Ein strombasiertes mathematisches Modell verteilter informationsverarbeitender Systeme - SysLab Systemmodell -

Bernhard Rumpe, Cornel Klein, Manfred Broy
Institut für Informatik, Technische Universität München
80333 München, Germany
e-mail: (broy|klein|rumpe)@informatik.tu-muenchen.de

24. März 1995

Zusammenfassung

Das SysLab-Systemmodell dient im Projekt SysLab als mathematisches Referenzmodell für die Formalisierung der Semantik von Beschreibungstechniken. Ausgehend von den Anforderungen, welche an ein solches abstraktes mathematisches Modell informationsverarbeitender Systeme zu stellen sind, definieren wir das Systemmodell mit seinen verschiedenen Sichten und ihren Beziehungen.

[0]Diese Arbeit entstand im Projekt Syslab, finanziert mit Unterstützung der DFG unter dem Leibnizpreis und durch die Firma Siemens-Nixdorf

[RKB95] B. Rumpe, C. Klein, M. Broy.
Ein strombasiertes mathematisches Modell verteilter informationsverarbeitender Systeme - Syslab Systemmodell.
Technical Report TUM-I9510, TU Munich, 1995.
www.se-rwth.de/publications

1 Einleitung

Methoden zur System- und Softwareentwicklung, wie OMT [RBP+92], Fusion [CAB+94], und GRAPES [Hel90], modellieren ein System auf unterschiedlichen Abstraktionsebenen und unter unterschiedlichen Sichtweisen. Zur Modellierung verwenden sie Beschreibungstechniken wie Entity-/Relationship-Diagramme und deren objektorientierte Erweiterungen, Zustandsautomaten, Sequence Charts oder Datenflußdiagramme. Problematisch bei den auf dem Markt befindlichen Methoden ist, daß sowohl die einzelnen Beschreibungstechniken, als auch die Beziehungen zwischen den verschiedenen Beschreibungstechniken einer Methode in der Regel nur informell erklärt sind. Aufgrund der dadurch entstehenden Mehrdeutigkeiten und Unklarheiten bezüglich der Semantik der Modellierungskonzepte ergeben sich eine Reihe von Problemen beim Methodeneinsatz:

- Die Kommunikation zwischen den Projektbeteiligten ist erschwert,
- es ist kein klares Konzept für die Übernahme von und die Beziehungen zwischen Informationen verschiedener Beschreibungsebenen definierbar,
- eine verbindliche Grundlage für die Werkzeugunterstützung fehlt,
- auch innerhalb einer Beschreibungsebene sind Fragen nach der Konsistenz und Vollständigkeit einer Menge von Beschreibungsdokumenten nur informell überprüfbar.

Vor diesem Hintergrund wird auch verständlich, warum Werkzeugsysteme für Methoden („CASE-Tools“) selten den gewünschten Produktivitätsgewinn herbeiführen: Aufgrund der mangelnden semantischen Fundierung der Methoden sind die mit ihnen erarbeiteten Informationen meist so wenig aussagekräftig, daß sich die Funktionalität der Werkzeuge auf reine Dokumenteneditierungs- und Verwaltungsfunktionen beschränken muß.

In jüngster Zeit gibt es deshalb einige Ansätze, Methoden zur System- und Softwareentwicklung zu formalisieren. Bekannt sind Formalisierungen durch sogenannte „Metamodelle“, wie sie aus dem Bereich der Werkzeugintegration kommen (vgl. [CDI92], [Tho89] und [HL93]). Dadurch wird jedoch meist nur die abstrakte Syntax der Beschreibungstechniken erfaßt. Ein Überblick über verschiedene Projekte welche die Integration strukturierter Methoden mit Techniken der formalen Spezifikation anstreben ist in [SFD92] zu finden. Weiter als diese Formalisierungsansätze geht die in [Hus94] beschriebene Formalisierung der britischen Standardmethode SSADM [AG90] in der algebraischen Spezifikationssprache SPECTRUM [BFG+93]. In dieser Arbeit gibt Hußmann ein mathematisches Modell der mit SSADM modellierten Informationssysteme an, zu dem er die verschiedenen in der Methode vorkommenden Beschreibungstechniken in Beziehung setzt. Dies erlaubt eine umfassende Analyse der Semantik der SSADM-Beschreibungstechniken und ihrer Beziehungen, die Angabe von Bedingungen für die Konsistenz und Vollständigkeit einer Menge von Beschreibungsdokumenten, und die einfache Erstellung von Prototypen durch funktionale Programme.

1.1 Rolle des Systemmodells in SysLab

Ziel des Projekts SysLab [Bro94] ist es, eine praktikable, wissenschaftlich fundierte Methode für die System- und Softwareentwicklung zu schaffen, welche den Nachteil einer mangelnden semantischen Fundierung nicht mehr aufweist, und einen Prototypen eines Werkzeugsystems zur Unterstützung dieser Methode zu erstellen. Die Formalisierung ist dabei nicht Selbstzweck, sondern sie liefert die wissenschaftliche Basis für die Überprüfung der Konsistenz der Konzepte. Die semantische Fundierung geschieht dabei durch die Verwendung eines SysLab-einheitlichen Systemmodells als Referenzmodell. Zu diesem abstrakten mathematischen Modell informationsverarbeitender Systeme werden alle in der SysLab-Methode vorkommenden Beschreibungstechniken, wie Objektdiagramme, Zustandsdiagramme, Datenflußdiagramme etc. und alle Transformationsregeln für die Transformation von Beschreibungsdokumenten in Beziehung gesetzt. Ein konkretes Beschreibungsdokument wie beispielsweise ein Objektdiagramm wird dabei als eine Aussage über dem mathematischen Systemmodell aufgefaßt.

Die Formalisierung von Beschreibungstechniken führt zunächst zu einem tieferen Verständnis der Bedeutung von Beschreibungen, der Aspekte, über welche sie Aussagen treffen, und ihrer Querbeziehungen. Beschreibungstechniken können damit zielgerichteter eingesetzt werden. Weiter können damit Bedingungen für die Konsistenz und Vollständigkeit einer Menge von Beschreibungsdokumenten angegeben werden und auch Beziehungen zwischen Beschreibungsdokumenten verschiedener Abstraktionsebenen definiert und analysiert werden. Letztlich ist die Formalisierung auch ein wichtiger Meilenstein auf dem Weg zu einer effektiveren Werkzeugunterstützung von Methoden, da nun bedeutungserhaltende Transformationen zwischen verschiedenen Beschreibungstechniken möglich sind bis hin zu einer Transformation in ausführbaren Code. Ferner wird dadurch der flexible Einsatz formaler Techniken ermöglicht, der in sicherheitskritischen Anwendungen erforderlich ist.

Das Systemmodell spielt damit eine zentrale Rolle im Projekt SysLab, sowohl für die Methode und die verwendeten Beschreibungstechniken als auch für die zu erstellenden Werkzeuge. Es ist Ziel dieses ersten SysLab-Berichts, das Systemmodell als Grundlage für die weiteren Arbeiten in SysLab vorzustellen und festzulegen. Die vorliegende Schrift wendet sich dabei nicht nur an SysLab-Projektbeteiligte, sondern an alle, welche mit der SysLab-Methode in Berührung kommen und die Bedeutung der in ihr vorkommenden Konzepte verstehen wollen, also auch an Methodenanwender oder an im Entwicklungsprozeß eines mit der SysLab-Methode erstellten Systems involvierte Benutzer. Aus diesem Grund wurde vor allem darauf geachtet, möglichst wenig mathematische Grundlagen vorauszusetzen, einen in sich abgeschlossenen Text zu erstellen, welcher auch ohne Kenntnis weiterer Literatur verständlich ist, und alle verwendeten mathematischen Konzepte sauber und verständlich einzuführen und zu erklären.

1.2 Anforderungen an das Systemmodell

Zweck des Systemmodells ist es, im Rahmen des SysLab-Projekts eine gemeinsame Ausgangsbasis für alle Projektbeteiligten zum Systembegriff und zur Definition der Semantik von Beschreibungstechniken festzulegen. Damit muß das Systemmodell alle Phasen und alle Beschreibungstechniken der SysLab-Methode abdecken, und es darf auch nicht auf eine spezielle Klasse von informationsverarbeitenden Systemen, wie betriebliche Informationssysteme, eingeschränkt sein. Hieraus ergibt sich die Forderung, daß ein möglichst *allgemeines* Modell informationsverarbeitender Systeme zu erstellen ist. Andererseits sollen für die Beschreibungstechniken, die im Rahmen des SysLab-Projekts entstehen, auf möglichst einfache Weise Semantiken auf Basis des Systemmodells definiert werden können. Daraus ergibt sich die Forderung, daß ein möglichst auf die ins Auge gefaßten Beschreibungstechniken zugeschnittenes Systemmodell zu erstellen ist.

Grundannahme bezüglich der Struktur informationsverarbeitender Systeme ist, daß solche Systeme hierarchisch und modular aus einer Reihe von *Komponenten* aufgebaut sind, welche parallel interagieren und ihrerseits als informationsverarbeitende Systeme aufgefaßt werden können. Wir sprechen in diesem Fall von einem *verteilten System*. Dabei gibt es Systeme, welche nicht mehr weiter parallelisiert oder verteilt werden sollen. Solche *Basiskomponenten* können beispielsweise durch einen Zustandsautomaten mit Ein- und Ausgabe beschrieben werden. Durch wiederholte Dekomposition eines Systems in Teilsysteme entsteht in natürlicher Weise ein hierarchisch aufgebautes System, dessen Struktur als ein Baum mit verteilten Systemen an den inneren Knoten und Basiskomponenten an den Blättern aufgefaßt werden kann.

Wir sind an einem Systemmodell interessiert, in dem jede Art der Interaktion konsequent durch Nachrichtenaustausch erfolgt. Solche Systeme besitzen *Eingabeports*, an denen sie Nachrichten von der Umgebung empfangen, und *Ausgabeports*, über die sie Nachrichten an ihre Umgebung senden können. Durch die Ports ist die *Schnittstelle* des Systems gegeben, sie bilden die einzige Möglichkeit der Interaktion zwischen einem System und seiner Umgebung. Das *Verhalten* eines Systems ist die Beziehung zwischen den Folgen von Nachrichten an seinen Eingabeports und den Folgen von Nachrichten auf seinen Ausgabeports. Systeme und ihre Komponenten sind sowohl Daten- als auch Prozeßkapseln: *Datenkapselung* bedeutet, daß der Zustand nach außen nicht sichtbar und nur durch explizite Kommunikation zu erfahren ist. *Prozeßkapselung* bedeutet, daß bei der Kommunikation keine Übertragung von Kontrolle stattfindet, und somit jede Komponente ein eigenständiger Prozeß ist.

Der Nachrichtenaustausch zwischen den Systemkomponenten erfolgt *asynchron*. Dies bedeutet, daß das Senden einer Nachricht unabhängig von einer möglichen Empfangsbereitschaft des Empfängers geschehen kann. Die Forderung nach asynchroner Kommunikation ergibt sich aus Erfahrungen im Projekt FOCUS [BDD+93]: Asynchron kommunizierende Systeme bilden das abstrakteste Modell für Systeme mit Nachrichtenaustausch. Sie können leicht durch stromverarbeitende Funktionen modelliert werden, für welche eine Vielfalt von leicht

handhabbaren Spezifikations- und Verfeinerungstechniken vorhanden ist. *Synchrone Kommunikation* kann jedoch auf asynchrone Kommunikation zurückgeführt werden, beispielsweise durch Einführung eines Protokolls zwischen den kommunizierenden Komponenten.

Das Systemmodell soll möglichst keine Festlegungen bezüglich der *Adressierung von Nachrichten* treffen. Eine Möglichkeit hierzu ist, die Ein- und Ausgabeports der Komponenten statisch durch *Kanäle* miteinander zu verbinden. Alternativ ist es in unserem Modell möglich, eine Adressierung mittels *Identifikatoren* vorzunehmen, wie sie insbesondere im Bereich objektorientierter Programmiersprachen Anwendung findet. Im Rahmen objektorientierter Sprachen erscheint es weiterhin notwendig, nicht von einer statischen Menge von Komponenten auszugehen, sondern das *dynamische Erzeugen* von Komponenten zuzulassen. Diese Anforderungen führen zu zwei Konzepten für die Kommunikation. Eines nutzt Ports und das andere nutzt Identifikatoren. Das Systemmodell ist für beide Konzepte vorzubereiten, so daß je nach Anwendungsfall eines oder auch eine Kombination der beiden genutzt werden kann.

Kein Platz ist im Systemmodell hingegen für weitere objektorientierte Konzepte wie Klassenbeschreibungen und Vererbungshierarchien. Diese werden als Beschreibungstechniken aufgefaßt, welche ihre Semantik durch das mathematische Systemmodell erhalten.

Für die Betrachtung von Systemen, in denen Zeit eine Rolle spielt, muß im Systemmodell über die (etwa durch die Monotonieanforderung bei stromverarbeitenden Funktionen formalisierte) Kausalität hinausgehend ein expliziter Zeitbegriff vorhanden sein. Hier nehmen wir an, daß eine *diskrete Zeit*, welche durch Unterteilung der Zeitachse in *äquidistante Zeitintervalle* entsteht, für die Zwecke von SysLab ausreichend ist.

Das Systemmodell ist ein Referenzmodell, auf welches sich die SysLab-Methodenbeschreibung, die Semantikdefinitionen für Beschreibungstechniken und die Werkzeugentwicklung beziehen. Als solches ist es vor allem als Grundlage für die Kommunikation zwischen den Projektbeteiligten aufzufassen und entsprechend zu präsentieren. Da in SysLab Fragen der Verfeinerbarkeit und Verifikation, wie sie beispielsweise in den Projekten FOCUS [BDD+93] und SPECTRUM [BFG+93] angegangen werden, zumindest zunächst eine untergeordnete Rolle spielen, erscheint es deshalb nicht erforderlich, eine konkrete Syntax oder gar einen Beweiskalkül für das Systemmodell anzugeben oder das Systemmodell in einer formalen Logik zu kodieren. Wir beschränken uns deshalb in diesem ersten Ansatz auf eine rein mathematische Darstellung des Systemmodells, wobei allerdings nicht ausgeschlossen ist, daß zukünftige Weiterentwicklungen des Systemmodells eine formalere Syntax und Semantik erhalten.

Der Bericht ist folgendermaßen aufgebaut: Im kommenden Abschnitt wird die Black-Box Sicht von Systemen vorgestellt. Hierzu wird zunächst die mathematische Struktur der Ströme beschrieben, anschließend werden stromverarbeitende Funktionen als Modell interaktiver Systeme vorstellt, und schließlich werden Identifikatoren für Komponenten eingeführt. In Abschnitt 3 werden dann die beiden Glass-Box Sichten „Das System als Basiskomponente“ und „Das Sy-

stem als verteiltes System“ vorgestellt, und in Abschnitt 4 wird das vorgestellte Systemmodell zu den in diesem Abschnitt aufgestellten Anforderungen in Beziehung gesetzt.

Alle Spezifikationen, die zur Definition des Systemmodells verwendet werden, charakterisieren die Komponenten dieses Systemmodells. Insbesondere sind daher rekursive Spezifikationen nicht als Definitionen unter Zuhilfenahme der Fixpunkttheorie, sondern als charakterisierende Eigenschaften zu verstehen.

2 Black-Box Sicht

Ein informationsverarbeitendes System ist eine Einheit, welche durch den Austausch von *Nachrichten* mit seiner Umgebung interagiert. Die Schnittstelle zwischen System und Umgebung besteht dabei aus sogenannten *Ports* (oft auch Kanäle genannt), über welche Daten fließen. Dabei unterscheiden wir zwischen *Eingabeports* und *Ausgabeports*. Eine graphische Darstellung einer Komponente mit den Eingabeports $port_1$ und $port_2$ und den Ausgabeports $port_3$, $port_4$ und $port_5$ ist in Abbildung 1 gegeben. Dabei nehmen wir an, daß alle Portnamen wie $port_1 \ldots port_5$ in der höchsten abzählbaren Menge P der *Portnamen* enthalten sind.

Abbildung 1: Black-Box Sicht eines Systems

Zur Laufzeit empfängt ein System auf seinen Eingabeports Nachrichten und sendet entsprechend seinem Verhalten auf den Ausgabeports Nachrichten. Im folgenden werden wir zunächst auf die mathematische Struktur der *Ströme* eingehen, welche die Kommunikationsgeschichte auf Ports modelliert, anschließend auf *stromverarbeitende Funktionen* als mathematisches Modell interaktiver Systeme und schließlich auf *Identifikatoren* von Komponenten in unserem Systemmodell.

2.1 Ströme

Das Verhalten von Systemen modellieren wir durch seine Abläufe, welche die Beziehung zwischen den auf den Eingabeports des Systems eingehenden Nachrichten und den auf den Ausgabeports des Systems ausgehenden Nachrichten beschreiben. Hierbei nehmen wir an, daß für jeden Ablauf die Ereignisse auf einem Port linear geordnet sind, so daß für zwei verschiedene Ereignisse stets

eines zeitlich und kausal vor dem anderen liegt. Dies erlaubt es, die Kommunikationsgeschichte auf Ports durch Ströme von Nachrichten zu modellieren.

Ein *Strom* ist ein endliche oder unendliche Folgen von Nachrichten. Bezeichnet M die Menge der Nachrichten, M^* die Menge der endlichen Folgen von Nachrichten und M^∞ die Menge der unendlichen Folgen von Nachrichten, so gilt für die Menge der Ströme über M, bezeichnet mit M^ω:

$$M^\omega = M^\infty \cup M^*$$

Wir verwenden die folgenden Operationen auf Strömen:

- $\hat{}\ : M^\omega \times M^\omega \rightarrow M^\omega$ bezeichnet die Konkatenation zweier Ströme, also den Strom welcher durch das Hintereinanderstellen der beiden Argumente entsteht. Der Operator $\hat{}$ wird in der Regel in Infixnotation geschrieben. Es gilt

  $$s \in M^\infty \Rightarrow s\hat{}t = s,$$

  das heißt die Konkatenation eines unendlichen Stroms s mit einen Strom t liefert wieder den Strom s. $\hat{}$ wird auch zur Konkatenation eines Elements mit einem Strom verwendet.

- $\# : M^\omega \rightarrow Nat \cup \{\infty\}$ Liefert die Länge eines Stroms als natürliche Zahl oder ∞, wenn der Strom unendlich lang ist.

- $Filter\ : \mathcal{P}(M) \times M^\omega \rightarrow M^\omega$ bezeichnet die Filterfunktion. $Filter(N, s)$ löscht aus dem Strom s alle nicht in der Menge N enthaltenen Folgenglieder von s heraus.

Unser Systemmodell verfügt zusätzlich zu der durch Ströme von Nachrichten modellierten linearen Ordnung von Ereignissen über einen expliziten Zeitbegriff. Hierbei nehmen wir wie in [Stø95] an, daß die Zeit in äquidistanten Zeitintervallen (Takten) voranschreitet, und kennzeichnen das Voranschreiten der Zeit um einen Takt in einer Kommunikationsgeschichte durch ein Zeitsignal $\surd \notin M$, genannt *Tick*. Wir bezeichnen mit $M^\surd$ die Menge $M \cup \{\surd\}$ und definieren:

$$M^{\overline{\infty}} = \{s \in (M^\surd)^\omega | \#(Filter(\{\surd\}, s)) = \infty\}$$
$$M^{\overline{*}} = (M^\surd)^*$$

Die Menge $M^{\overline{\infty}}$ ist die Menge aller unendlichen Folgen von Elementen aus $M \cup \{\surd\}$, welche unendlich viele $\surd$ enthält. Die Forderung nach unendlich vielen $\surd$ modelliert die Tatsache, daß die Zeit niemals endet und wir nur unendliche Kommunikationgeschichten betrachten. Ströme aus $M^\surd$ enthalten zwischen je zwei $\surd$ jeweils endlich viele Nachrichten aus M. Die Menge $M^{\overline{*}}$ werden wir im folgenden verwenden, um über endliche Präfixe von unendlichen Strömen zu sprechen.

Wenn wir annehmen, daß *In* die Menge der Eingabeports und *Out* die Menge der Ausgabeports eines Systems bezeichnet, so kann eine Kommunikationsgeschichte auf den Ein- und Ausgabeports durch Funktionen *in* und *out* beschrieben werden, welche den Ports Ströme von Nachrichten und Ticks zuordnen:

$$in : In \rightarrow M^{\overline{\infty}}$$
$$out : Out \rightarrow M^{\overline{\infty}}$$

Funktionen wie *in* und *out*, welche Portnamen auf gezeitete Nachrichtenströme abbilden, bezeichnen wir als *Bündel* von Nachrichtenströmen. Die Auswahl eines Nachrichtenstroms für den Port *portname* aus einem Bündel von Strömen b entspricht damit der Funktionsapplikation, welche wir in diesem Fall aus Gründen der besseren Lesbarkeit in der Form

$$b.portname$$

schreiben.

2.2 Stromverarbeitende Funktionen als Modell interaktiver Systeme

Das *Verhalten* eines Systems modellieren wir durch eine *gezeitete stromverarbeitende Funktion*, welche jeder Belegung der Eingabeports mit gezeiteten Nachrichtenströmen eine Belegung der Ausgabeports mit gezeiteten Nachrichtenströmen zuordnet:

$$Behaviour : (In \rightarrow M^{\overline{\infty}}) \rightarrow (Out \rightarrow M^{\overline{\infty}})$$

Nicht jede solche Funktion stellt allerdings ein sinnvolles Modell eines informationsverarbeitenden Systems dar: In der Realität ist es unmöglich, daß zu einem Zeitpunkt die Ausgabe eines Systems von einer möglicherweise erst später auftretenden Eingabenachricht abhängt. Um diesem Umstand Rechnung zu tragen, fordern wir eine entsprechende mathematische Eigenschaft. Dazu definieren wir zunächst die Funktion

$$\downarrow : M^{\overline{\infty}} \times Nat \rightarrow M^{\overline{*}},$$

welche wir in Infixnotation schreiben. $s \downarrow j$ liefert die ersten j Zeitabschnitte des Stroms s, das heißt $s \downarrow j$ ist derjenige Präfix von s, dessen letzte Nachricht der j.te Tick ist oder der leere Strom wenn j gleich 0 ist. $s \downarrow j$ enthält also genau j Ticks und ist ein Anfang von Strom s:

$$\#(Filter(\{\surd\}, s \downarrow j)) = j$$
$$\exists t : M^{\overline{\infty}} : (s \downarrow j)\hat{}t = s$$

Außerdem wird der Strom immer genau nach dem j.ten Tick abgeschnitten, beziehungsweise ist für $j = 0$ leer:

$$j = 0 \Rightarrow \#(s \downarrow j) = 0$$
$$j > 0 \Rightarrow \exists t : M^{\overline{*}} : s \downarrow j = t \hat{} \surd$$

Die Operation $\downarrow$ läßt sich auf Bündel unendlicher gezeiteter Ströme durch punktweise Anwendung auf einzelne Ströme fortsetzen. Sei $s : L \rightarrow M^{\overline{\infty}}$ mit $L \subseteq P$ ein solches Bündel, so gilt also:

$$(s \downarrow j).portname = (s.portname) \downarrow j$$

Die Funktion $\downarrow$ benutzen wir, um die Forderung, daß die Ausgabe einer Komponente zu einem Zeitpunkt nicht von möglicherweise erst in der Zukunft auftretenden Eingaben abhängen darf, formal zu fassen. Hierzu fordern wir, daß stromverarbeitende Funktionen zur Modellierung von Systemen *pulsgetrieben* sind. Für unsere Funktion *Behaviour* bedeutet dies, daß sie angewandt auf zwei Eingabestrombündel s und t, die bis zum Zeitpunkt j gleich sind, die gleiche Ausgabe bis zu diesem Zeitpunkt produzieren muß, formal:

$$s \downarrow j = t \downarrow j \Rightarrow Behaviour(s) \downarrow j = Behaviour(t) \downarrow j$$

Die stärkere Forderung

$$s \downarrow j = t \downarrow j \Rightarrow Behaviour(s) \downarrow j + 1 = Behaviour(t) \downarrow j + 1$$

kann erhoben werden, wenn wir die Zeitgranularität so fein wählen, daß für jede Komponente eine verzögerte Reaktion sichtbar wird. Dies stellt die Existenz eindeutiger Fixpunkte für Rückkopplungen sicher.

Stromverarbeitende Funktionen sind im folgenden Funktionen, welche Bündel von Eingabeströmen auf Bündel von Ausgabeströmen abbilden, und welche pulsgetrieben sind. Wir bezeichnen die Menge dieser Funktionen mit

$$(In \rightarrow M^{\overline{\infty}}) \xrightarrow{p} (Out \rightarrow M^{\overline{\infty}}).$$

Wir charakterisieren im folgenden die Menge der verteilten Systeme, die wir modellieren wollen. Dies tun wir, indem wir ein beliebiges Exemplar aus dieser Menge herausnehmen und seine Eigenschaften beschreiben. Dadurch wird nur die nachfolgende Beschreibung vereinfacht und damit erheblich verständlicher.

2.3 Identifikatoren

Wir wollen Systeme betrachten, in denen die Adressierung von Nachrichten durch die Angabe von *Identifikatoren* in Nachrichten geschehen kann, wie etwa bei Programmen in objektorientierten Programmiersprachen. Im Systemmodell verwenden wir hierfür die abzählbare Menge ID der Identifikatoren. Jeder Identifikator bezeichnet genau eine Systemkomponente, und jede Komponente hat genau einen Identifikator. Die Menge der Eingabeports eines Systems mit Identifikator id bezeichnen wir mit In_{id} und die Menge seiner Ausgabeports mit Out_{id}. Formal sind In und Out Funktionen, welche einem Identifikator eine Menge von Portnamen zuordnen:

$$In : ID \rightarrow \mathcal{P}(P)$$
$$Out : ID \rightarrow \mathcal{P}(P)$$

Die Anwendung von In und Out schreiben wir hier in der Form In_{id} und Out_{id}. Wir nehmen an, daß die Portmengen verschiedener Komponenten disjunkt sind:

$$id \neq id' \Rightarrow (In_{id} \cup Out_{id}) \cap (In_{id'} \cup Out_{id'}) = \emptyset$$

Diese Forderung schränkt die Mächtigkeit des Systemmodells nicht ein, sie vereinfacht aber die folgenden Ausführungen, da dadurch einem Portnamen eindeutig die Komponente, zu der der entsprechende Port gehört, zugeordnet werden kann.

Die stromverarbeitende Funktion, welche das Verhalten des Systems mit Identifikator id beschreibt, bezeichnen wir mit

$$Behaviour_{id} : (In_{id} \rightarrow M^{\overline{\infty}}) \xrightarrow{p} (Out_{id} \rightarrow M^{\overline{\infty}}).$$

Die Funktion $Behaviour_{id}$ legt für jede Belegung der Eingabeports der Komponente id mit gezeiteten Nachrichtenströmen fest, welche gezeiteten Nachrichtenströme über ihre Ausgabeports fließen.

3 Glass-Box Sichten

Wie bereits zu Beginn erwähnt, unterscheiden wir in Bezug auf den inneren Aufbau von Systemen zwischen

- Basiskomponenten und
- verteilten Systemen, die in eine Menge von Komponenten dekomponiert sind.

Auf der Ebene der Identifikatoren spiegelt sich dies dadurch wieder, daß wir die Menge ID in die disjunkten Mengen ID_b der Identifikatoren für Basiskomponenten und ID_v der Identifikatoren für verteilte Systeme unterteilen:

$$ID = ID_b \cup ID_v$$
$$ID_b \cap ID_v = \emptyset$$

Im folgenden behandeln wir die mathematische Modellierung von Basis- und verteilten Komponenten.

3.1 Basiskomponenten

Basiskomponenten sind Systeme, die nicht in weitere Komponenten zerlegt sind. Sie lassen sich durch stromverarbeitende Funktionen oder durch Zustandsmaschinen modellieren, welche ausgehend von einem gegebenen Startzustand durch eingehende Nachrichten in einen Folgezustand überführt werden und dabei auf ihren Ausgabeports Nachrichten aussenden. Mathematische Modelle für Basiskomponenten sind beispielsweise Zustands-Transitionssysteme [BDDW91] oder Zustandsautomaten mit Ein- und Ausgabe [LS89].

Eine Beschreibung von Basiskomponenten durch Zustandsmaschinen werden wir dann anwenden, wenn durch die betrachtete Beschreibungstechnik konkrete Annahmen über die Struktur des inneren Zustands von Systemen impliziert werden. Spricht eine Beschreibungstechnik hingegen rein über die Black-Box Aspekte von Systemen, so werden solche Zustandsmaschinen oft nicht explizit konstruieren, sondern lediglich deren Verhalten durch stromverarbeitende Funktionen beschreiben.

3.2 Verteilte Systeme

Eine andere Betrachtungsweise des inneren Aufbaus eines Systems ist dessen Dekomposition in eine Menge von *Komponenten*, welche ihrerseits wieder Systeme sind. Wir sprechen in diesem Fall von einem *verteilten System*. Die Menge der Identifikatoren der Komponenten eines verteilten Systems erhalten wir durch Anwendung der Funktion *Parts*:

$$Parts : ID_v \rightarrow \mathcal{P}(ID)$$

Durch die wiederholte Dekomposition von Systemen entsteht eine Hierarchie von Systemen. Die Funktion *Parts* beschreibt einen Baum, wobei ein Identifikator

$$RootSystem \in ID$$

die Wurzel dieses Baums darstellt. Durch diese Forderung erreichen wir, daß die übergeordneten Systeme jedes Systems eindeutig bestimmt sind, und somit jedes System eine eindeutige Stellung in der Hierarchie hat. Wir nutzen daher die Menge der Identifikatoren und die Funktion *Parts* zur Darstellung der hierarchischen Struktur von Systemen, während die Menge der Portnamen P zur Bestimmung von Kommunikationswegen dient.

Wir wollen nun auf die Beziehung des Verhaltens eines verteilten Systems zum Verhalten seiner Komponenten eingehen. Hierzu nehmen wir für diesen Abschnitt an, daß $id \in ID_v$ ein Identifikator eines verteilten Systems ist. Die Mengen $InParts_{id}$ und $OutParts_{id}$ sind die Mengen der Eingabe- und Ausgabeports aller Komponenten von id. Sie können wie folgt definiert werden:

$$InParts_{id} = \{p | \exists id' \in Parts(id) : p \in In_{id'}\}$$
$$OutParts_{id} = \{p | \exists id' \in Parts(id) : p \in Out_{id'}\}$$

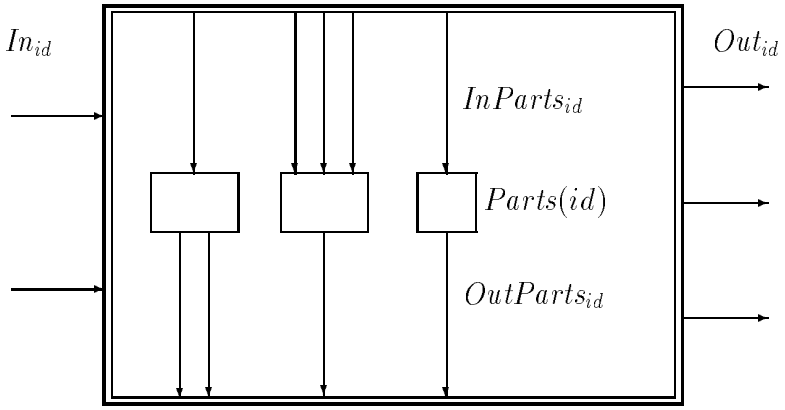

Abbildung 2: Ein verteiltes System

Eine graphische Darstellung eines verteilten Systems ist in Abbildung 2 angegeben. Ein verteiltes System besteht aus seinen Komponenten $Parts(id)$ und einem Kommunikationsmedium, das die im System vorhandenen Nachrichten an den richtigen Zielport des Empfängers weiterleitet. Dieses Kommunikationsmedium wirkt als „Membrane“ zwischen der Innenwelt der Komponente und ihrem Äußeren. Wir charaktisieren im folgenden den Nachrichtenfluß über diese Membran durch die Beziehung der Nachrichtenströme an den Ein- und Ausgabeports dieser Membran.

3.3 Das Kommunikationsmedium

Für die Beschreibung des Kommunikationsmediums definieren wir zunächst die Menge der *Nachrichtenquellen* $Origins_{id}$, welche aus den Eingabeports des Systems id und aus den Ausgabeports der Komponenten von id besteht, und die Menge der *Nachrichtenziele* $Destinations_{id}$, welche aus den Ausgabeports des Systems und aus den Eingabeports seiner Komponenten besteht:

$$Origins_{id} = In_{id} \cup OutParts_{id}$$
$$Destinations_{id} = Out_{id} \cup InParts_{id}$$

Wir nehmen an, daß jede Nachricht aus M selbst ihre Ziel- und Ursprungsadresse in Form eines Portnamen aus P beinhaltet. Wir lassen keine Nachrichtenadressierung nach dem „Broadcast-Prinzip“ zu, sondern jede Nachricht hat genau ein Ziel. Hierzu definieren wir zwei Funktionen

$$origin_{id} : M \rightarrow Origins_{id}$$
$$destination_{id} : M \rightarrow Destinations_{id},$$

die abhängig von Bezeichner des Systems id, durch dessen Membran die Nachricht gerade fließt, beschreiben, welcher Port Quelle und welcher Port Ziel der Nachricht ist. Wir stellen folgende fundamentale Anforderungen an den Nachrichtenfluß im Systemmodell:

- Je Eingabeport des Systems und je Ausgabeport der Komponenten muß die Reihenfolge der an ein Ziel abgesandten Nachrichten erhalten bleiben. Diese Anforderung entspricht der Auffassung, daß in Systemabläufen die Ereignisse auf jeder Verbindung linear geordnet ist.
- Der Inhalt von Nachrichten darf während der Übertragung nicht modifiziert werden. Nachrichten dürfen nicht dupliziert oder unterdrückt werden.

Systeme, in denen diese Anforderungen nicht gelten, können in unserem Systemmodell jedoch leicht nachgebildet werden. Hierzu sind die Nachrichten über Komponenten zu schicken, welche das Verhalten eines Übertragungskanals simulieren, welcher je nach seiner Spezifikation Nachrichten dupliziert, verschluckt oder in der Reihenfolge verändert. Wir fordern hingegen *nicht*, daß die Kommunikation über das Kommunikationsmedium verzögerungsfrei stattfinden muß. Insbesondere stellen wir keine Anforderungen an die Zeitdifferenz, die zwischen dem Abschicken einer Nachricht und seiner Ankunft höchstens verstreichen darf.

Wir werden nun ein Kommunikationsmedium spezifizieren, welches Nachrichten entsprechend den oben genannten Anforderungen verteilt. Hierzu beschreiben wir die Beziehung zwischen den Quell- und den Zielströmen von Nachrichten. Seien

$$ostreams : Origins_{id} \rightarrow M^{\overline{\infty}}$$
$$dstreams : Destinations_{id} \rightarrow M^{\overline{\infty}}$$

Belegungen der Nachrichtenquellen und Nachrichtenziele mit gezeiteten Nachrichtenströmen. Es muß gelten:

1. Die Quell- und Zielströme müssen für die Eingabe- und Ausgabeports von id die gleichen Belegungen mit Nachrichten aufweisen wie die Ein- und Ausgabeports des Systems id:

$$Behaviour_{id}(ostreams|_{In_{id}}) = dstreams|_{Out_{id}}$$

Dabei bezeichnet $f|_M$ die Restriktion einer Funktion $f : N \rightarrow L$ mit $M \subseteq N$ auf die Menge M. Genauer ausgedrückt gilt:

$$(f|_M)(x) = \begin{cases} f(x) & \text{wenn } x \in M \\ \text{undefiniert} & \text{sonst} \end{cases}$$

Durch die Restriktion $ostreams|_{In_{id}}$ wird das Teilbündel von *ostreams* betrachtet, welches die Nachrichtenströme enthält, die auf den Eingabeports des Systems ankommen. Umgekehrt werden durch $dstreams|_{Out_{id}}$ nur die Nachrichtenströme von *dstreams* ausgewählt welche für die Ausgabeports des Systems bestimmt sind.

2. Die Ein- und Ausgabeströme für die einzelnen Komponenten müssen entsprechend dem Verhalten der Komponente belegt sein:

$$\begin{aligned} id' \in Parts(id) \Rightarrow \quad & Behaviour_{id'}(dstreams|In_{id'}) \\ = \; & Behaviour_{id'}(ostreams|Out_{id'}) \end{aligned}$$

3. Jeder Zielstrom enthält alle für das entsprechende Nachrichtenziel bestimmten Nachrichten:

$$\begin{aligned} & Filter(\{m|origin_{id}(m) = s\} \cup \{\surd\}, dstreams.d) \\ = \; & Delay(Filter(\{m|destination_{id}(m) = d\} \cup \{\surd\}, ostreams.s)) \end{aligned}$$

Werden die Nachrichten vom Quellport s aus dem Zielstrom $dstreams.d$ gefiltert, so entsteht mit Ausnahme von dabei stattgefundener Verzögerung der gleiche Nachrichtenstrom, wie wenn aus dem Quellstrom $ostreams.s$ die Nachrichten für den Zielport d gefiltert werden. Damit wird gefordert, daß Nachrichten nicht verdoppelt werden, oder verloren gehen. Verzögerung modellieren wir mittels folgender gezeiteter stromverarbeitender Funktion:

$$\begin{aligned} & Delay : M^{\overline{\infty}} \xrightarrow{p} M^{\overline{\infty}} \\ & Filter(M, Delay(s)) = Filter(M, s) \end{aligned}$$

Aus der Definition gezeiteter Funktionen folgt, daß *Delay* tatsächlich eine verzögernde Funktion ist.

4 Diskussion

Zwei wichtige Anforderungen, welche im Hinblick auf die Behandlung objektorientierter Techniken an das Systemmodell gestellt wurden, waren die Möglichkeit einer Adressierung durch Identifikatoren und die Möglichkeit der dynamischen Erschaffung von Komponenten. Durch die Unterspezifikation des Nachrichtenvermittlungssystems wurde in Bezug auf die Adressierung ein sehr flexibles Grundkonzept geschaffen, welches eine große Anzahl möglicher Konkretisierungen offenhält.

Die dynamischen Erzeugbarkeit von Komponenten ist in unserem Modell implizit dadurch vorhanden, daß wir auch unendlich viele Komponenten zulassen. Wir können damit ein verteiltes System spezifizieren, welches die - meist unendliche - Gesamtheit aller erzeugbaren Komponenten beinhaltet. Eine Komponente wird dann dadurch erzeugt, daß ihr eine beliebige Nachricht zugesendet wird. Eine andere Möglichkeit ist, zum Erzeugen und Vernichten von Komponenten entsprechende Nachrichten auszuzeichnen. Dies könnte etwa durch Funktionen

$$\mathit{Create} : \mathit{ID} \rightarrow M$$
$$\mathit{Delete} : \mathit{ID} \rightarrow M$$

geschehen, welche zu einem Identifikator eine Nachricht liefern, welche das Erzeugen und das Vernichten der entsprechenden Komponente simuliert.

Darüber hinaus gibt es eine Reihe von weiteren offenen Punkten. Das Systemmodell steht in dem Zwiespalt zwischen der Notwendigkeit möglichst allgemein zu sein, um für verschiedenartige Anwendungen verwendbar zu sein, soll andererseits aber möglichst effektiv zur Semantikdefinition von Beschreibungstechniken genutzt werden können. Daher ist es sinnvoll eine zweite Modell-Schicht auf das Systemmodell aufzusetzen, das mehrere Adaptionen für verschiedene Anwendungen enthält. Interessante Anwendungen sind unter anderem Hardware, Kommunikationssysteme, technische Steuerungen und betriebliche Informationssysteme. In diesem Zusammenhang ist vor allem die Frage nach der Kommunikationsart, sowie die Frage nach Sichtbarkeitsbereichen für Identifikatoren von Interesse.

Ein weiterer Zwiespalt ergibt sich aus der Zielgruppe, für die das dieses Beschreibung des mathematischen Systemmodells geschrieben ist. Es soll einerseits einfach und leicht verständlich, andererseits aber formal definiert sein. Wir haben uns daher erlaubt, eine Vereinfachung durchzuführen, indem wir nicht, wie in FOCUS üblich, mit Spezifikation von Mengen stromverarbeitender Funktionen gearbeitet haben, sondern alle Eigenschaften unserer Komponenten auf Exemplarebene beschrieben haben. Wir haben somit nur ein Exemplar aus einer Menge von verteilten Systemen beschrieben, der unser eigentliches Interesse gilt. Eine Umsetzung aller exemplarischen Spezifikationen in Mengenspezifikationen ist jedoch schematisch und ist deshalb hier unterblieben. Dabei kann auch die durch rekursive Spezifikationen charakterisierte Komposition der Subsysteme in eine rekursive Mengenspezifikation übersetzt werden, deren Semantik durch den kleinsten Fixpunkt festgelegt werden kann.

Danksagung

Wir bedanken uns für anregende und fruchtbare Diskussionen und Hinweisen bei Radu Grosu, Christoph Hofmann, Stefan Merz, Barbara Paech, Ketil Stølen und insbesondere Rudi Hettler, sowie dem restlichen Team von SysLab.

Literatur

[AG90] C. Ashworth und M. Goodland, *SSADM: A Practical Approach*, McGraw-Hill, 1990.

[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", Technischer Bericht SFB 342/2/92 A, Technische Universität München, Institut für Informatik, 1993.

[BDDW91] M. Broy, F. Dederichs, C. Dendorfer und R. Weber, "Charakterizing the Behaviour of Reactive Systems by Trace Sets", Technischer Bericht SFB 342/2/91 A, TUM-I9102, Technische Universität München, Institut für Informatik, Februar 1991.

[BFG+93] M. Broy, C. Facchi, R. Grosu, R. Hettler, H. Hußmann, D. Nazareth, F. Regensburger, O. Slotosch und K. Stølen, "The Requirement and Design Specification Language Spectrum, An Informal Introduction, Version 1.0", Technischer Bericht TUM-I9311, Technische Universität München, Institut für Informatik, Mai 1993.

[Bro94] M. Broy, "SysLab - Forschungslabor für System- und Softwareentwicklung. Projektbeschreibung", Internes SysLab-Papier, 1994.

[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, Inc., Englewood Cliffs, New Jersey, 1994.

[CDI92] "Introduction to CDIF: The CASE Data Interchange Format Standards", April 1992, CDIF Technical Commitee, LBMS, Evelyn House, 62 Oxford Street, London W1N 9LF, United Kingdom.

[Hel90] G. Held, Hrsg., *Sprachbeschreibung GRAPES*, SNI AG, München Paderborn, 1990.

[HL93] H.J. Habermann und F. Leymann, *Repository - eine Einführung*, Handbuch der Informatik, Oldenbourg, 1993.

[Hus94] H. Hussmann, "Formal Foundations for SSADM: An Approach Integrating the Formal and Pragmatic Worlds of Requirements Engineering", Habilitationsarbeit, Technische Universität München, Juli 1994.

[LS89] N. Lynch und E. Stark, "A Proof of the Kahn Principle for Input/Output Automata", *Information and Computation*, 82:81–92, 1989.

[RBP+92] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy und W. Lorensen, *Object-Oriented Modeling and Design*, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1992.

[SFD92] L.T. Semmens, R.B. France und T.W.G. Docker, "Integrated Structued Analysis and Formal Specification Techniques", *The Computer Journal*, 35(6):600–610, 1992.

[Stø95] K. Stølen, "A Framework for the Specification and Development of Reactive Systems", Draft, 1995.

[Tho89] I. Thomas, "PCTE Interfaces: Supporting Tools in Software-Engineering Environments", *IEEE Software*, Seiten 15–23, November 1989.