Spezifikationsmethodik für mobile, dynamische FOCUS-Netze*

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

Zusammenfassung

Dieses Papier stellt eine methodische Vorgehensweise für die Modellierung dynamischer und mobiler verteilter Systeme in FOCUS vor. FOCUS ist eine formale Methodik zur Entwicklung und Spezifikation verteilter, reaktiver Systeme. Wir geben Leitfäden und Spezifikationsschemata für dynamische, mobile Systeme an, gefolgt von der Beschreibung einiger Fallstudien. Ziel ist es, auch Anwendern, die mit den formalen Techniken von FOCUS nur wenig vertraut sind, die Erstellung formaler Spezifikationen zu ermöglichen.

1. Einleitung

FOCUS ist eine formale Entwurfsmethodik zur Entwicklung und Spezifikation verteilter, reaktiver Systeme, die an der TU München am Lehrstuhl von Prof. Broy entwickelt wurde. In FOCUS werden verteilte Systeme als Netzwerke von Komponenten modelliert, die durch Nachrichtenaustausch über gerichtete Kanäle miteinander kooperieren. Die Systeme sind statisch, d.h. während des Systemablaufs können weder die Netzwerkstruktur noch die Anzahl der Komponenten verändert werden. Das semantische Modell von FOCUS beruht auf dem Konzept der Ströme (vgl. [Kah74] und [BDD+93]).
Bei der Modellierung verteilter Systeme ist es oft notwendig und wünschenswert, nicht nur von einer festen Anzahl von Komponenten auszugehen, sondern zusätzlich das dynamische Erzeugen neuer Komponenten während des Systemablaufs zuzulassen sowie die Kommunikationsstruktur variabel zu gestalten, also Mobilität zu modellieren.
Ein bekannter Ansatz zur Beschreibung von Mobilität ist der π-Kalkül, siehe [MPW92]. Dieser stellt eine Erweiterung der Prozeßalgebra CCS dar und beschreibt Kommunikationssysteme, deren Struktur sich aufgrund von Interaktionen zwischen den Prozessen ändert. Angeregt dadurch wurde der klassische Ansatz von FOCUS in neueren Arbeiten um Möglichkeiten zur Variation der Netzwerkstruktur und darüberhinaus auch um die Modellierung dynamischen Verhaltens erweitert. Die Erweiterungen des semantischen FOCUS-Modells werden detailliert in [GSB97] und [Gro96a] beschrieben.
Im vorliegenden Papier wird, ausgehend von dieser semantischen Erweiterung von FOCUS, eine Anleitung zur Spezifikation von dynamischen und mobilen Systemen vorgestellt. Damit werden Anwender, die mit FOCUS nicht vertraut sind, in der Lage sein, FOCUS als formale Spezifikationsmethode einzusetzen. Das Papier ist wie folgt aufgebaut: Zunächst erläutern wir kurz die wesentlichen Konzepte des semantischen Ansatzes. Anschließend

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

stellen wir Leitfäden und Schemata für die Spezifikation von dynamischen, mobilen Systemen vor, gefolgt von der Beschreibung einiger Fallstudien. Für eine Einführung in die grundlegenden Konzepte von FOCUS verweisen wir auf [BS97] und [BDD+93].

2. Semantische Konzepte und Begriffe

Dem erweiterten FOCUS-Ansatz liegt ein denotationelles Semantikmodell zugrunde. Systeme werden als Netzwerke von Komponenten beschrieben, die untereinander und mit der Systemumgebung durch gerichtete Kanäle verbunden sind und durch asynchronen Nachrichtenaustausch kooperieren . Welche Kanäle eine Komponente aktiv für Ein- und Ausgaben nutzen kann, wird durch die Vergabe von Lese- und Schreibrechten für die Kanäle geregelt. Das grundlegende Konzept zur Modellierung der Systemabläufe bilden Ströme, siehe hierzu auch [Kah74]. Ein Strom beschreibt den Nachrichtenfluß über einen Kanal. Das Verhalten einer Komponente wird durch Mengen von stromverarbeitenden Funktionen definiert, die Ströme von Eingabenachrichten auf Ströme von Ausgabenachrichten abbilden. Dadurch wird das Ein-/Ausgabeverhalten der Komponenten charakterisiert.
Mit dem Start eines Systemablaufs verfügt jede Komponente über initiale Kommunikationsverbindungen, die aus einer Menge I von Eingabe- und einer Menge O von Ausgabekanälen bestehen. Jeder Komponente steht eine Menge von privaten Kanalnamen P für das Erzeugen neuer Verbindungen zur Verfügung. Rechte an den Kanälen werden durch Ports beschrieben. Ein Port ist ein Kanalname, der mit einem Zugriffsrecht auf den Kanal (Lesen („?“) oder Schreiben („!“)) versehen ist. Über die Kanäle können neben Nachrichten nun auch Ports verschickt werden. Hierdurch wird *Mobilität* modelliert.
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, über die eine Komponente verfügen kann, aus $D \cup !M \cup ?M$. Für $!M \cup ?M$ schreiben wir auch $?!M$; mit $?!m$ kürzen wir $\{?m\} \cup \{!m\}$ ab.
Während des Systemablaufs ändert sich die Menge der verfügbaren Ein- und Ausgabekanäle einer Komponente. Wenn eine Komponente K eine neue Kommunikationsverbindung j aufgebaut hat – sie hat $!j$ mit $j \in P$ verschickt –, so darf sie Nachrichten von Kanal j lesen. Verschickt sie $?p$ mit $p \in P$, so darf sie Nachrichten auf p senden. Erhält K den Port $!q$, so darf sie auf q schreiben; erhält sie den Port $?q$, so darf sie von q lesen.
Durch das semantische Modell ist sichergestellt, daß sich die Komponenten *vertraulichkeitserhaltend* (privacy preserving) verhalten: eine Komponente greift nur auf solche Kanäle lesend oder schreibend zu, zu denen sie aktuell über die entsprechenden Zugriffsrechte verfügt. Damit ist das sogenannte *Channel Sharing* (Gemeinsames Verwenden von Kanälen) möglich: mehrere Komponenten können nacheinander das Schreibrecht auf dem gleichen Kanal erhalten; Interferenz, also das gleichzeitige Schreiben auf einen Kanal, ist ausgeschlossen. Channel Sharing wird durch das *Weiterleiten von Ports* („Forwarden“) hervorgerufen. Eine Komponente leitet Ports, die sie nicht selbst erzeugt, sondern empfangen hat, an andere Komponenten weiter, wodurch bestehende Kanalverbindungen umgelenkt werden. Damit wird die klassische FOCUS-Kommunikationsform *point-to-point*, bei der ein Kanal von genau zwei Komponenten verwendet wird, erweitert. Im folgenden behandeln wir point-to-point Kommunikation mit Channel Sharing. Eine Variante, die many-to-many Kommunikation, bei der mehrere Komponenten gleichzeitig lesend oder schreibend auf einen Kanal zugreifen dürfen, wird an einem Beispiel in Abschnitt 4.3 vorgestellt.

Die initiale Schnittstelle einer Komponente ist durch die Mengen I und O gegeben. Um die Veränderung der Schnittstelle einer Komponente K während des Systemablaufs zu erfassen und um zu kontrollieren, auf welche Kanäle sie aktuell zugreifen darf, werden die Mengen *pp* und *ap* im semantischen Modell eingeführt. *ap* (active ports) enthält sämtliche Ports, auf die K aktuell lesend oder schreibend zugreifen kann, und bildet somit die aktuelle Schnittstelle von K. *pp* (private ports) enthält dagegen die Menge aller Ports, die nur K selbst bekannt sind. Das semantische Model stellt sicher, daß diese Mengen verändert werden, sobald die K Ports empfängt oder versendet. Ein öffentlicher Port wird wieder privat, wenn eine Komponente einen Port empfängt, dessen Komplement sie als Ein- bzw. Ausgabeport besitzt (siehe dazu Abschnitt 3.2).

In [Gro96a] wird das Modell für mobile Netze um die *dynamische Erzeugung von Komponenten* erweitert. Das Erzeugen einer neuen Komponente während des Systemablaufs wird in der Spezifikation des Systems durch einen Verfeinerungsschritt modelliert (siehe dazu Abschnitt 3.2). Durch einen rekursiven Funktionsaufruf wird eine neuen Komponente erzeugt und in die bestehende Systemstruktur eingebunden.

3. Methodisches Vorgehen

Wir stellen vor, wie Komponenten in FOCUS durch die Angabe von Prädikaten spezifiziert werden, und geben eine Reihe von Spezifikationsschemata für typische Verhaltensmuster dynamischer, mobiler Systeme an.

3.1 Spezifikationskonzept

Das Verhalten einer mobilen Komponente wird als 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. Mobile Funktionen sind Funktionen, die die im vorherigen Abschnitt beschriebenen Eigenschaften des semantischen Modells erfüllen und sich insbesondere vertraulichkeitserhaltend verhalten. Eine genaue Definition findet sich in der bereits angegebenen Literatur.
Auf semantischer Ebene enthalten Ströme in FOCUS Zeitinformation. Es wird davon ausgegangen, daß innerhalb des Systems eine globale Uhr existiert, die die Zeitachse in diskrete Zeitintervalle unterteilt. Ein gezeiteter Strom ist eine unendliche Sequenz von endlichen Nachrichtensequenzen, wobei jede dieser Nachrichtensequenzen die Kommunikationsgeschichte innerhalb eines Zeitintervalls darstellt. Auf der Spezifikationsebene kann von der Zeitinformation abstrahiert werden – Ströme stellen dann endliche oder unendliche Nachrichtensequenzen dar.
Netzwerke werden durch Komposition von Komponenten gebildet. Hierfür werden die Operatoren $\otimes$ bzw. $\overline{\otimes}$ verwendet, wobei letzterer speziell für die Komposition von gezeiteten mit ungezeiteten Komponenten eingeführt worden ist (siehe [Gro96b]). Bei Spezifikationen werden die Operatoren auch auf funktionaler Ebene verwendet.
Wir spezifizieren im folgenden ausschließlich zeitunabhängig. Eine zeitunabhängig 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. In den Prädikaten

verwenden wir einen konstruktiven Spezifikationsstil auf der Basis von rekursiven Funktionsgleichungen. Eine Funktionsgleichung genügt im wesentlichen folgendem Schema:

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

Diese Funktionsgleichung steht für folgende textuelle Beschreibung:
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. Der Operator & konkateniert Nachrichten mit Strömen.

3.2 Spezifikationsschemata und Leitfäden

In FOCUS werden Systemspezifikationen in einem top-down-Verfahren erstellt. Zunächst wird die Struktur des Systems, also die Unterteilung in Subsysteme, auch Komponenten genannt, festgelegt. Anschließend erfolgt die Spezifikation des Verhaltens der Komponenten. Die Möglichkeit, Komponenten, Subsysteme und Systeme einheitlich zu beschreiben, ist durch das Konzept der *Hierarchie* in FOCUS gegeben. Auf diese Weise ist es beispielsweise möglich, eine Komponente ihrerseits als Netzwerk von Subkomponenten zu beschreiben. 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 verändern. Das System verhält sich in diesen Phasen also im wesentlichen wie ein statisches System. Innerhalb einer Phase können der strukturelle Aufbau des Systems und die Verbindung der Komponenten mit Systemstrukturdiagrammen (SSD) [HSS96, SS95] graphisch beschrieben werden. Komponenten, die direkt miteinander kooperieren, werden mittels Kanälen verbunden. Aus den SSDs kann die Information für eine vollständige formale Definition der Schnittstelle entnommen werden. FOCUS unterstützt eine modulare Systementwicklung. Das gesamte Systemverhalten wird durch die Komposition des Verhaltens der einzelnen Komponenten modelliert.
Das Verhalten einer Komponente kann in einem ersten Schritt informell festgelegt werden, wobei eine schematisierte textuelle Darstellung wie im Beispiel des Abschnitts 4.1 gewählt werden kann. Aus dieser Darstellung ist eine direkte Umsetzung in eine formale Spezifikation, die aus rekursiven Funktionsgleichungen besteht, leicht möglich.
Für das Erzeugen bzw. Löschen und Umlenken von Kommunikationsverbindungen sowie das Erzeugen bzw. Löschen von Komponenten können wir Leitfäden und genaue Spezifikationsschemata in Form von Funktionsgleichungen angeben. Somit kann dynamisches und mobiles Systemverhalten auf schematische Weise formal spezifiziert werden. Im folgenden stellen wir einige Spezifikationsschemata vor. Eine formale Darstellung aller Schemata ist in [Spi97] enthalten.

Erzeugen von Kanälen: Zwischen zwei Komponenten A und B soll eine neue Verbindung erzeugt werden, wobei A den Verbindungsaufbau initiiert und das zugehörige Schreibrecht behält. Dabei sind entsprechend des dazugehörigen Leitfadens folgende Fälle zu unterscheiden:

1. Von A nach B existiert eine direkte Kanalverbindung $AtoB_1$, d.h. A kann mit dem Port $!AtoB_1$ auf diesen Kanal schreiben und B kann mit $?AtoB_1$ von diesem lesen. Zusätzlich besitzt A einen Eingabekanal in_A. Bei Erhalt der Nachricht *new* auf in_A erzeugt A die neue Verbindung $AtoB_2$ zu B. Hierfür werden die Ports $?AtoB_2$ und $!AtoB_2$ aus der

Menge der privaten Ports von A entnommen; das Leserecht $?AtoB_2$ wird über $AtoB_1$ an B gesendet. f_A sei die Funktion, die dieses Verhalten von A beschreibt. f_A liest auf Kanal in_A die Nachricht new, reagiert darauf mit der Ausgabe von $?AtoB_2$ auf Kanal $AtoB_1$ und arbeitet auf dem Reststrom s weiter. Das zugehörige Schema lautet:

$$f_A\,(\{in_A \rightarrow new\}\ \&\ s)\ =\ \{AtoB_1 \rightarrow ?AtoB_2\}\ \&\ f_A(s)$$

2. Von A nach B existiert keine direkte Kanalverbindung, es existiert jedoch eine indirekte Verbindung, d.h. ein Pfad, der über weitere Komponenten führt. Aufgrund dieser indirekten Verbindung ist es möglich, Nachrichten von A an B zu senden. Auf diese Weise kann A als Initiator eine direkte Verbindung zu B erzeugen.

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

Schemata, wie sie in Punkt (1) vorgestellt wurden, müssen innerhalb der Komponentenspezifikationen nur noch mit den aktuell gegebenen Werten (z.B. Kanalnamen, Nachrichtennamen) belegt werden.

Umleiten von Kanälen: Eine Komponente A kann prinzipiell alle Ports, die ihr aktuell zur Verfügung stehen, als Nachrichten versenden. Dafür muß ihr für wenigstens einen der Kanäle ihrer aktuellen Schnittstelle das Schreibrecht zugeteilt sein. *Neue* Verbindungen werden dadurch erzeugt, daß die entsprechenden Ports aus der Menge der für die Komponente aktuell gültigen *privaten* Ports entnommen werden. Durch das Versenden eines Ports p, der zur aktuellen Schnittstelle von A gehört, es gilt also $p \in ap_A$, kann A eine *bestehende* Kanalverbindung *umlenken* und somit das Zugriffsrecht, das sie an diesem Kanal hatte, an eine andere Komponente weitergeben.
Das Schema lautet: Sei in_A ein Kanal, auf dem A aktuell Nachrichten empfangen kann. A sei mit Komponente B über Kanal $AtoB$ verbunden, wobei A über das Schreibrecht verfügt. Bei Erhalt der Nachricht $forward$ über Kanal in_A soll A das Leserecht zu Kanal in_A an B weitergeben. Die Spezifikation erfolgt nach dem Muster:

$$f_A\,(\{in_A \rightarrow forward\}\ \&\ s)\ =\ \{AtoB \rightarrow ?in_A\}\ \&\ f_A(s)$$

Löschen von Kanälen: Es ist neben dem Erzeugen von neuen Kanalverbindungen auch möglich, bestehende Kanalverbindungen zu löschen, indem eine Komponente ein Lese- oder Schreibrecht wieder an den Verbindungspartner zurückschickt.
Eine Komponente gibt das Zugriffsrecht zu dem zu löschenden Kanal an die Komponente zurück, die das komplementäre Recht an diesem Kanal besitzt. Sobald diese Komponente das Zugriffsrecht empfangen hat, verfügt sie über beide Rechte an dem betroffenen Kanal; der entsprechende Port wird aus der Menge der aktiven Ports gelöscht und beide Ports werden in die Menge der privaten Ports aufgenommen. Über den Kanal können keine Nachrichten mehr versendet werden, da der Kanal in der aktuellen Verbindungsstruktur des Systems nicht mehr sichtbar vorhanden ist.
Seien A und B Komponenten, die über den Kanal $AtoB$ verbunden sind, wobei A über das Schreib- und B über das Leserecht verfügt. Kanal $AtoB$ soll nach Empfang der Nachricht *delete* gelöscht werden, die A über in_A bzw. B über in_B empfangen kann.

1. Die Komponente, die über das Schreibrecht an $AtoB$ verfügt, also A, initiiert das Löschen von Kanal $AtoB$. Dies geschieht, indem sie $!AtoB$ über die noch bestehende Verbindung $AtoB$ sendet. Hierfür können wir folgendes Schema angeben:

$$f_A(\{in_A \rightarrow delete\}\ \&\ s)\ =\ \{AtoB \rightarrow !AtoB\}\ \&\ f_A(s)$$

2. B, also die Komponente, die über das Leserecht an $AtoB$ verfügt, soll das Löschen von $AtoB$ initiieren. Dies ist nur dann möglich, wenn eine der beiden folgenden Voraussetzungen erfüllt ist:
   1. Es gibt (mindestens) eine weitere direkte Verbindung zwischen A und B, auf die B schreibend zugreifen kann. B sendet dann $?AtoB$ über diesen Kanal an A.
   2. Es gibt (mindestens) eine indirekte Verbindung von B zu A über einen Pfad, auf den B schreiben darf. B sendet das Leserecht an Kanal $AtoB$ über diesen Pfad. Die Komponenten, über die der Pfad führt, sind in der Lage, den so empfangenen Port an A weiterzuleiten.

Erzeugen von Komponenten: Neben dem Erzeugen einer neuen Komponente muß auch die Eingliederung der Komponente in die vorhandene Systemstruktur spezifiziert werden. 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 an, mit dessen Hilfe das dynamische Erzeugen von Komponenten in ungezeiteten Spezifikationen schematisch durchgeführt werden kann.
Folgende Situation sei zu spezifizieren: Die Komponente *Parent* soll nach Erhalt des Eingabesignals *create* eine Komponente *Child* erzeugen. Innerhalb der FOCUS-Spezifikation von *Parent* wird wie folgt vorgegangen:

$$\exists\, child \in [\![\, Child]\!] :\ parent\,(state)\,(\{in \to create\}\ \&\ s)\ =$$
$$\{out_1 \to n_1,\, out_2 \to n_2,\, \ldots,\, out_m \to n_m\}\,\&\ (parent\,(state')\,\overline{\otimes}\,child\ (M_{ports}))\,(s)$$

Die Komponente *Parent*, beschrieben durch die Funktion *parent*, erzeugt nach Erhalt des Eingabesignals *create* auf ihrem Eingabekanal *in* eine Komponente *Child*. Dabei wird das Erzeugen der neuen Komponente *Child* durch den Aufruf der Funktion *child* modelliert. Die Existenz einer solchen Funktion wird in der Spezifikation durch die Forderung $\exists\, child \in [\![\, Child]\!]$ sichergestellt. *child* ist somit eine Funktion, die das Prädikat P_{Child} erfüllt und ein mögliches Verhalten der Komponente *Child* darstellt. Die neu erzeugte Komponente *Child* wird über den Kompositionsoperator $\overline{\otimes}$ in die bestehende Systemstruktur eingebunden. Mit der Menge der Ports M_{ports}, die der Funktion als Parameter übergeben werden, kann *Child* Kanalverbindungen in der vorhandenen Systemstruktur aufbauen. Sowohl die Funktion *parent* als auch die Funktion *child* arbeiten auf dem Eingabestrom s weiter.

Löschen von Komponenten: 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. 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. Wird die Schnittstelle einer Komponente gelöscht, so läßt sich das Verhalten der Komponente von außen nicht mehr beobachten. Das Löschen der Schnittstelle kann modelliert werden, indem die Komponente die Rechte zu allen Kanälen, die ihre aktuelle Schnittstelle bilden, an andere Komponenten abtritt, sie also über einen ihrer Ausgabekanäle an andere Komponenten sendet.

4. Fallstudien

Hier stellen wir die Anwendung der Leitfäden und einiger Schemata ausführlich anhand der Spezifikation eines Batchsystems vor. Anschließend erläutern wir kurz die Möglichkeiten, die das dynamische, mobile FOCUS-Modell für Prozeßdispatching und die semantische Fundierung des SDL-Konstrukts *Create* bietet.

4.1 Batchsystem

Wir spezifizieren ein einfaches Batchsystem, das sich mobil und dynamisch verhält, um die anstehenden Aufträge aus der Umgebung verteilt und gleichzeitig bearbeiten zu können. Die Aufträge werden jeweils ohne Unterbrechung ausgeführt. Das Batchsystem besteht aus einer zentralen Komponente, die die Aufträge an die ausführenden Komponenten verteilt und bei Bedarf neue Komponenten erzeugt, wobei die sich dabei entwickelnde Systemstruktur eine einheitliche Strukturierung aufweist. Von den funktionalen Eigenschaften des Systems haben wir stark abstrahiert; für eine detaillierte Beschreibung des Batchsystems verweisen wir auf [HS96].

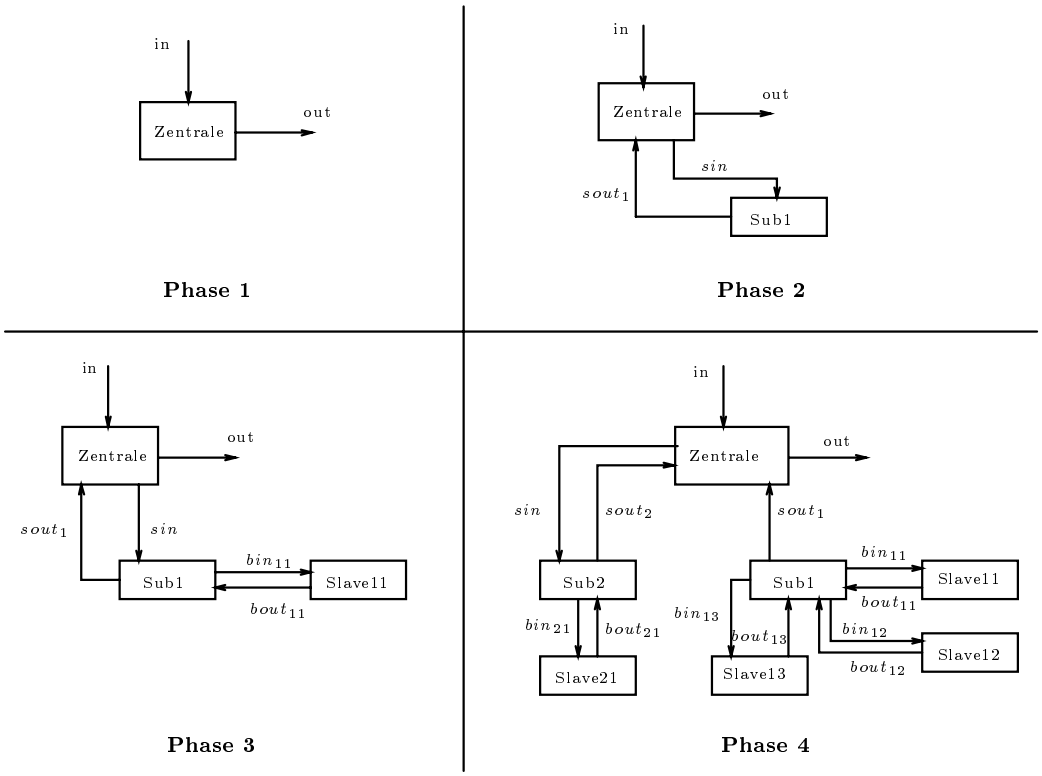

Abbildung 1: Phasen des Batchsystems

Der strukturelle Aufbau und die Kanalverbindungen jeder einzelnen Phase sind, wie in Abschnitt 3.1 beschrieben, mit SSDs graphisch spezifiziert. Abbildung 1 zeigt verschiedene Phasen des Batchsystems. Sei N die Menge aller vorhandenen Kanalnamen:

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

Die folgende Tabelle gibt für alle im Verlauf des Systems vorkommenden Komponenten die Mengen I, O und P an.

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

Für jede Phase geben wir nun die Mengen *ap* und *pp* der beteiligten Komponenten an. Hierbei können die Kanalbezeichner an den SSDs abgelesen werden. Mit *ap* und *pp* werden die verfügbaren Kanalrechte jeder Komponente festgehalten. Dadurch ist es möglich zu überprüfen, ob eine Komponente beim Erzeugen, Löschen oder Umlenken von Kanälen über die entsprechenden Kanalrechte verfügt. Zur kompakten Darstellung der Schnittstellen haben wir eine tabellarische Notation gewählt.

Phase 1: Das Batchsystem besteht anfangs aus einer Komponente *Zentrale*, die die Abarbeitung der Aufträge aus der Umgebung *Env* organisiert und an *Env* Statusmeldungen zurückgibt.

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

Phase 2: Die Zentrale erzeugt zunächst eine erste Subzentrale Sub_1 und leitet die Aufträge aus der Umgebung an diese weiter.

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

Phase 3: Für die Bearbeitung des ersten Auftrags hat Sub_1 die Slave-Komponente $Slave_{11}$ erzeugt.

	ap	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 von Sub_1 ist mit der Erzeugung von drei ausführenden Komponenten erschöpft. Die *Zentrale* hat eine weitere Subzentrale Sub_2 erzeugt, diese wiederum eine Slave-Komponente $Slave_{21}$.

	ap	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	$?sin,\ !sout_2,\ ?bout_{21},\ !bin_{21}$	$\{?!bin_{2j} \mid j \in I\!N, j \geq 2\} \cup \{?!bout_{2j} \mid j \in I\!N, j \geq 2\}$
$Slave_{21}$	$?bin_{21},\ !bout_{21}$	$\emptyset$

Im folgenden zeigen wir Ausschnitte aus der Spezifikation der *Zentrale* und verwenden dabei die zuvor vorgestellten Spezifikationsschemata. Eine vollständige Spezifikation aller Komponenten des Batchsystems ist in [HS96] zu finden.

Initialisierung: 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. Diese erhält bei ihrer Erzeugung die Rechte auf die Kanäle *sin* und $sout_1$ als Parameter und ist somit in die vorhandene Systemstruktur eingebunden. Das Verhalten von Sub_1 wird durch die Funktion sub_1 beschrieben; das Verhalten der Zentrale durch die Funktion g. Nach dem Erzeugen von Sub_1 wird das weitere Verhalten der Zentrale durch die Funktion h beschrieben.

$$s = \{in \ \rightarrow \ Login\} \ \& \ s' \quad \Longrightarrow$$
$$g(s) = \{out \ \rightarrow \ Conf \& New(Sub_1)\} \ \& \ (h \ \overline{\otimes} \ sub_1(?sin, !sout_1))(s')$$

Weiterleiten von Nachrichten: Wenn die Zentrale die Nachricht $New(Slave_{1j})$ (Sub_1 hat eine neue Slave-Komponente erzeugt) 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.

$$h(\{sout_1 \ \rightarrow \ New(Slave_{1j}\} \ \& \ t) = \{out \ \rightarrow \ New(Slave_{1j}\} \ \& \ h(t)$$

$$h(\{sout_1 \ \rightarrow \ State_{1j}\} \ \& \ t) = \{out \ \rightarrow \ State_{1j}\} \ \& \ h(t)$$

Auftrag aus Umgebung: Auf den Empfang eines Auftrags A aus der Umgebung über Kanal *in* reagiert die Zentrale wie folgt: sie sendet A über Kanal *sin* an die aktuelle Subzentrale Sub_1 und die Nachricht $A\_to\_Sub_1$ (zur Information) über Kanal *out* an die Umgebung. Die Zentrale leitet einen Auftrag A, der von Kanal *in* gelesen wurde, erst dann an Sub_1 weiter, wenn sichergestellt ist, daß deren Kapazität noch nicht erschöpft ist. Dies kann sie anhand der Statusmeldungen $NotFull(Sub_1)$ bzw. $Full(Sub_1)$ überprüfen.

Fall 1: Die Kapazität der Subzentrale ist noch nicht erschöpft:

$$h(\{in \ \rightarrow \ A, sout_1 \ \rightarrow \ NotFull(Sub_1)\} \ \& \ t) = \{out \ \rightarrow \ A\_to\_Sub_1, sin \ \rightarrow \ A\} \ \& \ h(t)$$

Fall 2: 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 an *in* an, wird die nächste Subzentrale Sub_2 kreiert. Die Meldungen $New(Sub_2)$, $A\_to\_Sub_2$ und alle anliegenden Statusmeldungen der übrigen Subzentralen werden nach außen gegeben. Die neue Subzentrale erhält über Kanal *sin* den Auftrag A.

$$h(\{in \rightarrow A, sout_1 \rightarrow Full(Sub_1) \ \& \ ?sin\} \ \& \ t) = (h \ \overline{\otimes} \ sub_2(?sin, !sout_2))(\{sin \rightarrow A\} \ \& \ t)$$

4.2 Prozeßdispatching

Eine der wichtigsten Managementaufgaben eines Betriebssystems besteht in der Prozessorverwaltung, also der Zuteilung des Prozessors an die im Rechensystem zur Bearbeitung anstehenden Aufträge, den rechenbereiten Prozessen.
Der semantische Ansatz von FOCUS zur Spezifikation mobilen, dynamischen Verhaltens eignet sich sehr gut für die Modellierung der Prozessorverwaltung, insbesondere für das Dispatchen von Prozessen und Prozessor. Der Dispatcher hat die Aufgabe, rechenbereite Prozesse mit dem Prozessor zu koppeln. Diese Kopplung modellieren wir dadurch, daß der Dispatcher den Aufbau neuer Kanalverbindungen zwischen dem ausgewählten Prozeß und dem Prozessor initiiert und gleichzeitig einen Zeitgeber startet, der den Ablauf der zugehörigen Zeitscheibe mißt. Dieses Verhalten kann im wesentlichen mit den Schemata aus Abschnitt 3.2 modelliert werden. Da sich im allgemeinen mehrere rechenbereite Prozesse um die Zuteilung des Prozessors bewerben, müssen diese Anforderungen alle berücksichtigt werden und keine Anforderung darf verloren gehen. Dieser Fall entspricht einer klassischen Situation bei der Modellierung mit FOCUS. Die Anforderungen müssen

„fair gemischt“ und an den Dispatcher weitergeleitet werden. Aufgrund des point-to-point-Modells ist es notwendig, hierfür eine weitere Komponente in das System einzufügen. Dies entspricht einem Scheduler in der Prozessorverwaltung. Eine detaillierte Modellierung dieses Verhaltens und der methodische Einsatz von Spezifikationsschemata werden detailliert in [Spi97] vorgestellt.

4.3 Prozeßerzeugung in SDL

In der Spezifikationssprache SDL ist es möglich, mit dem Sprachkonstrukt *Create* während des Systemablaufs neue Komponenten, SDL-Prozesse, zu erzeugen. Die formale Fundierung dieses Ablaufs kann mittels der dynamischen, mobilen Erweiterung von FOCUS erfolgen. Hierbei bietet sich die Verwendung der Kommunikationsform many-to-many an. Ein SDL-Prozeß erhält auf mehreren Signalwegen Eingabesignale zugesendet, die in den Eingabepuffer des Prozesses geschrieben werden, der nach dem FIFO-Prinzip abgearbeitet wird. Signale, die den Prozeß gleichzeitig erreichen, werden in beliebiger Reihenfolge in den Puffer eingetragen.
Bei Verwendung der many-to-many Kommunikation entfällt die explizite Spezifikation des Eingabepuffers; die Interferenz, die auftritt, wenn mehrere Komponenten auf einen Kanal gleichzeitig zugreifen, wird im semantischen Modell aufgelöst. In den Netzwerkoperator $\otimes$ sind implizite Fair-Merge-Komponenten eingebaut. Diese mischen die Ströme fair und ohne zeitliche Verzögerung. Demzufolge können wir sämtliche Signalwege, die einen SDL-Prozeß erreichen, in FOCUS in einen Kanal münden lassen, wobei dieser Kanal den Eingabepuffer des Prozesses darstellt.
Damit läßt sich unter Verwendung der Spezifikationsschemata für das Erzeugen neuer Komponenten und dem Erzeugen und Weiterleiten von Kanälen eine adäquate formale Fundierung der dynamischen Prozeßerzeugung von SDL erzielen (siehe [Hin]).

5. Abschließende Bemerkung

Im vorliegenden Beitrag haben wir ausgehend von dem in [GS96] beschriebenen semantischen Modell eine Methode für die Spezifikation mobiler, dynamischer Netze in FOCUS vorgestellt. Dabei stand die Vorgabe von Leitfäden und Spezifikationsschemata für dynamisches, mobiles Systemverhalten im Vordergrund. Unser Vorgehen wurde mittels der Spezifikation eines einfachen Batchsystems veranschaulicht und anhand zweier weiterer Anwendungsbereiche vorgestellt.

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 Batchsystems hat zu umfangreichen Spezifikationen geführt. Die Verwendung von tabellarischen Übersichten und Spezifikationen sowie von Spezifikationsschemata 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 der in diesem Bericht vorgestellten Beispiele bewährt. Dank der Leitfäden und Spezifikationsschemata ist es möglich, sich auf die Erfassung und Lösung des Anwendungsproblems zu konzentrieren und technische Details von FOCUS außer acht zu lassen.

Danksagung

Wir möchten uns 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, Monika Schmidt und Oscar Slotosch für ihre hilfreichen Kommentare.

Literatur

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

[BS97] M. Broy and K. Stølen. FOCUS on System Development – A Method for the Development of Interactive Systems, 1997. Manuskript.

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

[Gro96b] R. Grosu. Recursive Networks and Time Abstraction, 1996.

[GS96] R. Grosu and K. Stølen. A Model for Mobile Point-to-Point Data-flow Networks without Channel Sharing. In M. Nivat M. Wirsing, editor, *AMAST'96*. LNCS 1101, 1996.

[GSB97] R. Grosu, K. Stølen, and M. Broy. A model for mobile point-to-point data-flow networks with channel sharing. Technical Report SFB 342/17/97A, Technische Universität München, 1997.

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

[HS96] U. Hinkel and K. Spies. Anleitung zur Spezifikation von mobilen, dynamischen FOCUS-Netzen. SFB-Bericht 342/16/96 A, Technische Universität München, November 1996.

[HSS96] F. Huber, B. Schätz, and K. Spies. AutoFocus - Ein Werkzeugkonzept zur Beschreibung verteilter Systeme . In Ulrich Herzog Holger Hermanns, editor, *Formale Beschreibungstechniken für verteilte Systeme*, pages 165–174. Universität Erlangen-Nürnberg, 1996. Erschienen in: Arbeitsbereichte des Insituts für mathematische Maschinen und Datenverarbeitung, Bd.29, Nr. 9.

[Kah74] G. Kahn. The Semantics of a Simple Language for Parallel Programming. *Information Processing*, pages 471 – 475, 1974.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part i, ii. *Information and Computation*, 100:1–40, 1992.

[Spi97] K. Spies. Spezifikationsmethodik für die Modellierung von Betriebssystemkonzepten, 1997. Dissertation, *vorläufige Version*.

[SS95] B. Schätz and K. Spies. Formale Syntax zur logischen Kernsprache der FOCUS-Entwicklungsmethodik. SFB-Bericht 342/16/95 A, Technische Universität München, Institut für Informatik, 1995.