. Dies hat Auswirkungen auf die Mengen ep und pp von B: Das Schreib- und das Leserecht für k werden aus der Menge ep entfernt; die Menge pp wird um $?!k$ erweitert.

Es ist zu beachten, daß in der eben beschriebenen Situation ein Pfad zwischen A und B existieren muß, über den A den Port $?k$ an B senden kann. Ergreift hingegen B die Initiative zum Löschen des Kanals k, so ist über den Kanal k bereits eine direkte Kanalverbindung zwischen B und A gegeben, über die B senden kann. Diese Verbindung wird erst gelöscht, nachdem B sein Schreibrecht, nämlich $!k$, über eben diese Kanalverbindung k an A gesendet hat.

3.2.3 Erzeugen einer neuen Komponente

In [Gro96a] wird basierend auf dem Modell für mobile Netze die dynamische Erzeugung von Komponenten während des Systemablaufs eingeführt. Anhand der Modellierung eines Bankensystems ([Gro96b]) wird zudem gezeigt, wie sich die neu erzeugten Komponenten in den Systemablauf eingliedern lassen, indem die gegebene Kanalstruktur verändert wird. Einer Komponente können bei ihrer Erzeugung Ports als Parameter übergeben werden. Mit Hilfe dieser Ports wird der neu erzeugten Komponente eine initiale Schnittstelle zu bereits existierenden Komponenten mitgegeben, und zudem kann sie den Aufbau neuer Kanalverbindungen initiieren.

Im folgenden geben wir einen **Leitfaden für das Erzeugen einer neuen Komponente** an, mit Hilfe dessen das dynamische Erzeugen von Komponenten in ungezeiteten Spezifikationen schematisch durchgeführt werden kann. Eine semantische Fundierung des dynamischen Erzeugens von Komponenten findet sich in [Gro96a].

Folgende Situation sei zu spezifizieren: Die Komponente *Parent* soll nach Erhalt des Eingabesignals *init* eine Komponente *Child* erzeugen. Innerhalb der FOCUS-Spezifikation von *Parent* wird wie folgt vorgegangen:

$\exists\, child \in [[\, Child \,]]$

$\vdots$

$$parent\ (\{k_i \rightarrow init\}\ \&\ s) \ = \ (parent \,\overline{\otimes}\, child\ (M_{ports}))\ \ (\{k_{j1} \rightarrow t_{j1}, \ldots, k_{jl} \rightarrow t_{jl}\}\ \&\ s)$$

1. *parent* ist eine Funktion, die das Verhalten der Komponente *Parent* in der Spezifikation beschreibt.
2. Auf Kanal k_i trifft das Signal *init* ein.
3. Das Erzeugen der neuen Komponente *Child* wird durch den Aufruf der Funktion *child* modelliert.
4. Die Existenz einer solchen Funktion wird in der Spezifikation durch die Forderung $\exists\, child \in [[\, Child \,]]$ sichergestellt. *child* ist somit eine Funktion, die ein mögliches Verhalten der Komponente *Child* darstellt. Eine FOCUS-Spezifikation der Komponente *Child* muß vorliegen.
5. Die neu erzeugte Komponente *Child* wird über den Kompositionsoperator $\overline{\otimes}$ in die bestehende Systemstruktur eingebunden. Da *parent* eine Funktion auf ungezeiteten Strömen darstellt, *child* jedoch eine Funktion auf gezeiteten Strömen ist, verwenden wir eine Variante des Kompositionsoperators $\otimes$ aus Abschnitt 2.2. Dieser Operator stützt sich auf der Definition von $\otimes$ ab und ermöglicht es, mobile Funktionen mit Funktionen, die auf ungezeiteten Strömen arbeiten, zu komponieren. Das Ergebnis dieser Komposition liefert eine Funktion auf ungezeiteten Strömen. Auf diese Weise ist es möglich, auch in zeitunabhängigen Spezifikationen Komponenten neu zu kreieren ([Gro96c]).
6. Bei der Erzeugung von *Child* wird die Schnittstelle von *Child* initialisiert. Dazu erhält die Funktion *child* eine Menge von Ports, M_{ports}. Die in der Menge M_{ports} enthaltenen Ports stammen aus der Menge *ep* bzw. *pp* der erzeugenden Komponente

Parent. Mit diesen Ports kann *Child* Kanalverbindungen in der vorhandenen Systemstruktur aufbauen. Die zu den Ports gehörenden Kanäle bilden die statische Schnittstelle von *Child*.

7. Sowohl die Funktion *parent* als auch die Funktion *child* arbeiten auf dem Eingabestrom s weiter. Es ist allerdings möglich, der Komponente *Child* zusätzliche Nachrichten mitzugeben. Diese Nachrichten werden dem Strom s vorangestellt. Dabei müssen die Nachrichten den einzelnen Kanälen zugeordnet werden. Natürlich ist es nur sinnvoll, Nachrichten auf Kanälen zu senden, auf die *Child* zugreifen kann; das sind Kanäle, deren Ports in der Menge M_{ports} der Ports liegen, die *Child* von *Parent* übergeben wird.

   Für obige Gleichung gilt: $\{?k_{j1}, \ldots, ?k_{jl}\} \subseteq M_{ports}$. Auf den Kanälen k_{j1} bis k_{jl} werden die Nachrichten t_{j1} bis t_{jl} an *Child* gesendet. Diese Nachrichten kann nur *Child* lesen, da *Parent* für die Kanäle k_{j1} bis k_{jl} kein Leserecht besitzt.

3.2.4 Löschen einer Komponente

In der Semantik der mobilen, dynamischen Netze existiert kein Konzept, mit dem das Löschen von Komponenten direkt realisiert werden kann. Es ist allerdings möglich, das Löschen einer Komponente auf der Spezifikationsebene zu modellieren. Wir gehen dabei von folgender Überlegung aus: Das Verhalten einer Komponente wird als Beziehung zwischen den Nachrichten, die sie auf ihren Eingabekanälen erhält, und den Nachrichten, die sie als Reaktion darauf auf ihren Ausgabekanälen ausgibt, definiert. Löscht man nun die Schnittstelle einer Komponente, so kann diese weder Nachrichten empfangen noch Nachrichten ausgeben. Dadurch läßt sich das Verhalten der Komponente von außen nicht mehr beobachten: die Komponente weist kein sichtbares Verhalten mehr auf. Dies betrachten wir als adäquate Modellierung für das Löschen einer Komponente aus der vorhandenen Systemstruktur.

Das Löschen der Schnittstelle, die sich aus den Ein- und Ausgabekanälen der Komponente bildet, kann wie folgt modelliert werden: Die Komponente sendet die zu ihren Eingabekanälen gehörigen Leserechte an eine andere Komponente oder an die Umgebung. Dann sendet sie die zu ihren Ausgabekanälen gehörigen Schreibrechte über einen Kanal ebenfalls an eine andere Komponente oder an die Umgebung, zuletzt das Schreibrecht des Kanals, über den sie das Versenden vornimmt.

Damit besitzt die Komponente keine Rechte mehr, auf Ein- oder Ausgabekanäle zuzugreifen. Wie die anderen Komponenten mit den ihnen zugesandten Rechten der gelöschten Komponente verfahren, hängt vom Verhalten ab, das das spezifizierte System erfüllen soll. Es gibt keine Möglichkeit mehr, eine Kanalverbindung zu der „gelöschten" Komponente herzustellen.

4 Spezifikation des Batch-Systems

Im folgenden werden wir ein einfaches Batch-System spezifizieren, das sich mobil und dynamisch verhält, um die anstehenden Aufträge verteilt und gleichzeitig bearbeiten zu können. Von einer hier nicht näher spezifizierten Umgebung werden Aufträge an das System gegeben. Diese Aufträge werden jeweils ohne Unterbrechung ausgeführt; für die Bearbeitung eines Auftrags ist kein Dialog mit der Umgebung erforderlich. Das Batch-System besteht aus einer zentralen Komponente, die neue Komponenten erzeugt, wobei die sich dabei entwickelnde Systemstruktur eine einheitliche Strukturierung aufweist. Wir haben von den funktionalen Eigenschaften des Systems stark abstrahiert, um in erster Linie das Erzeugen und Löschen von Kanälen sowie das Erzeugen von Komponenten anhand der Leitfäden aus Abschnitt 3.2 zu demonstrieren. Es lassen sich jedoch durchaus Systeme in der Realität finden, deren Struktur und Verhalten unserem gewählten Beispiel nahe kommen. Zunächst geben wir eine informelle Beschreibung des Batch-Systems. Das dynamische Verhalten des Systems erläutern wir anhand von Phasen, die das System durchläuft, und geben dazu die jeweiligen Mengen der öffentlichen und privaten Ports zu den im System vorhandenen Komponenten an. Anschließend folgt die formale Spezifikation des Systems in Focus.

4.1 Informelle Beschreibung des Batch-Systems

Das zu modellierende Batch-System besteht anfangs aus einer zentralen Komponente *Zentrale*, die die Abarbeitung von Aufträgen, die sie aus der Umgebung *Env* erhält, organisiert und an die Umgebung Statusmeldungen zurückgibt. Da die Ausführung der Aufträge sehr rechenaufwendig ist und mehrere Aufträge gleichzeitig ausgeführt werden sollen, werden diese von Komponenten ausgeführt, die je nach Bedarf zur Ausführung dieser Aufträge erzeugt werden. Die Zentrale ist dadurch in der Lage, mehrere Aufträge gleichzeitig abzuwickeln, und übernimmt die Rolle eines Pförtners oder auch Vermittlers zwischen der Umgebung (den Auftraggebern) und dem Rechensystem, von dem die Aufträge bearbeitet werden. Die Kommunikation zwischen der Zentrale und dem Auftraggeber in der Umgebung wird nicht durch lange Berechnungen unterbrochen.

Als mögliches Anwendungsbeispiel für das eben beschriebene System ist eine einfache Rechenanlage vorstellbar. Der Benutzer meldet sich mit *Login* bei dem Rechner an. Die Login-Shell übernimmt die Rolle der Zentrale und erzeugt eine weitere Shell. Die Aufträge des Benutzers sind zum Beispiel der Start eines Editors oder das Kompilieren eines Programms. In einer Shell sind nur eine begrenzte Anzahl von Prozessen ausführbar, so daß bei wachsender Zahl der Aufträge von der Login-Shell weitere Shells erzeugt werden. Die Login-Shell gibt Nachrichten über den Systemzustand und von den Unterkomponenten erarbeitete Ergebnisse aus, die der Benutzer auf dem Monitor lesen kann.

Die Struktur des Systems entwickelt sich folgendermaßen: Die Zentrale erzeugt zunächst eine erste Subzentrale, an die sie die Aufträge aus der Umgebung weiterleitet. Diese Subzentrale erzeugt ihrerseits für jeden Auftrag eine Komponente *Slave*, die den Auftrag bearbeitet. Wir gehen davon aus, daß ein Prozeß unbegrenzt Zeit für die Bearbeitung des Auftrags benötigt und an die Umgebung Nachrichten ausgibt, die Informationen über die Abarbeitung des Auftrags enthalten, wie z. B. Zwischenergebnisse oder Statusmeldungen. Eine Subzentrale kann allerdings nur eine begrenzte Anzahl von *Slave*-Komponenten erzeugen. Ist ihre Kapazität erschöpft, meldet sie dies der Zentrale. Sobald daraufhin ein neuer Auftrag aus der Umgebung eintrifft, erzeugt die Zentrale eine neue Subzentrale, die das gleiche Verhalten wie die bereits vorhandenen Subzentralen aufweist. Die Aufträge werden nun solange an die neue Subzentrale weitergeleitet, bis deren Kapazität ebenfalls ausgelastet ist und die Zentrale eine weitere Subzentrale erzeugen muß. Da die Zahl der Subzentralen unbegrenzt ist, kann die Zentrale beliebig viele Aufträge aus der Umgebung annehmen und bearbeiten lassen. Wir modellieren das System so, daß die Subzentralen unterschiedliche Kapazitäten besitzen können, die von der Zentrale durch eine Nachricht k_i festgelegt werden. Damit läßt sich natürlich auch ein System modellieren, deren Subzentralen alle gleichgroße Kapazität besitzen. Wir gehen davon aus, daß eine Subzentrale keine Nachrichten mehr von der Zentrale erhält, sobald sie die ihr zustehenden Aufträge empfangen und die für die Abarbeitung notwendigen Komponenten erzeugt hat. Somit genügt es, wenn lediglich ein Kanal zwischen der Zentrale und der Subzentrale existiert, auf den die Zentrale schreiben kann, und zwar zu derjenigen Subzentrale, die die nächsten Aufträge erhalten wird. Eine Subzentrale, deren Kapazität erschöpft ist, sendet folglich ihr Leserecht auf dem Kanal zwischen ihr und der Zentrale zurück an die Zentrale, die es der nächsten neu erzeugten Subzentrale übergibt.

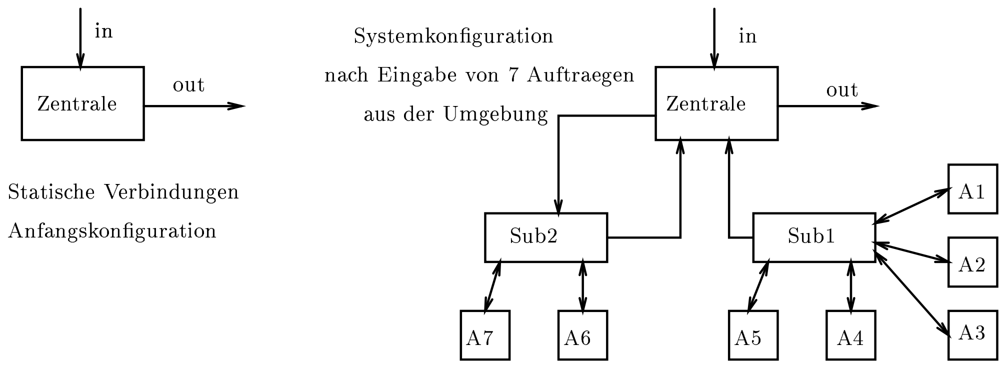

Abbildung 2: Entwicklung der Systemstruktur

Abbildung 2 zeigt, wie sich aus der anfangs statischen Systemkonfiguration eine Systemkonfiguration mit mehreren Subzentralen und Komponenten zur Auftragsbearbeitung entwickelt hat. In dem Beispiel liegt die Kapazität der ersten Subzentrale bei fünf Komponenten. Die sieben eingegangenen Aufträge wurden auf sieben *Slave*-Komponenten verteilt.

Das Verhalten des Systems wird aus der Systemumgebung angestoßen. Sobald aus der Systemumgebung das Signal *Login* eintrifft, bestätigt die Zentrale das erfolgreiche Einloggen mit dem Signal *Conf* an die Umgebung und erzeugt die erste Subzentrale. Die Aufträge aus der Umgebung werden durch die Nachrichten A_i repräsentiert. Die Zentrale teilt der Umgebung das Erzeugen neuer Subzentralen und neuer Slave-Komponenten über die Nachrichten $New(Sub_i)$ bzw. $New(Slave_{ij})$ mit und geben mit $A_i\_to\_Sub_i$ die Nachricht aus, an welche Subzentrale der Auftrag weitergegeben wurde. Die Slave-Komponenten geben mit den Nachrichten $State_{ij}$ Statusmeldungen aus, die von den Subzentralen an die Zentrale weitergegeben werden. Die Zentrale gibt diese Nachrichten an die Umgebung weiter. Nach dem Empfang eines Auftrags A_i senden die Subzentralen mit den Nachrichten $NotFull(Sub_i)$ bzw. $Full(Sub_i)$ Statusmeldungen über noch vorhandene bzw. bereits erschöpfte Kapazität an die Zentrale.

4.2 Die dynamische Entwicklung des Batch-Systems

In diesem Abschnitt werden die Kanalbezeichner und die zugehörigen Nachrichtentypen für alle im System auftretenden Komponenten deklariert. Wir verwenden die Tabellen zur Beschreibung der statischen Schnittstellen und der Mengen *ep* und *pp*, die wir bereits in Kapitel 3 für das einführende Beispiel verwendet haben.

Abbildung 3 zeigt, wie das System während des Systemablaufs wächst und seine Kanalstruktur verändert.

Kanäle und Nachrichten:

Sei N die Menge aller vorhandenen Kanalnamen:

$$N \;=\; \{in, out\} \;\cup\; \{sin\} \;\cup\; \{sout_i \,|\, i \in I\!N\} \;\cup\; \{bin_{ij} \,|\, i,j \in I\!N\} \;\cup\; \{bout_{ij} \,|\, i,j \in I\!N\}$$

Die folgende Tabelle gibt die Nachrichtenbelegung S_n der einzelnen Kanäle $n \in N$ an, wobei die Kanalverbindungen zum Teil erst während des Systemablaufs erzeugt werden. Jede Nachrichtenmenge enthält die Menge $?!N$ aller Ports, die sich aus der Menge N ableiten lassen und während des Systemablaufs zwischen den Komponenten ausgetauscht werden können.

Kanal $n \in N$	Nachrichten S_n
in	$\{Login\} \cup \{A_k \mid k \in I\!N\} \cup ?!N$
out	$\{Conf\} \cup \{New(Sub_i) \mid i \in I\!N\} \cup \{New(Slave_{ij}) \mid i,j \in I\!N\}$ $\cup \{A_k\_to\_Sub_j \mid k,j \in I\!N\} \cup \{State_{ij} \mid i,j \in I\!N\} \cup ?!N$
sin	$\{A_k \mid k \in I\!N\} \cup ?!N \cup \{k_i \mid i \in I\!N\}$
$sout_i$	$\{New(Slave_{ij}) \mid i,j \in I\!N\} \cup \{State_{ij} \mid i,j \in I\!N\}$ $\cup \{Full(Sub_i),\ NotFull(Sub_i) \mid i \in I\!N\} \cup ?!N$
bin_{ij}	$\{A_k \mid k \in I\!N\} \cup ?!N$
$bout_{ij}$	$\{State_{ij} \mid i,j \in I\!N\} \cup ?!N$

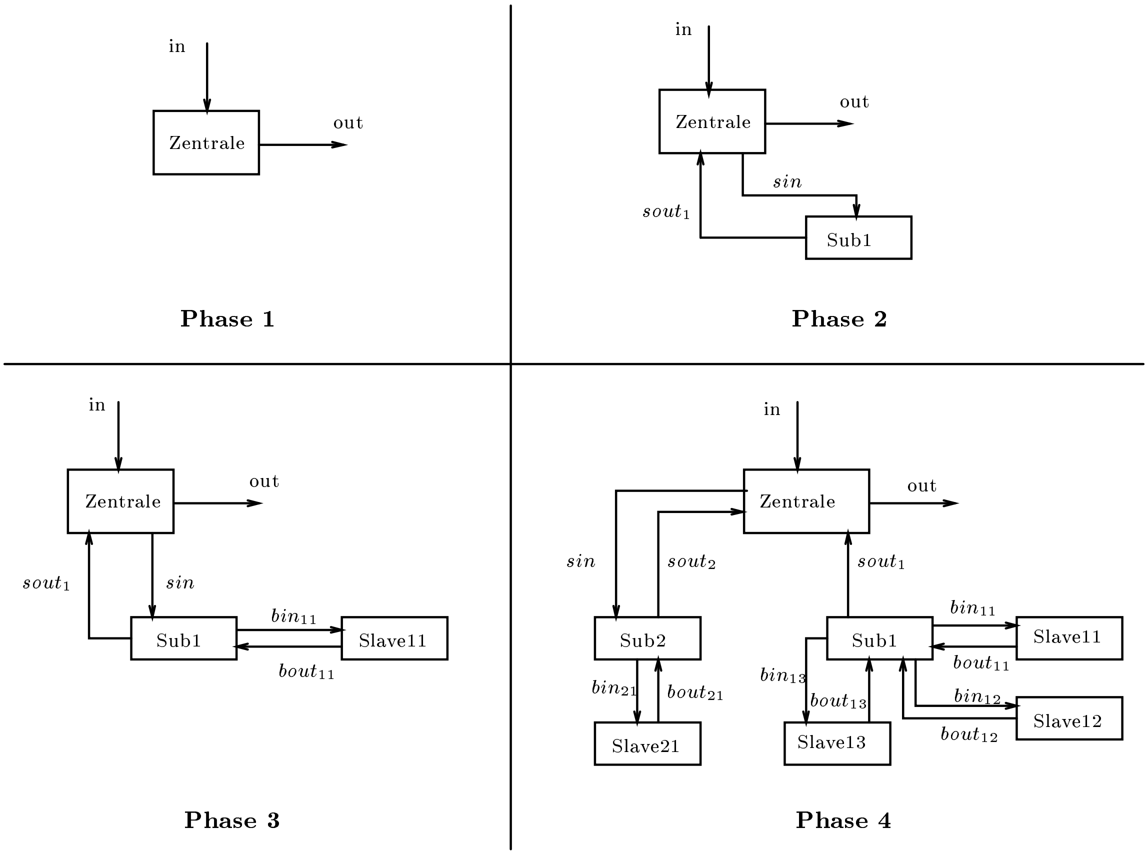

Abbildung 3: Dynamische Entwicklung des Systems

Statische Schnittstellen und private Kanalnamen:

Die folgende Tabelle gibt für alle im Verlauf des Systems vorkommenden Komponenten die Mengen I, O und P an. Die Ports, die die initiale Schnittstelle einer neu erzeugten Komponente definieren, werden von der Elternkomponente als Parameter übergeben.

	I	O	P
$Zentrale$	in	out	$\{sin\} \cup \{sout_i \,\vert\, i \in I\!N\}$
Sub_i	sin	$sout_i$	$\{bin_{ij} \mid j \in I\!N\} \cup \{bout_{ij} \mid j \in I\!N\}$
$Slave_{ij}$	bin_{ij}	$bout_{ij}$	$\emptyset$

Systemablauf:

Abbildung 3 zeigt einige Schnappschüsse aus dem Systemablauf. Für die Phasen 1 bis 4 beschreiben wir kurz das Verhalten des Systems und geben die Mengen *pp* und *ep* für die jeweils vorhandenen Komponenten an.

Phase 1 zeigt die Anfangskonfiguration des Systems. Einzig vorhandene Komponente ist die *Zentrale*.

	ep	pp
Zentrale	$?in,\ !out$	$\{?!sin\} \cup \{?!sout_i \mid i \in I\!N,\ i \geq 1\}$

Phase 2: Aus der Umgebung hat die Komponente *Zentrale* die Nachricht *Login* erhalten und daraufhin die Komponente Sub_1 erzeugt, die die Funktion einer Subzentrale übernimmt. Bei der Erzeugung von Sub_1 werden die Ports $?sin_1$ und $!sout_1$ als Parameter übergeben, so daß eine Schreib- und Leseverbindung zwischen der Zentrale und der Subzentrale aufgebaut wird.

	ep	pp
Zentrale	$?in,\ !out,\ ?sout_1,\ !sin$	$\{?!sout_i \mid i \in I\!N,\ i \geq 2\}$
Sub_1	$?sin,\ !sout_1$	$\{?!bin_{ij} \mid j \in I\!N\} \cup \{?!bout_{ij} \mid j \in I\!N\}$

Phase 3: Aus der Umgebung ist ein Auftrag A_1 eingetroffen, den die *Zentrale* an die Subzentrale Sub_1 weitergeleitet hat. Für die Bearbeitung dieses Auftrags hat die Subzentrale die Komponente $Slave_{11}$ erzeugt und den Auftrag an sie weitergeschickt. Als Parameter wurden bei der Erzeugung die Ports $?bin_{11}$ und $!bout_{11}$ übergeben, so daß eine Verbindung zwischen der Subzentrale und der neuen Komponente existiert.

	ep	pp
Zentrale	$?in,\ !out,\ ?sout_1,\ !sin$	$\{?!sout_i \mid i \in I\!N,\ i \geq 2\}$
Sub_1	$?sin,\ !sout_1,\ ?bout_{11},\ !bin_{11}$	$\{?!bin_{1j} \mid j \in I\!N, j \geq 2\} \cup \{?!bout_{1j} \mid j \in I\!N, j \geq 2\}$
$Slave_{11}$	$?bin_{11},\ !bout_{11}$	$\emptyset$

Phase 4: Die Kapazität der Subzentrale Sub_1 ist mit der Erzeugung von 3 ausführenden Komponenten ausgeschöpft, da wir hier die Kapazität 3 gewählt haben. Sie meldet dies der Zentrale durch die Nachricht $Full(Sub_1)$ und sendet ihr Leserecht auf Kanal *sin* zurück an die Zentrale. Sobald ein neuer Auftrag aus der Umgebung eintrifft, erzeugt diese eine weitere Subzentrale Sub_2 und übergibt ihr das Leserecht auf Kanal *sin*. Dann leitet sie den Auftrag an die neue Subzentrale weiter. Diese erzeugt wiederum eine Komponente, welche die Bearbeitung des Auftrags übernimmt.

	ep	pp
Zentrale	$?in,\ !out,\ ?sout_1,\ !sin$	$\{?!sout_i \mid i \in I\!N,\ i \geq 3\}$
Sub_1	$!sout_1,\ ?bout_{11},\ !bin_{11},$ $?bout_{12},\ !bin_{12},\ ?bout_{13},\ !bin_{13}$	$\{?!bin_{1j} \mid j \in I\!N, j \geq 4\} \cup \{?!bout_{1j} \mid j \geq 4\}$
$Slave_{11}$	$?bin_{11},\ !bout_{11}$	$\emptyset$
$Slave_{12}$	$?bin_{12},\ !bout_{12}$	$\emptyset$
$Slave_{13}$	$?bin_{13},\ !bout_{13}$	$\emptyset$
Sub_2	$?sin,\ !sout_2,\ ?bout_{21},\ !bin_{21}$	$\{?!bin_{2j} \mid j \in I\!N, j \geq 2\} \cup \{?!bout_{2j} \mid j \in I\!N, j \geq 2\}$
$Slave_{21}$	$?bin_{21},\ !bout_{21}$	$\emptyset$

4.3 Formale Spezifikation des Batch-Systems

Im folgenden werden die Komponenten bzw. das System, das in Kapitel 4.1 informell beschrieben wurde, mit dem in Kapitel 2 erklärten Ansatz zur Modellierung dynamischer Netze auf der Basis von FOCUS formal spezifiziert. Hierzu werden die funktionalen Eigenschaften der Komponenten *Zentrale*, der Subzentralen Sub_i und der ausführenden Komponenten $Slave_{ij}$ (mit $i, j \in I\!N$) zunächst informell beschrieben und anschließend in eine formale Spezifikation umgesetzt. Die Erstellung dieser formalen Spezifikation hält sich – bezogen auf die dynamischen Eigenschaften des Systems – an die in Kapitel 3 angegebenen Leitfäden für den Umgang mit mobilen, dynamischen Netzen. Die Spezifikation wird in einem konstruktiven Spezifikationsstil durch Gleichungen über mobilen Funktionen angegeben. Da wir keine expliziten Zeitbedingungen an das Verhalten der Zentrale stellen, spezifizieren wir die Zentrale mit dem zeitunabhängigen Spezifikationsformat (siehe auch Abschnitt 2.3).

4.3.1 Die Zentrale

Die *Zentrale* des Batch-Systems beschreibt den statischen Anteil des hier spezifizierten Systemaufbaus. Die initiale Schnittstelle der Zentrale wird durch den Eingabekanal *in* und den Ausgabekanal *out* definiert. Die Zentrale ist für die Erzeugung der Subzentralen zuständig und bildet die Verbindung des gesamten Systems zu seiner Umgebung durch die Entgegennahme der Aufträge A_i und die Ausgabe der diversen Statusmeldungen. Für das Verhalten der Komponente *Zentrale* des Batch-Systems und die dynamische Entwicklung des gesamten Systems gelten folgende Eigenschaften:

1. Das Verhalten des gesamten Systems (und dies gilt insbesondere für die Spezifikation der Zentrale) wird nur für den Fall spezifiziert, daß über den Eingabekanal *in* der Zentrale die Nachricht *Login* empfangen wird.

2. Auf den Empfang der Nachricht *Login* reagiert die Zentrale mit dem Senden der Nachrichten *Conf* (als Bestätigung für den Start des Systems) und $New(Sub_1)$ (als Information über die Erzeugung der ersten Subzentrale) über den Ausgabekanal *out* an die Umgebung und kreiert die erste Subzentrale Sub_1 mit den Kanälen *sin* und $sout_1$. (Der Eingabekanal *sin* wird jeweils der Subzentrale zur Verfügung gestellt, die die aktuellen Aufträge entgegennimmt.) Als erste Nachricht erhält die Subzentrale eine natürliche Zahl k_1 über den Kanal *sin*, mit der ihre Kapazität festgelegt wird.

3. Auf den Empfang eines Auftrags A_m aus der Umgebung über Kanal *in* reagiert die Zentrale wie folgt: sie sendet A_m über Kanal *sin* an die aktuelle Subzentrale Sub_1 und die Nachricht $A_m\_to\_Sub_1$ (zur Information) über Kanal *out* an die Umgebung. Die Zentrale leitet einen Auftrag A_m, der von Kanal *in* gelesen wurde, erst dann an eine Subzentrale weiter, wenn sichergestellt ist, daß die Kapazität der aktuellen

Subzentrale noch nicht erschöpft ist. Dies kann sie anhand der Statusmeldungen $NotFull(Sub_1)$ bzw. $Full(Sub_1)$ überprüfen, die sie von der Subzentrale erhält.

4. Wenn die Zentrale die Nachricht $New(Slave_{1j})$ oder die Statusmeldung $State_{1j}$ über den Kanal $sout_1$ empfängt, so leitet sie diese Nachrichten (zur Information) über Kanal *out* an die Umgebung weiter. (Diese Eigenschaft gilt für $j \in \{1, 2, \cdots, k_1\}$.)

5. Sobald die Zentrale die Nachrichten $Full(Sub_1)$ und $?sin$ über Kanal $sout_1$ erhält, registriert sie, daß die Kapazität der ersten Subzentrale erschöpft ist. Liegt nun ein weiterer Auftrag A_m an *in* an, wird die nächste Subzentrale Sub_2 kreiert. Die Meldungen $New(Sub_2)$, $A_m\_to\_Sub_2$ und alle anliegenden Statusmeldungen der übrigen Subzentralen werden nach außen gegeben. Die neue Subzentrale erhält über Kanal *sin* den Auftrag A_m.

Die Eigenschaften, die wir für die Zentrale und ihre Beziehung zur ersten Subzentrale beschrieben haben, treffen auch auf die Beziehungen zu den weiteren Subzentralen zu, d.h. sie gelten somit für $i, j, m, k_i \in I\!N$ mit $1 \leq j \leq k_i$.

Wir gehen bei der Modellierung davon aus, daß eine Komponente die an ihren aktuellen Eingabekanälen vorliegenden Nachrichten simultan liest und dann gemäß der Spezifikation reagiert.

Zunächst wird die Schnittstelle der Komponente festgelegt. Zur Spezifikation der Komponente wird das Prädikat $P_{Zentrale}$ definiert. Die in diesem Prädikat eingeführte Funktion g wird in einem konstruktiven Spezifikationsstil angegeben, indem wir die in den informellen Anforderungen vorgegebenen Eigenschaften jeweils durch Funktionsgleichungen über der mobilen Funktion g formulieren. Die Nummern vor den Gleichungen verweisen auf die zuvor informell aufgeführten Eigenschaften der Zentrale. Bei der Erzeugung von Kanälen und mobilen Komponenten halten wir uns an die in Kapitel 3 vorgestellten Leitfäden. Mit den Gleichungen $(3'), (4')$ und $(5')$ werden die informellen Anforderungen $3, 4$ und 5 für die Phasen der Zentrale spezifiziert, in denen bereits mehrere Subzentralen und Slave-Komponenten im Batch-System existieren.

Spezifikation der Komponente Zentrale

Initiale Schnittstelle:

$$I \triangleq \{in\} \qquad \text{(statische Eingabekanäle)}$$

$$O \triangleq \{out\} \qquad \text{(statische Ausgabekanäle)}$$

Private Kanalnamen:

$$P \triangleq \{sin\} \cup \{sout_i \mid i \in \mathbb{N}\}$$

Semantik der Spezifikation:

$$[\![\ Zentrale\ ]\!] = \{\ f :: TypeZ_{I,O,P} \mid P_{Zentrale}.f\ \}$$

mit dem Prädikat

$$P_{Zentrale} :: TypeZ_{I,O,P} \longrightarrow \mathbb{B}$$

wobei

$$TypeZ_{I,O,P} \triangleq \prod_{n \in N} [S_n^*] \xrightarrow{I,O,P} \prod_{n \in N} [S_n^*]$$

(Signatur der mobilen Funktionen)

$$\overline{TypeZ}_{I,O,P} \triangleq \prod_{n \in N} S_n^\omega \xrightarrow{I,O,P} \prod_{n \in N} S_n^\omega$$

(Signatur mit Zeitabstraktion)

Prädikat zur Verhaltensbeschreibung:

$$P_{Zentrale}\ (\ in\ \rhd\ out\ \square\ \{\ \{sin\} \cup \{sout_i \mid i \in \mathbb{N}\}\ \}\ ).f$$

$$\triangleq\ \forall\, \theta \in \prod_{n \in N} [S_n^*], \exists\, g :: \overline{TypeZ}_{I,O,P}\ :\ \overline{f(\theta)} = g(\overline{\theta})$$

Die Spezifikation des Prädikats erfolgt zeitunabhängig und

durch Angabe von Funktionsgleichungen für g

Funktionsgleichungen zur Definition von g

$\forall\ s, s', t \in \prod_{n \in N} S_n^\omega, \quad \forall i, j, k_i, l \in I\!N, l \geq 2, \ \ \forall\ w_i \in \{New(Slave_{ij}), State_{ij}\}$

$\exists\ h :: \overline{TypeZ}_{I,O,P}, \exists\ sub_i \in [\![\ Sub_i\ ]\!]\ :$

Initialisierung:

(1) $s = \{in\ \to\ Login\}\ \&\ s' \quad \Longrightarrow$

(2) $g(s) \ = \ \{out\ \to\ Conf\&New(Sub_1)\}\ \&$
$(h\ \overline{\otimes}\ sub_1(?sin, !sout_1))(\{sin\ \to\ k_1\}\ \&\ s')$

Die erste Subzentrale:

(3) $h(\{in\ \to\ A_i, sout_1\ \to\ NotFull(Sub_1)\}\ \&\ t)$
$= \ \{out\ \to\ A_i\_to\_Sub_1, sin\ \to\ A_i\}\ \&\ h(t)$

(4) $h(\{in\ \to\ A_i, sout_1\ \to\ w_1\}\ \&\ t)$
$= \ \{out\ \to\ w_1\}\ \&\ h(\{in\ \to\ A_i\ \}\ \&\ t)$

(5) $h(\{in\ \to\ A_i, sout_1\ \to\ Full(Sub_1)\&?sin\}\ \&\ t)$
$= \ \{out\ \to\ New(Sub_2)\}\ \&$
$(\ h\ \overline{\otimes}\ sub_2(?sin, !sout_2)\ )(\{sin\ \to\ k_2,\ in\ \to\ A_i\}\ \&\ t)$

Alle weiteren Subzentralen:

(3′) $h(\{in\ \to\ A_i,$
$sout_1\ \to\ State_{1j}, \cdots, sout_{l-1}\ \to\ State_{l-1j}, sout_l\ \to\ NotFull(Sub_l)\}\ \&\ t)$
$= \ \{out\ \to\ A_i\_to\_Sub_l\&State_{l-1j}\&\cdots\&State_{1j}, sin\ \to\ A_i\}\ \&\ h(t)$

(4′) $h(\{in\ \to\ A_i,$
$sout_1\ \to\ State_{1j}, \cdots, sout_{l-1}\ \to\ State_{l-1j}, sout_l\ \to\ w_l\}\ \&\ t)$
$= \ \{out\ \to\ w_l\&State_{l-1j}\&\cdots\&State_{1j}\}\ \&\ h(\{in\ \to\ A_i\ \}\ \&\ t)$

(5′) $h(\{in\ \to\ A_i,$
$sout_1\ \to\ State_{1j}, \cdots, sout_{l-1}\ \to\ State_{l-1j}, sout_l\ \to\ Full(S_l)\&?sin\}\ \&\ t)$
$= \ \{out\ \to\ New(Sub_{l+1})\&State_{l-1j}\&\cdots\&State_{1j}\}\ \&$
$(\ h\ \overline{\otimes}\ sub_{l+1}(?sin, !sout_{l+1})\ )(\{sin\ \to\ k_{l+1},\ in\ \to\ A_i\ \}\ \&\ t)$

Im folgenden geben wir einige Anmerkungen zur Erläuterung der oben aufgeführten Formeln an. Wir verweisen jedoch zum besseren Verständnis der Spezifikation vor allem auf Kapitel 3. Die einzelnen Gleichungen stellen die direkte Formalisierung der in Kapitel 4.1 angegebenen informellen Beschreibung dar. Aus diesem Grund verzichten wir hier auf detaillierte Erläuterungen zu jeder einzelnen Gleichung.

- Die Punkte (1) und (2) spezifizieren die Initialisierung unseres Batch-Systems. Wir geben die Spezifikation ausschließlich für den Fall an, in dem sich die Umgebung mit einer *Login*-Nachricht über Kanal *in* bei der Zentrale anmeldet. Die Statusmeldungen werden an die Umgebung ausgegeben. Das Verhalten des Systems wird im folgenden durch die Spezifikation eines Netzwerks, bestehend aus der *Zentrale* als Initiator-Komponente und der ersten *Subzentrale* als erzeugte Komponente, angegeben. Dieser Schritt erfolgt exakt gemäß der in Kapitel 3 angegebenen Vorgehensweise. Über den neu eingeführte Kommunikationsverbindung *sin* wird initial die Nachricht k_1 mitgegeben; k_1 wird explizit dem Strom vorangestellt, mit dem die Berechnung fortgesetzt wird.
- Wir beschreiben die Zentrale ausschließlich in der Initialisierungsphase durch die mobile, stromverarbeitende Funktion g. Sobald die Abarbeitung der Aufträge gestartet wurde, wird die Zentrale durch die Funktion h spezifiziert.
- Die Erzeugung weiterer Subzentralen erfolgt nach dem gleichen Schema in den Gleichungen (5) bzw. $(5')$.
- Solange ein von der Umgebung vorliegender Auftrag durch die Zentrale aufgrund des aktuellen Systemstatus noch nicht weitergeleitet werden kann, wird diese Nachricht A_i gepuffert, d.h. sie wird nicht aus dem Eingabestrom gelöscht. Dies wird beispielsweise in den Gleichungen $(4), (5), (4')$ und $(5')$ spezifiziert.

4.3.2 Spezifikation mit Tabellen

Als Alternative zu der oben vorgestellten konstruktiven Spezifikation mit Funktionsgleichungen der Zentrale und auch zu der noch folgenden Spezifikation der Subzentralen können wir eine verkürzte Notation – Spezifikation mit Hilfe von Tabellen – wählen. Bei der Durchsicht der Funktionsgleichungen fällt auf, daß diese sehr viele FOCUS-spezifische Operatoren und sich wiederholende Sonderzeichen enthalten. Die für den jeweiligen Spezifikationsschritt wesentliche Information ist in diesem Fall oft nur für den geübten Leser sofort zu erkennen. Dies erschwert das Verständnis der Spezifikationen vor allem für Anwender, die mit der speziellen FOCUS-Notation oder dem Umgang mit logischen und mathematischen Formeln nicht vertraut sind. Eine Möglichkeit, die Funktionsgleichungen durch eine Notation zu ersetzen, die vor allem die für jede Gleichung relevante Information enthält und somit eine verkürzte Schreibweise der Funktionsgleichungen darstellt, sind **Tabellen**. Aus diesem Grund können Tabellen auch wieder in Funktionsgleichungen umgesetzt werden; hierbei können die eben erwähnten Operatoren und Sonderzeichen schematisch in die formale Spezifikation eingesetzt werden.

Die Tabellen bestehen aus Spalten, die den Ein- und Ausgabekanälen und den in der Spezifikation verwendeten Ausgangs- und Folgezuständen zugeordnet sind. Jede Zeile einer Tabelle entspricht einer Funktionsgleichung und somit einer der Anforderungen, denen die Komponente genügen muß. Im Fall der Spezifikation eines verteilten Systems mit statischer Struktur genügt hierbei eine Tabelle, deren Spaltenanzahl der statisch definierten Schnittstelle des Systems entspricht. Da eines der wesentlichen Charakteristika der mobilen, dynamischen Netze darin besteht, daß sich die syntaktische Schnittstelle einer Systemkomponente während des Systemverlaufs verändern kann, siehe Kapitel 4.2 für die Zentrale des Batch-Systems, können diese Tabellen zur Beschreibung der mobilen, dynamischen Netze nicht ohne Anpassung verwendet werden. Bei Spezifikation mit *einer* Tabelle müßte sich die Anzahl der Spalten mit der Schnittstelle der Komponente dynamisch verändern. Um dies zu vermeiden, geben wir im folgenden eine erweiterte Tabellennotation für die Spezifikation mobiler, dynamischer Komponenten an, die an die Mobilität der Komponentenschnittstelle angepaßt ist.

Wie wir in den Kapiteln 3 allgemein und in 4.2 für die Zentrale des Batch-Systems bereits gezeigt haben, können wir für ein mobiles, dynamische System Phasen angeben, in denen sich weder die Vernetzung der Komponenten untereinander durch Kanalverbindungen noch die Anzahl der im System existierenden Komponenten verändert. Das System verhält sich in diesen Phasen also im wesentlichen wie ein statisches System. Um zu vermeiden, daß die Übersichtlichkeit der Tabellenschreibweise verloren geht, geben wir für die Verhaltensspezifikation einer Komponente jeweils eine Tabelle zu jeder der oben beschriebenen Phasen an, in der die Schnittstelle der Komponente unverändert bleibt. Auf diese Weise können die für die statischen Systeme bereits definierten Tabellen, siehe [Spi94], auch bei Spezifikationen mit mobilen Netzen verwendet werden, und die klare Struktur der Tabellenspezifikationen bleibt erhalten.

Die Spalten entsprechen den für die jeweilige Phase definierten Ein- und Ausgabekanälen der Komponente. Solange der Übergang in die nächste Phase nicht erfolgt, können die Tabelleneinträge in der für statische Systeme bekannten Art und Weise verwendet werden. Sobald sich die Schnittstelle einer Komponente verändert, geht die Komponente in ihre nächste Phase über. Die Spezifikation des Verhaltens, das dieser Phase entspricht, erfolgt in einer gesonderten Tabelle. Die Tabellen müssen jedoch um die Möglichkeit erweitert werden, den Teil der Spezifikation, durch den die dynamische Änderung initiiert wird, zu notieren und hervorzuheben. Hierfür werden die Tabellen um Spalten erweitert, die den dynamischen Anteil enthalten. Zur besseren Strukturierung der Tabellen fassen wir die statischen bzw. dynamischen Anteile der Tabelle jeweils zu einem Block zusammen, der mit dem Schlüsselwort *Static* bzw. *Dynamic* gekennzeichnet ist.

Im folgenden stellen wir für die in Kapitel 4.3 angegebenen Spezifikation der Zentrale die alternative Spezifikation in Tabellennotation vor. Jede Tabelle beschreibt das Verhalten der Komponente *Zentrale* für die Phase, in der sich die Schnittstelle der Komponente nicht verändert. Die Tabellen bestehen aus Spalten, die für die Ein- und Ausgabekanäle der Komponente in der aktuellen Phase stehen, und weiteren Spalten, die die wesentlichen Informationen enthalten, die einen Phasenwechsel auslösen. In die Spalten der Kanäle werden Nachrichten eingetragen, wobei Ports (konsistent zu der Semantik der mobilen, dynamischen Netze) wie Nachrichten behandelt werden. In den Spalten, die die Erzeugung neuer Komponenten betreffen, können Bezeichner für die Funktionen, die eine neu kreierte Komponente beschreiben, und deren initiale Schnittstelle aufgenommen werden. Oftmals ist es auch nötig, den Strom, mit dem die Funktionen weiterarbeiten sollen, explizit zu notieren; dies wird ebenfalls durch eine separate Spalte in die Tabellenspezifikation aufgenommen. Einträge in diese Spalten werden jedoch nur dann vorgenommen, wenn die Erzeugung neuer Komponenten spezifiziert wird.

Phase 1: Phase 1 beschreibt die Anfangskonfiguration des Systems, und die Tabellenspezifikation entspricht den Punkten (1) und (2) in der Spezifikation von Abschnitt 4.3.1.

Static		**Dynamic**		
in	**out**	**Parent**	**Child**	**sin**
$Login$	$Conf\&New(Sub_1)$	h	$sub_1(?sin, !sout_1)$	k_1

Die mit *Parent* markierte Spalte betrifft die erzeugende Komponente, die in unserer Spezifikation durch die mobile, stromverarbeitende Funktion h spezifiziert wird. Die Spalte *Child* enthält die Information, die die neu erzeugte Komponente betreffen. *sin* ist der neu erzeugte Kanal von der Zentrale zu der ersten Subzentrale, über den die initiale Nachricht k_1 gesendet wird.

Phase 2: Phase 2 beschreibt die Konfiguration, in der die erste Subzentrale bereits existiert und Aufträge entgegennimmt, solange die Kapazität noch nicht erreicht ist. Sobald diese Kapazität erreicht ist, wird die zweite Subzentrale kreiert. Die Tabelle entspricht den Punkten (3) bis (4) der Spezifikation von Abschnitt 4.3.1.

Mit $w_1 \in \{New(Slave_{1j}), State_{1j}\}$

Static				**Dynamic**		
in	**sout**$_1$	**out**	**sin**	**Parent**	**Child**	**sin**
A_i	$NotFull(Sub_1)$	$A_i\_to\_Sub_1$	A_i	$--$	$--$	$--$
A_{i-}	w_1	w_1	$--$	$--$	$--$	$--$
A_{i-}	$Full(Sub_1)\&?sin$	$New(Sub_2)$	$--$	h	$sub_2(?sin, !sout_2)$	k_2

Die abkürzende Schreibweise $A_{i\_}$ liefert die Notation dafür, daß die Zentrale die an Kanal in anliegende Nachricht A_i speichert, bis sie an die zuständige Subzentrale weitergeleitet werden kann.

Phase l: Phase l beschreibt die Konfiguration, in der im System bereits l Subzentralen existieren. Die aktuellen Aufträge werden an die $l$te Subzentrale weitergeleitet. Sobald deren Kapazität erreicht ist, wird die nächste Subzentrale erzeugt. Die Tabelle entspricht den Punkten (3′) bis (5′) der Spezifikation in Abschnitt 4.3.1.

Mit $w_l \in \{New(Slave_{lj}), State_j\}, \forall\ l, j \in I\!N, l \geq 2$ und $1 < m < l$

Static						**Dynamic**		
in	**sout**$_1$	**sout**$_m$	**sout**$_l$	**out**	**sin**	**Parent**	**Child**	**sin**
A_i	$State_{1j}$	$State_{mj}$	$NotFull(Sub_l)$	$A_i\_to\_Sub_l$	A_i	$--$	$--$	$--$
$A_{i\_}$	$State_{1j}$	$State_{mj}$	w_l	w_l	$--$	$--$	$--$	$--$
$A_{i\_}$	$State_{1j}$	$State_{mj}$	$Full(Sub_l)\&?sin$	$New(Sub_{l+1})$	$--$	h	$sub_{l+1}(?sin, !sout_{l+1})$	k_{l+1}

Die so vorgestellten Spezifikationen im Tabellenstil sind wesentlich überschaubarer als die Gleichungsspezifikationen aus Abschnitt 4.3.1, da sie nur die für die Spezifikation des Verhaltens wesentliche Information enthalten. Bei der formalen Spezifikation von Komponenten, deren Komplexität in einem überschaubaren Umfang bleibt, hat sich die Verwendung der Tabellenschreibweise durchaus bewährt. Werden die Anforderungen an die funktionalen Eigenschaften des Systems jedoch komplexer, sollte sich der Anwender darüber im Klaren sein, daß auch Tabellen mit großem Umfang weniger überschaubar werden. Die Umsetzung in die in Abschnitt 4.3.1 verwendeten Funktionsgleichungen kann schematisch erfolgen und sollte offensichtlich sein. Wir verzichten auf die Erläuterung dieser Umsetzung und werden auch die im nächsten Abschnitt folgende Spezifikation der Subzentralen nur im Gleichungsstil angeben.

Eine besondere Eigenschaft der hier vorgestellten Tabellennotation besteht in der Möglichkeit, eine Konsistenzüberprüfung zwischen den in einer Tabelle in den Spalten verwendeten Kanalnamen und den in Kapitel 2 definierten Mengen *pp* und *ep* durchzuführen. Die Menge der im Block *Static* verwendeten Kanalbezeichner muß mit der für die mobile Funktion aktuell definierten Menge der öffentlichen Ports *ep* übereinstimmen. Die Kanalbezeichner, die im Block *Dynamic* verwendet werden, müssen in der aktuellen Menge der privaten Ports der Komponente enthalten sein. Ist dies nicht der Fall, kann die Komponente nicht die Rolle des Initiators für die Kanalerzeugung übernehmen, siehe auch Kapitel 3. Durch derartige Konsistenzüberprüfungen können Fehler in der Spezifikation schematisch aufgedeckt werden.

4.3.3 Die Subzentralen

Die für jede der *Subzentralen* geltenden funktionalen Eigenschaften werden im folgenden zunächst informell angegeben:

Die Subzentralen werden direkt von der Zentrale erzeugt. Sie erhalten bei ihrer Erzeugung die statische Schnittstelle mit dem Eingabekanal sin und dem Ausgabekanal $sout_i$. Die Subzentralen bilden die Schnittstelle zwischen der Zentralen und den ausführenden Komponenten $Slave_{ij}$. Sie sind für die Erzeugung der ihnen zugeordneten ausführenden Komponenten $Slave_{ij}$ zuständig und leiten die Aufträge von der Zentrale an die ausführenden Komponenten bzw. die Statusmeldungen von den ausführenden Komponenten an die Zentrale weiter.

1. Jede Subzentrale kann eine feste Anzahl von ausführenden Komponenten erzeugen. Diese feste Anzahl richtet sich nach der Kapazitätsangabe, die jeder Subzentrale bei ihrer Erzeugung als Nachricht durch die Zentrale mitgegeben wird. Dementsprechend wird das Verhalten einer Subzentrale nur für den Fall spezifiziert, daß über den Eingabekanal sin die zugehörige Kapazitätsangabe in Form einer natürlichen Zahl k_i empfangen wurde. (Dies gilt für alle Kapazitätsangaben k_i mit $i \in I\!N$.)

2. Jede Subzentrale sendet nach ihrer Erzeugung zunächst die Nachricht $NotFull(Sub_i)$ über Kanal $sout_i$ an die Zentrale; diese Nachricht wird immer gesendet, da wir nur Subzentralen mit einer Kapazität ≥ 1 erzeugen. Im weiteren Verlauf verhält sich eine Subzentrale dann wie eine Komponente, deren obere Kapazitätsgrenze durch k_i bestimmt ist.

3. Sobald eine Subzentrale Sub_i die Nachricht A_m über den Kanal sin empfängt, erzeugt sie eine neue Komponente $Slave_{ij}$ mit den Kanälen bin_{ij} und $bout_{ij}$ für die Ausführung des Auftrags A_m. Der Auftrag A_m wird der neu erzeugten Komponente über den Kanal bin_{ij} weitergegeben. Zusätzlich gibt die Subzentrale – solange ihre Kapazitätsgrenze noch nicht erreicht ist – die Meldungen $NotFull(Sub_i)$, $New(Slave_{ij})$ und alle von den anderen ausführenden Komponenten anliegenden Statusmeldungen über Kanal $sout_i$ an die Zentrale weiter. (Dies gilt für alle $i, j, m \in I\!N$, mit $1 < j \leq k_i$.)

4. Wenn eine Subzentrale Sub_i eine Nachricht A_m empfängt und damit die obere Kapazitätsgrenze erreicht ist, so erzeugt sie zum letzten Mal eine ausführende Komponente $Slave_{ik_i}$ mit der statischen Schnittstelle bin_{ik_i} und $bout_{ik_i}$. Der Auftrag A_m wird der neuen Komponente über den Kanal bin_{ik_i} weitergegeben. Außerdem meldet die Subzentrale der Zentralen mittels der Nachricht $Full(Sub_i)$, daß ihre Kapazität für die Ausführung weiterer Aufträge erschöpft ist. Zusätzlich sendet sie die Meldungen $New(Slave_{ik_i})$ sowie das Leserecht für Kanal sin und alle von den anderen ausführenden Komponenten anliegenden Statusmeldungen über Kanal $sout_i$ an die Zentrale. (Dies gilt für alle $i, m \in I\!N$.)

5. Eine Subzentrale, deren Kapazität erschöpft ist, kann keine weiteren Aufträge empfangen und wird somit keine ausführenden Komponenten mehr erzeugen. Sie leitet nur noch die über die Kanäle bin_{ij} empfangenen Statusmeldungen $State_{ij}$ der ausführenden Komponenten über Kanal $sout_i$ an die Zentrale weiter. (Dies gilt für alle $i, j, k_i \in I\!N$ mit $1 \leq j \leq k_i$.)

Im folgenden wird – analog zur formalen Spezifikation der Zentrale – die Spezifikation der *Subzentralen*-Komponenten angegeben.

Spezifikation der Komponenten Subzentrale

Initiale Schnittstelle:

$$I_i \triangleq \{sin\} \quad \text{(statische Eingabekanäle)}$$

$$O_i \triangleq \{sout_i\} \quad \text{(statische Ausgabekanäle)}$$

Private Kanalnamen:

$$P_i \triangleq \{bin_{ij} \mid j \in I\!N\} \cup \{bout_{ij} \mid j \in I\!N\}$$

Semantik der Spezifikation:

$$[\![\ SubZentrale_i\ ]\!] = \{f :: TypeSub\_i_{I_i,O_i,P_i} \mid P_{Sub_i}.f \}$$

mit dem Prädikat

$$P_{Sub_i} :: TypeSub\_i_{I_i,O_i,P_i} \longrightarrow I\!B$$

wobei

$$TypeSub\_i_{I_i,O_i,P_i} \triangleq \prod_{n \in N} [S_n^*] \xrightarrow{I_i,O_i,P_i} \prod_{n \in N} [S_n^*]$$

(Signatur der mobilen Funktionen)

$$\overline{TypeSub\_i}_{I_i,O_i,P_i} \triangleq \prod_{n \in N} S_n^\omega \xrightarrow{I_i,O_i,P_i} \prod_{n \in N} S_n^\omega$$

(Signatur mit Zeitabstraktion)

Prädikat zur Verhaltensbeschreibung:

$$P_{Sub_i}\ (\ sin\ \rhd\ sout_i \ \Box \ \{\ \{bin_{ij}\ j \in I\!N\}\ \} \cup \{bout_{ij} \mid j \in I\!N\}\ \}\ ).f_i$$

$$\triangleq \forall\, \theta \in \prod_{n \in N} [S_n^*], \exists\, g_i :: \overline{TypeSub_i}_{I_i,O_i,P_i} \ :\ \overline{f_i(\theta)} = g_i(\overline{\theta})$$

Die Spezifikation des Prädikats erfolgt zeitunabhängig und

durch Angabe von Funktionsgleichungen für g_i

Funktionsgleichungen zur Definition von g_i

$\forall\ s, s', t \in \prod_{n \in N} S_n^{\omega}, \quad \forall i, j, k_i, l, m \in I\!N, l \geq 2$

$\exists\ h_i :: I\!N_0\ \times\ \overline{TypeSub\_i}_{I_i, O_i, P_i}, \exists\ slave_{ij} \in [\![\ Slave_{ij}\ ]\!], \ :$

Initialisierung:

(1) $s = \{sin\ \rightarrow\ k_i\}\ \ \&\ \ s' \quad \Longrightarrow$

(2) $\quad g_i(s)\ \ =\ \ \{sout_i\ \rightarrow\ NotFull(Sub_i)\}\ \&\ h_i(k_i)(s')$

Für $k_i = 1$:

(3) $h_i(1)(\{sin\ \rightarrow\ A_m\}\&t)$

$$= \{sout_i\ \rightarrow\ Full(Sub_i)\&?sin\&New(Slave_{i1})\}\ \& \\ (h_i(0)\ \overline{\otimes}\ slave_{i1}(?bin_{i1}, !bout_{i1})\ )(\{bin_{i1}\ \rightarrow\ A_m\}\&t)$$

Für $k_i \geq 2$:

(4) $h_i(k_i)(\{sin\ \rightarrow\ A_m\}\&t)$

$$= \{sout_i\ \rightarrow\ NotFull(Sub_i)\&New(Slave_{i1}) \\ (h_i(k_i - 1)\ \overline{\otimes}\ slave_{i1}(?bin_{i1}, !bout_{i1})\ )(\{bin_{i1}\ \rightarrow\ A_m\}\&t)$$

und mit $1 < l < k_i$ und $j\ =\ k_i - l + 1$:

(5) $h_i(l)(\{sin\ \rightarrow\ A_m, bout_{i1}\ \rightarrow\ State_{i1}, \cdots, bout_{ij-1}\ \rightarrow\ State_{ij-1}\}\&t)$

$$= \{sout_i\ \rightarrow\ NotFull(Sub_i)\&New(Slave_{ij})\&State_{ij-1}\&\cdots\&State_{i1}\}\ \& \\ (h_i(l-1)\ \overline{\otimes}\ slave_{ij}(?bin_{ij}, !bout_{ij})\ )(\{bin_{ij}\ \rightarrow\ A_m\}\&t)$$

(6) $h_i(1)(\{sin\ \rightarrow\ A_m, bout_{i1}\ \rightarrow\ State_{i1}, \cdots, bout_{ik_i-1}\ \rightarrow\ State_{ik_i-1}\}\&t)$

$$= \{sout_i\ \rightarrow\ Full(Sub_i)\&?sin\&New(Slave_{ik_i})\&State_{ik_i-1}\&\cdots\&State_{i1}\}\ \& \\ (h_i(0)\ \overline{\otimes}\ slave_{ik_i}(?bin_{ik_i}, !bout_{ik_i})\ )(\{bin_{ik_i}\ \rightarrow\ A_m\}\&t)$$

Für $k_i \in I\!N$:

(7) $h_i(0)(\{bout_{i1}\ \rightarrow\ State_{i1}, \cdots, bout_{ik_i}\ \rightarrow\ State_{ik_i}\}\&t)$

$$= \{sout_i\ \rightarrow\ State_{ik_i}\&\cdots\&State_{i1}\}\ \&\ h_i(0)(t)$$

Wie bereits bei der Spezifikation der Zentrale gehen wir davon aus, daß von allen an einer Komponente anliegenden Eingabekanälen simultan gelesen werden kann.

Zum Verständnis der hier gezeigten formalen Spezifikation verweisen wir wieder auf die im vorhergehenden Abschnitt gegebene informelle Spezifikation und auf die allgemeine Vorgehensweise, die bereits an der Spezifikation der Zentrale und zudem in Kapitel 3 vorgestellt wurde. Die einzelnen Funktionsgleichungen sollten daher direkt nachvollziehbar sein.

In dieser Lösung erhält die Subzentrale als erste Nachricht die Zahl k_i, durch die die Kapazität der Subzentrale festgelegt wird. Die Subzentrale wird durch mobile Funktionen spezifiziert, die einen Parameter mit sich führen, der die jeweilige aktuell verbleibende Kapazität der Subzentrale beschreibt. Dieser Parameter wird immer dann dekrementiert, wenn eine neue Slave-Komponente erzeugt wird. Eine alternative Modellierung kann auch durch die Inkrementierung des Kapazitätsparameters erstellt werden. Hierfür muß in den Gleichungsspezifikationen der Index l in Form von $l < k_i + 1$ definiert werden. Bei der Wahl dieser Lösung handelt es sich um eine Designentscheidung bzgl. der Realisierung der gestellten informellen Anforderungen. Wir haben uns für die hier detailliert vorgestellte Lösung entschieden und wollen die zweite Alternative nicht weiter verfolgen.

4.3.4 Die Slave-Komponenten

Zunächst geben wir – analog zu den vorangegangenen Abschnitten – die formale Spezifikation der *Slave*-Komponenten an.

Die nachfolgende Spezifikation der Funktionsgleichungen ist sehr kurz. Wir fordern ausschließlich, daß die Komponente eine unendliche Zahl von Nachrichten $State_{ij}$ ausgibt, nachdem sie als erstes Eingabesignal die Nachricht A erhalten hat. A ist dabei der Auftrag, den die Komponente zu bearbeiten hat. $State_{ij}$ sind die Informationsnachrichten, die sie während der Bearbeitung an die Umgebung weiterleitet. Da es im Ablauf des Batch-Systems nicht vorgesehen ist, daß Slave-Komponenten neue Kanalverbindungen aufbauen, ist jeder Slave-Komponente eine leere Menge privater Kanalnamen P_{ij} zugeordnet.

Spezifikation der Komponenten Slave

Initiale Schnittstelle:

$$I_{ij} \triangleq \{bin_{ij}\} \qquad \text{(statische Eingabekanäle)}$$

$$O_{ij} \triangleq \{bout_{ij}\} \qquad \text{(statische Ausgabekanäle)}$$

Private Kanalnamen:

$$P_{ij} \triangleq \emptyset$$

Semantik der Spezifikation:

$$[\![\ Slave_{ij} \ ]\!] = \{ \ f \ :: \ TypeSlave\_ij_{I_{ij},O_{ij},P_{ij}} \mid P_{Slave_{ij}}.f \ \}$$

mit dem Prädikat

$$P_{Slave_{ij}} \ :: \ TypeSlave\_ij_{I_{ij},O_{ij},P_{ij}} \longrightarrow \mathbb{B}$$

wobei

$$TypeSlave\_ij_{I_{ij},O_{ij},P_{ij}} \triangleq \prod_{n \in N} [S_n^*] \xrightarrow{I_{ij},O_{ij},P_{ij}} \prod_{n \in N} [S_n^*]$$

(Signatur der mobilen Funktionen)

$$\overline{TypeSlave}\_ij_{I_{ij},O_{ij},P_{ij}} \triangleq \prod_{n \in N} S_n^\omega \xrightarrow{I_{ij},O_{ij},P_{ij}} \prod_{n \in N} S_n^\omega$$

(Signatur mit Zeitabstraktion)

Prädikat zur Verhaltensbeschreibung:

$$P_{Slave_{ij}} \ (\ bin_{ij} \ \triangleright \ bout_{ij} \ \Box \ \emptyset \ ).f$$

$$\triangleq \ \forall \ \theta \in \prod_{n \in N} [S_n^*], \exists \ g :: \overline{TypeSlave}\_ij_{I_{ij},O_{ij},P_{ij}} \ : \ \overline{f(\theta)} = g(\overline{\theta})$$

Die Spezifikation des Prädikats erfolgt zeitunabhängig und
durch Angabe von Funktionsgleichungen für g

Funktionsgleichungen zur Definition von g

$$\forall\, s \in \prod_{n \in N} S_n^{\omega} \;\; \exists\, h \;\in\; \overline{TypeSlave\_ij}_{I_{ij},O_{ij},P_{ij}} :$$

$$\begin{array}{lcl} g\ (A\ \&\ s) & = & State_{ij}\ \&\ h\ (s) \\ h\ (s) & = & State_{ij}\ \&\ h\ (s) \end{array}$$

5 Zusammenfassung

In dem vorliegenden Bericht haben wir ausgehend von dem in [GS96c] beschriebenen semantischen Modell eine Vorgehensweise für die Spezifikation mobiler, dynamischer Netze entwickelt.

Mit diesem Bericht liegen nun

- eine für den Anwender verständliche Einführung in die semantischen Grundlagen von mobilen, dynamischen Netzwerken,
- Leitfäden für den Umgang mit dem Modell der mobilen, dynamischen Netze zur Spezifikation der Änderung von Vernetzungsstruktur und Komponentenanzahl sowie
- eine detaillierte Spezifikation eines Batch-Systems, das als Referenzbeispiel dienen kann,

vor.

Bei dem von uns gewählten Batch-System aus Kapitel 3 steht die Spezifikation der mobilen und dynamischen Aspekte im Vordergrund. Das funktionale Verhalten der einzelnen Komponenten haben wir nur sehr eingeschränkt beschrieben. Wichtig war es uns, das schematische Vorgehen bei der Erstellung der Spezifikation hervorzuheben und dem Anwender somit ein Referenzbeispiel anzubieten.

Eine Reihe von praktisch eingesetzten Methoden ermöglicht es, neben statischen Aspekten auch dynamische Anteile in die Systembeschreibung einzubeziehen. Gerade in objektorientierten Methoden wie etwa OMT ([Rum91]) oder Fusion ([CAB$^+$94]) ist es üblich, nicht von einer statisch fest vorgebenen Anzahl von Komponenten auszugehen, sondern das dynamische Erzeugen neuer Komponenten während des Systemablaufs in die Modellierung einzubeziehen. Mit der Erweiterung von FOCUS ist nun die Basis geschaffen, um mobile und dynamische Aspekte von Systemen adäquat zu modellieren. Für die Spezifikationsund Beschreibungssprache SDL ([IT93]) wurde im Rahmen dieser Erweiterung bereits gezeigt, wie sich die dynamische Prozeßerzeugung mittels des Create-Konstrukts formal nach Focus umsetzen läßt.

In anderen Ansätzen, wie etwa [KRB96], wird die dynamische Erzeugung von Komponenten nur implizit modelliert. Eine Komponente, die während des Systemablaufs neu erzeugt wird, wird als statische Komponente modelliert, die während des gesamten Systemablaufs existiert, jedoch erst bei Erhalt einer bestimmten Nachricht sichtbares Verhalten zeigt. Dabei wird vorausgesetzt, daß sich das System aus unendlich vielen Komponenten zusammensetzt, so daß beliebig viele neue Komponenten aktiviert werden können. Wird die Anzahl der gleichzeitig vorhandenen Komponenten eines Systems beschränkt, so ist bereits in der klassischen Version von Focus die Umsetzung von Komponentenerzeugung möglich. Dabei sind allerdings die Komponenten und die Kommunikationsverbindungen während des gesamten Systemablaufs vorhanden.

Die diesem Bericht zugrundeliegenden Arbeiten haben gezeigt, daß das Erfassen von dynamischen Aspekten eines Systems an und für sich eine komplexe Aufgabe darstellt, selbst wenn, wie in unserem Fall, mit der Erweiterung von Focus eine adäquate Modellierungstechnik für mobile, dynamische Systeme vorliegt. Bereits die Umsetzung des vorgestellten einfachen Batch-Systems hat zu umfangreichen Spezifikationen geführt. Die Verwendung von tabellarischen Übersichten und Spezifikationen stellt eine Möglichkeit dar, dem Anwender den Umgang mit der formalen Spezifikation zu erleichtern und es ihm zu erlauben, die angegebenen Spezifikationen nachzuvollziehen. Die von uns entwickelten Leitfäden haben sich schon während der Erstellung des in diesem Bericht vorgestellten Beispiels bewährt. Ausgehend von diesem Beispiel und unter Verwendung dieser Leitfäden entstanden die formale Umsetzung des *Create*-Konstrukts von SDL in Focus und ein Beispiel hierfür, siehe [Hin96], sowie die Spezifikation eines einfachen Beispiels aus dem Betriebssystemebereich, das aus dem Bereich der Scheduling- und Ressourcenmanagment-Problematik herausgenommen wurde, siehe [Spi].

Danksagung

Wir möchten uns vor allem bei Radu Grosu für die zahlreichen Gespräche und Erklärungen zu den mobilen, dynamischen Focus-Systemen und bei Jan Philipps für die konstruktive Kritik an Vorversionen dieses Berichts bedanken. Weiterhin danken wir Manfred Broy, Bernhard Schätz und Oscar Slotosch für ihre hilfreichen Kommentare zu Vorversionen dieser Arbeit.

Literatur

[BDD+93] M. Broy, F. Dederichs, C. Dendorfer, M. Fuchs, T. F. Gritzner und R. Weber. *The Design of Distributed Systems — An Introduction to* FOCUS. SFB-Bericht 342/2/92 A, Technische Universität München, 1993.

[BS] Manfred Broy und Ketil Stølen. *FOCUS on System Development – A Method for the Development of Interactive Systems.* In Vorbereitung.

[CAB+94] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes und P. Jeremaes. *Object-Oriented Development - The Fusion Method.* Prentice Hall, 1994.

[Gro96a] R. Grosu. *About Recursive Mobile Networks*, 1996. Interne Ausarbeitung.

[Gro96b] R. Grosu. *Banken und Konten: Ein Beispiel zur Spezifikation nebenläufiger Informationssysteme*, 1996. Interne Ausarbeitung.

[Gro96c] R. Grosu. *Recursive Networks and Time Abstraction*, 1996.

[GS96a] R. Grosu und K. Stølen. *A Denotational Model for Mobile Many-to-Many Dataflow Networks.* Technischer Bericht TUM-I9622, Technische Universität München, 1996.

[GS96b] R. Grosu und K. Stølen. *A Denotational Model for Mobile Point-to-Point Dataflow Networks with Channel Sharing.* Technischer Bericht, Technische Universität München, 1996.

[GS96c] Radu Grosu und Ketil Stølen. *A Model for Mobile Point-to-Point Dataflow Networks without Channel Sharing.* in M. Nivat M. Wirsing (Hrsg.) , *AMAST'96.* LNCS 1101 1996.

[Hin] U. Hinkel. *Formale Fundierung und Verifikationsmethodik für SDL.* Dissertation, *vorläufige Version.*

[Hin96] U. Hinkel. *SDL-Projekt: Berücksichtigung von Daten und Funktionen, Projektabschnitt 3*, 1996. Interner Bericht.

[IT93] ITU-T. *Recommendation Z.100, Specification and Description Language (SDL).* ITU, 1993.

[KRB96] Cornel Klein, Bernhard Rumpe und Manfred Broy. *A stream-based mathematical model for distributed information processing systems - SysLab system model -* . in Jean-Bernard Stefani Elie Naijm (Hrsg.) , *FMOODS'96 Formal Methods for Open Object-based Distributed Systems.* ENST France Telecom 1996, S. 323–338.

[MPW92a] R. Milner, J. Parrow und D. Walker. *A calculus of mobile processes, part I.* Information and Computation **100** (1992), 1–40.

[MPW92b] R. Milner, J. Parrow und D. Walker. *A calculus of mobile processes, part II.* Information and Computation **100** (1992), 41–77.

[Rum91] J. Rumbaugh. *Object-Oriented Modelling and Design.* Prentice Hall, 1991.

[Spi] K. Spies. *Spezifikation und methodische Entwicklung von Betriebssystemkonzepten.* Dissertation, *vorläufige Version.*

[Spi94] K. Spies. *Funktionale Spezifikation eines Kommunikationsprotokolls.* SFB-Bericht 342/08/94 A, Technische Universität München, 1994.

SFB 342: Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

bisher erschienen :

Reihe A

342/1/90 A	Robert Gold, Walter Vogler: Quality Criteria for Partial Order Semantics of Place/Transition-Nets, Januar 1990
342/2/90 A	Reinhard Fößmeier: Die Rolle der Lastverteilung bei der numerischen Parallelprogrammierung, Februar 1990
342/3/90 A	Klaus-Jörn Lange, Peter Rossmanith: Two Results on Unambiguous Circuits, Februar 1990
342/4/90 A	Michael Griebel: Zur Lösung von Finite-Differenzen- und Finite-Element-Gleichungen mittels der Hierarchischen Transformations-Mehrgitter-Methode
342/5/90 A	Reinhold Letz, Johann Schumann, Stephan Bayerl, Wolfgang Bibel: SETHEO: A High-Performance Theorem Prover
342/6/90 A	Johann Schumann, Reinhold Letz: PARTHEO: A High Performance Parallel Theorem Prover
342/7/90 A	Johann Schumann, Norbert Trapp, Martin van der Koelen: SETHEO/PARTHEO Users Manual
342/8/90 A	Christian Suttner, Wolfgang Ertel: Using Connectionist Networks for Guiding the Search of a Theorem Prover
342/9/90 A	Hans-Jörg Beier, Thomas Bemmerl, Arndt Bode, Hubert Ertl, Olav Hansen, Josef Haunerdinger, Paul Hofstetter, Jaroslav Kremenek, Robert Lindhof, Thomas Ludwig, Peter Luksch, Thomas Treml: TOPSYS, Tools for Parallel Systems (Artikelsammlung)
342/10/90 A	Walter Vogler: Bisimulation and Action Refinement
342/11/90 A	Jörg Desel, Javier Esparza: Reachability in Reversible Free- Choice Systems
342/12/90 A	Rob van Glabbeek, Ursula Goltz: Equivalences and Refinement
342/13/90 A	Rob van Glabbeek: The Linear Time - Branching Time Spectrum
342/14/90 A	Johannes Bauer, Thomas Bemmerl, Thomas Treml: Leistungsanalyse von verteilten Beobachtungs- und Bewertungswerkzeugen

Reihe A

342/15/90 A	Peter Rossmanith: The Owner Concept for PRAMs
342/16/90 A	G. Böckle, S. Trosch: A Simulator for VLIW-Architectures
342/17/90 A	P. Slavkovsky, U. Rüde: Schnellere Berechnung klassischer Matrix-Multiplikationen
342/18/90 A	Christoph Zenger: SPARSE GRIDS
342/19/90 A	Michael Griebel, Michael Schneider, Christoph Zenger: A combination technique for the solution of sparse grid problems
342/20/90 A	Michael Griebel: A Parallelizable and Vectorizable Multi- Level-Algorithm on Sparse Grids
342/21/90 A	V. Diekert, E. Ochmanski, K. Reinhardt: On confluent semi-commutations-decidability and complexity results
342/22/90 A	Manfred Broy, Claus Dendorfer: Functional Modelling of Operating System Structures by Timed Higher Order Stream Processing Functions
342/23/90 A	Rob van Glabbeek, Ursula Goltz: A Deadlock-sensitive Congruence for Action Refinement
342/24/90 A	Manfred Broy: On the Design and Verification of a Simple Distributed Spanning Tree Algorithm
342/25/90 A	Thomas Bemmerl, Arndt Bode, Peter Braun, Olav Hansen, Peter Luksch, Roland Wismüller: TOPSYS - Tools for Parallel Systems (User's Overview and User's Manuals)
342/26/90 A	Thomas Bemmerl, Arndt Bode, Thomas Ludwig, Stefan Tritscher: MMK - Multiprocessor Multitasking Kernel (User's Guide and User's Reference Manual)
342/27/90 A	Wolfgang Ertel: Random Competition: A Simple, but Efficient Method for Parallelizing Inference Systems
342/28/90 A	Rob van Glabbeek, Frits Vaandrager: Modular Specification of Process Algebras
342/29/90 A	Rob van Glabbeek, Peter Weijland: Branching Time and Abstraction in Bisimulation Semantics
342/30/90 A	Michael Griebel: Parallel Multigrid Methods on Sparse Grids
342/31/90 A	Rolf Niedermeier, Peter Rossmanith: Unambiguous Simulations of Auxiliary Pushdown Automata and Circuits
342/32/90 A	Inga Niepel, Peter Rossmanith: Uniform Circuits and Exclusive Read PRAMs
342/33/90 A	Dr. Hermann Hellwagner: A Survey of Virtually Shared Memory Schemes
342/1/91 A	Walter Vogler: Is Partial Order Semantics Necessary for Action Refinement?
342/2/91 A	Manfred Broy, Frank Dederichs, Claus Dendorfer, Rainer Weber: Characterizing the Behaviour of Reactive Systems by Trace Sets
342/3/91 A	Ulrich Furbach, Christian Suttner, Bertram Fronhöfer: Massively Parallel Inference Systems

Reihe A

Reihe A

342/24/91 A	Johann M.Ph. Schumann: Parallelization of Inference Systems by using an Abstract Machine
342/25/91 A	Eike Jessen: Speedup Analysis by Hierarchical Load Decomposition
342/26/91 A	Thomas F. Gritzner: A Simple Toy Example of a Distributed System: On the Design of a Connecting Switch
342/27/91 A	Thomas Schnekenburger, Andreas Weininger, Michael Friedrich: Introduction to the Parallel and Distributed Programming Language ParMod-C
342/28/91 A	Claus Dendorfer: Funktionale Modellierung eines Postsystems
342/29/91 A	Michael Griebel: Multilevel algorithms considered as iterative methods on indefinite systems
342/30/91 A	W. Reisig: Parallel Composition of Liveness
342/31/91 A	Thomas Bemmerl, Christian Kasperbauer, Martin Mairandres, Bernhard Ries: Programming Tools for Distributed Multiprocessor Computing Environments
342/32/91 A	Frank Leßke: On constructive specifications of abstract data types using temporal logic
342/1/92 A	L. Kanal, C.B. Suttner (Editors): Informal Proceedings of the Workshop on Parallel Processing for AI
342/2/92 A	Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas F. Gritzner, Rainer Weber: The Design of Distributed Systems - An Introduction to FOCUS
342/2-2/92 A	Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas F. Gritzner, Rainer Weber: The Design of Distributed Systems - An Introduction to FOCUS - Revised Version (erschienen im Januar 1993)
342/3/92 A	Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas F. Gritzner, Rainer Weber: Summary of Case Studies in FOCUS - a Design Method for Distributed Systems
342/4/92 A	Claus Dendorfer, Rainer Weber: Development and Implementation of a Communication Protocol - An Exercise in FOCUS
342/5/92 A	Michael Friedrich: Sprachmittel und Werkzeuge zur Unterstüt- zung paralleler und verteilter Programmierung
342/6/92 A	Thomas F. Gritzner: The Action Graph Model as a Link between Abstract Relation Algebras and Process-Algebraic Specifications
342/7/92 A	Sergei Gorlatch: Parallel Program Development for a Recursive Numerical Algorithm: a Case Study
342/8/92 A	Henning Spruth, Georg Sigl, Frank Johannes: Parallel Algorithms for Slicing Based Final Placement
342/9/92 A	Herbert Bauer, Christian Sporrer, Thomas Krodel: On Distributed Logic Simulation Using Time Warp
342/10/92 A	H. Bungartz, M. Griebel, U. Rüde: Extrapolation, Combination and Sparse Grid Techniques for Elliptic Boundary Value Problems

Reihe A

342/11/92 A M. Griebel, W. Huber, U. Rüde, T. Störtkuhl: The Combination Technique for Parallel Sparse-Grid-Preconditioning and -Solution of PDEs on Multiprocessor Machines and Workstation Networks
342/12/92 A Rolf Niedermeier, Peter Rossmanith: Optimal Parallel Algorithms for Computing Recursively Defined Functions
342/13/92 A Rainer Weber: Eine Methodik für die formale Anforderungsspezifkation verteilter Systeme
342/14/92 A Michael Griebel: Grid– and point–oriented multilevel algorithms
342/15/92 A M. Griebel, C. Zenger, S. Zimmer: Improved multilevel algorithms for full and sparse grid problems
342/16/92 A J. Desel, D. Gomm, E. Kindler, B. Paech, R. Walter: Bausteine eines kompositionalen Beweiskalküls für netzmodellierte Systeme
342/17/92 A Frank Dederichs: Transformation verteilter Systeme: Von applikativen zu prozeduralen Darstellungen
342/18/92 A Andreas Listl, Markus Pawlowski: Parallel Cache Management of a RDBMS
342/19/92 A Erwin Loibl, Markus Pawlowski, Christian Roth: PART: A Parallel Relational Toolbox as Basis for the Optimization and Interpretation of Parallel Queries
342/20/92 A Jörg Desel, Wolfgang Reisig: The Synthesis Problem of Petri Nets
342/21/92 A Robert Balder, Christoph Zenger: The d-dimensional Helmholtz equation on sparse Grids
342/22/92 A Ilko Michler: Neuronale Netzwerk-Paradigmen zum Erlernen von Heuristiken
342/23/92 A Wolfgang Reisig: Elements of a Temporal Logic. Coping with Concurrency
342/24/92 A T. Störtkuhl, Chr. Zenger, S. Zimmer: An asymptotic solution for the singularity at the angular point of the lid driven cavity
342/25/92 A Ekkart Kindler: Invariants, Compositionality and Substitution
342/26/92 A Thomas Bonk, Ulrich Rüde: Performance Analysis and Optimization of Numerically Intensive Programs
342/1/93 A M. Griebel, V. Thurner: The Efficient Solution of Fluid Dynamics Problems by the Combination Technique
342/2/93 A Ketil Stølen, Frank Dederichs, Rainer Weber: Assumption / Commitment Rules for Networks of Asynchronously Communicating Agents
342/3/93 A Thomas Schnekenburger: A Definition of Efficiency of Parallel Programs in Multi-Tasking Environments
342/4/93 A Hans-Joachim Bungartz, Michael Griebel, Dierk Röschke, Christoph Zenger: A Proof of Convergence for the Combination Technique for the Laplace Equation Using Tools of Symbolic Computation
342/5/93 A Manfred Kunde, Rolf Niedermeier, Peter Rossmanith: Faster Sorting and Routing on Grids with Diagonals

Reihe A

342/6/93 A	Michael Griebel, Peter Oswald: Remarks on the Abstract Theory of Additive and Multiplicative Schwarz Algorithms
342/7/93 A	Christian Sporrer, Herbert Bauer: Corolla Partitioning for Distributed Logic Simulation of VLSI Circuits
342/8/93 A	Herbert Bauer, Christian Sporrer: Reducing Rollback Overhead in Time-Warp Based Distributed Simulation with Optimized Incremental State Saving
342/9/93 A	Peter Slavkovsky: The Visibility Problem for Single-Valued Surface (z = f(x,y)): The Analysis and the Parallelization of Algorithms
342/10/93 A	Ulrich Rüde: Multilevel, Extrapolation, and Sparse Grid Methods
342/11/93 A	Hans Regler, Ulrich Rüde: Layout Optimization with Algebraic Multigrid Methods
342/12/93 A	Dieter Barnard, Angelika Mader: Model Checking for the Modal Mu-Calculus using Gauß Elimination
342/13/93 A	Christoph Pflaum, Ulrich Rüde: Gauß' Adaptive Relaxation for the Multilevel Solution of Partial Differential Equations on Sparse Grids
342/14/93 A	Christoph Pflaum: Convergence of the Combination Technique for the Finite Element Solution of Poisson's Equation
342/15/93 A	Michael Luby, Wolfgang Ertel: Optimal Parallelization of Las Vegas Algorithms
342/16/93 A	Hans-Joachim Bungartz, Michael Griebel, Dierk Röschke, Christoph Zenger: Pointwise Convergence of the Combination Technique for Laplace's Equation
342/17/93 A	Georg Stellner, Matthias Schumann, Stefan Lamberts, Thomas Ludwig, Arndt Bode, Martin Kiehl und Rainer Mehlhorn: Developing Multicomputer Applications on Networks of Workstations Using NXLib
342/18/93 A	Max Fuchs, Ketil Stølen: Development of a Distributed Min/Max Component
342/19/93 A	Johann K. Obermaier: Recovery and Transaction Management in Write-optimized Database Systems
342/20/93 A	Sergej Gorlatch: Deriving Efficient Parallel Programs by Systemating Coarsing Specification Parallelism
342/01/94 A	Reiner Hüttl, Michael Schneider: Parallel Adaptive Numerical Simulation
342/02/94 A	Henning Spruth, Frank Johannes: Parallel Routing of VLSI Circuits Based on Net Independency
342/03/94 A	Henning Spruth, Frank Johannes, Kurt Antreich: PHIroute: A Parallel Hierarchical Sea-of-Gates Router
342/04/94 A	Martin Kiehl, Rainer Mehlhorn, Matthias Schumann: Parallel Multiple Shooting for Optimal Control Problems Under NX/2

Reihe A

342/05/94 A	Christian Suttner, Christoph Goller, Peter Krauss, Klaus-Jörn Lange, Ludwig Thomas, Thomas Schnekenburger: Heuristic Optimization of Parallel Computations
342/06/94 A	Andreas Listl: Using Subpages for Cache Coherency Control in Parallel Database Systems
342/07/94 A	Manfred Broy, Ketil Stølen: Specification and Refinement of Finite Dataflow Networks - a Relational Approach
342/08/94 A	Katharina Spies: Funktionale Spezifikation eines Kommunikationsprotokolls
342/09/94 A	Peter A. Krauss: Applying a New Search Space Partitioning Method to Parallel Test Generation for Sequential Circuits
342/10/94 A	Manfred Broy: A Functional Rephrasing of the Assumption/Commitment Specification Style
342/11/94 A	Eckhardt Holz, Ketil Stølen: An Attempt to Embed a Restricted Version of SDL as a Target Language in Focus
342/12/94 A	Christoph Pflaum: A Multi-Level-Algorithm for the Finite-Element-Solution of General Second Order Elliptic Differential Equations on Adaptive Sparse Grids
342/13/94 A	Manfred Broy, Max Fuchs, Thomas F. Gritzner, Bernhard Schätz, Katharina Spies, Ketil Stølen: Summary of Case Studies in FOCUS - a Design Method for Distributed Systems
342/14/94 A	Maximilian Fuchs: Technologieabhängigkeit von Spezifikationen digitaler Hardware
342/15/94 A	M. Griebel, P. Oswald: Tensor Product Type Subspace Splittings And Multilevel Iterative Methods For Anisotropic Problems
342/16/94 A	Gheorghe Ştefănescu: Algebra of Flownomials
342/17/94 A	Ketil Stølen: A Refinement Relation Supporting the Transition from Unbounded to Bounded Communication Buffers
342/18/94 A	Michael Griebel, Tilman Neuhoeffer: A Domain-Oriented Multilevel Algorithm-Implementation and Parallelization
342/19/94 A	Michael Griebel, Walter Huber: Turbulence Simulation on Sparse Grids Using the Combination Method
342/20/94 A	Johann Schumann: Using the Theorem Prover SETHEO for verifying the development of a Communication Protocol in FOCUS - A Case Study -
342/01/95 A	Hans-Joachim Bungartz: Higher Order Finite Elements on Sparse Grids
342/02/95 A	Tao Zhang, Seonglim Kang, Lester R. Lipsky: The Performance of Parallel Computers: Order Statistics and Amdahl's Law
342/03/95 A	Lester R. Lipsky, Appie van de Liefvoort: Transformation of the Kronecker Product of Identical Servers to a Reduced Product Space
342/04/95 A	Pierre Fiorini, Lester R. Lipsky, Wen-Jung Hsin, Appie van de Liefvoort: Auto-Correlation of Lag-k For Customers Departing From Semi-Markov Processes

Reihe A

342/05/95 A	Sascha Hilgenfeldt, Robert Balder, Christoph Zenger: Sparse Grids: Applications to Multi-dimensional Schrödinger Problems
342/06/95 A	Maximilian Fuchs: Formal Design of a Model-N Counter
342/07/95 A	Hans-Joachim Bungartz, Stefan Schulte: Coupled Problems in Microsystem Technology
342/08/95 A	Alexander Pfaffinger: Parallel Communication on Workstation Networks with Complex Topologies
342/09/95 A	Ketil Stølen: Assumption/Commitment Rules for Data-flow Networks - with an Emphasis on Completeness
342/10/95 A	Ketil Stølen, Max Fuchs: A Formal Method for Hardware/Software Co-Design
342/11/95 A	Thomas Schnekenburger: The ALDY Load Distribution System
342/12/95 A	Javier Esparza, Stefan Römer, Walter Vogler: An Improvement of McMillan's Unfolding Algorithm
342/13/95 A	Stephan Melzer, Javier Esparza: Checking System Properties via Integer Programming
342/14/95 A	Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Point-to-Point Dataflow Networks
342/15/95 A	Andrei Kovalyov, Javier Esparza: A Polynomial Algorithm to Compute the Concurrency Relation of Free-Choice Signal Transition Graphs
342/16/95 A	Bernhard Schätz, Katharina Spies: Formale Syntax zur logischen Kernsprache der Focus-Entwicklungsmethodik
342/17/95 A	Georg Stellner: Using CoCheck on a Network of Workstations
342/18/95 A	Arndt Bode, Thomas Ludwig, Vaidy Sunderam, Roland Wismüller: Workshop on PVM, MPI, Tools and Applications
342/19/95 A	Thomas Schnekenburger: Integration of Load Distribution into ParMod-C
342/20/95 A	Ketil Stølen: Refinement Principles Supporting the Transition from Asynchronous to Synchronous Communication
342/21/95 A	Andreas Listl, Giannis Bozas: Performance Gains Using Subpages for Cache Coherency Control
342/22/95 A	Volker Heun, Ernst W. Mayr: Embedding Graphs with Bounded Treewidth into Optimal Hypercubes
342/23/95 A	Petr Jančar, Javier Esparza: Deciding Finiteness of Petri Nets up to Bisimulation
342/24/95 A	M. Jung, U. Rüde: Implicit Extrapolation Methods for Variable Coefficient Problems
342/01/96 A	Michael Griebel, Tilman Neunhoeffer, Hans Regler: Algebraic Multigrid Methods for the Solution of the Navier-Stokes Equations in Complicated Geometries
342/02/96 A	Thomas Grauschopf, Michael Griebel, Hans Regler: Additive Multilevel-Preconditioners based on Bilinear Interpolation, Matrix Dependent Geometric Coarsening and Algebraic-Multigrid Coarsening for Second Order Elliptic PDEs

Reihe A

342/03/96 A	Volker Heun, Ernst W. Mayr: Optimal Dynamic Edge-Disjoint Embeddings of Complete Binary Trees into Hypercubes
342/04/96 A	Thomas Huckle: Efficient Computation of Sparse Approximate Inverses
342/05/96 A	Thomas Ludwig, Roland Wismüller, Vaidy Sunderam, Arndt Bode: OMIS — On-line Monitoring Interface Specification
342/06/96 A	Ekkart Kindler: A Compositional Partial Order Semantics for Petri Net Components
342/07/96 A	Richard Mayr: Some Results on Basic Parallel Processes
342/08/96 A	Ralph Radermacher, Frank Weimer: INSEL Syntax-Bericht
342/09/96 A	P.P. Spies, C. Eckert, M. Lange, D. Marek, R. Radermacher, F. Weimer, H.-M. Windisch: Sprachkonzepte zur Konstruktion verteilter Systeme
342/10/96 A	Stefan Lamberts, Thomas Ludwig, Christian Röder, Arndt Bode: PFSLib – A File System for Parallel Programming Environments
342/11/96 A	Manfred Broy, Gheorghe Ştefănescu: The Algebra of Stream Processing Functions
342/12/96 A	Javier Esparza: Reachability in Live and Safe Free-Choice Petri Nets is NP-complete
342/13/96 A	Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Many-to-Many Data-flow Networks
342/14/96 A	Giannis Bozas, Michael Jaedicke, Andreas Listl, Bernhard Mitschang, Angelika Reiser, Stephan Zimmermann: On Transforming a Sequential SQL-DBMS into a Parallel One: First Results and Experiences of the MIDAS Project
342/15/96 A	Richard Mayr: A Tableau System for Model Checking Petri Nets with a Fragment of the Linear Time μ -Calculus
342/16/96 A	Ursula Hinkel, Katharina Spies: Anleitung zur Spezifikation von mobilen, dynamischen Focus-Netzen

SFB 342 : Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

Reihe B

342/1/90 B	Wolfgang Reisig: Petri Nets and Algebraic Specifications
342/2/90 B	Jörg Desel: On Abstraction of Nets
342/3/90 B	Jörg Desel: Reduction and Design of Well-behaved Free-choice Systems
342/4/90 B	Franz Abstreiter, Michael Friedrich, Hans-Jürgen Plewan: Das Werkzeug runtime zur Beobachtung verteilter und paralleler Programme
342/1/91 B	Barbara Paech1: Concurrency as a Modality
342/2/91 B	Birgit Kandler, Markus Pawlowski: SAM: Eine Sortier- Toolbox -Anwenderbeschreibung
342/3/91 B	Erwin Loibl, Hans Obermaier, Markus Pawlowski: 2. Workshop über Parallelisierung von Datenbanksystemen
342/4/91 B	Werner Pohlmann: A Limitation of Distributed Simulation Methods
342/5/91 B	Dominik Gomm, Ekkart Kindler: A Weakly Coherent Virtually Shared Memory Scheme: Formal Specification and Analysis
342/6/91 B	Dominik Gomm, Ekkart Kindler: Causality Based Specification and Correctness Proof of a Virtually Shared Memory Scheme
342/7/91 B	W. Reisig: Concurrent Temporal Logic
342/1/92 B	Malte Grosse, Christian B. Suttner: A Parallel Algorithm for Set-of-Support Christian B. Suttner: Parallel Computation of Multiple Sets-of-Support
342/2/92 B	Arndt Bode, Hartmut Wedekind: Parallelrechner: Theorie, Hardware, Software, Anwendungen
342/1/93 B	Max Fuchs: Funktionale Spezifikation einer Geschwindigkeitsregelung
342/2/93 B	Ekkart Kindler: Sicherheits- und Lebendigkeitseigenschaften: Ein Literaturüberblick
342/1/94 B	Andreas Listl; Thomas Schnekenburger; Michael Friedrich: Zum Entwurf eines Prototypen für MIDAS