Institut für Informatik

der Technischen Universität München

Eine Methodik für die formale Anforderungsspezifikation verteilter Systeme

Rainer Weber

Vollständiger Abdruck der von der Fakultät für Mathematik und Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation

Vorsitzender: ……

Prüfer der Dissertation:

1.

……

2.

……

3.

……

Die Dissertation wurde am ..... bei der Technischen Universität München eingereicht und durch die Fakultät für Mathematik und Informatik am .....

angenommen.

Zusammenfassung

In dieser Arbeit wird ein Vorgehensmodell für die formale Anforderungsspezifikation verteilter Systeme vorgestellt, das auf dem *Spurformalismus* basiert. Anders als in vielen Arbeiten auf dem Gebiet der formalen Methoden liegt der Schwerpunkt nicht auf der Beschreibung eines Formalismus, sondern auf der *Methodik*, d.h. dessen zielgerichteten Einsatz.

Ausgangspunkt der Arbeit ist ein methodischer Rahmen für eine durchgängige Systementwicklung, der für jede Entwicklungsphase formale Beschreibungstechniken sowie rudimentäre methodische Leitlinien bereitstellt. Soweit es die Anforderungsspezifikation betrifft baue ich diese Ansätze zu einer umfassenden Methodik aus. Besonders berücksichtigt wird hierbei der Übergang zur nächsten Phase, der Entwurfsspezifikation, in der nicht mehr Spuren, sondern stromverarbeitende Funktionen als Spezifikationsmittel eingesetzt werden.

Mein Vorgehensmodell gliedert die Anforderungsspezifikation in zwei Teilphasen: die *globale* und die *komponentenorientierte Spezifikation* eines verteilten Systems. In der globalen Spezifikation werden Anforderungen an das gesamte *System* gerichtet, worunter ich sowohl das zu erstellende Hardware/Software-Produkt als auch die Umgebung verstehe, in die es eingebettet ist. In der komponentenorientierten Spezifikation werden die Anforderungen an das Produkt von denen an die Umgebung getrennt. Dadurch ist die *Schnittstelle* zwischen Produkt und Umgebung festgelegt. Insofern läßt sich eine komponentenorientierte Spezifikation als ein Kontrakt zwischen einem Kunden und einem Systementwickler verstehen. Technisch lassen sich in beiden Teilphasen die ablauforientierte Spezifikation durch Spurformeln und die transitionsorientierte Spezifikation durch Transitionssysteme einsetzen und kombinieren.

Aus diesem Vorgehensmodell resultieren mehrere Fragen, die in meiner Arbeit detailliert behandelt werden: Auf welche Weise lassen sich Anforderungsspezifikationen strukturieren? Wie lassen sich unterschiedliche Beschreibungsformen eines verteilten Systems (globale Spezifikation, komponentenorientierte Spezifikation, Entwurfsspezifikation) zueinander in Beziehung setzen? Wie lassen sich "sanfte" Übergänge zwischen (Teil-)Phasen des Entwicklungsprozesses erreichen? Wie läßt sich der ablauforientierte mit dem transitionsorientierten Spezifikationsstil kombinieren? Wie lassen sich globale Anforderungen in lokale aufspalten?

Ausgehend von den methodischen Erfordernissen erweitere ich den Spurformalismus an einigen Stellen: um Sprachmittel für die komponentenorientierte Spezifikation sowie zur Erfassung von Echtzeiteigenschaften und um eine komfortable Notation für Transitionssysteme.
.

Ich danke Herrn Prof. Dr. Manfred Broy für seine von Anfang an gewährte Unterstützung und Herrn Prof. Dr. Wolfgang Reisig für zahlreiche Anmerkungen zu einer Vorversion meiner Arbeit. Mein Dank gilt auch Frank Dederichs und Claus Dendorfer, die Teile von vorläufigen Fassungen lasen.

Inhaltsverzeichnis

1. Einleitung

1.1 Motivation

Wie können wir auf methodische Weise korrekte *verteilte Systeme*[1] erstellen? Dies ist eine Kernfrage der heutigen Informatikforschung. Die wirtschaftliche Bedeutung verteilter Systeme ist groß: Immer leistungsfähigere Parallelrechner und Rechnernetze kommen auf den Markt, Anwender verlangen nach immer höherer Rechenleistung und Zuverlässigkeit, viele Anwendungen, etwa in der Prozeßautomatisierung und Bürokommunikation, sind inhärent nebenläufig. Die Programmentwicklung für verteilte Systeme ist jedoch ein noch wenig beherrschtes Problem.

Formale Methoden versprechen hier Abhilfe. *Formale Methoden* - darunter verstehen wir den zielgerichteten, systematischen Einsatz von theoretisch fundierten Formalismen zur Konstruktion korrekter Programme. Eine Sammlung formaler Methoden, die für eine durchgängige Systementwicklung aufeinander abgestimmt sind, nennen wir eine *formale Methodik*. Der Einsatz formaler Methoden ist von großer Bedeutung für eine ingenieurmäßige und zum Teil automatisierbare und daher durch Rechner unterstützbare Programmentwicklung. Wesentlich ist, daß sich die Korrektheit eines Programms (gegenüber einer formalen Spezifikation) nur mit formalen Methoden nachweisen läßt.

In den letzten Jahren entstanden mehrere formale Methodiken, die auf den Entwurf verteilter Systeme ausgerichtet sind. Wichtige Vertreter sind UNITY ([Chandy, Misra 88]), Lamports Transitionsaxiom-Methode ([Lamport 89]) und FOCUS ([Broy 89a], [Broy et al. 91b]). Meine Arbeit bezieht sich auf letzteren Ansatz. FOCUS gliedert den Entwicklungsprozeß in vier Phasen:

- Anforderungsspezifikation,
- Entwurfsspezifikation,
- abstrakte Implementierung und
- konkrete Implementierung.

Ein ähnlicher Rahmen findet sich in vielen Phasenmodellen der Softwaretechnik. In FOCUS verbinden sich mit den Namen der Phasen spezielle Formalismen und Techniken:

---

[1] Den Begriff "verteiltes System" verwende ich als Synonym bzw. Oberbegriff für "paralleles System", "kommunizierendes System" und "nebenläufiges System". Eine Präzisierung dieses Begriffs erfolgt in Kap. 2.

Die *Anforderungen* werden in einer *Spurspezifikation* (*trace specification*) erfaßt. Eine *Spur* (*trace)* modelliert einen Ablauf des zu erstellenden verteilten Systems. Technisch gesehen ist eine Spur eine endliche oder unendliche Sequenz (*Strom*) von Aktionen, die in diesem System auftreten können. Eine Spurspezifikation ist eigenschaftsorientiert: Sie beschreibt eine Reihe von Anforderungen an Abläufe des Systems, operationelle Vorstellungen über dessen Arbeitsweise müssen dabei nicht bestehen. Über die physische Verteilung und interne Struktur des Systems muß anfangs noch keine Aussage getroffen werden.

In einer *Entwurfsspezifikation* wird ein verteiltes System als ein Netz von Agenten beschrieben, die miteinander über asynchronen Nachrichtenaustausch kommunizieren. Die Agenten werden durch *stromverarbeitende Funktionen* modelliert; das sind Funktionen, die Ströme von Eingaben auf Ströme von Ausgaben abbilden. Auf diese Weise ergibt sich eine funktionale Sicht eines verteilten Systems. Die Spezifikation der Agenten muß in dieser Phase nicht ausführbar sein.

Ausführbarkeit wird erst von einer *abstrakten Implementierung* verlangt. Sie wird in einer funktionalen Programmiersprache angegeben, mit der sich stromverarbeitende Funktionen beschreiben lassen.

Konkrete Implementierungen werden dagegen in einer imperativen Programmiersprache für verteilte Systeme notiert. Hier wird besonderer Wert auf eine effiziente Realisierung gelegt.

Das Besondere an FOCUS ist, daß bekannte Formalismen derart kombiniert sind, daß eine durchgängige Systementwicklung von einer abstrakten, anwendungsorientierten Beschreibung bis zu einer konkreten, maschinenorientierten Realisierung möglich ist. Während der Systementwicklung müssen immer wieder verschiedene Systembeschreibungen auseinander abgeleitet und zueinander in Beziehung gesetzt werden. Dazu ist es nötig, daß die Formalismen gut aufeinander abgestimmt sind. Doch selbst dann ist die systematische Herleitung einer konkreteren Beschreibung aus einer abstrakteren schwierig und im allgemeinen nicht automatisierbar.

Die angesprochene Phaseneinteilung gibt nur einen groben Rahmen vor. Um die Methodik handhabbar zu machen, ist weitere Arbeit erforderlich: die Spezifikationstechnik *in* den einzelnen Phasen ist zu verfeinern und zu erweitern, die Übergänge *zwischen* den Phasen sind zu glätten.

Meine Arbeit verfolgt das Ziel, einen Beitrag zu diesen beiden Aspekten für die Phase der *Anforderungsspezifikation* und ihre *Schnittstelle zur Entwurfsspezifikation* zu leisten: Ich untergliedere die Anforderungsspezifikation in Teilphasen, stelle die Beziehung zwischen ihnen her, gebe für jede Teilphase eine methodische Vorgehensweise an und behandle den Übergang von einer Anforderungs- zu einer Entwurfsspezifikation.

1. Einleitung

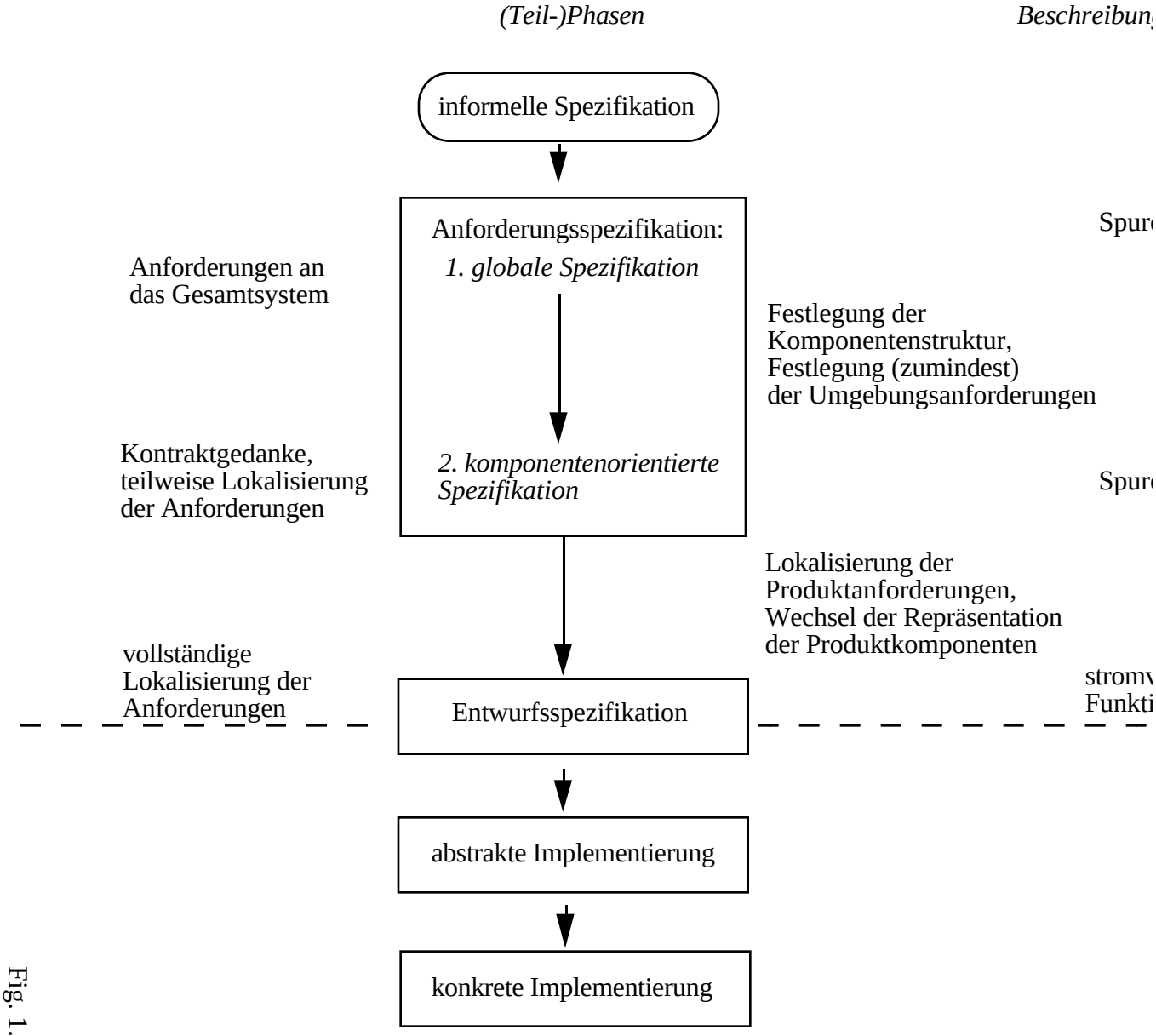

(Teil-)Phasen
Beschreibun
informelle Spezifikation
Anforderungsspezifikation:
1. globale Spezifikation
Spur
Anforderungen an
das Gesamtsystem
Festlegung der
Komponentenstruktur,
Festlegung (zumindest)
der Umgebungsanforderungen
Kontraktgedanke,
teilweise Lokalisierung
der Anforderungen
2. komponentenorientierte
Spezifikation
Spur
Lokalisierung der
Produktanforderungen,
Wechsel der Repräsentation
der Produktkomponenten
vollständige
Lokalisierung der
Anforderungen
stromv
Funkti
Entwurfsspezifikation
abstrakte Implementierung
konkrete Implementierung

Fig. 1.1

1.2 Die Methodik der Anforderungsspezifikation im Überblick

Ich gebe einen Überblick über das Vorgehensmodell für die Anforderungsspezifikation, das ich in dieser Arbeit vorschlage (vgl. Fig. 1.1). Auf technische Details verzichte ich an dieser Stelle, dies ist den folgenden Kapiteln vorbehalten.

Ich gehe von der weitverbreiteten Vorstellung aus, daß eine Anforderungsspezifikation einen *Kontrakt* zwischen einem *Kunden* und einem *Systementwickler* darstellt. Wenn ich (verkürzend) von *dem* Kunden und *dem* Systementwickler spreche, meine ich die entsprechenden Personen*gruppen*; insbesondere reicht die Gruppe der Systementwickler vom Systemanalytiker bis zum Programmierer.[1] Dieser Kontrakt regelt, welche Eigenschaften das zu erstellende System haben muß. In meiner Arbeit betrachte ich nicht beliebige Systemeigenschaften, ich fasse den Begriff der Anforderungsspezifikation etwas enger: Eine Anforderungsspezifikation sei eine formale Beschreibung der *funktionalen* Eigenschaften des zu erstellenden Systems. *Nicht-funktionale* Anforderungen wie die Verwendung einer bestimmten Implementierungssprache lasse ich außer Betracht. Zu den funktionalen Eigenschaften zähle ich auch Echtzeiteigenschaften, da sich diese in meinem Ansatz wie rein funktionale Eigenschaften spezifizieren lassen.

Der Begriff "verteiltes System" wurde bisher verwendet, ohne ihn zu präzisieren; für den Augenblick wollen wir uns darunter eine Menge von Komponenten vorstellen, die miteinander in Wechselwirkung stehen. Komponenten sind offene Teilsysteme des verteilten Systems. Sie werden in die Klassen der *Produkt-* und *Umgebungskomponenten* eingeteilt. Die Produktkomponenten werden im Laufe der Systementwicklung in Hardware oder Software realisiert, während die Umgebungskomponenten vom Kunden bereitzustellen sind. Ich verwende die Unterscheidung Produkt - Umgebung statt der geläufigeren Unterscheidung System - Umgebung, um die Rolle dieser Teilsysteme in einem Kontrakt herauszustellen und da ich den Begriff "System" für den gesamten Komplex Produkt *und* Umgebung gebrauche; dies ist die Sichtweise eines geschlossenen Systems. Im einfachsten Fall liegt genau eine Produkt- und eine Umgebungskomponente vor. Eine weitergehende Präzisierung des Begriffs "verteiltes System" findet sich in Kapitel 2.

Den Anfang der Systementwicklung bildet die Gewinnung der Anforderungen: informelle Anforderungen des Kunden werden gesammelt und in formale umgesetzt, die die Grundlage

---

[1] Ich gebrauche hier der Einfachheit halber nur die männliche Form; selbstverständlich können diese Personen auch weiblichen Geschlechts sein (Kundinnen, Systemanalytikerinnen, ...).

für die weitere Systementwicklung sind. Diese Umsetzung geschieht nicht in einem Schritt, sondern in einer Rückkopplungsschleife (vgl. z.B. [Dubois et al. 90]).

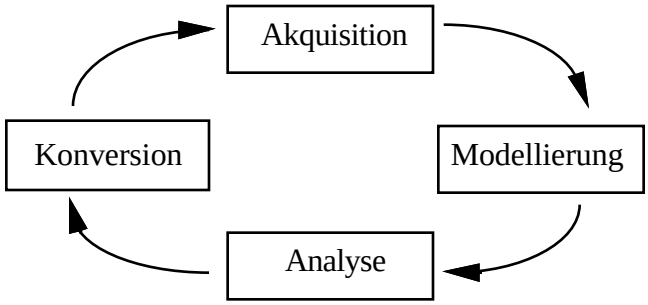


Fig. 1.2

Durch Interviews, Diskussionen zwischen Kunden und Systemanalytikern, Beobachtungen, etc., werden bei der *Akquisition* die informellen Anforderungen ermittelt. Bei der *Modellierung* werden die informellen Anforderungen in formale umgesetzt, wobei gewisse Heuristiken zur Anwendung kommen. Die *Analyse* ist eine Validation der formalen gegenüber den informellen Anforderungen. Die *Konversion* betrifft die Umwandlung (von Teilen) der formalen Beschreibung in eine für den Kunden leichter aufnehmbare Darstellung; hier läßt sich z.B. Graphik einsetzen.

Von diesen vier Tätigkeiten behandle ich lediglich die Modellierung und die Analyse. Für die Akquisition und Konversion, die für die Anwendbarkeit und Akzeptanz einer Methodik ebenfalls äußerst wichtig sind, vergleiche man z.B. [Dubois et al. 90] oder [Finkelstein et al. 91]; das dort geschilderte Vorgehensmodell läßt sich - obwohl auf andere Formalismen ausgerichtet - auch in unserer Methodik verwenden.

Bei der Modellierung wird zunächst festgelegt, welche *Aktionen* im verteilten System auftreten können. Aktionen werden als zeitlich atomar angenommen, sie haben die Zeitdauer Null; länger andauernde Vorgänge werden durch zwei Aktionen modelliert: die eine gibt den Anfang des Vorgangs an, die andere sein Ende (vgl. Kap. 2).

Nach der Festlegung der Aktionen werden Anforderungen an die möglichen *Abläufe* des Systems gerichtet. Ein Ablauf wird durch eine endliche oder unendliche Sequenz von Aktionen, eine *Spur*, modelliert. Die Formalisierung der Anforderungen geschieht durch eine Kombination der *ablauforientierten* und der *transitionsorientierten* Spezifikation. Bei der ablauforientierten Spezifikation werden durch Prädikate über Spuren direkt Anforderungen an Abläufe formuliert. Bei der transitionsorientierten Spezifikation besteht eine Indirektionsstufe: ein Transitionssystem wird angegeben, dem sich eine Spurmenge zuordnen läßt. Damit wird die Anforderung formuliert, daß erlaubte Spuren in dieser Spurmenge liegen müssen. Ich schlage eine Kombination dieser beiden Spezifikationsarten vor, da sich manche Anforderungen relativ leicht mit der einen, dagegen nur äußerst umständlich mit der anderen formulieren lassen.

Auf dieser Stufe des Entwicklungsprozesses besteht keine Trennung zwischen den Anforderungen an das zu erstellende Produkt und den Anforderungen an seine Umgebung. Wir nennen eine solche Spezifikation eine *globale Spezifikation*. Dieser Vorgehensweise liegt die weitverbreitete Erfahrung zugrunde, daß ein Kunde am Anfang einer Systementwicklung häufig Wünsche dahingehend formuliert, wie das "globale" Verhalten des gesamten Systems aussehen soll. Die Anforderungen werden noch nicht einzelnen Komponenten zugeordnet (vgl. auch [Dubois et al. 90], [Chandy, Misra 88]).

Die Trennung der Zuständigkeiten geschieht in einem zweiten Schritt, der *komponentenorientierten Spezifikation* (vgl. Fig. 1.3). In dieser Phase legt der Kunde zunächst fest, aus welchen Komponenten das verteilte System bestehen soll. Er muß die Komponentenstruktur nur soweit bestimmen, als es für die Anforderungsspezifikation wichtig ist. Dazu gehört zumindest die Festlegung einer oder mehrerer Umgebungskomponenten sowie einer Produktkomponente. In manchen Fällen ist es natürlich, mehr als eine Produktkomponente festzulegen, etwa bei einer Protokollspezifikation, in der zwei Protokollentitäten (Produktkomponenten) aufbauend auf einem "einfacheren" Kommunikationsdienst (Umgebungskomponente) einen "höheren" Kommunikationsdienst (globale Spezifikation) erbringen. Im Laufe der Produktentwicklung lassen sich die Produktkomponenten selbstverständlich weiter modular in Subkomponenten strukturieren; diese Strukturierung ergibt sich aus Entwurfsentscheidungen und bedarf nicht der Mitwirkung des Kunden. Die Aktionen, die in der ersten Teilphase uniform behandelt wurden, werden in Ein- und Ausgabeaktionen für die einzelnen Komponenten eingeteilt. Die Ein- und Ausgabeaktionen dienen der Kommunikation und Synchronisation der Komponenten. Zwischen zwei Komponenten besteht eine Verbindung, wenn eine Ausgabe der einen Komponente zugleich eine Eingabe der anderen ist. Die Kommunikation zwischen Komponenten geschieht asynchron.

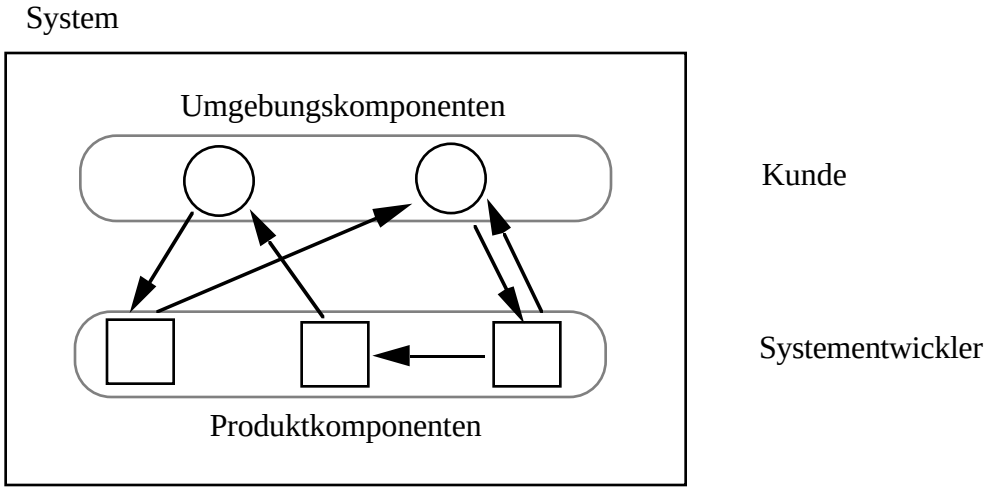


Fig. 1.3

Die Anforderungsspezifikation ist erst dann vollständig, wenn die Aufgabenverteilung zwischen dem Kunden und dem Systementwickler geregelt ist. Dies läßt sich auf verschiedene Weise erreichen: Zum einen kann der Kunde die Anforderungen der globalen Spezifikation explizit den Produkt- und Umgebungskomponenten zuordnen, zum anderen ist auch eine implizite Festlegung der Produktanforderungen möglich. Im letzteren Fall gibt der Kunde neben der globalen Spezifikation an, auf welches Verhalten der Umgebungskomponenten sich der Systementwickler verlassen kann. Daraus ergeben sich die Anforderungen an die Produktkomponenten in kanonischer Weise. Die explizite Festlegung des Umgebungsverhaltens durch den Kunden ist erforderlich, da sich die Umgebungsanforderungen i.a. nicht in eindeutiger Weise allein aus der globalen Spezifikation und der Komponentenstruktur des Systems ergeben. Auch Mischformen zwischen einer expliziten und einer impliziten Festlegung der Produktanforderungen sind möglich.

Der Übergang von einer globalen Spezifikation zu einer komponentenorientierten Spezifikation bedeutet, von der Sicht eines geschlossenen Systems zur Sicht mehrerer offener Teilsysteme zu wechseln.

Bei der Entwurfsspezifikation wird eine funktionale Agentenbeschreibung für die Produktkomponenten entwickelt. Hierzu bietet sich die schrittweise Verfeinerung einer komponentenorientierten Spezifikation an.

Dieser Überblick zeigt, daß meine Arbeit nicht *alle* methodischen Aspekte der Anforderungsspezifikation abdeckt, z.B. sind Managementaspekte nicht behandelt; der Schwerpunkt liegt auf der *Spezifikationsmethodik*.

Globale Spezifikationen werden in Kapitel 3 behandelt, komponentenorientierte Spezifikationen in Kapitel 4, der Übergang von einer Anforderungs- zu einer Entwurfsspezifikation in Kapitel 5.

1.3 Ergebnisse der Arbeit

Viele Arbeiten auf dem Gebiet der formalen Methoden konzentrieren sich auf die Vorstellung oder theoretische Untersuchung eines Formalismus; ich stütze mich hingegen auf einen im wesentlichen *bekannten* Formalismus. Der Schwerpunkt meiner Arbeit liegt auf der *Methodik*, ihr Hauptergebnis ist ein *methodisches Vorgehensmodell* für die formale Anforderungsspezifikation verteilter Systeme, das auf den Spurformalismus abgestimmt ist.

Die dabei verwendeten Ideen sind nicht vollständig neu, vielmehr handelt es sich in der Regel um eine Kombination und Erweiterung verschiedener bewährter Konzepte, etwa der Phasengliederung des Entwurfsprozesses, der schrittweisen Verfeinerung einer Spezifikation,

der Strukturierung von Anforderungen in Sicherheits- und Lebendigkeitsanteile. Wesentlich ist hierbei, daß diese in der Literatur an verschiedenen Stellen eingeführten Konzepte auf das formale Beschreibungsmittel der Spuren ausgerichtet und zu einem einheitlichen Vorgehensmodell verbunden werden; hiermit läßt sich eine durchgängige Entwicklung von der Formulierung beliebiger globaler Anforderungen an das Gesamtsystem über eine kontraktmäßige Trennung der Zuständigkeit des Systementwicklers von der des Kunden bis zu einer funktionalen Entwurfsspezifikation durchführen, was in ähnlichen Ansätzen nur eingeschränkt möglich ist (vgl. [Olderog 88] sowie Abschnitt 2.3, Punkt 3). Um diese Durchgängigkeit zu erreichen führe ich verschiedene grundsätzliche Untersuchungen durch, die durch die Methodik motiviert sind, etwa Untersuchungen zur Aufspaltung globaler Anforderungen in lokale in den Abschnitten 4.2 und 4.3.

Meine Arbeit basiert auf dem durch FOCUS vorgegebenen methodischen Rahmen. Ausgehend von dem Überblick über die methodische Vorgehensweise im vorigen Abschnitt stelle ich nun dar, welche neuen Ergebnisse ich in FOCUS einbringe. Ein weitergehender Vergleich meiner Arbeit mit ähnlichen Ansätzen findet sich in Abschnitt 2.3.

1. Methodik der Anforderungsspezifikation

Während bisher in FOCUS (vgl. z.B. [Broy 88a]) allein eine globale Spezifikation als Anforderungsspezifikation verwendet wurde, teile ich den Vorgang der Anforderungsspezifikation in zwei Teilphasen auf: die *globale Spezifikation* und die *komponentenorientierte Spezifikation*. Dies bietet zwei Vorteile: Zum einen läßt sich damit auf der Anforderungsebene festlegen, welches Verhalten von dem zu erstellenden Produkt verlangt wird und welches Verhalten die vom Kunden bereitzustellende Umgebung zeigen muß; gemäß des Kontraktgedankens ist diese Festlegung ein notwendiger Bestandteil einer Anforderungsspezifikation. Bisher wurde diese Trennung der Zuständigkeiten nicht berücksichtigt. Zum anderen erleichtert die Komponentensicht den Übergang von der Anforderungs- zur Entwurfsspezifikation: ein Agent in einer Entwurfsspezifikation ist ein Spezialfall einer Komponente, für die ausschließlich "lokale" Anforderungen bestehen und die durch stromverarbeitende Funktionen beschrieben ist; der Übergang geschieht demnach durch die "Lokalisierung" der Anforderungen und den Wechsel der Repräsentation einer Komponente (siehe Punkt 2).

Durch die Bestimmung von Komponenten wird das geschlossene Systeme in mehrere offene Teilsysteme strukturiert. Als weitere, dazu orthogonale Strukturierung wird die Einteilung der Anforderungen in Sicherheits- und Lebendigkeitsanforderungen eingesetzt. Wird eine *Sicherheitsanforderung* verletzt, so ist dies bereits nach einem endlichen Teilablauf feststellbar. Im Gegensatz dazu kann die Verletzung einer *Lebendigkeitsanforderung* nur bei Kenntnis des gesamten, unter Umständen unendlichen Ablaufs festgestellt werden. Die Strukturierung in

Sicherheits- und Lebendigkeitsanforderungen ist nicht neu (vgl. z.B. [Lamport 83a], [Li, Maibaum 88], [Jonsson 87]). Neu sind umfangreiche grundsätzliche Untersuchungen zu diesem Begriffspaar, die durch die Methodik motiviert sind. Für die neu eingeführte komponentenorientierte Spezifikation werden diese Begriffe erst von mir definiert.

Weiterhin stelle ich Sprachmittel vor, mit denen sich Anforderungen flexibel beschreiben lassen. Neben Spurformeln, die schon vielfach in Fallstudien verwendet wurden, motiviere ich den Gebrauch von Transitionssystemen zur Spezifikation spezifischer Eigenschaften und führe eine komfortable Notation ein. Für die komponentenorientierte Spezifikation erweitere ich die Sprachmittel einer globalen Spezifikation um Strukturinformation.

2. Übergange zwischen verschiedenen Systembeschreibungen

Wie wir in Abschnitt 1.1 gesehen haben, fallen in FOCUS naturgemäß Übergänge von abstrakteren zu konkreteren Systembeschreibungen an. Ich behandle zwei solche Übergänge: den Übergang von einer globalen zu einer komponentenorientierten Spezifikation und den Übergang von einer Komponenten- zu einer Agentenspezifikation.

Ersterer bildet einen Schwerpunkt meiner Arbeit; er wurde bisher im Rahmen von Spurspezifikationen nicht untersucht. Ich zeige, daß sich die (getrennten) Anforderungen an die Produkt- und an die Umgebungskomponenten im allgemeinen nicht aus einer globalen Spezifikation ergeben. Daher ist es im allgemeinen nötig, die Anforderungen an die Umgebungskomponenten explizit festzulegen, womit aber zugleich die Anforderungen an das Produkt bestimmt sind. Die oben angesprochene Aufspaltung der globalen Anforderungen in Produkt- und Umgebungsanforderungen untersuche ich mit einem Begriffsapparat, in dem das Konzept der "Realisierbarkeit einer Spurmenge durch Strategien" zentral ist.

In einer komponentenorientierten Spezifikation auf der Anforderungsebene liegen bereits lokale Anforderungen an die Umgebungskomponenten vor, aber möglicherweise noch globale Anforderungen an die Produktkomponenten: zwar sind die Anforderungen an das Teilsystem "Produktkomponenten" als ganzes bestimmt, nicht jedoch das Zusammenspiel jeder einzelnen Produktkomponente mit dem Rest des Systems. Die Lokalisierung der Anforderungen geschieht durch Entwurfsschritte im Rahmen der schrittweisen Verfeinerung der komponentenorientierten Spezifikation. Der Übergang von einer lokalen Komponentenbeschreibung zu einer (impliziten) Agentenbeschreibung ist relativ einfach. Ich verwende und erweitere für diesen Übergang Ergebnisse aus [Jonsson 87], worin eine Spursemantik für I/O-Automaten angegeben wird.

3. Erweiterungsmöglichkeiten für Echtzeitsysteme

Bisher wurden Echtzeitsysteme im Rahmen von Spurspezifikationen wenig untersucht; ein Ansatz ist in [Reed, Roscoe 86] zu finden. In meiner Arbeit stelle ich verschiedene Möglichkeiten vor, Echtzeiteigenschaften durch Spuren zu beschreiben, und vergleiche sie. Ich behandle sowohl die globale Spezifikation als auch die komponentenorientierte Spezifikation und Agentenspezifikation von Echtzeiteigenschaften. Das Ziel ist hier, das für zeitfreie Anforderungen vorgestellte Vorgehensmodell auch für zeitkritische nutzen zu können. Bei der Annahme eines globalen Zeitbegriffs wird dieses Ziel auch erreicht; weitere Untersuchungen sind erforderlich, wenn man diese Annahme nicht treffen kann (vgl. Kap. 6).

Die Untersuchungen meiner Arbeit sind auf eine spezielle Entwurfsmethodik ausgerichtet. Ich glaube jedoch, daß viele Ergebnisse von darüber hinausgehendem Interesse sind, etwa für zustandsorientierte Formalismen. Ich nenne hier den Übergang von einer globalen Spezifikation zu einer komponentenorientierten Spezifikation (vgl. [Dubois et al. 90] und [Abadi, Lamport 90]), die Strukturierung der Anforderungen in Sicherheits- und Lebendigkeitsanforderungen ([Lamport 89], [Li, Maibaum 88]), das Konzept der Realisierbarkeit ([Abadi et al. 89], [Abadi, Lamport 90]), den Übergang von einer Spur- zu einer Agentenbeschreibung (vgl. [Olderog 88]). Verschiedene Techniken aus anderen Ansätzen ergänzen meine Methodik bzw. setzen sie fort: Zum Beispiel läßt sich das Einbringen von Entwurfsschritten nach [Chandy, Misra 88] und [Li, Maibaum 88] mit Spurformeln nachvollziehen. Insgesamt erwarte ich in der Zukunft eine stärkere wechselseitige Beeinflussung der verschiedenen methodischen Ansätze, die ähnliche Fragen in unterschiedlichen Formalismen untersuchen.

Zur Orientierung des Lesers findet sich am Ende des folgenden Abschnitts eine Klassifizierung der Kapitel und Abschnitte in Bekanntes und Neues.

1.4 Aufbau der Arbeit

Kapitel 2 enthält einen Überblick über den Spurformalismus. Ich stelle zunächst die Sichtweise und die Annahmen vor, die der Modellierung eines verteilten Systems durch Spuren zugrundeliegen. Anschließend begründe ich, warum sich der Spurformalismus zur Anforderungsspezifikation verteilter Systeme eignet. Ein Vergleich mit meiner Arbeit ähnlichen Ansätzen beschließt das Kapitel.

In den Kapiteln 3 und 4 wird die Methodik der Anforderungsspezifikation ausführlich dargestellt.

Kapitel 3 befaßt sich mit der globalen Spezifikation. Sprachmittel zur Formulierung von Anforderungen und Hinweise für ihre methodische Verwendung werden angegeben: Die sog.

Spurformeln unterstützen einen ablauforientierten Spezifikationsstil, Transitionssysteme einen transitionsorientierten; beide Stile lassen sich flexibel kombinieren. Auch Echtzeitanforderungen, für deren Beschreibung ich eine Erweiterung des Spurformalismus vorstelle, und Nicht-Echtzeitanforderungen lassen sich kombinieren. Zentrale Punkte sind eine tabellenorientierte Notation für Transitionssysteme sowie umfangreiche Untersuchungen zu Sicherheits- und Lebendigkeitseigenschaften. Ich verwende dieses Begriffspaar zur anforderungsorientierten Strukturierung von Spezifikationen.

Kapitel 4 behandelt die komponentenorientierte Spezifikation, insbesondere den Übergang von einer globalen Spezifikation zu einer komponentenorientierten Spezifikation. Ich untersuche, ob sich die Spezifikation von Komponenten schematisch aus einer globalen Spezifikation ableiten läßt. Hierbei ist das Konzept der Realisierbarkeit zentral. Es ist Gegenstand grundsätzlicher Untersuchungen, die mit einem an die Spieltheorie angelehnten Begriffsapparat ("Strategien") durchgeführt werden. Schließlich gehe ich noch auf die Echtzeitspezifikation von Komponenten ein.

In Kapitel 5 wird der Übergang von einer Anforderungs- zu einer Entwurfsspezifikation erläutert. Dieser umfaßt zwei Dimensionen: die "Lokalisierung" der Anforderungen an die Produktkomponenten und die Beschreibung der Produktkomponenten durch stromverarbeitende Funktionen. Für die erste Dimension bietet sich die Technik der schrittweisen Verfeinerung einer Spezifikation an, für die zweite ist eine Verbindung zwischen einer Spur- und einer funktionalen Beschreibung nötig. Sie erfolgt über eine Spursemantik für stromverarbeitende Funktionen. Ich führe dazu neben stromverarbeitenden Agenten die ebenfalls aus der Literatur bekannten I/O-Automaten ein, um Ergebnisse zur Spursemantik von I/O-Automaten auf stromverarbeitende Agenten zu übertragen. Das Konzept der Realisierbarkeit durch Strategien aus Kapitel 4 wird zur Realisierbarkeit durch stromverarbeitende Agenten und I/O-Automaten in Beziehung gesetzt. Zum Schluß des Kapitels wird exemplarisch die Spezifikation von zeitbehafteten Agenten in Form spezieller I/O-Automaten vorgeführt.

In Kapitel 6 gebe ich Anregungen zu einer weiteren Verwendung der Ergebnisse sowie zu weitergehenden Untersuchungen.

Die methodische Vorgehensweise verdeutliche ich durchgehend anhand eines einfachen Kommunikationssystems ("Postfachsystem"). Dieses Beispiel lehnt sich an die formale Beschreibung des Postfachsystems eines existierenden Betriebssystems an ([Bemmerl et al. 90], [Weber 90b], [Dendorfer 91]). Es ist jedoch stark vereinfacht dargestellt, um den Umfang gering zu halten und trotzdem möglichst viele Gesichtspunkte der Methodik anzusprechen. Daneben verwende ich zahlreiche sehr einfache Beispiele, die Aussagen im Bereich der methodischen Grundlagen verdeutlichen, begründen oder widerlegen.

Die folgende Klassifizierung der Kapitel und Abschnitte in Bekanntes und Neues soll der Orientierung des Lesers dienen. Die Darstellung von Bekanntem wird knapp gehalten; für eine ausführliche Behandlung siehe die angegebenen Literaturhinweise.

Die Kapitel 1, 2 und 6 bringen keine technisch neuen Ergebnisse. Trotzdem liefert gerade das zweite Kapitel einen wichtigen Beitrag zu meiner Arbeit, es begründet die Adäquatheit meines Ansatzes.

In Kapitel 3 enthalten die Abschnitte 3.1, 3.2 und 3.3 Bekanntes (vgl. hierzu [Broy 89a] sowie die Fallstudien zu FOCUS, etwa [Broy 88a]). Neu ist die Integration des transitionsorientierten Spezifikationsstils in FOCUS (Abschnitt 3.4, insbesondere Abschnitt 4.3.1), die Untersuchungen zu Sicherheits- und Lebendigkeitseigenschaften (Abschnitt 3.5; ich stütze mich allerdings auf exisitierende formale Definitionen dieser Begriffe) sowie die spurbasierte Echtzeitspezifikation (Abschnitt 3.6).

Fast ausschließlich neue Ergebnisse sind in Kapitel 4 zu finden; lediglich der Begriff der Strategie in Abschnitt 4.3 ist bekannt ([Broy et al. 91a]), nicht jedoch die Untersuchungen, die damit durchgeführt werden.

In Kapitel 5 ist das Konzept der stromverarbeitenden Agenten (Abschnitt 5.1) bekannt. Das Ergebnis des Abschnitts 5.2 ist die Adaption von Ergebnissen aus [Jonsson 87] für unsere Entwurfsmethodik, wozu verschiedene Anpassungsschritte nötig sind. Neues präsentieren die Abschnitte 5.3, 5.4 und 5.5. In Abschnitt 5.3 geht Erfahrung aus (anderen) Ansätzen der schrittweisen Verfeinerung von Spezifikationen ein, in Abschnitt 5.4 ein Ergebnis aus [Broy et al. 91].

Die eingeführten Definitionen und Sätze sind - soweit nicht anders angegeben - neu.

2. Spezifikation durch Spuren - Überblick und Vergleich mit ähnlichen Ansätzen

Das vorige Kapitel gab bereits einen Eindruck von der methodischen Vorgehensweise bei der Spurspezifikation; der eigentliche *Formalismus* wurde dagegen nur kurz angesprochen. Auf diesen wollen wir uns nun stärker konzentrieren: Ich erläutere, welche Modellierungssicht ihm zugrundeliegt (Abschnitt 2.1), begründe seine Eignung zur Anforderungsspezifikation verteilter Systeme (Abschnitt 2.2) und stelle einen Vergleich mit ähnlichen Formalismen und Methoden an (Abschnitt 2.3).

Mit diesem Kapitel soll erreicht werden, daß der Leser die Spurspezifikation in das umfangreiche und vielgestaltige Feld der Formalismen und Methoden zur Beschreibung und zum Entwurf verteilter Systeme einordnen kann.

2.1 Die Modellierungssicht der Spurspezifikation

Eine Spurspezifikation gibt ein mathematisches *Modell* der erlaubten Abläufe eines verteilten *Systems* wieder. Bei einer Spurspezifikation wird in der Regel ein Modell allerdings nicht explizit angegeben, sondern es wird vielmehr implizit über seine Eigenschaften ("eigenschaftsorientiert", "axiomatisch") charakterisiert. Die Begriffe "Modell" und "System" gebrauche ich wie in der Systemtheorie üblich:

"Der Begriff *System* umschreibt eine Realität mit allen für den *Untersuchungszweck* relevanten Wirkbeziehungen zwischen ihren Bestandteilen [...]. Das mathematische *Modell* soll, soweit es der *Untersuchungszweck* erfordert, diese Realität mit Hilfe mathematischer Zeichen und unter Beachtung mathematischer Regeln abbilden.“ ([Meyer 83], S. 2 m.)

Unser "Untersuchungszweck" ist die Hardware/Software-Entwicklung. Mit dem Spurformalismus lassen sich nicht alle Aspekte eines verteilten Systems beschreiben. Wie wir jedoch feststellen werden ist er ausdrucksstark genug, um als Basis für die Hardware/Software-Entwicklung zu dienen. Unter einem *verteilten* System verstehen wir eine Menge von *Komponenten*, die räumlich oder logisch voneinander getrennt sind, nebenläufig arbeiten und miteinander durch Kommunikation in Wechselwirkung treten. Einige der Komponenten sind im Laufe der Systementwicklung zu entwerfen; sie werden *Produktkomponenten* genannt. Die restlichen Komponenten heißen *Umgebungskomponenten*; sie sind vom Kunden bereitzustellen (vgl. Abschnitt 1.2). Tatsächlich spielt die Komponentenstruktur des verteilten Systems erst in der zweiten Teilphase der Anforderungsspezifikation, der *komponentenorientierten Spezifikation*, eine Rolle, nicht dagegen in der ersten Teilphase, der *globalen Spezifikation*.

Nach diesen Vorbemerkungen zur Modellierung verteilter Systeme erläutere ich nun die Modellierungssicht der Spurspezifikation. Jeder Modellierungsannahme stelle ich eine andere, meist gegensätzliche, aber ebenfalls weitverbreitete Modellierungsannahme gegenüber, um die Modellierungssicht der Spurspezifikation von der anderer Formalismen abzugrenzen. Zunächst beziehe ich mich allein auf die globale Spezifikation.

1. Aktions- vs. Zustandssicht

Der Spurformalismus ist *aktionsorientiert*: das Verhalten eines Systems wird durch das Auftreten von Aktionen im Zeitablauf wiedergegeben. Ein Beispiel für eine Aktion ist das Senden einer Nachricht an ein bestimmtes Postfach in einem Postfachsystem. Eine Aktion kann in einem Ablauf mehrfach auftreten.

Weit verbreitet sind auch *zustandsorientierte* Formalismen. Ihnen liegt die Vorstellung zugrunde, daß ein verteiltes System aus einer Menge von *Objekten* besteht, die sich zu einem gegebenen Zeitpunkt in gewissen *Zuständen* befinden, d.h. gewisse Werte annehmen (z.B. "Postfach p enthält Nachricht n"). Der Gesamtzustand des Systems ergibt sich aus den Zuständen aller Objekte im System. Gemäß dieser Sichtweise läßt sich ein verteiltes System z.B. durch die Sequenzen von Gesamtzuständen beschreiben, die es im Laufe der Zeit einnimmt.

Obwohl Aktionen das Grundkonzept des Spurformalismus sind, können auch Zustände als Hilfskonzept eingeführt werden (siehe Abschnitt 3.4); sie bilden die Basis für einen transitionsorientierten Spezifikationsstil.

Häufig wird die Dualität oder gar Äquivalenz der Begriffe "Aktion" und "Zustand" betont (vgl. z.B. [Abadi, Lamport 90]); diese Dualität zeigt sich auch an einigen Stellen meiner Untersuchungen (vgl. hierzu die Anmerkungen in Abschnitt 2.3, Punkt 9).

2. Atomare vs. nicht-atomare Aktionen

Aktionen können eine zeitliche Ausdehnung haben oder als zeitlich *atomar* betrachtet werden. Im Spurformalismus wird letzteres angenommen und ein Vorgang mit einer zeitlichen Ausdehnung wird durch zwei Aktionen modelliert; die erste gibt seinen Anfang an, die zweite sein Ende. Zeitlich nicht-atomare Aktionen können im Laufe der Systementwicklung in "primitivere" Aktionen verfeinert werden (vgl. [Vogler 91]), bei zeitlich atomaren Aktionen erscheint mir das nicht sinnvoll. Die Annahme der zeitlichen Atomarität rechtfertigt die Interleaving-Sicht, die dem Spurformalismus zugrundeliegt:

3. Interleaving vs. explizite Nebenläufigkeit

Im Spurformalismus wird, bedingt durch die Vorstellung zeitlich atomarer Aktionen, Nebenläufigkeit nicht explizit repräsentiert; wir sprechen von der *Interleaving-Sicht*, die in [Hoare 85], S. 41, folgendermaßen charakterisiert ist:

"Imagine there is an observer with a notebook who watches the process and writes down the name of each event as it occurs. We can validly ignore the possibility that two events occur simultaneously; for if they did, the observer would still have to record one of them first and then the other, and the order in which he records them would not matter."

Es sei darauf hingewiesen, daß auch vielen zustandsorientierten Formalismen die Interleaving-Sicht zugrundeliegt, etwa der *linear time temporal logic* (vgl. z.B. [Lamport 80]). Wenn auch durch Interleaving Nebenläufigkeit nicht explizit repräsentiert wird, anders also als z.B. bei der Modellierung durch Petrinetze ([Reisig 82]), so lassen sich damit doch viele wichtige Eigenschaften verteilter Systeme hinreichend gut beschreiben.

4. Explizite Modellierung vs. implizite Modellierung von unendlichem Verhalten

Unbeschränktes bzw. unendliches Verhalten ist ein typisches Phänomen verteilter Systeme. Weit verbreitet, insbesondere bei Prozeßalgebren (vgl. Abschnitt 2.3, Punkt 3), ist die Modellierung des Systemverhaltens durch alle endlichen Approximationen von Abläufen. Wie Abschnitt 3.5 zeigen wird, werden dadurch nur Sicherheitseigenschaften eines Systems modelliert. Im Spurformalismus sind neben endlichen auch unendliche Spuren nötig, um beliebige Lebendigkeitseigenschaften verteilter Systeme ausdrücken zu können.

5. Axiomatische vs. modellorientierte Beschreibung

Wie bereits am Anfang dieses Abschnitts erwähnt verwende ich bei der Spurspezifikation eine (weitgehend) *axiomatische, eigenschaftsorientierte* Beschreibung. Bei einer *modellorientierten Beschreibung* gibt man dagegen explizit ein Modell für das System an. Häufig haben modellorientierte Beschreibungen eine operationelle Semantik, d.h. sie sind ausführbar. Eine axiomatische Beschreibung kann widersprüchlich sein; dies ist bei einer modellorientierten Beschreibung wegen der Existenz eines Modells nicht möglich.

In manchen Anwendungen ist es nützlich, statt eine rein axiomatische Beschreibung durch logische Formeln (Spurformeln, vgl. Abschnitt 3.3) zu erstellen, einen Teil der Beschreibung durch ein Transitionssystem wiederzugeben (vgl. Abschnitt 3.4); indirekt werden auch damit Spuren beschrieben. Diese Kombination eines axiomatischen und eines modellorientierten Spezifikationsstils verursacht keine semantischen Probleme. Zwar hat eine Spezifikation eines Transitionssystems in meinem Ansatz immer ein Modell (außer wenn die algebraische Spezifikation der zugrundeliegenden Datentypen widersprüchlich ist), die Übergangsrelation kann jedoch auf axiomatische Weise spezifiziert sein, daher ist die Beschreibung eines Transitionssystems in meinem Ansatz nicht unbedingt ausführbar.

Bei den bisherigen Modellierungsannahmen hatten wir die globale Spezifikation im Auge. Da auch eine Komponente auf der Anforderungsebene durch eine Spurmenge beschrieben wird, gelten die Modellierungsannahmen 1 - 5 ebenfalls für die komponentenorientierte Spezifikation. Anders als bei einer globalen Spezifikation wird bei einer komponentenorientierten Spezifikation zwischen Ein- und Ausgabeaktionen unterschieden. Die Art der Kommunikation zwischen Komponenten wird in der letzten Modellierungsannahme festgelegt:

6. Asynchrone vs. synchrone Kommunikation

Abgestimmt auf unsere Agentenmodellierung bei der Entwurfsspezifikation verwenden wir die asynchrone Kommunikation. Sie bietet zudem den Vorteil, daß aus der Zusammenschaltung der einzelnen Komponenten eine Spurmenge resultiert, die sich auf einfache Weise aus den Spurmengen der Komponenten ergibt ([Jonsson 87]).

Die obige Gegenüberstellung von Modellierungsannahmen soll nicht als Koordinatensystem verstanden werden, in das sich jeder Formalismus zur Beschreibung verteilter Systeme einordnen läßt. Außerdem soll sie nicht nahelegen, daß die getroffenen Annahmen die einzige "vernünftige" Wahl sind (vgl. Abschnitt 2.3). Ich will jedoch motivieren, welche Vorteile gerade diese Kombination von Modellierungsannahmen bringt:

- Spuren lassen sich sowohl zur globalen Spezifikation benutzen als auch zur modularen Spezifikation von Komponenten.
- Mit Spuren lassen sich asynchron kommunizierende Komponenten hinreichend genau

beschreiben; dies ermöglicht in einfacher Weise die Anknüpfung an eine Agentenspezifikation. Bei synchron kommunizierenden Agenten ist dies nicht in gleicher Weise möglich (vgl. [Olderog 88] und Abschnitt 2.3, Punkt 3).
- Sicherheits- und Lebendigkeitseigenschaften verteilter Systeme lassen sich flexibel ausdrücken.
- Der Formalismus ist einfach handhabbar (übersichtliche Spezifikation, relativ einfache Beweisführung).

2.2 Spurspezifikationen als Anforderungsspezifikationen

In diesem Abschnitt begründe ich, warum sich der Spurformalismus zur Anforderungsspezifikation eignet. Sehen wir uns dazu eine Definition des Begriffs *Anforderungsspezifikation* (*requirements specification*) an:

"A specification that sets forth the requirements for a system or system component; for example, a software configuration item. Typically included are functional requirements, interface requirements, design requirements, and development standards." [IEEE 83]

Wir erkennen, daß über zwei Arten von Anforderungen gesprochen wird: *funktionale Anforderungen* und *nichtfunktionale Anforderungen*. Funktionale Anforderungen an ein gesamtes System oder eine Systemkomponente lassen sich durch Spuren spezifizieren. Nichtfunktionale Anforderungen wie Entwicklungsstandards, z.B. die Verwendung einer bestimmten Implementierungssprache, werden bei der Spurspezifikation nicht berücksichtigt (vgl. [Davis 90] für eine Abgrenzung der funktionalen Anforderungen *(behavioural requirements)* von den nichtfunktionalen *(non-behavioural requirements)*).

Die IEEE-Definition läßt viele Freiheitsgrade, insbesondere bzgl. des Grades der Formalisierung der funktionalen Anforderungen. Ich will daher auf Eigenschaften des Spurformalismus hinweisen, die mir für die Spezifikation funktionaler Anforderungen nützlich erscheinen:

1. Präzision: FOCUS sieht die Anforderungsspezifikation als einen Kontrakt zwischen einem Kunden und einem Systementwickler (vgl. Abschnitt 1.2). Zu diesem Zweck ist eine präzise Beschreibung der Anforderungen nötig. Mit dem Spurformalismus wird dies erreicht. Ob die so erlangte Präzision durch einen zu hohen Aufwand für die Formalisierung erkauft wird und - allgemeiner - der Nutzen formaler Methoden in der Industrie wird kontrovers diskutiert. Ich will die industrielle Einsetzbarkeit von formalen Methoden hier nicht weiter ansprechen, verweise aber auf [Denert 91], [Parnas et al. 90] und [Hall 90], die unterschiedliche Meinungen zu diesem Thema vertreten.

2. Anwendungsorientierung: Eine Anforderungsspezifikation soll anwendungs- und nicht maschinenorientiert sein. Im Spurformalismus lassen sich die einzuführenden Konzepte (Aktionen und Zustände, wenn Transitionssysteme verwendet werden) nach der Sicht des Kunden wählen.

3. Globale und lokale Anforderungen: Sowohl globale Anforderungen, d.h. Anforderungen an das System als ganzes, als auch lokale Anforderungen, d.h. Anforderungen an eine bestimmte Komponente des Systems, lassen sich im Spurformalismus spezifizieren.

4. Flexible Ausdrucksweise: Durch Spurformeln lassen sich Eigenschaften eines verteilten Systems beschreiben, ohne ein Modell explizit anzugeben. Insbesondere muß eine Anforderungsspezifikation durch Spuren nicht ausführbar sein. Wenn wir es wünschen, können wir jedoch auch einige Systemeigenschaften durch die Angabe eines Modells (Transitionssystem) festlegen. Gerade die Möglichkeit, den eigenschaftsorientierten mit dem modellorientierten Spezifikationsstil kombinieren zu können, halte ich im Sinne einer hohen Flexibilität für einen großen Vorteil. Allerdings lassen sich auch rein-eigenschaftsorientierte Spurspezifikationen angeben (vgl. z.B. [Broy, Streicher 87], [Broy 88a]). Dies ist möglich, da Spurformeln ausdrucksstark genug sind, um beliebige Sicherheits- und Lebendigkeitseigenschaften formulieren zu können; die Kombination einer eigenschaftsorientierten Spezifikation mit einer modellorientierten hat allein pragmatische Vorteile.

5. Einbindung in eine Entwurfsmethodik: Die Anforderungsspezifikation darf nicht isoliert von den restlichen Phasen des Entwicklungsprozesses gesehen werden. Insbesondere muß die Schnittstelle zur nächsten Phase, in unserer Methodik zur Entwurfsspezifikation (vgl. Abschnitt 1.1), klar definiert sein. Dieses Ziel wird mit meinem Vorgehensmodell erreicht (siehe Kap. 5). Die Einbindung in eine Entwurfsmethodik ist sowohl für den Systementwickler als auch für den Kunden von Bedeutung: Der Systementwickler weiß, wie ausgehend von der Anforderungsspezifikation mit der Systementwicklung fortzufahren ist, wie die Korrektheit einer Entwurfsspezifikation bzgl. einer Anforderungsspezifikation bewiesen werden kann. Der Kunde kann verfolgen, ob das schließlich erstellte Produkt der Anforderungsspezifikation genügt, da die verschiedenen Formalismen der Entwurfsmethodik aufeinander abgestimmt sind.

6. Änderbarkeit: Gemäß Abschnitt 1.2 ist die Ermittlung der Anforderungen ein interaktiver Prozeß; Änderungen und Präzisierungen von Anforderungen sind dabei die Regel. Daher halte ich es für wichtig, daß sich einzelne Anforderungen leicht durch neue ersetzen lassen. Die leichte Änderbarkeit von Anforderungen ist im Spurformalismus zumindest dann gegeben, wenn der ablauforientierte, axiomatische Spezifikationsstil verwendet wird. Bei Anforderungen, die durch ein Transitionssystem beschrieben werden, ist dies schwieriger; hier kann eine größere Änderung des gesamten Modells, z.B. des Zustandsraums, erforderlich sein, wenn sich einzelne Anforderungen ändern.

7. Verwendung vertrauter Konzepte: Für die Akzeptanz eines Spezifikationsformalismus ist es von Vorteil, wenn ein Benutzer des Formalismus mit dessen Konzepten vertraut ist. Im Spurformalismus werden Prädikatenlogik, Transitionssysteme und algebraische Spezifikation verwendet. Diese drei Konzepte gehören zum Standardrepertoire der Informatik; zumindest die ersten beiden sind auch den zahlreichen Anwendern mit einer technischen Ausbildung bekannt. Außerdem bieten diese Konzepte den Vorteil, daß sie theoretisch gründlich untersucht sind.

Ich will auch auf zwei Eigenschaften hinweisen, die der Spurformalismus höchstens eingeschränkt hat:

Häufig wird die Möglichkeit der *graphischen Beschreibung* als wünschenswert für einen Spezifikationsformalismus betrachtet. Der Spurformalismus ist dagegen textuell. Graphik läßt sich jedoch an manchen Stellen als Hilfsmittel einsetzen: ich nenne hier die tabellenorientierte Notation von Transitionssystemen (vgl. Abschnitt 3.4) sowie die Darstellung von Komponentennetzen durch Datenflußgraphen (vgl. Abschnitt 4.1). Ist die Zustandsmenge eines Transitionssystems hinreichend klein, so bietet sich eine graphische Beschreibung auch dort an.

Es wird auch oft als vorteilhaft angesehen, wenn eine Anforderungsspezifikation ausführbar ist bzw. relativ schnell ein Prototyp eines Systems aus ihr erstellt werden kann (*rapid prototyping*). Dies ist z.B. bei der Sprache Paisley ([Zave, Schell 86]) möglich, nicht jedoch im Spurformalismus, wie wir in Abschnitt 2.1 gesehen haben. Eine große Ausdruckskraft und Ausführbarkeit sind unvereinbare Ziele; für eine Anforderungsspezifikation halte ich vorrangig ersteres für wichtig.

2.3 Vergleich mit verwandten Ansätzen

In Abschnitt 2.1 wurden den Modellierungsannahmen des Spurformalismus Alternativen gegenübergestellt, ohne jedoch weiter auf konkrete, aus der Literatur bekannte Ansätze einzugehen. Dies hole ich nun nach. Um den Umfang gering zu halten beschränke ich mich auf Formalismen und methodische Ansätze, die mit der Spurspezifikation eng verwandt sind.

Zunächst ein Vergleich mit einigen aktionsorientierten Formalismen:

1. Theorie formaler Sprachen und Automatentheorie

Faßt man Aktionen als Elemente eines Zeichenvorrats auf, so sind Spuren endliche und unendliche Worte im Sinne der Theorie formaler Sprachen und der Automatentheorie. Die Theorie formaler Sprachen befaßt sich mit der Untersuchung von Sprachklassen und mit Formalismen zu deren Beschreibung (Automaten, Grammatiken); das (technische) Ergebnis einer Anforderungsspezifikation kann dagegen jede beliebige Spurmenge sein. Meine Arbeit befaßt sich im Gegensatz zur Theorie formaler Sprachen nicht mit der theoretischen Untersuchung eines Formalismus, sondern mit der methodischen Anwendung eines Formalismus.

2. Spurspezifikation nach Bartussek und Parnas

Eine frühe Verwendung von Spuren als Spezifikationsmittel findet sich in [Bartussek, Parnas 77]. Bartussek und Parnas benutzen Spuren von Prozeduraufrufen zur Analyse des Verhaltens von Programmen. Die Modellierungssicht unterscheidet sich also von der meiner Arbeit.

3. Spuren bei Prozeßalgebren

Auch bei Prozeßalgebren gibt es Ansätze, (in der Regel endliche) Spuren zur Spezifikation von Systemeigenschaften zu verwenden (vgl. z.B. [Hoare 85]). Eine Spur beschreibt dort eine endliche Approximation an das Verhalten eines Systems; unendliches Verhalten wird lediglich implizit modelliert. Dadurch lassen sich nicht beliebige Lebendigkeitseigenschaften ausdrücken; so ist es nicht möglich, Systeme, bei denen eine beliebig große, aber endliche Anzahl von Aktionen nacheinander möglich ist, von solchen zu unterscheiden, bei denen unendlich viele

Aktionen nacheinander stattfinden müssen. In [Hoare 85] wird das semantische Modell der Spuren verfeinert, um solche Unterschiede wiederzugeben; hierfür gibt es verschiedene Möglichkeiten (z.B. Spuren mit Verweigerungsmengen). Wenn solche Modelle auch zur Semantikdefinition von Agenten brauchbar sein mögen, so erscheinen sie mir für die Anforderungsspezifikation doch zu komplex. Im Gegensatz zu Prozeßalgebren lassen sich in meinem Ansatz interagierende Komponenten durch Spuren hinreichend genau beschreiben, da hier Komponenten nicht synchron, sondern asynchron kommunizieren und neben endlichen auch unendliche Spuren verwendet werden (vgl. auch Punkt 4). Neuere Arbeiten über Spuren bei Prozeßalgebren finden sich in [Drost 90].

Den Übergang von einer Spur- zu einer Agentenspezifikation untersucht Olderog in seiner Habilitationsschrift ([Olderog 88]). Mein Ansatz ist insofern allgemeiner als der von Olderog, als ich auf der Spurebene beliebige Lebendigkeitseigenschaften beschreiben kann; anders als bei Olderog ist das Ziel der Entwicklung in meinem Ansatz nicht ein abstraktes Programm in einer Prozeßalgebra (mit synchroner Kommunikation), sondern eine (noch "abstraktere") Entwurfsspezifikation durch stromverarbeitende Funktionen (mit asynchroner Kommunikation; vgl. Kap. 5).

Die Beschreibung von Echtzeitsystemen wird auch im Rahmen von Prozeßalgebren untersucht. Das in [Reed, Roscoe 86] vorgestellte erweiterte Spurmodell zur Beschreibung von Echtzeiteigenschaften ähnelt den zeitbehafteten Spuren in Abschnitt 3.6.

4. Spurmodell nach Jonsson

In [Jonsson 87] wird eine kompositionale Spursemantik für I/O-Automaten angegeben, die eine semantische Grundlage für die Spezifikation und kompositionale Verifikation verteilter Systeme liefert. I/O-Automaten modellieren ebenso wie stromverarbeitende Funktionen asynchron kommunizierende Agenten. Der Begriff "Spur" ist der gleiche wie in meiner Arbeit, zur Spezifikation von Spuren verwendet Jonsson dagegen allein Transitionssysteme mit gewissen Lebendigkeitsannahmen.

Ich stütze mich in meiner Arbeit auf einige von Jonssons Ergebnissen. So ist das semantische Modell einer komponentenorientierten Spezifikation (vgl. Kap. 4) von Jonsson übernommen; auch die Spursemantik von Agenten (Kap. 5) ist von ihm beeinflußt. Im Gegensatz zu Jonssons Arbeit ist meine wesentlich stärker methodisch ausgerichtet: Die flexible, größtenteils axiomatische Beschreibung von Systemeigenschaften und der Übergang von globalen Spezifikationen zu komponentenorientierten Spezifikationen steht im Vordergrund; dies wird von Jonsson nicht behandelt. Ich führe keinen neuen Formalismus ein, sondern untersuche die methodische Verwendung eines weitgehend bestehenden Formalismus.

5. Spurtheorie für den Schaltungsentwurf

In den letzten Jahren wurde die Spurtheorie (*trace theory*) für den Entwurf geschwindigkeitsunabhängiger Schaltungen entwickelt (siehe z.B. [Snepscheut 85], [Rem 85], [Dill 89]). Eine Schaltung wird durch eine "Spurstruktur" *(trace structure)*, d.h. eine Aktionenmenge und eine Spurmenge beschrieben; für Spurstrukturen werden Kompositionsoperatoren definiert. Zur Modellierung wurden zunächst nur endliche Spuren verwendet ([Snepscheut 85], [Rem 85]), in der letzten Zeit aber auch unendliche, so daß sich

neben Sicherheitseigenschaften auch Lebendigkeitseigenschaften ausdrücken lassen ([Dill 89]). Die Modellierungssicht von Dill ist der von Jonsson (s.o.) ähnlich. Im Gegensatz zu meiner Arbeit ist die Spurtheorie für den Schaltungsentwurf auf die Modellierung eines Spezialfalls verteilter Systeme, nämlich Schaltungen, sowie auf Fragen der Verifikation ausgerichtet; weitere methodische Gesichtspunkte spielen eine untergeordnete Rolle.

6. Spurtheorie nach Mazurkiewicz

Mazurkiewicz ([Mazurkiewicz 86]) verwendet den Begriff "Spurtheorie" *(trace theory)* in einem anderen Sinn: Systemabläufe werden nicht allein durch eine endliche oder unendliche Sequenz von Aktionen modelliert, sondern zusätzlich durch eine "Abhängigkeitsrelation"; nicht abhängige Aktionen können gleichzeitig stattfinden. Insofern ähnelt dieser Ansatz eher den *Pomsets* (s.u.; tatsächlich sind Spuren nach Mazurkiewicz spezielle Pomsets).

7. Pomsets

Bisher habe ich Formalismen vorgestellt, die verteilte Systeme aktionsorientiert nach der Interleaving-Sicht modellieren; eine Ausnahme ist der Ansatz von Mazurkiewicz: er kann als Zwischenglied zwischen der Interleaving-Sicht und der Sicht der expliziten Nebenläufigkeit gewertet werden. Explizite Nebenläufigkeit modellieren die *Pomsets (partially ordered multisets of events)* nach [Pratt 86]. Pratt beschreibt verteilte Systeme durch Pomset-Ausdrücke, die den aus der Theorie formaler Sprachen bekannten regulären Ausdrücken ähneln. Mein Eindruck ist, daß sich meine Methodik auch auf die Halbordnungssemantik von Pomsets übertragen läßt; ein erster Ansatz dazu wurde in [Broy 89a] unternommen. Ich glaube jedoch, daß die Pomset-Beschreibung weniger handlich als der Spurformalismus ist.

Bislang war allein von aktionsorientierten Formalismen die Rede. Es besteht jedoch auch eine enge Verbindung zwischen einigen zustandsorientierten Formalismen und dem Spurformalismus. Hier ist vor allem die temporale Logik zu nennen, in methodischer Hinsicht insbesondere die Arbeiten im Umfeld von Lamport.

8. Temporale Logik

Mit der temporalen Logik läßt sich das Verhalten verteilter Systeme im Zeitablauf modellieren, ohne einen Zeitparameter einzuführen. Zwei verschiedene Ansätze sind gebräuchlich: Die *linear time temporal logic*, ein Interleaving-Formalismus, und die *branching time logic*, die stärker den Nichtdeterminismus berücksichtigt. Für eine ausführliche Diskussion dieser beiden Richtungen sei auf [Lamport 80] und [Emerson, Halpern 86] verwiesen. Ich beziehe mich im folgenden ausschließlich auf die *linear time temporal logic*, die dem Spurformalismus näher steht als die *branching time temporal logic*. Neben Standardoperatoren wie "immer", "irgendwann" oder "im nächsten Schritt" führen manche Autoren weitere Operatoren ein (z.B. Vergangenheitsoperatoren), die die Logik ausdrucksmächtiger machen. Die temporale Logik dient ebenso wie der Spurformalismus der axiomatischen, ablauforientierten Spezifikation. Sie unterscheidet sich vom Spurformalismus vor allem dadurch, daß Zustandssequenzen als Modellbegriff gewählt werden und eher implizit über derartige Sequenzen gesprochen wird (für einen genaueren Vergleich zwischen temporaler Logik und dem Spurformalismus siehe Abschnitt 3.3).

9. Lamports Transitionsaxiom-Methode

Lamport spezifiziert in seiner Transitionsaxiom-Methode verteilte Systeme durch die Kombination von Transitionssystemen mit Formeln einer temporalen Logik ([Lamport 83a], [Lamport 89]). Auch in meinem Ansatz ist die Kombination des transitionsorientierten mit dem ablauforientierten Spezifikationsstil möglich, anders als Lamport spezifiziere ich jedoch nicht nur Lebendigkeitseigenschaften ablauforientiert, sondern auch (möglicherweise alle) Sicherheitseigenschaften eines Systems.

Besonders in der Arbeit von Lamport sehe ich viele Gemeinsamkeiten mit meinem methodischen Vorgehensmodell. Dies betrifft die Trennung von Sicherheits- und Lebendigkeitseigenschaften, die Kombination des ablauf- mit dem transitionsorientierten Spezifikationsstil, die Aufteilung in Produkt- und Umgebungsanforderungen und das Konzept der Realisierbarkeit. Gerade zu den letzten beiden Punkten entstanden parallel zu meiner Arbeit ähnliche Ansätze ([Abadi, Lamport 90], [Abadi, Plotkin 91]; vgl. Abschnitt 4.3). Man kann daran Fragestellungen ablesen, die unabhängig davon sind, ob ein aktions- oder ein zustandsorientierter Formalismus gewählt wird. Ich erwarte mir in der Zukunft eine weitere wechselseitige Beeinflussung dieser beiden Ansätze. Im Rahmen von FOCUS verspreche ich mir allerdings einen Vorteil von einem aktionsorientierten Formalismus: ich glaube, die Anknüpfung einer Anforderungsspezifikation an eine Entwurfsspezifikation durch stromverarbeitende Funktionen gelingt besser in einem aktionsorientierten Formalismus (vgl. Kap. 5).

10. UNITY

Wie FOCUS ist UNITY ([Chandy, Misra 88]) eine Methodik zur Konstruktion paralleler Programme. Die Anforderungsspezifikation erfolgt in einer Logik, die mit der temporalen Logik eng verwandt ist. Es zeigen sich im wesentlichen die Gemeinsamkeiten und Unterschiede zum Spurformalismus wie unter Punkt 8 beschrieben. Wie in meiner Arbeit besteht auch in UNITY die empfohlene Vorgehensweise darin, zunächst eine globale Spezifikation zu erstellen und sich erst dann zu überlegen, welche Aufgaben dem zu erstellenden Produkt und welche der Umgebung zugeordnet werden. Während der Spurformalismus kompositional ist (vgl. Kap. 4), gilt dies für UNITY nur eingeschränkt (vgl. [Sanders 91]). Die in UNITY wesentliche Technik der schrittweisen Verfeinerung von Spezifikationen spielt in meiner Arbeit beim Übergang von einer (komponentenorientierten) Anforderungsspezifikation zu einer Entwurfsspezifikation eine wichtige Rolle. Die Vorgehensweise in beiden Methodiken ist ähnlich (vgl. Abschnitt 5.3)

11. Formale Requirements-Engineering-Sprachen

Auf dem Gebiet des Requirements Engineering für verteilte Systeme finden in der letzten Zeit formale Methoden verstärkt Einzug. Es wurden Sprachen entwickelt, die auf die Formalisierung von Anforderungen abgestimmt sind. Ein Beispiel ist die Sprache ERAE ([Dubois et al. 88]), die das aus der Datenbankwelt stammende Entity-Relationship-Modell mit temporaler Logik kombiniert. Wie die temporale Logik ist diese Sprache zustandsorientiert. Ein verwandter Ansatz ist die Sprache MAL ([Jeremaes et al. 86]), die auf einer speziellen modalen Logik basiert. Ebenso wie in meiner Arbeit wird in beiden Ansätzen die Notwendigkeit einer Methodik betont, die auf den jeweiligen Formalismus abgestimmt ist. Nach meiner Beobachtung wird bei Requirements-Engineering-Sprachen dem Übergang zur nächsten Phase,

in FOCUS der Entwurfsspezifikation, wenig Beachtung geschenkt; dies wird in meiner Arbeit ausführlich behandelt.

12. Formale Anforderungsspezifikation transformationeller Systeme

Zum Schluß dieses Abschnitts will ich begründen, warum sich bewährte Techniken der Spezifikation transformationeller ("sequentieller") Programme nicht (uneingeschränkt) für die Anforderungsspezifikation verteilter Systeme heranziehen lassen. In [Harel, Pnueli 85] werden *reaktive* Systeme von *transformationellen* abgegrenzt. Ein *transformationelles* System bekommt genau eine Eingabe und transformiert diese in genau eine Ausgabe[1]. Transformationelle Systeme können durch Funktionen bzw. Relationen oder durch zwei Zustände (Vorbedingung - Nachbedingung) beschrieben werden. Bei verteilten Systemen ist ein Denken in nur zwei Zuständen nicht mehr sinnvoll, vielmehr muß ein Verhalten im Zeitablauf mit vielen wechselseitigen Beeinflussungen nebenläufig agierender Komponenten modelliert werden. Wir sprechen von einem *reaktiven* oder *interaktiven* System. Die Techniken zur Spezifikation transformationeller Systeme (z.B. die algebraische Spezifikation ([Wirsing 90]), VDM ([Bjørner, Jones 82]) oder die Methodik von Gries ([Gries 81]); für einen Überblick vgl. [Sanella 88]) lassen sich für reaktive Systeme höchstens in modifizierter Form einsetzen. Vor- und Nachbedingungen müssen zu Spielregeln verallgemeinert werden, an die sich die nebenläufigen Komponenten während eines gesamten Ablaufs zu halten haben. Dies ist die Vorstellung des Annahme/Verpflichtung-Konzepts nach [Jones 83] (vgl. auch Abschnitt 4.2.2).

[1] Bei nichtdeterministischen Systemen gibt es zwar mehrere *Möglichkeiten* für eine Ausgabe, jedoch findet auch hier nur genau eine statt.

3. Globale Spezifikation

Dieses Kapitel erläutert die Sprachmittel und Methoden, die zur Formulierung und Validierung globaler Anforderungen verwendet werden. Das Attribut "global" drückt aus, daß sich hier im Gegensatz zur komponentenorientierten Spezifikation die Anforderungen an das System als ganzes richten, d.h. an das zu erstellende Produkt *und* seine Umgebung. Neben algebraischen und logischen Techniken werden Transitionssysteme eingesetzt, um Anforderungen möglichst flexibel ausdrücken zu können. Auf diese Weise läßt sich ein ablauforientierter mit einem transitionsorientierten Spezifikationsstil kombinieren.

Abschnitt 3.1 behandelt die Beschreibung von Aktionen, das Grundkonzept des Spurformalismus. Die ablauforientierte und transitionsorientierte Spezifikation sowie ihre Verbindung sind in den Abschnitten 3.3 und 3.4 dargestellt. Bei der ablauforientierten Spezifikation durch Spurformeln werden gewisse Standardoperationen für Spuren verwendet, die in einem abstrakten Datentyp festgelegt sind (Abschnitt 3.2). Ein wichtiges methodisches Instrument, die Strukturierung einer globalen Spezifikation in Sicherheits- und Lebendigkeitsanforderungen, ist in Abschnitt 3.5 beschrieben. Abschnitt 3.6 zeigt schließlich, wie sich globale Echtzeitanforderungen durch eine einfache Erweiterung des Spurformalismus wie rein funktionale Anforderungen formalisieren lassen.

Es sei darauf hingewiesen, daß ich algebraische Spezifikationen und prädikatenlogische Formeln (Spurformeln) in meiner Arbeit ausschließlich als eine komfortable Schreibweise verwende. Auf Untersuchungen des zugrundeliegenden logischen Apparates und kalkülmäßiges Beweisen gehe ich nicht ein. Entsprechend sind die Beweise in meiner Arbeit durchweg "mathematisch", nicht kalkülmäßig. Mit der Verwendung von algebraischer Spezifikation und logischer Formeln deute ich an, daß auch eine vollständig formale Behandlung sowie eine mittels algebraischer Spezifikation strukturierte Darstellung möglich ist.

3.1 Beschreibung von Aktionen

Im ersten Schritt einer Systementwicklung wird festgelegt, welche *Aktionen* im System auftreten sollen bzw. dürfen. Diese Festlegung orientiert sich an der Erfahrungswelt des Kunden, nämlich an den in der Realität auftretenden bzw. geplanten Tätigkeiten und Objekten. Tätigkeiten werden in der Regel in Aktionen umgesetzt, die Objekte als Parameter haben. Zum Beispiel treten in einem Postfachsystem die Objekte Nachrichten, Postfächer sowie sendende und empfangende Prozesse auf; eine Aktion wäre hier das Senden einer Nachricht von einem Prozeß zu einem Postfach.

Zur Verdeutlichung der methodischen Vorgehensweise verwende ich durchgehend das Beispiel "Postfachsystem". Es ähnelt dem in [Weber 90b] auf der Spurebene und in [Dendorfer 91] funktional beschriebenen, ist jedoch stark vereinfacht. Insbesondere werden Echtzeitaspekte sowie Möglichkeiten zum dynamischen Erzeugen und Löschen von Postfächern nicht berücksichtigt.

Beispiel (Postfachsystem: informelle Beschreibung): Das Postfachsystem enthält eine feste Anzahl von Postfächern, die Nachrichten aufnehmen können, sowie sendende und empfangende Prozesse. Für jedes Postfach gibt es eine Obergrenze für die Anzahl der

Nachrichten, die es speichern kann. Sendende Prozesse schreiben Nachrichten in die Postfächer, empfangende lesen (zerstörend) Nachrichten aus ihnen heraus. Das Schreiben und Lesen geschieht nach der FIFO-Disziplin. Jeder Sende- und Empfangsversuch wird durch eine Rückmeldung quittiert. Bei der Rückmeldung wird angegeben, ob die Kommunikation erfolgreich verlief. Eine Kommunikation ist erfolglos, wenn in ein volles Postfach geschrieben oder aus einem leeren Postfach gelesen werden sollte. Für die Flußkontrolle[1] sind die sendenden und empfangenden Prozesse verantwortlich. Sie verpflichten sich, nach jedem Sende- bzw. Empfangsversuch die entsprechende Rückmeldung abzuwarten, bevor sie mit weiteren Kommunikationsaktionen fortfahren ("Stop-and-Wait-Protokoll"). □

Die formale Festlegung der Aktionen und Objekte erfolgt durch Angabe eines oder mehrerer algebraisch spezifizierter abstrakter Datentypen (für die algebraische Spezifikation vgl. [Wirsing 90]). Der Datentyp der Aktionen dient im wesentlichen als eine konkrete Syntax für die Menge der möglichen Aktionen. Die Verbindung zur informellen Beschreibung wird häufig klar sein, sollte zur Dokumentation aber auch in Form von Kommentaren niedergelegt werden.

Beispiel (Postfachsystem: Aktionen): Bei dem Postfachsystem werden

- das Senden einer Nachricht von einem Prozeß an ein Postfach,
- das Quittieren einer Sendeaktion mit einer Rückmeldung (ok oder error),
- der Empfangswunsch eines Prozesses an ein Postfach und
- das Quittieren eines Empfangswunsches mit einer Rückmeldung und einer Nachricht (nil, falls der Empfang nicht möglich war)

als beobachtbare Aktionen betrachtet. Später wird festgelegt, daß Sendeaktionen von der Komponente "sendende Prozesse" ausgeführt werden, Empfangswünsche von der Komponente "empfangende Prozesse", Quittierungsmeldungen von der Komponente "Postfachserver" (siehe Abschnitt 4.1). Diese Festlegung spielt aber in diesem Kapitel noch keine Rolle. Die Aktionen lassen sich durch den folgenden abstrakten Datentyp formalisieren:

spec ACTION ≡
  based on MBX
    sort Act,
    snd: Tsk × Mbx × Msg → Act,
    s-ack: Tsk × Mbx × Sig → Act,
    rec: Tsk × Mbx → Act,
    r-ack: Tsk × Mbx × Msg × Sig → Act
end

Im abstrakten Datentyp ACTION werden die Sorte der Aktionen Act sowie die Aktionen snd *(send)*, s-ack *(send acknowledgement)*, rec *(receive)* und r-ack *(receive acknowledgement)* beschrieben. Dieser Datentyp erweist sich als sehr einfach: es werden keine Axiome angegeben.

---

[1] Die Flußkontrolle dient in einem Kommunikationssystem dazu, daß die Kommunikationspartner nicht von Nachrichten "überflutet" werden. Hierzu gibt es verschiedene Protokolle (vgl. [Tanenbaum 81]).

Er stützt sich auf den Datentyp MBX, in dem Prozeßidentifikatoren (Tsk), Postfachidentifikatoren (Mbx), Nachrichten (Msg) und Rückmeldesignale (Sig) spezifiziert sind. Dort seien die entsprechenden Sorten Tsk, Mbx, Msg und Sig eingeführt. Die mit Sig bezeichnete Menge bestehe aus den Elementen ok und error.

Aktionen sind nicht mit Funktionsaufrufen zu verwechseln, die ein Ergebnis liefern; die Terme der Sorte Act dienen nur zur strukturierten Darstellung von Aktionen. So bezeichnet z.B. für jeden Prozeß ts aus Tsk, jedes Postfach mb aus Mbx und jede Nachricht ms aus Msg der Term snd(ts,mb,ms) eine Sendeaktion. □

In vielen Fallstudien zur Spurspezifikation (z.B. [Broy, Streicher 87], [Broy 88a]) wurden die Aktionen ähnlich einfach beschrieben. Der Grund dafür liegt darin, daß dort die Abläufe des Systems im Vordergrund stehen, nicht aufwendige Datenmanipulationen. Will man solche modellieren, so werden die Objekte und die Aktionen komplexer (vgl. [Broy 87b]). Die Technik der algebraischen Spezifikation eignet sich gut zu deren Beschreibung.

Im obigen Beispiel können die Aktionen auf natürliche Weise als zusammengehörig betrachtet werden. Für größere Systeme ist es jedoch sinnvoll, Teilklassen von zusammengehörigen Objekten und Aktionen zum Zweck der Strukturierung durch verschiedene abstrakte Datentypen zu beschreiben.

3.2 Der Datentyp der Ströme

Spuren lassen sich ebenfalls durch algebraische Techniken beschreiben. Sie werden auch *Ströme (von Aktionen)* genannt; je nachdem, ob der Verwendungszweck der Ablaufbeschreibung oder die Datenstruktur betont werden soll, wird der Ausdruck "Spur" oder "Strom" bevorzugt. Unabhängig von der Wahl der Aktionen gibt es häufig benutzte Operationen auf Strömen. Ich stelle nun jene Operationen vor, die in Fallstudien zur Spurspezifikation immer wieder vorkommen. Der Formalismus geht jedoch nicht von einem Satz von Standardoperationen aus, sondern gestattet es, bei Bedarf neue Operationen durch algebraische Spezifikation zu definieren. Insbesondere ist es oft hilfreich, neue Operationen als Abkürzungen für komplexe zusammengesetzte Operationen einzuführen.

Der unten angegebene abstrakte Datentyp STREAM beschreibt bei einer vorgegebenen Aktionenmenge, dargestellt durch die Sorte Act, endliche oder unendliche Sequenzen von Aktionen:

$$Act^{\omega} = Act^{*} \cup Act^{\infty},$$

wobei Act^{*} die Menge der endlichen und Act^{∞} die Menge der unendlichen Sequenzen von Aktionen bezeichnet. Die Beschreibung dieses Datentyps orientiert sich an [Broy 89a].

In Erweiterung der Grundkonzepte der algebraischen Spezifikation (vgl. [Wirsing 90]) nehmen wir eine Signatur (S,Op) als gegeben an; S sei eine Menge von Sorten, Op eine Menge von Funktions- und Relationssymbolen. Die Sorten können auch funktionale und Mengensorten sowie weitere generische Sorten enthalten. Überladung von Funktions- und Relationssymbolen (*overloading*) und Mixfix-Notation sind erlaubt.

Für eine Sorte S bezeichnen wir mit **set** S die Potenzmenge über der S entsprechenden Menge. Nat_∞ bezeichnet die Menge der natürlichen Zahlen, die um die Zahl ∞ ("unendlich") erweitert ist, mit der üblichen Ordnung. Ausgezeichnete Relationssymbole sind =, DEF und $\sqsubseteq$, welche für die Gleichheit, Definiertheit und partielle Ordnung stehen.

Ein Modell für einen Datentyp ist eine Algebra, bei der alle Sorten vollständigen Halbordnungen (*complete partially ordered sets*) mit kleinstem Element $\bot$ (*"bottom"*, "undefiniert") entsprechen. Alle Funktionssymbole stehen für totale Funktionen, Relationssymbole entsprechen Relationen. Terme und Formeln sind wie üblich definiert.

Die Bedeutung von DEF wird durch das folgende Axiom gegeben:

$$x = \bot \Leftrightarrow \neg DEF(x)$$

1. Ströme

Der Datentyp der Ströme enthält die folgenden Grundoperationen: ε bezeichnet den leeren ("undefinierten") Strom, also das $\bot$-Element der der Sorte S^ω entsprechenden Trägermenge. ft(s) liefert das erste Element des Stroms s, falls dieser nicht leer ist. rt(s) ergibt den Strom, bei dem das erste Element von s entfernt ist. Die Operation $\circ$ *(prefixing, left-append)* wird in Infixschreibweise notiert. Dies wird durch die folgende Spezifikation formalisiert:

spec STREAM = **for given sort** S:
sort S^ω,
ε: S^ω,
ft: $S^\omega \to S$,
rt: $S^\omega \to S^\omega$,
$\_\circ\_$: $S \times S^\omega \to S^\omega$
axioms $\forall x \in S,\ s \in S^\omega$:
$ft(x \circ s) = x$,
$DEF(x) \Rightarrow rt(x \circ s) = s$,
$DEF(x \circ s) \Leftrightarrow DEF(x)$,
$DEF(rt(s)) \Rightarrow DEF(s)$,
$\neg DEF(\varepsilon)$
end

Der Datentyp der Ströme enthält nur die Grundoperationen auf Strömen. Er läßt sich erweitern um weitere wichtige Funktionen und Relationen, die sich aufbauend auf den Grundoperationen definieren lassen:

2. Präfixrelation

Die Präfixrelation wird mit $\sqsubseteq$ bezeichnet. $s \sqsubseteq t$ gilt genau dann, wenn der Strom s ein Anfangsstück (*Präfix*) des Stroms t ist. Die Präfixrelation wird rekursiv definiert:

$$s \sqsubseteq t \Leftrightarrow s = t \vee s = \varepsilon \vee (ft(s) = ft(t) \wedge rt(s) \sqsubseteq rt(t)).$$

Die Ströme bilden eine vollständige Halbordnung mit der Präfixrelation als Ordnung und dem leeren Strom als kleinstem Element.

3. Relation in

x *in* s gibt an, ob das Element x im Strom s vorkommt.

$$\_in\_: S \times S^{\omega} \to \text{Bool}$$

wird definiert durch:

$$x \; in \; s \Leftrightarrow \exists i \in \text{Nat}: ft(rt^i(s)) = x,$$

wobei $rt^0(s) =_{def} s$, $rt^{i+1}(s) =_{def} rt(rt^i(s))$.

4. Filteroperationen

Ich führe zwei Filteroperationen für Ströme ein, wobei die zweite eine Verallgemeinerung der ersten ist:

$$\_©\_: S \times S^{\omega} \to S^{\omega},$$
$$\_©\_: \textbf{set}\ S \times S^{\omega} \to S^{\omega},$$

a © s bzw. M © s liefert für ein Element a bzw. eine Menge M den Teilstrom von s, der nur aus a- bzw. M-Elementen besteht.

Die erste Filteroperation ist definiert durch:

$$a © \varepsilon = \varepsilon,$$
$$a = b \Rightarrow a © (b^{\circ}s) = b^{\circ}(a © s),$$
$$a \neq b \Rightarrow a © (b^{\circ}s) = a © s,$$

die zweite durch:

$$M © \varepsilon = \varepsilon,$$
$$b \in M \Rightarrow M © (b^{\circ}s) = b^{\circ}(M © s),$$
$$b \notin M \Rightarrow M © (b^{\circ}s) = M © s.$$

5. Längenoperation

#(s) (bzw. in einer klammersparenden Schreibweise #s) ergibt die Länge von s, für unendliche Ströme ist das Ergebnis ∞. Die Längenoperation

$$\#: S^{\omega} \to \text{Nat}_{\infty},$$

wird definiert durch:

$$\#(\varepsilon) = 0,$$
$$DEF(x) \Rightarrow \#(x^{\circ}s) = 1 + \#(s)$$

Man beachte, daß bei dieser Definition eine vollständige Fallunterscheidung vorliegt: ein Strom ist definiert, wenn sein erstes Element definiert ist, ansonsten undefiniert, d.h. gleich ε.

6. Konkatenation

Die Konkatenation zweier Ströme ist eine (bzgl. der Präfixordnung) nichtmonotone und daher

nichtstetige Funktion. Sie wird wie das *prefixing* mit $\circ$ bezeichnet (Elemente aus S werden somit mit den Strömen der Länge 1 identifiziert):

$$\_\circ\_: S^\omega \times S^\omega \rightarrow S^\omega,$$

Die Konkatenation ist definiert gemäß (seien s und t Ströme, a ein Term der Sorte S):

$$\varepsilon \circ s = s,$$
$$(a \circ s) \circ t = a \circ (s \circ t),$$

3.3 Ablauforientierte Spezifikation

Vorbemerkung zur Kombination eines ablauforientierten mit einem transitionsorientierten Spezifikationsstil

Ich stelle zwei kombinierbare Möglichkeiten zur Festlegung von Anforderungen vor: *Spurformeln* zur ablauforientierten Spezifikation in diesem Abschnitt, *Transitionssysteme* zur transitionsorientierten Spezifikation im folgenden Abschnitt. Nach meiner Erfahrung gibt es nämlich zwei prinzipiell verschiedene Arten Anforderungen an Abläufe festzulegen: zum einen können Anforderungen an einen *gesamten* Ablauf gestellt werden *(ablauforientierter Spezifikationsstil)*, zum anderen können Anforderungen an Einzelschritte eines Ablaufs gerichtet werden *(transitionsorientierter Spezifikationsstil)*. Beim transitionsorientierten Spezifikationsstil hat man nicht einen ganzen Ablauf im Auge, sondern einzelne Zustandsübergänge: man stellt sich vor, wie das System von einer Momentaufnahme zur nächsten übergeht.

Inwieweit die Bevorzugung eines der beiden Spezifikationsstile vom Problem, der Person oder nur der Gewohnheit abhängt, will ich hier nicht untersuchen. Ein Formalismus zur Anforderungsspezifikation sollte es gestatten, informelle Anforderungen möglichst direkt in formale umsetzen zu können. Dies ist der methodische Grund für die Verwendung zweier Spezifikationsstile.

In meinem Ansatz unterscheiden sich ablauforientierte und transitionsorientierte Spezifikationen in der Ausdruckskraft: Transitionsorientiert lassen sich hier nur Sicherheitsanforderungen beschreiben, ablauforientiert dagegen sowohl Sicherheits- als auch Lebendigkeitsanforderungen. Dadurch läßt sich auch rein ablauforientiert spezifizieren (vgl. [Broy, Streicher 87], [Broy 88a]). (Für die Gründe, weswegen ich keine Lebendigkeitsanforderungen durch Transitionssysteme festlege vgl. Abschnitt 3.4.1.) Einen Schritt weiter geht Lamport in seiner Transitionsaxiom-Methode ([Lamport 89]): Sicherheitsanforderungen müssen darin allein durch ein Transitionssystem spezifiziert werden, Lebendigkeitsanforderungen ablauforientiert mit temporaler Logik.

Die Kombination der Spezifikationsstile verursacht zwar keine semantischen Probleme (die Konjunktion einer transitionsorientiert formulierten Anforderung mit einer ablauforientiert formulierten ist einfach der Schnitt der entsprechenden Spurmengen), die technische Handhabbarkeit, insbesondere die Beweisführung, wird jedoch schwieriger. Der Grund liegt darin, daß die Systementwicklung nicht mehr in einem einheitlichen Rahmen stattfindet,

sondern Querverbindungen zwischen beiden Spezifikationsstilen herzustellen sind (vgl. Abschnitt 3.4.3).

Nach dieser Motivation der verschiedenen Spezifikationsstile nun zur ablauforientierten Spezifikation durch Spurformeln. Die erlaubten Systemabläufe werden - wie auch bei der transitionsorientierten Spezifikation - durch eine Spurmenge wiedergegeben. Die Spezifikation einer Spurmenge (und damit der erlaubten Abläufe) erfolgt hier durch eine Reihe von Anforderungen, die jede Spur dieser Menge erfüllen muß. Technisch wird dies durch die Angabe von Spurformeln $P_1, ..., P_n$ realisiert, die jeweils eine freie Variable der Sorte Act^ω enthalten und eine (Teil-) Anforderung beschreiben. Ihre Konjunktion ergibt die gesamte Anforderung an die Spurmenge:

$$P(t) =_{def} P_1(t) \wedge ... \wedge P_n(t)$$

Statt $P(t)$ schreibe ich auch oft $t \in P$; implizit wird damit ein Prädikat mit der Spurmenge identifiziert, die es beschreibt.

"Spurformeln" sind Formeln einer mehrsortigen Prädikatenlogik, bei der Spuren eine ausgezeichnete Sorte sind. Die Prädikatenlogik ist ein fundierter und weitverbreiteter Formalismus, der eine große Ausdruckskraft besitzt. Gerade letzteres halte ich bei einer Anforderungsspezifikation für sehr wichtig.

Sehen wir uns einige typische Beispiele von Spurformeln an:

Eine häufig verwendete Art von Spurformeln sind *Invarianten.* Dies sind Formeln der Bauart

$$P(t) \equiv \forall s \in Act^*: s \sqsubseteq t \Rightarrow Q(s)$$

Diese Formel besagt, daß alle endlichen Präfixe von t die Eigenschaft Q erfüllen; damit wird eine Menge von endlichen Beobachtungen des Systemverhaltens zusammengefaßt.

Oft werden in Invarianten die Filteroperationen in Verbindung mit der Längenoperation benutzt.

Beispiel (Postfachsystem: Spurformel-Anforderungen):

$$\text{S-ack\_safe}(t) \equiv \forall s \in Act^*: s \sqsubseteq t \Rightarrow \#(\{snd(ts,mb,ms) \mid ms \in Msg\} \copyright s) \geq \#(\{s\text{-}ack(ts,mb,sg) \mid sg \in Sig\} \copyright s)$$

Damit wird ausgedrückt, daß im Postfachsystem zu keinem Zeitpunkt mehr Sende-Rückmeldungen als Sendeaktionen stattgefunden haben (bezogen auf einen bestimmten Prozeß und ein bestimmtes Postfach). Bezüglich der Flußkontrolle (vgl. Abschnitt 3.1) besteht eine ähnliche Anforderung:

$$\text{Snd\_safe}(t) \equiv \forall s \in Act^*: s \sqsubseteq t \Rightarrow \#(\{snd(ts,mb,ms) \mid ms \in Msg\} \copyright s) \leq \#(\{s\text{-}ack(ts,mb,sg) \mid sg \in Sig\} \copyright s) + 1$$

Dies besagt, daß nach einer Sendeaktion die Rückmeldung abgewartet werden muß, bevor eine erneute Sendeaktion stattfindet (wiederum bezogen auf einen bestimmten Prozeß und ein bestimmtes Postfach).

Für Empfangsaktionen und ihre Rückmeldungen lassen sich entsprechende Anforderungen formulieren. □

Ein Spezialfall von Invarianten sind *Anforderungen an die Vorgeschichte*. Dies sind Formeln der Art:

$$P(t) \equiv \forall r \in Act^*, a \in Act: (r \circ a \sqsubseteq t \wedge Q(a)) \Rightarrow R(r)$$

Dies bedeutet: Eine bestimmte Aktion (ausgezeichnet durch das Prädikat Q) darf nur vorkommen, wenn vorher bestimmte Bedingungen erfüllt sind. Die Vorgeschichte (hier wiedergegeben durch die Teilspur r) läßt sich als *impliziter Zustand* auffassen. Die Interpretation der obigen Formel lautet damit: In einem Zustand r wird nur dann eine Aktion a ausgeführt, die Q erfüllt, wenn der Zustand das Prädikat R erfüllt. Es ergibt sich somit eine Verbindung zu der zustandsorientierten Spezifikation mit Transitionssystemen (vgl. Abschnitt 3.4). Die Zustände eines Transitionssystems werden *explizite Zustände* genannt, da sie explizit eingeführt werden.

Beispiel (Postfachsystem: Spurformel-Anforderungen, Forts.): Ein Beispiel für eine Anforderung an die Vorgeschichte ist das Prädikat S-ack_safe':

$$\text{S-ack\_safe}'(t) \equiv \forall r \in Act^*: \; r \circ \text{s-ack}(ts,mb,sg) \sqsubseteq t \Rightarrow$$
$$\#(\{\text{snd}(ts,mb,ms) \mid ms \in Msg\} \copyright r) > \#(\{\text{s-ack}(ts,mb,sg) \mid sg \in Sig\} \copyright r)$$

Wenn eine Rückmeldung erfolgt, müssen vorher bereits mehr Sendeaktionen als Rückmeldungen stattgefunden haben; damit ist diese Anforderung äquivalent zu S-ack_safe(t). Somit wird eine Aussage über die Aktivierungsbedingung *(enabling condition)* im impliziten Zustand r getroffen. □

Dual zu Anforderungen an die Vorgeschichte sind *Anforderungen an die Zukunft*, z.B.

$$P(t) \equiv \forall r \in Act^*, s \in Act^\omega: t = r \circ a \circ s \Rightarrow b \; \textit{in} \; s,$$

wobei a und b bestimmte Aktionen sind. Dies drückt aus: Wenn irgendwann a vorkommt, dann geschieht irgendwann später b.

Beispiel (Postfachsystem: Spurformel-Anforderungen, Forts.): Für das Postfachsystem gibt es eine ähnliche Anforderung:

$$\text{S-ack\_live}(t) \equiv \forall r \in Act^*, s \in Act^\omega: \; t = r \circ \text{snd}(ts,mb,ms) \circ s \Rightarrow$$
$$\exists sg \in Sig: \; \text{s-ack}(ts,mb,sg) \; \textit{in} \; s$$

Dies bedeutet: Jede Sendeaktion erhält irgendwann ihre Rückmeldung. □

Ich will die Technik der Formalisierung von Anforderungen durch Spurformeln hier nicht weiter erläutern; sie ist in mehreren Fallstudien dokumentiert (vgl. z.B. [Broy, Streicher 87], [Broy 88a]). Die Strukturierung von Anforderungen wird in Abschnitt 3.5 angesprochen.

Für die Analyse von mit Spurformeln formulierten Anforderungen und ihre Validation bietet sich die logische Ableitung von Varianten und Implikationen der Anforderungen an. Dadurch läßt sich feststellen, ob gewisse Eigenschaften, die man von einem System erwartet, tatsächlich

aus den Anforderungen folgen. Die Verwendung von Logik ermöglicht diese deduktive Analysetechnik; sie ist in [Dubois, Hagelstein 87] für eine temporale Logik dargestellt, die Vorgehensweise überträgt sich auf Spurformeln. Wie man beweist, daß ablauforientiert spezifizierte Eigenschaften aus transitionsorientiert spezifizierten folgen, zeigt Abschnitt 3.4.3.

Die Spezifikationsweise mit Spurformeln ähnelt der der temporalen Logik. Ich weise an dieser Stelle auf einige wichtige Unterschiede hin. Abgesehen von der Dualität "Aktion vs. Zustand" besteht der wesentliche Unterschied zwischen beiden Formalismen darin, daß in Spurformeln explizit über Sequenzen gesprochen wird - Spurvariablen treten in den Formeln auf -, in der temporalen Logik dagegen implizit. Durch den expliziten Bezug auf Sequenzen lassen sich die Zähl- und Filteroperationen leicht definieren, darüber hinaus kann man in einfacher Weise auf Teilsequenzen Bezug nehmen; dies hat sich in verschiedenen Fallstudien als hilfreich erwiesen. Solche Operationen sind in den gängigen Varianten der temporalen Logik nicht enthalten (vgl. z.B. [Pnueli 86]).

In der temporalen Logik geht man davon aus, daß es die Spezifikation und Verifikation vereinfacht, wenn man Abläufe (Zustandssequenzen) nicht syntaktisch repräsentiert, sondern nur unter Verwendung der temporalen Operatoren über sie spricht. Die Idee dabei ist, die temporalen Operatoren so zu wählen, daß sie gerade die für die Spezifikation verteilter, reaktiver Systeme relevanten zeitlichen Beziehungen wiedergeben, um so zu einem möglichst einfachen logischen Rahmen zu gelangen. Über die Frage, welche Kombination temporaler Operatoren am nützlichsten ist, bestehen unterschiedliche Meinungen (vgl. z.B. [Lamport 83b], [Lichtenstein et al. 85]).

Temporale Logik läßt sich vorteilhaft bei der Spezifikation von Systemen mit endlich vielen Zuständen einsetzen: solche Systeme können mit propositionaler temporaler Logik spezifiziert werden; mit sog. *Model-checking*-Algorithmen läßt sich dann zeigen, ob ein gegebenes Programm (mit endlich vielen Zuständen) eine solche Spezifikation erfüllt (siehe z.B. [Emerson 90]). Offen ist dagegen, ob und wie sich *Model-checking* für (Teilklassen) von Spurformeln einsetzen läßt.

Ein anderer Aspekt, der einen Unterschied zwischen temporaler Logik und dem Spurformalismus verdeutlicht, ist die Aktionsverfeinerung von Spezifikationen. Lamports temporale Logik ist so angelegt, daß Verfeinerung möglich ist (Verzicht auf den "next step"-Operator, siehe [Lamport 83b]); bei Spurformeln erscheint mir Aktionsverfeinerung schwierig, aber methodisch auch nicht angebracht: die Verfeinerung von Spezifikationen wird in FOCUS in der Entwurfsphase und den darauf folgenden Phasen durchgeführt ([Broy 91] beschreibt einen Ansatz der Aktionsverfeinerung auf der Entwurfsebene).

Ein Vorteil von Spurformeln (und allgemeiner: der Spurspezifikation) ist, daß modulare Spezifikationen genauso einfach zu formulieren sind wie globale Spezifikationen, Schnittstellen lassen sich elegant beschreiben. Dies ist in der temporalen Logik noch nicht in dieser Weise gelöst.

3.4 Transitionsorientierte Spezifikation

Ich erläutere zunächst in Abschnitt 3.4.1, was ich unter einem Transitionssystem verstehe und welche Spurmenge dadurch beschrieben wird, in Abschnitt 3.4.2 präsentiere ich eine tabellenorientierte Notation für Transitionssysteme, in Abschnitt 3.4.3 stelle ich die beweistechnische Verbindung zur ablauforientierten Spezifikation her.

Ich schildere die transitionsorientierte Spezifikation hier ausführlicher als die ablauforientierte, weil sie bei der Spurspezifikation in FOCUS bisher wenig berücksichtigt wurde. Neu ist insbesondere die Notation für Transitionssysteme.

3.4.1 Der Begriff des Transitionssystems

Transitionssysteme, auch Automaten, Zustandsmaschinen und Zustandsübergangssysteme genannt, sind in der Informatik weitverbreitet. Sie kommen dort in verschiedenen Ausprägungen vor, z.B. als endliche erkennende Automaten, Mealy-Automaten und Kellerautomaten. Wir wollen unter einem Transitionssystem ein Tupel

$$(A, State, —;>, Init),$$

verstehen, wobei:

A	eine *Aktionenmenge*,
$State$	eine (nicht notwendigerweise endliche) Menge von *Zuständen*,
$—;>$	eine Teilmenge von $State \times A \times State$ *(Übergangsrelation)* und
$Init$	eine Teilmenge von $State$ *(Anfangszustände)*

ist. Häufig wird $—;>$ eine (partielle) Funktion der Funktionalität $State \times A \to State$ sein. Auch gibt es oft genau *einen* Anfangszustand. Die Elemente von $—;>$ werden *Transitionen* oder *(Zustands-)Übergänge* genannt. Für $(s,a,s') \in —;>$ schreibe ich $s —;^{a}> s'$.

Ein Transitionssystem TS beschreibt ein Prädikat über Spuren, das informell folgendermaßen charakterisiert ist:

$P_{TS}(t) \equiv$ "alle endlichen Präfixe von t liegen auf einem Pfad von TS"

Um dies zu formalisieren, definiere ich zunächst (in üblicher Weise) die erweiterte Übergangsrelation und die Pfade (Berechnungen) eines Transitionssystems.

Die auf endliche Spuren erweiterte Übergangsrelation, eine Teilmenge von $State \times A^* \times State$, bezeichne ich ebenfalls mit $—;>$; sie ist folgendermaßen definiert (seien $s, s', r \in State$, $a \in A$, $t \in A^*$):

$$s —\!—;^{\varepsilon}> s$$
$$s —\!—;^{a^\frown t}> s' \Leftrightarrow \exists r \in State: s —\!—;^{a}> r \ \wedge \ r —\!—;^{t}> s'$$

$s —;^{t}> r$ gilt also genau dann, wenn es einen endlichen Pfad von s zu r gibt, und die Konkatenation der Aktionen auf dem Pfad t ergibt. Dabei ist ein *Pfad* eine endliche oder unendliche Folge von Zuständen s_i und Aktionen a_i der Art

$$\langle s_0,a_1,s_1,a_2,\ldots,s_i,a_{i+1},\ldots\rangle,$$

wobei $s_0 \in \text{Init}$ gilt (d.h. s_0 ist ein Anfangszustand) und $s_i \text{ —;}^{a_i}\text{> } s_{i+1}$ gilt für alle i. Eine endliche Folge wird mit einem Zustand abgeschlossen.

Ein Transitionssystem TS repräsentiert das Prädikat (über Spuren aus Act^ω):

$$P_{TS}(t) \equiv \forall u \in A^\omega: u = A \copyright t \Rightarrow (\forall v \in A^*: v \sqsubseteq u \Rightarrow \exists s,s' \in \text{State}: s \in \text{Init} \wedge s \text{ —;}^{v}\text{> } s') \quad (*)$$

Zur Erläuterung dieser Formel sei angemerkt, daß im Transitionssystem nur über Aktionen aus A gesprochen wird, wobei A eine Teilmenge der in der globalen Spezifikation betrachteten Aktionenmenge Act sei. Es ist nämlich manchmal sinnvoll, mit einem Transitionssystem lediglich Reihenfolgebeziehungen für einen *Teil* der im verteilten System auftretenden Aktionen festzulegen. Da t aus Act^ω stammt, ist die Prämisse $u = A \copyright t$ nötig. Falls $A = \text{Act}$ ist, vereinfacht sich die Formel zu:

$$P_{TS}(t) \equiv \forall v \in \text{Act}^*: v \sqsubseteq t \Rightarrow \exists s,s' \in \text{State}: s \in \text{Init} \wedge s \text{ —;}^{v}\text{> } s'$$

Für $P_{TS}(t)$ schreibe ich auch $t \in \text{Traces(TS)}$ oder t **is trace of** TS.

Gemäß (*) wird durch ein Transitionssystem lediglich eine Sicherheitsanforderung festgelegt (vgl. Abschnitt 3.5.1).

Der Unterschied zwischen den hier betrachteten Transitionssystemen und den wohlbekannten endlichen, erkennenden Automaten besteht darin, daß

1. die Zustandsmenge nicht notwendigerweise endlich ist,
2. keine Endzustandsmenge angegeben ist. Auf diese Weise werden aber keine Lebendigkeitsanforderungen ausgedrückt[1].

Den ersten Punkt halte ich für wichtig, da bei Anforderungsspezifikationen beliebig große Zustandsräume auftreten können. In verschiedenen transitionsorientierten Ansätzen, etwa in der Sprache Estelle zur Protokollbeschreibung ([Hogrefe 89]), stellt man Transitionssysteme durch parametrisierte endliche Automaten dar. Die Zustände können Variablen enthalten, die beliebig große Werte annehmen können. Somit handelt es sich auch dort um unendliche Automaten und die Einführung von Variablen dient nur der Strukturierung.

Der zweite Punkt impliziert, daß Lebendigkeitsanforderungen auf andere Weise anzugeben sind. Ich verwende dafür allein Spurformeln. In der Literatur finden sich auch Ansätze, die Lebendigkeitsanforderungen in ein Transitionssystem integrieren (vgl. z.B. [Pnueli 86], [Jonsson 87], [Chandy, Misra 88], [Thomas 90]); meine Entscheidung, dies nicht zu tun, hat folgende Gründe:

a) Nach meiner Erfahrung lassen sich Lebendigkeitseigenschaften leicht ablauforientiert formulieren. Dies liegt m.E. daran, daß sich Lebendigkeitsanforderungen immer an einen *gesamten* Ablauf richten. Der Nutzen des transitionsorientierten Spezifikationsstils macht sich

---

[1] bzw. nur die triviale Lebendigkeitsanforderung "jede Spur ist lebendig": $P(t) \equiv \text{true}$

allein bei Sicherheitsanforderungen bemerkbar: es wird beschrieben, wie das System von einem erlaubten ("sicheren") Zustand zum nächsten übergeht.

b) Lebendigkeitsanforderungen werden für Transitionssysteme häufig durch Fairneßbedingungen ausgedrückt und damit in "standardisierter" Form behandelt. Diese Standardisierung ist manchmal zu unflexibel und führt zum Teil zu nicht vorhergesehenen Effekten (siehe [Broy et al. 91a]).

Es sei auf die Gefahr hingewiesen, daß durch das Einführen von Zuständen schon gewisse Entwurfsentscheidungen gefällt bzw. nahegelegt werden können *(overspecification)*. Deshalb sollten die Zustände so "abstrakt" wie möglich gewählt werden. (Für das Spezifizieren mit abstrakten Zuständen verweise ich auf die Spezifikationssprache VDM, vgl. [Bjørner, Jones 82].)

3.4.2 Eine tabellenorientierte Notation für Transitionssysteme

Bisher wurde lediglich der Begriff des Transitionssystems eingeführt. Zur methodischen Verwendung von Transitionssystemen ist es jedoch auch erforderlich, komfortable Sprachmittel zu deren Spezifikation bereitzustellen. Ein solches ist die tabellenorientierte Notation für Transitionssysteme, die ich nun vorstelle.

1. Die Notation im Überblick

Diese Notation hat zwei Grundlagen:

- die algebraische Spezifikation von Datentypen und
- die relationale Notation nach [Lam, Shankar 90] und - verwandt damit - die Notation nach [Lamport 83a].

Algebraisch spezifiziert werden die Sorten, Funktionssymbole und Prädikatsymbole, die bei der Spezifikation eines Transitionssystems verwendet werden.

Ebenso wie in [Lam, Shankar 90] werden Zustandsmengen sowie Mengen gleichartiger Übergänge durch Prädikate ("relational") beschrieben. Unterschiede zu [Lam, Shankar 90] bestehen in der Abstützung auf die algebraische Spezifikation, der flexiblen Einführung und Handhabung von Zustandskomponenten (es lassen sich verschiedene Zustandsselektorsorten einführen; über Variable einer Selektorsorte kann quantifiziert werden; Aktionen können Zustandsselektoren als Parameter enthalten) und einer anderen Syntax, bei der Übergänge tabellenorientiert notiert werden. Zudem ist hier die formale Semantik einer Spezifikation eines Transitionssystems präzise aufgeschrieben. Dies wird in [Lam, Shankar 90] nicht behandelt, in [Lamport 83a] findet sich eine Übersetzung der dort verwendeten Transitionssystem-Notation in temporale Logik. Dagegen gebe ich als Semantik direkt ein Transitionssystem an. Eine Übersetzung in Spurformeln (analog zu [Lamport 83a]) ist möglich, bringt jedoch keinen methodischen Vorteil: Die semantische Verbindung zu Spurformeln ergibt sich über die Spurmenge, die ein Transitionssystem beschreibt; die beweistechnische Verbindung erfordert an Transitionssysteme angelehnte Beweistechniken (vgl. Abschnitt 3.4.3)

Ich will zunächst die Idee vermitteln, die der Notation zugrundeliegt. Dabei verzichte ich auf die Darstellung notationeller Besonderheiten, die dem leichteren Erstellen und der leichteren Lesbarkeit einer Spezifikation dienen. Eine präzise Darstellung der Notation findet sich im nächsten Teilabschnitt.

Der Zustandsraum eines Transitionssystems ist die Menge aller Abbildungen der Funktionalität Sel → Val, wobei Sel (Zustandsselektoren, Bezeichner für Zustandskomponenten) und Val (Werte) frei gewählte Mengen sind. Die Menge der Anfangszustände wird durch ein Prädikat Init(s) über Zuständen angegeben (s stehe für einen Zustand, also für eine Abbildung aus Sel → Val). Die Übergangsrelation wird durch einige *Übergangsformeln*, d.h. Formeln der Bauart

from P(r) **with** a **to** Q(r,s) (*)

spezifiziert, die Mengen von Zustandsübergängen als Wert haben. P sei ein Prädikat über Zuständen, Q ein Prädikat über Paaren von Zuständen, a eine Aktion. r und s stehen für den Zustand vor bzw. nach der Ausführung eines Übergangs. Statt der Schlüsselworte **from**, **with** und **to** kann auch eine dreispaltige Tabelle verwendet werden (vgl. das Postfachbeispiel weiter unten). Auf diese Weise ergibt sich eine Anknüpfung an die weitverbreiteten Automatentafeln.

(*) beschreibt die Menge von Zustandsübergängen

$$\{\ r \text{—};^{a}\!> s \mid P(r) \wedge Q(r,s)\ \}$$

Die Anfangszustände und die Übergangsrelation werden also auf schematische Weise durch Formeln beschrieben. Die Schematisierung setzen wir noch fort: Wir erlauben auch, daß P, a und Q von Parametern abhängen können:

from P(r; x) **with** a(x) **to** Q(r,s; x) (**)

x sei ein Parameter, genauer gesagt eine getypte Variable; natürlich können auch mehrere Parameter verwendet werden.

(**) beschreibt die Menge von Zustandsübergängen

$$\{\ r \text{——};^{a(p)}\!> s \mid P(r; p) \wedge Q(r,s; p) \wedge p \in \mathrm{Par}\}$$

Par sei die Menge aller Werte, die x annehmen kann.

Betrachten wir ein Beispiel für die Spezifikation eines Transitionssystems in unserer Notation. Hier erscheinen nun auch Schreiberleichterungen, von denen ich oben noch abstrahiert habe. Insbesondere wird die Tabellennotation für Übergangsformeln verwendet. An dieser Stelle will ich nur einen Eindruck von der Gestalt einer derartigen Spezifikation vermitteln, die einzelnen Bestandteile der Spezifikation werden anschließend genau erklärt. Anmerkungen erscheinen in der Spezifikation in geschweiften Klammern.

Beispiel (Postfachsystem: Transitionssystem): Manche Eigenschaften unseres Postfachsystems lassen sich leicht durch ein Transitionssystem wiedergeben. Die Zustandskomponenten sind die Inhalte der Postfächer und eine Warteschlange noch nicht abgesandter Rückmeldungen.

transition system MAILBOX *{Name des Transitionssystems}*

based on MBX, ACTION *{Namen algebraisch spezifizierter Datentypen, auf die sich die Spezifikation des Transitionssystems stützt}*

state components *{Vereinbarung der Zustandskomponenten}*
Mbx : Msg^*,
acks : Act^*

actions Act *{Sorte der Aktionen; sie muß aus den unter* **based on** *angegebenen Typen stammen, d.h. es darf keine neue Sorte eingeführt werden}*

initially $'acks = \varepsilon \wedge \forall mb \in Mbx: 'mb = \varepsilon$
{ein Prädikat, das die Anfangszustände spezifiziert}

transitions *{die Übergangsrelation wird durch mehrere sog. Übergangsformeln spezifiziert}*

variables $mb \in Mbx$, $ts \in Tsk$, $ms \in Msg$, $sg \in Sig$

from	**with**	**to**
¬full('mb)	snd(ts,mb,ms)	$mb' = 'mb \circ ms \wedge$ $acks' = 'acks \circ$ s-ack(ts,mb,ok) $\wedge$ $\forall m \in Mbx: m \neq mb \Rightarrow m' = 'm$
full('mb)	snd(ts,mb,ms)	$acks' = 'acks \circ$ s-ack(ts,mb,error) $\wedge$ $\forall m \in Mbx: m' = 'm$
¬empty('mb)	rec(ts,mb)	$mb' =$ rt('mb) $\wedge$ $acks' = 'acks \circ$ r-ack(ts,mb,ft('mb),ok) $\wedge$ $\forall m \in Mbx: m \neq mb \Rightarrow m' = 'm$
empty('mb)	rec(ts,mb)	$acks' = 'acks \circ$ r-ack(ts,mb,nil,error) $\wedge$ $\forall m \in Mbx: m' = 'm$
¬empty('acks)	ft('acks)	$acks' =$ rt('acks) $\wedge$ $\forall m \in Mbx: m' = 'm$

end

Der Typ ACTION ist in Abschnitt 3.1 beschrieben.

Tatsächlich erscheinen in acks nur Rückmeldeaktionen, nicht beliebige Aktionen. full('mb) gilt genau dann, wenn #'mb gleich der maximalen Kapazität des Postfachs mb ist (dies sei eine vordefinierte natürliche Zahl). Analog gilt: empty('mb) $\Leftrightarrow$ #'mb = 0. nil bezeichnet eine fehlerhafte bzw. Dummy-Nachricht.

Die erste Übergangsformel bedeutet informell gesprochen: Von einem Zustand aus, bei dem ein Postfach mb nicht voll ist, kommt man mit der Aktion snd(ts,mb,ms) zu einem neuen Zustand, bei dem die Nachricht ms in dieses Postfach geschrieben wurde und eine entsprechende Rückmeldung fällig ist. □

2. Die Notation im Detail

Sehen wir uns nun genauer an, wie die Bestandteile

- Zustandskomponenten,
- Aktionen,
- Übergangsrelation und
- Anfangszustände

eines Transitionssystems spezifiziert werden.

a) Zustandskomponenten

Die Transitionssysteme, die wir betrachten wollen, haben *strukturierte Zustände*, d.h. jeder Zustand ist aus mehreren *Zustandskomponenten* aufgebaut. Formal läßt sich ein strukturierter Zustand als eine Abbildung einer Menge *Sel* von *(Zustands-)Selektoren*, d.h. Namen für die Zustandskomponenten, in eine Menge *Val* von Werten auffassen. Der (semantische) Zustandsraum ist damit gegeben als $Sel \rightarrow Val$, wobei noch eine Typbedingung gelten muß (s.u.). Semantische Objekte wie die Menge *Sel* werden hier und im folgenden kursiv gedruckt, syntaktische Objekte wie die Sorte Sel_1 (siehe weiter unten) in Standardschrift.

Zur Beschreibung des Zustandsraums dient eine Sequenz der Art

Sel_1 : Val_1,
.....
Sel_n : Val_n

$Sel_1,...,Sel_n$ und $Val_1,...,Val_n$ müssen Sorten sein, die in den unter **based on** eingeführten Datentypen definiert sind. Die Sorten $Sel_1,...,Sel_n$ seien verschieden; wir nennen sie *Selektorsorten*, $Val_1,...,Val_n$ nennen wir *Wertesorten*. Terme einer Selektorsorte werden *Selektorterme* genannt. Wir erkennen die Ähnlichkeit zur Deklaration von Programmvariablen in höheren Programmiersprachen, bemerken aber zugleich einen Unterschied: $Sel_1,...,Sel_n$ sind Sorten, d.h. Mengen von Selektoren werden vereinbart, nicht nur einzelne Programmvariable.

Zur Schreibvereinfachung ist es auch erlaubt, ein neues Symbol sel als Selektorkonstante einzuführen und ihm eine Sorte Val für den Wertebereich zuzuordnen. Dies entspricht der Deklaration einer Programmvariablen in einer höheren imperativen Programmiersprache. Das Symbol sel wird ausschließlich in der Spezifikation des Transitionssystems verwendet. Die Verwendung der Schreibweise sel : Val kürzt die folgenden drei Schritte ab:

- Einführung eines neuen Datentyps (sei SEL ein beliebiger, sonst nicht verwendeter Name)

spec SEL
sort Sel
sel : Sel
end

- Abstützen auf SEL in der Spezifikation des Transitionssystems (**based on** SEL, ...)
- Zuordnung Sel : Val bei der Deklaration der Zustandskomponenten.

Auf den konkreten Namen Sel nehmen wir bei der Spezifikation des Transitionssystems nie Bezug.

Wesentlich für diese Notation ist, daß wir nicht explizit eine Sorte State einführen, sondern lediglich über Zustandsselektoren auf Zustände Bezug nehmen.

Zur Definition der Semantik des Transitionssystems gehen wir aus von einer Interpretation I der unter **based on** eingeführten Datentypen. I ist eine Abbildung, die jeder Sorte eine Trägermenge zuordnet, jedem Funktionssymbol eine Funktion und jedem Prädikatsymbol ein Prädikat (jeweils der entsprechenden Funktionalität). Die Selektormenge Sel ist definiert als

$$Sel = I(\mathrm{Sel}_1) \cup ... \cup I(\mathrm{Sel}_n),$$

die Wertemenge Val als

$$Val = I(\mathrm{Val}_1) \cup ... \cup I(\mathrm{Val}_n).$$

Der Zustandsraum besteht aus der Menge aller Funktionen $s \in Sel \rightarrow Val$ mit $sel \in I(\mathrm{Sel}_k) \Rightarrow s(sel) \in I(\mathrm{Val}_k)$ für alle $k \in \{1,...,n\}$. Der Zustandsraum ist damit isomorph zu

$$(I(\mathrm{Sel}_1) \rightarrow I(\mathrm{Val}_1)) \times ... \times (I(\mathrm{Sel}_n) \rightarrow I(\mathrm{Val}_n)),$$

wenn die $I(\mathrm{Sel}_i)$ disjunkt sind.

Beispiel (Vereinbarung von Zustandskomponenten):

Mbx : Msg^* {eine Selektormenge $Mbx = I(\mathrm{Mbx})$ (mailboxes) wird eingeführt; der Wertebereich jedes Elements aus Mbx ist $Msg^* = I(\mathrm{Msg}^*)$ (endliche Sequenzen von Nachrichten)}

acks : Act^* {ein Selektor $acks = I(\mathrm{acks})$ wird eingeführt; acks (Rückmeldungen) ist von einer anonymen Sorte mit Trägermenge $\{acks\}$; der Wertebereich von $acks$ ist $Act^* = I(\mathrm{Act}^*)$} □

Man beachte, daß bei acks die erwähnte Schreibvereinfachung verwendet wird.

b) Aktionen

Durch

actions Act

mit einer Sorte Act, die in den unter **based on** eingeführten Datentypen definiert ist, wird die Aktionenmenge $I(\mathrm{Act})$ spezifiziert.

Parameter einer Aktion können auch einer Selektorsorte entstammen:

Beispiel: Bei

snd: Tsk × Mbx × Msg → Act

(vgl. Abschnitt 3.1) ist Mbx eine Selektorsorte (siehe obiges Beispiel). □

c) Übergangsrelation

Die Übergangsrelation wird durch mehrere Formeln der Bauart

from P **with** a **to** Q (*)

spezifiziert *(Übergangsformeln)*; als alternative Syntax für Übergangsformeln dient, wie im Postfachbeispiel demonstriert, die Tabellennotation. Hierbei seien P und Q erweiterte prädikatenlogische Formeln (s.u.) und a ein Term einer unter **actions** aufgeführten Sorte. Grob gesprochen bedeutet (*), daß man von einem Zustand, der P erfüllt, mit einer Aktion a zu einem neuen Zustand gelangen kann, der in einer durch Q festgelegten Beziehung zum alten Zustand steht. Die präzise Semantik von (*) findet sich weiter unten.

Im Gegensatz zu üblichen prädikatenlogischen Formeln dürfen in P und Q auch *dekorierte* Selektorterme vorkommen. Es gibt davon zwei Arten (t sei ein Selektorterm):

- Apostroph vorne: 't
- Apostroph hinten: t'

Die Kombination von Dekorationen ist verboten. In P und a dürfen keine Terme der Form t' vorkommen. Die Sorte der dekorierten Terme 't und t' ist die *Werte*sorte von t. Daher ist z.B. 'mb0 = ε eine wohlgeformte Formel, wenn mb0 ein Grundterm der Sorte Mbx ist, weil 'mb0 ein dekorierter Selektorterm der Sorte Msg^* ist.

Diese syntaktischen Anforderungen sind darin begründet, daß bei der Semantik der Formeln P und Q dekorierte Selektorterme *dereferenziert* werden, d.h. der Wert der entsprechenden Zustandskomponente interessiert, nicht deren Name. Ein Selektorterm 't steht für den Wert der entsprechenden Zustandskomponente *vor* der Ausführung eines Zustandsübergangs, ein Selektorterm t' für deren Wert *nach* der Ausführung des Übergangs. Daß in P und a keine Terme der Form t' vorkommen dürfen, ist dadurch motiviert, daß die Anwendbarkeit einer Aktion in einem Zustand nicht vom Folgezustand abhängig sein soll.

Die Semantik einer Übergangsformel formalisiert diese intuitiven Vorstellungen. Sie läßt sich knapp formulieren, wenn wir unsere Schreibvereinfachung in gewisser Weise rückgängig machen und die Übergangsformeln in reine Prädikatenlogik umsetzen. Hierzu führen wir die Funktionssorte State über einen geeigneten Sortenausdruck explizit ein.[1]

---

[1] Die Einführung der Sorte State ist nicht nötig, vereinfacht aber die Beschreibung der Semantik.

Für eine Übergangsformel **from** P **with** a **to** Q geht die Formel $P;\tilde{}(r)$ aus P dadurch hervor, daß alle Selektorterme der Art 't in P durch r(t) ersetzt werden; r sei eine (neue) Variable der Funktionssorte State. Analog sei $a;\tilde{}(r)$ definiert. Die Formel $Q;\tilde{}(r,s)$ gehe aus Q dadurch hervor, daß außerdem alle Selektorterme der Art t' durch s(t) ersetzt werden; s sei ebenfalls eine (neue) Variable der Funktionssorte State.

Wir erweitern den Definitionsbereich einer Interpretationsfunktion auf Übergangsformeln: Eine Interpretationsfunktion *I* bilde jede wohlgeformte Übergangsformel auf ein Prädikat aus $State \times Act \times State \rightarrow Bool$ ab. Die Semantik einer Übergangsformel ist wie folgt definiert: Seien *p* und *q* Zustände und *a* eine Aktion (*p,q* und *a* sind Objekte der *semantischen* Ebene).

I (**from** P **with** a **to** Q) (*p,a,q*) = *true* genau dann, wenn gilt:

Es gibt eine Belegung σ, so daß

$I(\ P;\tilde{}(r) \wedge\ v = a;\tilde{}(r)\ \wedge\ Q;\tilde{}(r,s)\ )\ (\sigma[p/r, q/s, a/v]) = true$

v sei eine neue Variable der Sorte Act. $I(R)(\sigma)$, der Wert der Formel R bei der Interpretation *I* und der Belegung σ der logischen Variablen, ist wie üblich in der Prädikatenlogik definiert (vgl. [Loeckx, Sieber 87]). Die Belegung $\sigma[d/x]$ ist definiert durch:

$$(\sigma[d/x])(y) = \begin{cases} d & \text{falls } x = y; \\ \sigma(y) & \text{sonst} \end{cases}$$

Die Erweiterung auf die Überschreibung mehrerer Werte ist offensichtlich.

I (**from** ...) beschreibt ein Prädikat auf Zustandsübergängen, damit eine Menge von Zustandsübergängen. Die Übergangsrelation wird als das logische Oder aller dieser Prädikate definiert, mengenmäßig gesehen also als die Vereinigung der entsprechenden Mengen.

Um die Spezifikation des Transitionssystems übersichtlich zu gestalten, sollte Q so beschaffen sein, daß es keinen Beitrag zur Anwendbarkeitsbedingung eines Zustandsübergangs leistet. Dies ist dann der Fall, wenn gilt:

$$P \Rightarrow \exists v_1 \in V_1,...,v_n \in V_n\colon Q[v_1/t_1',...,v_n/t_n']$$

wobei $t_1',...,t_n'$ die (dekorierten) Terme einer Selektorsorte sind, die in Q vorkommen. V_i ist die Wertesorte von t_i. $Q[v_1/t_1',...,v_n/t_n']$ bezeichnet die simultane Substitution der t_i' durch die v_i.

Beispiel (Übergänge): Formal beschreibt die erste Übergangsformel im Postfachbeispiel

from ¬full('mb) **with** snd(ts,mb,ms)
to mb' = 'mb ◦ ms ∧ acks' = 'acks ◦ s-ack(ts,mb,ok) ∧ ∀m∈ Mbx: m ≠ mb ⇒ m' = 'm

die Menge der Übergänge $p \xrightarrow{a} q$, für die gilt:

Es gibt eine Belegung σ der logischen Variablen, so daß gilt:

$\neg\ full(p(\sigma(mb))$

und $a = snd(\sigma(ts),\sigma(mb),\sigma(ms))$

und $q(\sigma(\text{mb})) = p(\sigma(\text{mb})) \circ \sigma(\text{ms})$
und $q(acks) = p(acks) \circ s\text{-}ack(\sigma(\text{ts}),\sigma(\text{mb}),ok)$
und $q(m) = p(m)$ für m aus $I(\text{Mbx})$ mit $m \neq \sigma(\text{mb})$ □

d) Anfangszustände

Die Anfangszustände werden durch eine prädikatenlogische Formel spezifiziert, die keine Terme der Art t' enthält und implizit als allquantifiziert angenommen wird. Der Wert dieser Formel ist wie unter c) beschrieben festgelegt. Bei einer gegebenen Interpretation legt eine solche Formel eine Menge von (Anfangs)-Zuständen fest.

Beispiel (Anfangszustände): Die im Postfachbeispiel unter **initially** angegebene Formel

$$'acks = \varepsilon \wedge \forall mb \in Mbx: 'mb = \varepsilon$$

besagt, daß ein Zustand s ein Anfangszustand ist, wenn gilt:

$s(acks) = \varepsilon$ (keine Rückmeldungen stehen an)
und für alle mb aus $I(\text{Mbx})$ gilt: $s(mb) = \varepsilon$ (alle Postfächer sind leer)

Im Postfachbeispiel gibt es somit genau einen Anfangszustand. □

3.4.3 Beweistechnische Verbindung zur ablauforientierten Spezifikation

Ein wichtiges Instrument der Validation von Anforderungsspezifikationen ist ihre Analyse, d.h. die Feststellung und der Nachweis spezifischer Eigenschaften. Sind die Anforderungen mit Spurformeln formuliert, so bieten sich offensichtlich logische Techniken an, die sich in einem Kalkül formalisieren lassen. Wie aber lassen sich ablauforientierte Eigenschaften eines Transitionssystems in unserer Notation nachweisen? Gesucht ist also eine beweistechnische Verbindung zwischen dem transitions- und dem ablauforientierten Spezifikationsstil.

Ich zeige hier, daß sich die klassische *Invariantentechnik* auf unseren Formalismus übertragen läßt. Ich stütze mich hierbei auf eine Version dieser Technik, die z.B. in [Lamport, Lynch 90] erläutert wird; sie gibt an, wie sich (z.B. mit temporaler Logik formulierte) Eigenschaften von Transitionssystemen nachweisen lassen.

Wir wenden diese Technik in der folgenden Situation an: Uns liegt eine Beschreibung eines Transitionssystems TS in unserer Notation vor und wir vermuten, daß eine Sicherheitseigenschaft S für die Spuren des Transitionssystems gilt, d.h. daß $t \in \text{Traces(TS)} \Rightarrow S(t)$ gilt. Dies wollen wir mit unserer Invariantentechnik auf schematische Weise zeigen. Ich beschränke mich auf Sicherheitseigenschaften, denn nur solche Eigenschaften formuliere ich mit Transitionssystemen.

Zunächst schildere ich die Beweisidee und die auszuführenden Schritte, ohne auf unsere Notation einzugehen. Anschließend zeige ich, wie die Beweisschritte abgestimmt auf unsere Notation aussehen.

Die Idee ist, eine Formel Inv *(Invariante)* zu finden, die von einem Zustand und einer Spur abhängt und für die gilt[1]:

1) Die Invariante gilt in jedem Anfangszustand:
   Inv(init,ε) für alle Anfangszustände init
2) Jeder mögliche Übergang erhält die Invariante:
   $(Inv(r,t) \wedge r \text{ —;}^{a}\text{>} s) \Rightarrow Inv(s,t \circ a)$ für alle Zustände r und s und jede Spur t
3) Die Invariante impliziert die zu beweisende Formel S:
   $Inv(r,t) \Rightarrow S(t)$ für jeden Zustand r und jede Spur t

Findet man eine Invariante, die 1 - 3 für ein Transitionssystem TS erfüllt, so gilt offensichtlich:

$t \in Traces(TS) \Rightarrow S(t)$ (*)

Prinzipiell läßt sich für jedes S, für das (*) gilt, eine Invariante finden, so daß sich (*) mit der Invariantentechnik zeigen läßt (nämlich $Inv(r,t) \equiv \exists s \in Init: s \text{ —;}^{t}\text{>} r$); um einen möglichst einfachen Beweis führen zu können, müssen problemangepaßte Invarianten gefunden werden, was nicht immer einfach ist.

Für unsere Notation lassen sich diese Schritte in der folgenden Weise anpassen bzw. handhabbar machen: Wir führen eine neue Zustandskomponente t mit Wertebereich Act^* im Transitionssystem ein. Im folgenden sei Inv eine Formel, in der keine Terme der Art q' vorkommen. Der Wert dieser Formel hängt somit lediglich von einem um die Komponente t erweiterten Zustand ab. 1 - 3 lassen sich für unsere Notation in die folgenden Beweisschritte umsetzen:

1') Init $\wedge$ 't = ε $\Rightarrow$ Inv wobei Init die bei **initially** angegebene Formel ist
2') Für jede Übergangsformel **from** P **with** a **to** Q ist zu zeigen:
   $(Inv \wedge P \wedge Q \wedge t' = 't \circ a) \Rightarrow Inv'$
3') $Inv \Rightarrow S('t)$

Inv' definieren wir als diejenige Formel, die sich aus Inv durch Umbenennung von Termen der Art 'q in q' ergibt; dies bedeutet, daß Inv' im Zustand *nach* der Ausführung des Übergangs ausgewertet wird. In 1' und 3' wird jeweils nur über einen Zustand eine Aussage gemacht. Offensichtlich entsprechen 1' - 3' den Bedingungen 1 - 3.

Beim Postfachsystem läßt sich diese Beweistechnik auf die folgende Weise anwenden:

Beispiel: Wir wollen zeigen: Die Spuren des Postfach-Transitionssystems erfüllen die Anforderung

---

[1] Inv (bzw. $\exists s: Inv(s,t)$) muß zwar formal keine Sicherheitseigenschaft sein, man sieht jedoch leicht, daß Inv als Sicherheitseigenschaft gewählt werden *kann.*

S-ack_safe(t) ≡ ∀s∈ Act*: s ⊑ t ⇒ #({snd(ts,mb,ms) | ms∈ Msg} © s)
≥ #({s-ack(ts,mb,sg) | sg∈ Sig} © s)

Wir geben eine Invariante Inv an

Inv ≡ ∀tk∈ Tsk,m∈ Mbx:
sum_snd(tk,m,'t) = sum_s-ack(tk,m,'t) + sum_acks(tk,m,'acks),

wobei

sum_snd(tk,m,'t) = #({snd(tk,m,ms) | ms∈ Msg} © 't)
"Summe der bisherigen Sendeaktionen von Prozeß tk zu Postfach mb bei Wert der Zustandskomponente 't ",
sum_s-ack(tk,m,'t) = #({s-ack(tk,m,sg) | sg∈ Sig} © 't)
"Summe der bisherigen Senderückmeldungen",
sum_acks(tk,m,'acks) = #({s-ack(tk,m,sg) | sg∈ Sig} © 'acks)
"Summe der anstehenden Senderückmeldungen"

Prüfen wir nun 1' bis 3':

1': Anfangs gilt: sum_snd(tk,m,ε) = 0, aber auch:
sum_s-ack(tk,m,ε) = 0 und
sum_acks(tk,m,'acks) = sum_acks(tk,m,ε) = 0

2': Betrachten wir zunächst die erste Übergangsformel. Wir müssen zeigen:

((∀tk∈ Tsk, m∈ Mbx: sum_snd(tk,m,'t) = sum_s-ack(tk,m,'t) + sum_acks(tk,m,'acks))
∧ ¬full('mb) ∧ mb' = 'mb ◦ ms ∧ acks' = 'acks ◦ s-ack(ts,mb,ok)
∧ ∀m∈ Mbx: m ≠ mb ⇒ m' = 'm
∧ t' = 't ◦ snd(ts,mb,ms))
⇒ (∀tk∈ Tsk,m∈ Mbx: sum_snd(tk,m,t') = sum_s-ack(tk,m,t') + sum_acks(tk,m,acks'))

Wir sehen:

sum_snd(tk,m,t') = \B\LC\{(\A\AL\CO1(sum_snd(tk,m,'t) + 1 falls tk = ts ∧ m = mb; sum_snd(tk,m,'t) sonst)),

sum_s-ack(tk,m,t') = sum_s-ack(tk,m,'t),

sum_acks(tk,m,acks') = \B\LC\{(\A\AL\CO1(sum_acks(tk,m,'acks) + 1 falls tk = ts ∧ m = mb ; sum_acks(tk,m,'acks) sonst)),

Damit stimmt die Bilanz auch für t'.

Bei den übrigen Übergangsformeln zeigen wir den Erhalt der Invariante auf die gleiche Weise.

3': Da sum_acks(ts,mb,'acks) ≥ 0, gilt mit Inv insbesondere
sum_snd(ts,mb,'t) ≤ sum_s-ack(ts,mb,'t) = S-ack_safe('t). □

3.5 Strukturierung in Sicherheits- und Lebendigkeitsanforderungen

In den letzten Abschnitten wurden die Begriffe "Sicherheit" und "Lebendigkeit" bereits erwähnt, aber - wie in vielen anderen Arbeiten - nicht formal definiert. Dies hole ich nun nach: ich zitiere (bekannte) Definitionen dieser Begriffe, erläutere ihren Charakter und untersuche die Kombination von Sicherheits- und Lebendigkeitsanforderungen. Die Ergebnisse dieser Untersuchungen bilden die Grundlage für die Strukturierung einer Spezifikation in Sicherheits- und Lebendigkeitsanforderungen, eine Strukturierung, die auch in anderen Ansätzen verfolgt wird (vgl. [Lamport 89], [Li, Maibaum 88]).

3.5.1 Begriffsbestimmung

Informell gesprochen schließen *Sicherheitsanforderungen* (*safety conditions*) das Auftreten unerwünschter (nach endlichen Teilabläufen beobachtbarer) Effekte aus, im Englischen oft umschrieben mit "nothing bad does happen". *Lebendigkeitsanforderungen* (*liveness conditions*) stellen dagegen sicher, daß ein bestimmter Zustand oder eine bestimmte Aktion schließlich eintritt ("something good happens eventually"). Eine präzisere Charakterisierung ist die folgende: Ob eine Sicherheitsanforderung verletzt ist, ist nach jedem endlichen Teilablauf feststellbar, ob eine Lebendigkeitsanforderung verletzt ist dagegen erst bei Kenntnis des gesamten, evtl. unendlichen Ablaufs.

Statt von einer Sicherheits*anforderung* sprechen wir auch oft - je nach dem Verwendungszweck - von einer Sicherheits*eigenschaft* oder einem Sicherheits*prädikat*: von einer Eigenschaft, wenn allgemein die Beschreibung eines (evtl. sogar existierenden) Systems im Vordergrund steht, von einem Prädikat, wenn wir an einer formalen Charakterisierung interessiert sind. Gleiches gilt für den Begriff der Lebendigkeit.

Formale Definitionen dieser Begriffe finden sich in [Alpern, Schneider 85]. Zwar beziehen sich diese Definitionen auf den Formalismus der (unendlichen) Zustandssequenzen, doch sie lassen sich leicht auf Aktionssequenzen übertragen.

Von einem Sicherheitsprädikat wird gefordert, daß ein Ablauf genau dann sicher ist (d.h. er erfüllt das Sicherheitsprädikat), wenn alle seine endlichen Teilabläufe sicher sind.

Definition (Sicherheit): P ist ein *Sicherheitsprädikat*, wenn gilt:

$$\forall t \in Act^{\omega}: (P(t) \Leftrightarrow \forall s \in Act^{*}: s \sqsubseteq t \Rightarrow P(s)) \qquad (1)$$ □

Offensichtlich ist *false* das kleinste Sicherheitsprädikat, $P(t) \equiv t = \varepsilon$ das nächstgrößere und *true* das größte Sicherheitsprädikat.

Von einem Lebendigkeitsprädikat wird gefordert, daß jeder endliche Ablauf zu einem lebendigen erweitert werden kann.

Definition (Lebendigkeit): P ist ein *Lebendigkeitsprädikat*, wenn gilt:

$$\forall s \in Act^{*}: \exists t \in Act^{\omega}: P(s \circ t) \qquad (2)$$ □

true ist das größte Lebendigkeitsprädikat, es gibt jedoch kein kleinstes, insbesondere gibt es keines, das kleiner als die Lebendigkeitsprädikate $P_1(t) \equiv \#t = \infty$ und $P_2(t) \equiv \#t \neq \infty$ ist.

Sicherheits- und Lebendigkeitseigenschaften haben einen qualitativ unterschiedlichen Charakter, weswegen sich z.B. Beweismethoden für die beiden Arten von Eigenschaften unterscheiden (vgl. [Lamport 89]). Daher ist sinnvoll, die Anforderungen in Sicherheits- und Lebendigkeitsanforderungen zu strukturieren, wenn Eigenschaften einer Spezifikation nachgewiesen oder verschiedene Spezifikationen zueinander in Beziehung gesetzt werden sollen. Einen Schritt in die Richtung einer strukturierten Darstellung der Anforderungen sind wir bereits bei der Wahl unserer Sprachmittel gegangen: Transitionssysteme beschreiben in unserem Formalismus allein Sicherheitseigenschaften; eine mit Spurformeln formulierte Anforderung kann zwar sowohl einen Sicherheits- als auch einen Lebendigkeitsanteil enthalten, wie wir sehen werden, lassen sich jedoch auch solche Anforderungen immer schematisch in reine Sicherheits- und Lebendigkeitsanforderungen aufspalten.

Neben dem Nutzen bei der Verifikation bietet die Strukturierung einen pragmatischen Vorteil: Fallstudien zur Spurspezifikation zeigen, daß eine Spezifikation durchsichtiger ist, wenn Sicherheits- und Lebendigkeitsanforderungen getrennt spezifiziert sind. Trotzdem möchte ich das getrennte Spezifizieren dieser Anforderungen im Sinne einer großen Flexibilität bei der Anforderungsspezifikation nicht vorschreiben, sondern nur empfehlen. Ich verstehe die Strukturierung in Sicherheits- und Lebendigkeitsanforderungen als ein hilfreiches Stil- und Analysemittel.

Beispiel (Postfachsystem: Sicherheit und Lebendigkeit): Bei unserem Postfachsystem sind S-ack_safe, Snd_safe und S-ack_safe' (vgl. Abschnitt 3.3) Sicherheitsprädikate, S-ack_live ist dagegen ein Lebendigkeitsprädikat. Das in Abschnitt 3.4 angegebene Transitionssystem beschreibt (naturgemäß) ein Sicherheitsprädikat. □

Sehen wir uns nun Sicherheits- und Lebendigkeitsprädikate getrennt voneinander genauer an.

Die Definition eines Sicherheitsprädikats und die Invariantenschreibweise (vgl. Abschnitt 3.3) sind eng verwandt. Daher stellt sich die Frage, ob sich nicht jedes Sicherheitsprädikat durch eine Formel in Invariantenform darstellen läßt. Ist S ein Sicherheitsprädikat, so ist offensichtlich die Invariante $\forall s \in Act^*: s \sqsubseteq t \Rightarrow S(s)$ äquivalent zu $S(t)$. Wie steht es jedoch mit "nichttrivialen" Invarianten, d.h. läßt sich $S(t)$ immer auch durch eine Formel $\forall s \in Act^*: s \sqsubseteq t \Rightarrow P(s)$ darstellen, bei der P *kein* Sicherheitsprädikat ist? Der folgende Satz beantwortet diese Frage:

Satz (Invariantendarstellung von Sicherheitsprädikaten): Sei S ein Sicherheitsprädikat. Für jedes Prädikat P gilt die Aussage

$$\forall t \in Act^\omega: (\forall s \in Act^*: s \sqsubseteq t \Rightarrow P(s)) \Leftrightarrow S(t) \qquad (1)$$

genau dann, wenn

$$\forall s \in Act^*: (S(s) \Rightarrow P(s)) \wedge (P(s) \Rightarrow Maxsafe(S)(s)) \qquad (2),$$

gilt, wobei $Maxsafe(S)(t) \equiv t \in Act^* \Rightarrow S(t) \vee (\exists r \in Act^*: r \sqsubset t \wedge \neg S(r))$.

Anmerkung: $r \sqsubset t \Leftrightarrow_{def} r \sqsubseteq t \wedge r \neq t$. $Maxsafe(S)$ enthält alle unendlichen Spuren sowie diejenigen endlichen Spuren, die sicher sind oder nicht-sichere Spuren echt verlängern.

Beweis: Im Beweis wird $Maxsafe(S)$ mit M abgekürzt.

a) "⇒": Gelte (1).

i) Zunächst wird gezeigt: $\forall s \in Act^*: S(s) \Rightarrow P(s)$. Gelte $s \in Act^*$ und $S(s)$. Dann gilt mit (1) offensichtlich auch $P(s)$.

ii) Nun zur Aussage: $\forall s \in Act^*: P(s) \Rightarrow M(s)$. Gelte $s \in Act^*$ und $P(s)$. Zu zeigen ist $M(s)$. Falls für alle Präfixe u von s auch $P(u)$ gilt, so gilt auch $S(s)$ und damit auch $M(s)$. Ansonsten gibt es ein echtes Präfix u von s mit $\neg P(u)$. Dann gilt $\neg S(u)$, daher auch $M(s)$.

b) "⇐": Gelte (2).

i) Gezeigt wird zunächst: $(\forall s \in Act^*: s \sqsubseteq t \Rightarrow P(s)) \Rightarrow S(t)$.

Gelte $\forall s \in Act^*: s \sqsubseteq t \Rightarrow P(s)$. Zu zeigen ist: $S(t)$ gilt, was gleichbedeutend mit $\forall s \in Act^*: s \sqsubseteq t \Rightarrow S(s)$ ist. Dies wird wiederum durch Induktion über die Präfixe von t gezeigt:

Induktionsanfang: $s = \varepsilon$. Da $P(\varepsilon)$ und (2) gilt , gilt auch $M(\varepsilon)$, daher $S(\varepsilon)$, denn es gibt kein echtes Präfix von ε.

Induktionsschritt: Gelte $\forall u \in Act^*: u \sqsubseteq r \Rightarrow S(u)$ mit $r \in Act^*$ und sei $r \circ a \sqsubseteq t$ mit $a \in Act$. Zu zeigen ist: $S(r \circ a)$. Da $P(r \circ a)$ gilt, gilt auch $M(r \circ a)$. Falls $\exists s \in Act^*: s \sqsubset r \circ a \wedge \neg S(s)$, so muß es ein $s \sqsubseteq r$ geben mit $\neg S(s)$ - ein Widerspruch zur Induktionsannahme. Daher gilt $S(r \circ a)$.

ii) Nun ist zu zeigen: $S(t) \Rightarrow (\forall s \in Act^*: s \sqsubseteq t \Rightarrow P(s))$.

Gelte $S(t)$. Dann gilt auch $\forall s \in Act^*: s \sqsubseteq t \Rightarrow S(s)$ und mit (2) daher $\forall s \in Act^*: s \sqsubseteq t \Rightarrow P(s)$. □

Dieser Satz besagt, daß es für ein Sicherheitsprädikat i.a. keine eindeutige Invariantenschreibweise gibt, sondern ein ganzes Spektrum von Möglichkeiten. Nur im Trivialfall, wenn $S \equiv \mathit{true}$ oder $S \equiv \mathit{false}$ ist P (auf endlichen Spuren) eindeutig bestimmt; dann gilt für alle endlichen Spuren t: $S(t) \equiv Maxsafe(S)(t)$. Identifiziert man Prädikate mit den Mengen, die sie beschreiben, so gilt: $S \cap Act^*$ ist die kleinste Menge, für die (1) gilt, $Maxsafe(S)$ ist die größte Menge, für die (1) gilt. Alle Mengen "dazwischen" erfüllen auch (1).

Während Sicherheitseigenschaften aufgrund ihres Invariantencharakters anschaulich und beweistechnisch relativ einfach handhabbar sind (vgl. auch Abschnitt 3.4.3), ist der Umgang mit Lebendigkeitseigenschaften häufig schwieriger. Bei reaktiven Systemen gibt es im Gegensatz zu transformationellen Systemen (vgl. Kap. 2) nicht nur die verhältnismäßig einfache Lebendigkeitseigenschaft der Terminierung, die man mit Noetherschen Ordnungen in Griff bekommt. Gerade bei einer Anforderungsspezifikation können nahezu beliebige Lebendigkeitsanforderungen auftreten. Es folgen deshalb noch einige Anmerkungen, die dem besseren Verständnis von Lebendigkeitseigenschaften dienen sollen.

Betrachten wir hierzu ein Lebendigkeitsprädikat L und eine unendliche Folge $t_0, t_1, ...$ von endlichen Spuren, für die $\#t_i = i$ gilt. Diese Folge läßt sich als unendlicher "lückenloser" Berechnungspfad deuten. Gemäß unserer Definition eines Lebendigkeitsprädikats muß es für alle t_i ein $t \sqsupseteq t_i$ geben, so daß $L(t)$ gilt. Dies kann auf verschiedene Weise erreicht werden:

1. L wird erst im Unendlichen erreicht: es gilt $L(\bigsqcup t_i)$.
2. L wird "immer wieder" erfüllt: $|\{i \mid L(t_i)\}| = \infty$
3. Ab einer gewissen Stelle gilt L "stabil": $\exists k: \forall i: i \geq k \Rightarrow L(t_i)$

4. Auf dem Pfad $t_0, t_1, ...$ wird L nicht erreicht, eine Abzweigung von diesem Pfad ist zu wählen: $\exists k: \forall i: i \geq k \Rightarrow (t \sqsupseteq t_i \wedge L(t) \Rightarrow \exists l: \neg\, t \sqsupseteq t_l)$

Die Fälle 1 und 2 sowie 1 und 3 lassen sich auch kombinieren. Fall 3 ist ein Sonderfall von 2.

Zum Schluß dieses Abschnitts prüfen wir, ob ein Prädikat sowohl ein Sicherheits- als auch ein Lebendigkeitsprädikat sein kann. Der folgende Satz zeigt, daß dies nur in einem Trivialfall möglich ist:

Satz: Ein Prädikat ist genau dann ein Sicherheits- *und* Lebendigkeitsprädikat, wenn es identisch *true* ist.
Beweis: "$\Rightarrow$": Sei P Sicherheits- und Lebendigkeitsprädikat. Annahme: $\neg\, P \equiv$ true. Sei $t \in Act^{\omega}$ mit $\neg P(t)$.
Fall 1: $t \in Act^{*}$. Dann gilt für alle s mit $t \sqsubseteq s$: $\neg P(s)$, da P ein Sicherheitsprädikat ist. Also gibt es eine endliche Spur, die nicht zu einer Spur in P verlängert werden kann. Deshalb kann P kein Lebendigkeitsprädikat sein.
Fall 2: $t \in Act^{\infty}$. Da $\neg P(t)$ gibt es ein endliches Präfix s von t, so daß $\neg P(s)$, da P ein Sicherheitsprädikat ist. Damit ist man wieder bei Fall 1 angelangt.
"$\Leftarrow$": *true* erfüllt sowohl die Charakterisierung eines Sicherheits- als auch eines Lebendigkeitsprädikats. □

3.5.2 Kombination von Sicherheits- und Lebendigkeitsanforderungen

Im letzten Abschnitt wurden Sicherheits- und Lebendigkeitsprädikate getrennt voneinander betrachtet. Nun wird auf ihr Zusammenwirken eingegangen.

Gemäß den obigen Definitionen von Sicherheits- und Lebendigkeitsprädikaten kann es zu einer Anomalie kommen, wenn man ein beliebiges Sicherheitsprädikat mit einem beliebigen Lebendigkeitsprädikat kombiniert: Es kann der Fall eintreten, daß sich gewisse sichere, aber nicht lebendige (Teil-) Spuren nicht zu total korrekten (d.h. sicheren und lebendigen) Spuren verlängern lassen. Es ist fraglich, ob man solche Spuren sicher nennen soll. Hierzu wird in [Dederichs, Weber 90] Stellung genommen und es wird eine alternative Definition vorgeschlagen, die solche Anomalien ausschließt. Aufgrund dieser Definition können Sicherheits- und Lebendigkeitsanforderungen an ein System nicht unabhängig voneinander gewählt werden.

Lebendigkeit (alternative Definition): Sei S ein Sicherheitsprädikat. L heißt *Lebendigkeitsprädikat bzgl.* S, wenn gilt:

$$\forall s \in Act^{*}: (S(s) \Rightarrow \exists t \in Act^{\omega}: L(s \circ t) \wedge S(s \circ t)) \qquad (3)$$ □

Man sollte sich darüber bewußt sein, daß bei einer unabhängigen Spezifikation von Sicherheits- und Lebendigkeitsanforderungen die oben genannte Anomalie auftreten kann. Insbesondere läßt sich eine Spezifikation auf solche "Sackgassen" hin analysieren. Für die getrennte Spezifikation von Sicherheits- und Lebendigkeitseigenschaften bringt die alternative Definition jedoch keinen Vorteil: ob eine Anforderung eine Lebendigkeitsanforderung (bzgl. des Sicherheitsanteils der

Spezifikation) ist, kann nicht mehr "lokal", pro Anforderung, sondern nur nach Kenntnis aller Sicherheitsanforderungen (und - wie wir noch sehen werden - auch der übrigen Lebendigkeitsanforderungen) beurteilt werden.

In [Abadi, Lamport 90] findet sich eine zu (3) ähnliche und - wie ich zeigen werde - tatsächlich äquivalente Anforderung an ein Lebendigkeitsprädikat. Abadi und Lamport betrachten Spezifikationen der Form $(S, S \cap L)$, die aus einer Sicherheitseigenschaft S und der Kombination von S mit einer Lebendigkeitseigenschaft L (gemäß der Definition nach [Alpern, Schneider 85]) bestehen. Eine solche Spezifikation nennen sie *machine-closed*, wenn $S = Close(S \cap L)$ gilt, wobei $Close(M)$ als die kleinste Menge definiert ist, die M enthält und gegen Präfix- und Supremumsbildung abgeschlossen ist[1].

Im folgenden stelle ich die Beziehung zwischen dem Begriff *machine-closed* und der obigen (alternativen) Charakterisierung von Sicherheits- und Lebendigkeitseigenschaften her. Hierzu definiere ich $Close$ sowie die Hilfsoperatoren $Pref$ und Lim formal, die verschiedene Abschlüsse einer Spurmenge bilden und untersuche Bezüge zwischen diesen Abschlußoperatoren. Ich gehe auf diese Operatoren näher ein, da sie sich auch zur Definition der Begriffe "Sicherheit" und "Lebendigkeit" heranziehen lassen und bei der formalen Aufspaltung einer Spezifikation in einen Sicherheits- und einen Lebendigkeitsanteil eine Rolle spielen (s.u.).

Definition (Pref, Lim, Close): Sei M eine Spurmenge. Wir definieren:

a) $Pref(M) =_{def} \{x \in Act^{\omega} \mid \exists y \in M: x \sqsubseteq y\}$

b) $Lim(M) =_{def} \{x \in Act^{\omega} \mid$ es gibt eine Kette C in M mit $x = \bigsqcup C\}$

c) $Close(M)$ sei die kleinste Menge, die die Fixpunktgleichung

$$X = Pref(X) \cup Lim(X) \cup M \qquad (*)$$

erfüllt. □

(*) hat einen kleinsten Fixpunkt, da - wie man sich leicht überzeugt - $Pref$, Lim und $\cup$ bzgl. der Halbordnung $(\wp(Act^{\omega}), \subseteq)$ monoton sind und somit auch $Close$. Daher ist $Close(M)$ für jedes M wohldefiniert. Man sieht leicht, daß $Pref(Pref(M)) = Pref(M)$ und $Lim(Lim(M)) = Lim(M)$.

Der nächste Satz zeigt eine explizite Darstellung von $Close(M)$:

Satz: Seien $Close$, Lim und $Pref$ wie oben definiert. Es gilt:

$$Close(M) = Lim(Pref(M))$$

Beweis: a) $Lim(Pref(M))$ ist eine Lösung von (*): $Pref(Lim(Pref(M))) \cup Lim(Lim(Pref(M))) \cup M = Lim(Pref(M))$. Die $\supseteq$-Richtung ist klar. Zur $\subseteq$-Richtung: Offensichtlich gilt für beliebige N: $N \subseteq Pref(N)$ und $N \subseteq Lim(N)$, damit auch $M \subseteq Lim(Pref(M))$. Zum anderen gilt $Pref(Lim(Pref(M)) \subseteq Lim(Pref(M))$ (vgl. folgendes Lemma).

---

[1] Wir betrachten hier statt eines Prädikats über Spuren direkt die Spurmenge, die dieses beschreibt.

b) Lim(Pref(M)) ist die *kleinste* Menge, die (*) erfüllt: Sei Y eine Lösung von (*). Dann gilt $M \subseteq Y$, $Pref(M) \subseteq Pref(Y) \subseteq Y$, da Pref monoton und wegen (*), und daher $Lim(Pref(M)) \subseteq Lim(Y) \subseteq Y$, da auch Lim monoton und wiederum wegen (*). □

Lemma: Seien Pref und Lim wie oben definiert. Es gilt:

$$Pref(Lim(Pref(M))) \subseteq Lim(Pref(M))$$

Beweis: Sei $x \in Pref(Lim(Pref(M)))$, d.h. es gibt eine Kette C in Pref(M) mit $x \sqsubseteq \bigsqcup C$ (**).

Falls x unendlich ist, so kann x kein *echtes* Präfix von $\bigsqcup C$ sein, daher $x = \bigsqcup C$ und somit $x \in Lim(Pref(M))$.

Falls x endlich ist, so gibt es ein $z \in C$ mit $x \sqsubseteq z$ (***) aus dem folgenden Grund: Für alle $y \in C$ gilt $y \sqsubseteq \bigsqcup C$. Andererseits gilt aber auch $x \sqsubseteq \bigsqcup C$, daher $x \sqsubseteq y$ oder $y \sqsubseteq x$ für $y \in C$. Falls für alle $y \in C$ die Ungleichung $y \sqsubseteq x$ gilt, so gilt auch $\bigsqcup C \sqsubseteq x$, mit (**) daher $x = \bigsqcup C$ und da x endlich ist dann $x \in C$. Wegen (***) gilt also: $x \in Pref(Pref(M)) = Pref(M)$, also $x \in Lim(Pref(M))$. □

Korollar: $Close(Close(M)) = Close(M)$. □

Satz: Seien Pref und Lim wie oben definiert. Es gilt:

$$Pref(Lim(M)) \subseteq Lim(Pref(M)),$$

aber im allgemeinen gilt nicht $Lim(Pref(M)) \subseteq Pref(Lim(M))$.

Beweis: a) $Pref(Lim(M)) \subseteq Lim(Pref(M))$:

Sei $x \in Pref(Lim(M))$, d.h. es gibt eine Kette C in M mit $x \sqsubseteq \bigsqcup C$.

Falls x endlich ist, so muß es ein $y \in C$ geben mit $x \sqsubseteq y$ (die Argumentation läuft hier parallel zur Argumentation im obigen Lemma). Also gilt $x \in Pref(M)$ und damit $x \in Lim(Pref(M))$.

Falls x unendlich ist, so gilt $x \in Lim(M)$ und da $M \subseteq Pref(M)$ auch $x \in Lim(Pref(M))$.

b) Ein Gegenbeispiel für die umgekehrte Richtung ist $M = a^*b$ (in der Notation regulärer Ausdrücke), denn $Lim(Pref(M)) = a^*b \cup a^* \cup a^\infty$, aber $Pref(Lim(M)) = a^*b \cup a^*$. □

Beobachtung: Sicherheitsprädikate sind offensichtlich gerade diejenigen Prädikate S, für die $Close(S) = S$ gilt, Lebendigkeitsprädikate dagegen diejenigen Prädikate L, für die $Act^* \subseteq Pref(L)$ gilt.

Der folgende Satz besagt, daß die hier vorgestellte alternative Definition des Begriffs "Lebendigkeit" und Lamports *"machine-closed"* äquivalent sind.

Satz (Zusammenhang mit "machine-closed"): Sei S eine Sicherheitseigenschaft nach (1), L eine Lebendigkeitseigenschaft nach (2). Es gilt: $(S, S \cap L)$ ist genau dann *machine-closed*, wenn L die alternative Definition einer Lebendigkeitseigenschaft (bzgl. S) (3) erfüllt.

Beweis: "⇒": Gelte $Close(S \cap L) = S$. Zu zeigen ist: $\forall s \in Act^*: (s \in S \Rightarrow \exists t \in Act^\omega: s \cdot t \in S \cap L)$. Sei $s \in S$ und s endlich. Da $s \in Close(S \cap L) = Lim(Pref(S \cap L))$, liegt s in $Pref(S \cap L)$. Daher gibt es ein t, so daß $s \cdot t \in S \cap L$.

"⇐": a) Zu zeigen ist: $Close(S \cap L) \subseteq S$. Wenn $x \in Close(S \cap L)$, dann $x \in Close(S) = S$, da Close monoton und S ein Sicherheitsprädikat ist.

b) Zu zeigen ist: $S \subseteq Close(S \cap L)$.
Sei x endlich und $x \in S$. Gemäß der Lebendigkeitsdefinition gibt es ein y, so daß $x \sqsubseteq y$ und $y \in S \cap L$. Also $x \in Close(S \cap L)$.
Sei x unendlich und $x \in S$. Falls $x \in S \cap L$, dann $x \in Close(S \cap L)$. Sonst: $x \in S$ und $x \notin L$. Dann sind mit $x \in S$ auch alle x' mit $x' \sqsubseteq x$ Elemente aus S. Diese sind gemäß der Lebendigkeitsdefinition erweiterbar zu einer Spur in $S \cap L$, also sind alle Präfixe von Spuren x in $Close(S \cap L)$. Da $x = \bigsqcup\{x' \mid x' \sqsubseteq x \text{ und } x' \text{ endlich}\}$, gilt auch $x \in Close(S \cap L)$. □

Im folgenden untersuche ich die Verknüpfung von Sicherheits- und Lebendigkeitsprädikaten, wobei die Definitionen (1) und (2) zugrundeliegen. Die Auswirkungen der Untersuchungsergebnisse auf die Methodik stelle ich anschließend dar.

Satz (Kombination von Sicherheits- und Lebendigkeitsprädikaten): Seien S, S' Sicherheitsprädikate und L, L' Lebendigkeitsprädikate, M ein beliebiges Prädikat über Spuren. Es gilt:

(1)	$S \wedge S'$	ist ein Sicherheitsprädikat,
(2)	$S \vee S'$	ist ein Sicherheitsprädikat,
(3)	$\neg S$	ist i.a. weder ein Sicherheits- noch ein Lebendigkeitsprädikat,
(4)	$S \Rightarrow S'$	ist i.a. kein Sicherheitsprädikat,
(5)	$L \wedge L'$	ist i.a. kein Lebendigkeitsprädikat,
(6)	$L \vee M$	ist ein Lebendigkeitsprädikat,
(7)	$\neg L$	ist i.a. weder ein Sicherheits- noch ein Lebendigkeitsprädikat,
(8)	$L \Rightarrow L'$	ist ein Lebendigkeitsprädikat,
(9)	$S \wedge L$	ist i.a. weder ein Sicherheits- noch ein Lebendigkeitsprädikat,
(10)	$S \vee L$	ist i.a. kein Sicherheits-, aber ein Lebendigkeitsprädikat,
(11)	$S \Rightarrow L$	ist i.a. kein Sicherheits-, aber ein Lebendigkeitsprädikat,
(12)	$L \Rightarrow S$	ist i.a.weder ein Sicherheits- noch ein Lebendigkeitsprädikat.

Beweis: Ich zeige hier nur (1), (2), (5), (6) und (9). Die Beweise für die übrigen Aussagen laufen analog. (8), (10) und (11) sind Spezialfälle von (6).
Zu (1): Zu zeigen: $S(t) \wedge S'(t) \Leftrightarrow (\forall s \in Act^*: s \sqsubseteq t \Rightarrow S(s) \wedge S'(s))$
"$\Rightarrow$": Wenn $S(t) \wedge S'(t)$ gilt, dann gilt auch für alle endlichen Präfixe s von t $S(s)$ und $S'(s)$, da S und S' Sicherheitsprädikate sind.
"$\Leftarrow$": Wenn für alle endlichen Präfixe s von t $S(s)$ und $S'(s)$ gilt, dann gilt auch $S(t)$ und $S'(t)$, da S und S' Sicherheitsprädikate sind.
Zu (2): Zu zeigen: $S(t) \vee S'(t) \Leftrightarrow (\forall s \in Act^*: s \sqsubseteq t \Rightarrow S(s) \vee S'(s))$
"$\Rightarrow$": Gelte $S(t) \vee S'(t)$ und sei $s \in Act^*$ und $s \sqsubseteq t$. Falls $S(t)$ gilt, dann gilt auch $S(s)$, also $S(s) \vee S'(s)$. (Für S' analog).
"$\Leftarrow$": Gelte $\forall s \in Act^*: s \sqsubseteq t \Rightarrow S(s) \vee S'(s)$. Dann gilt:
$(\forall s \in Act^*: s \sqsubseteq t \Rightarrow S(s)) \vee (\forall s \in Act^*: s \sqsubseteq t \Rightarrow S'(s))$ (*)
Sonst gäbe es s, s', so daß $\neg S(s)$ und $\neg S'(s)$ gilt. Dann gälte aber auch $\neg S(t)$ und $\neg S'(t)$ und (*) wäre falsch, da S und S' Sicherheitsprädikate sind.
Zu (5): $L(t) \equiv even(\#t) \wedge \#t < \infty$, $L'(t) \equiv \neg even(\#t) \wedge \#t < \infty$ (hierbei gelte $even(n)$ gdw. n ist

gerade).
Zu (6): Zu zeigen ist, daß es für alle $s \in Act^*$ ein $t \in Act^\omega$ gibt mit $L(s \circ t) \vee M(s \circ t)$. Dies gilt, da L ein Lebendigkeitsprädikat ist und $L \Rightarrow L \vee M$.
Zu (9): i.a. kein Sicherheitsprädikat: Gegenbeispiel: $S \equiv true$, $L(t) \equiv \#t = \infty$; i.a. kein Lebendigkeitsprädikat: Gegenbeispiel: $S = false$. □

Von methodischer Bedeutung sind vor allem (1), (2) und (5). (1) und (2) besagen, daß die Konjunktion und Disjunktion von Sicherheitsprädikaten wiederum ein Sicherheitsprädikat ist. (5) legt dar, daß dies bei Lebendigkeitsprädikaten nicht notwendigerweise der Fall ist. Diesen Punkt greife ich nach einigen Bemerkungen zu den übrigen Aussagen des Satzes noch einmal auf.

Gemäß (6) darf man Lebendigkeitsprädikate beliebig vergrößern (z.B. um andere Sicherheits- oder Lebendigkeitsprädikate). (8) und (11) bringen zum Ausdruck, daß man eine Lebendigkeitsanforderung von einer anderen Lebendigkeits- oder Sicherheitsanforderung abhängig machen darf und auf diese Weise wiederum eine Lebendigkeitsanforderung erhält. Für Sicherheitsanforderungen gibt es keine analoge Aussage.

(3) und (7) verdeutlichen die wohlbekannte Tatsache, daß eine Sicherheitseigenschaft nicht einfach als Negation einer Lebendigkeitseigenschaft verstanden werden darf (und umgekehrt; für eine ausführlichere Diskussion siehe [Lamport 89]). (9) zeigt, daß die Kombination eines Sicherheits- mit einem Lebendigkeitsprädikat im allgemeinen weder ein Sicherheits- noch eine Lebendigkeitsprädikat ist.

Sehen wir uns nun die Auswirkung der Aussage (5) genauer an. Sei eine Anforderungsspezifikation in der Form

$$P(t) \equiv S_1(t) \wedge ... \wedge S_m(t) \wedge L_1(t) \wedge ... \wedge L_n(t)$$

mit Sicherheitsanforderungen $S_1, ..., S_m$ und Lebendigkeitsanforderungen $L_1, ..., L_n$ notiert, d.h. wie in unserer Methodik empfohlen in Sicherheits- und Lebendigkeitsanforderungen strukturiert. $S_1(t) \wedge ... \wedge S_m(t)$ stellt zwar eine Sicherheitsprädikat dar, $L_1(t) \wedge ... \wedge L_n(t)$ jedoch nicht unbedingt ein Lebendigkeitsprädikat. $L \wedge L'$ ist genau dann kein Lebendigkeitsprädikat, wenn

$$\exists s \in Act^*: \forall t \in Act^\omega: \neg(L(s \circ t) \wedge L'(s \circ t)) \qquad (*)$$

gilt. Das heißt, daß es für einen Teilablauf keine Verlängerung gibt, die sowohl in L als auch in L' liegt. $L \wedge L'$ ist in diesem Fall kein reines Lebendigkeitsprädikat, es enthält auch einen Sicherheitsanteil (die Erfahrung zeigt, daß dies bei "realen" Spezifikationen eher selten eintritt). Für eine echte Trennung der Sicherheits- und Lebendigkeitsanforderungen benötigt man noch einen Anpassungsschritt; er bestimmt den enthaltenen Sicherheitsanteil und den "wirklichen" Lebendigkeitsanteil. Gemäß [Schneider 87] kann dieser Anpassungsschritt mechanisch durchgeführt werden. Dort wird gezeigt, daß sich jede beliebige Eigenschaft P in einen Sicherheitsanteil S und einen Lebendigkeitsanteil L aufspalten läßt, es kann jedoch mehr als eine Aufspaltungsmöglichkeit geben. Eine Möglichkeit ist,

$S = P \cup Close(P)$ und

$L = P \cup \neg Close(P)$.

zu setzen. Diese Aufspaltungsmöglichkeit ergibt den kleinsten Sicherheitsanteil und den größten Lebendigkeitsanteil (vgl. [Dederichs, Weber 90]).

Da Sicherheitseigenschaften i.a. leichter als Lebendigkeitseigenschaften zu beweisen sind, wäre es nützlich, wenn es auch eine Aufspaltung eines Prädikats in einen größten Sicherheitsanteil und einen kleinsten Lebendigkeitsanteil gäbe. Das folgende Beispiel zeigt, daß eine solche Aufspaltung i.a. nicht möglich ist:

Beispiel: Seien $P(t) \equiv t = \varepsilon$ und $S_i(t) \equiv \#t \leq i \vee t = \varepsilon$, $L_i(t) \equiv \#t > i \vee t = \varepsilon$ für $i \in Nat$. Es gilt: $S_i(t) \wedge L_i(t) \Leftrightarrow P(t)$. Angenommen, es gäbe einen größten Sicherheitsanteil S_{max} und einen kleinsten Lebendigkeitsanteil L_{min} von P, so gälte:
$\forall i \in Nat$: $S_i(t) \Rightarrow S_{max}(t)$, d.h. $\#t \leq \infty \Rightarrow S_{max}(t)$, daher $S_{max}(t) \Leftrightarrow \mathit{true}$, da S_{max} Sicherheitsprädikat.
$\forall i \in Nat$: $L_{min}(t) \Rightarrow L_i(t)$, d.h. $\forall i \in Nat$: $L_{min}(t) \Rightarrow \#t > i \vee t = \varepsilon$, daher $L_{min}(t) \Leftrightarrow \#t = \infty \vee t = \varepsilon$.
Damit gälte $S_{max}(t) \wedge L_{min}(t) \Leftrightarrow L_{min}(t) \neq P(t)$. □

3.6 Echtzeitspezifikation

Verteilte Systeme werden in Anwendungsbereichen eingesetzt, in denen Echtzeiteigenschaften eine Rolle spielen; ein typisches Beispiel ist die Prozeßautomatisierung. In derartigen Anwendungen ist es sinnvoll, schon in einer Anforderungsspezifikation Echtzeitanforderungen formal zu erfassen. Die meisten Spezifikationsformalismen abstrahieren von quantitativen Zeitbetrachtungen, nur wenige erlauben die Formulierung von Zeiteigenschaften.

In diesem Abschnitt untersuche ich, wie sich Echtzeiteigenschaften durch Spuren beschreiben lassen. Zu diesem Zweck vergleiche ich verschiedene naheliegende und zum Teil in ähnlicher Form existierende Erweiterungsmöglichkeiten des Spurformalismus:

1. zeitbehaftete Spuren,
2. Spuren mit Zeitstempeln und
3. Spuren von Multimengen von Aktionen.

In allen drei Fällen läßt sich sowohl der ablauforientierte als auch der transitionsorientierte Spezifikationsstil einsetzen.

3.6.1 Zeitbehaftete Spuren

In diesem Ansatz wird eine neue Aktion $\surd$ ("Tick") eingeführt, die das Ticken einer Uhr modelliert. Technisch werden Spuren aus $(Act \cup \{\surd\})^{\omega}$ mit den bei Spurspezifikationen üblichen Mitteln beschrieben.

Wir gehen von der folgenden Modellvorstellung aus: Alle Modellierungsannahmen gemäß Abschnitt 2.1 gelten weiterhin, insbesondere die zeitliche Atomarität. Zudem gibt es eine Uhr in unserem System, deren Ticken wir durch $\surd$-Aktionen modellieren. Die Uhr läuft fortwährend, so daß in jedem Ablauf des Systems unendlich viele $\surd$-Aktionen vorkommen. Aktionen zwischen zwei $\surd$-Aktionen geschehen während eines Zeittakts; ich nenne dies bewußt nicht

gleichzeitig - Gleichzeitigkeit gibt es im Spurformalismus nicht. Die √-Aktionen geben ein Zeitraster vor, das je nach Anwendung beliebig grob oder fein gewählt werden kann; dies entspricht einer gröberen oder feineren Zeitmessung. Insofern unterscheidet sich diese Modellierung von der in [Broy 90] vorgestellten: Dort verkörpert √ eine "inhaltsleere" Nachricht.

Für eine vorgegebene Menge Act bezeichnen wir die Menge der zeitbehafteten Spuren mit $Act_{\surd}^{\infty}$. Formal: $Act_{\surd}^{\infty} =_{def} \{s \in (Act \cup \{\surd\})^{\omega} \mid \#\surd\copyright s = \infty\}$. $Act_{\surd}^{\infty}$ ist eine Lebendigkeitseigenschaft.

$Act_{\surd}^{*}$ steht für die Menge der endlichen Teilabläufe; endliche *vollständige* Abläufe stammen aus $Act_{\surd}^{*} \circ \{\surd\}^{\infty}$.

Ich zeige nun, wie sich Echtzeiteigenschaften mit Spurformeln beschreiben lassen. Dazu stütze ich mich auf eine in [Dasarathy 85] angegebene Klassifizierung von Echtzeitanforderungen und formalisiere einen typischen Vertreter aus jeder Klasse.

1) *Maximum:* "Auf eine Aktion a folgt nach höchstens N Zeiteinheiten (ZE) eine Aktion b."

$$Max(s) \equiv \forall p \in Act_{\surd}^{*}: p \circ a \sqsubseteq s \Rightarrow \exists q \in Act_{\surd}^{*}: p \circ a \circ q \circ b \sqsubseteq s \wedge \#\surd\copyright q \leq N$$

2) *Minimum:* "Auf eine Aktion a folgt frühestens nach N ZE eine Aktion b."

$$Min(s) \equiv \forall p \in Act_{\surd}^{*}: p \circ a \sqsubseteq s \Rightarrow \exists q \in Act_{\surd}^{*}: p \circ a \circ q \circ b \sqsubseteq s \wedge \neg a \; in \; q \wedge \neg b \; in \; q \wedge \#\surd\copyright q \geq N$$

3) *Andauern eines Vorgangs:* "Nach einem a (Beginn des Vorgangs) folgt genau N ZE später ein b (Ende des Vorgangs) und dazwischen weder a noch b."
Damit wird festgelegt, daß der Vorgang genau N ZE dauert.

$$Duration(s) \equiv Max(s) \wedge Min(s)$$

4) *Zeitanforderung für eine Sequenz von Aktionen:* "Innerhalb von N ZE müssen die Aktionen a_1 bis a_m (in dieser Reihenfolge) stattgefunden haben." (Verallgemeinerung von 1)

$$Seq(s) \equiv \forall p \in Act_{\surd}^{*}: p \circ a_1 \sqsubseteq s \Rightarrow \exists q_1, q_2, ..., q_{m-1} \in Act_{\surd}^{*}: q_1, q_2, ..., q_{m-1}: p \circ a_1 \circ q_1 \circ a_2 \circ ... \circ q_{m-1} \circ a_m \sqsubseteq s \wedge \neg\{a_1,...,a_m\} \; in \; q_1 \wedge ... \wedge \neg\{a_1,...,a_m\} \; in \; q_{m-1} \wedge \#\surd\copyright a_1 \circ q_1 \circ a_2 \circ ... \circ q_{m-1} \circ a_m \leq N$$

Die Definitionen von Sicherheits- und Lebendigkeitseigenschaften gemäß Abschnitt 3.5.1 lassen sich leicht auf zeitbehaftete Spuren übertragen; es ergeben sich nur geringfügige Änderungen in der Notation:

Definition (Sicherheit): P ist ein *Sicherheitsprädikat*, wenn gilt:

$$\forall t \in Act_{\surd}^{\infty}: (P(t) \Leftrightarrow (\forall s \in Act_{\surd}^{*}: s \sqsubseteq t \Rightarrow \exists u \in Act_{\surd}^{\infty}: P(s \circ u)))$$ □

Erfolgt die Verletzung einer Sicherheitsanforderung durch eine √-Aktion, so liegt ein *Zeitfehler* vor. In diesem Fall hätte vor dieser √-Aktion mindestens eine weitere Aktion aus Act stattfinden müssen.

Die Definition eines Lebendigkeitsprädikats, angepaßt auf zeitbehaftete Spuren, liest sich folgendermaßen:

Definition (Lebendigkeit): P ist ein *Lebendigkeitsprädikat*, wenn gilt:

$$\forall s \in Act_{\surd}^{*}: \exists t \in Act_{\surd}^{\infty}: P(s \circ t)$$ □

Setzt man für das Eintreffen einer bestimmten Aktion eine zeitliche obere Schranke, so werden Eigenschaften, die sich ohne Zeitbetrachtung nur als Lebendigkeitseigenschaften (dann allerdings ohne obere Zeitschranke) formulieren lassen, zu Sicherheitseigenschaften. Setzen wir etwa in der Anforderung

$$P(t) \equiv \forall r \in Act^{*}, s \in Act^{\omega} : t = r \circ a \circ s \Rightarrow b \; \textit{in} \; s$$

für das Eintreffen der Aktion b eine zeitliche obere Schranke von N Zeiteinheiten, so erhalten wir die Anforderung Max(t) (s.o.), also eine Sicherheitsanforderung. Generell ist zu erwarten, daß Sicherheitseigenschaften bei Echtzeitsystemen eine noch größere Bedeutung zukommt als bei Nicht-Echtzeitsystemen.

In der Regel sind nicht sämtliche Anforderungen an ein zu erstellendes System zeitkritisch; ich verstehe zeitunabhängige Anforderungen als einen Spezialfall von Echtzeitanforderungen. Ist ein Prädikat zeitunabhängig, so darf man in die dadurch beschriebenen Spuren an beliebigen Stellen √-Aktionen einfügen oder solche aus ihnen entnehmen - vorausgesetzt, daß die Anzahl der √-Aktionen unendlich bleibt. Die Zeitunabhängigkeit von Anforderungen wird durch die folgende Definition formal erfaßt:

Definition (zeitunabhängig): Ein Prädikat P über Spuren aus $Act_{\surd}^{\infty}$ heißt *zeitunabhängig*, wenn gilt: $\forall s,t \in Act_{\surd}^{\infty}: Act \copyright s = Act \copyright t \Rightarrow (P(s) \Leftrightarrow P(t))$. □

Ein Prädikat P über Spuren aus Act^{ω} läßt sich als jenes zeitbehaftete (aber zeitunabhängige) Prädikat $P_{\surd}$ über Spuren aus $Act_{\surd}^{\infty}$ verstehen, das definiert ist durch: $P_{\surd}(s) \equiv P(Act \copyright s)$.

Der Vorteil der Modellierung durch zeitbehaftete Spuren besteht darin, daß sie die zeitfreie Modellierung auf natürliche Weise erweitert. Echtzeitanforderungen lassen sich wie (andere) funktionale Anforderungen formulieren und mit Nicht-Echtzeitanforderungen kombinieren. Nach meiner Auffassung sind die anderen Erweiterungsmöglichkeiten des Spurformalismus Spuren mit Zeitstempeln und Spuren von Multimengen von Aktionen, weniger gut handhabbar; ich stelle sie im folgenden zum Vergleich dar.

3.6.2 Weitere Möglichkeiten

Spuren mit Zeitstempeln

Quantitative Zeitinformation läßt sich in Spuren auch dadurch einbringen, daß jede Aktion in einer Spur mit einem Zeitstempel versehen wird. Bei Annahme einer diskreten Zeit, repräsentiert durch natürlichzahlige Zeitstempel, werden Elemente aus $(Act \times Nat)^\omega$ beschrieben. Die Zeitstempel müssen monoton aufsteigend sein:

$$Monoton(s) \equiv \forall q \in (Act \times Nat)^*, a, b \in Act, m, n \in Nat: q \circ \langle a,m \rangle \circ \langle b,n \rangle \sqsubseteq s \Rightarrow m \leq n,$$

und nur endlich viele Aktionen dürfen in einem Zeitintervall stattfinden:

$$Fin(s) \equiv \forall n \in Nat: \#(Act \times \{n\}) \copyright s < \infty$$

Spuren mit Zeitstempeln sind dem in [Reed, Roscoe 86] vorgestellten Spurmodell für Echtzeit-CSP ähnlich. Die Unterschiede bestehen darin, daß ich nicht nur endliche, sondern auch unendliche Spuren beschreibe und eine diskrete Zeit annehme (Reed und Roscoe verwenden nichtnegative reelle Zahlen als Zeitstempel). Tatsächlich sind zeitbehaftete Spuren und Spuren mit Zeitstempeln eng verwandt, was sich aus den folgenden Übersetzungen entnehmen läßt:

Von zeitbehafteten Spuren kommt man zu Spuren mit Zeitstempeln mittels der Funktion timestamp: $Act\surd^\infty \to (Act \times Nat)^\omega$, die wie folgt definiert sind:

$$timestamp(s) = ht(s,0),$$

wobei ht: $Act\surd^\infty \times Nat \to (Act \times Nat)^\omega$ definiert ist durch:

$$ht(\surd^\infty,n) = \varepsilon,$$
$$a \in Act \Rightarrow ht(a \circ s,n) = \langle a,n \rangle \circ ht(s,n),$$
$$ht(\surd \circ s,n) = ht(s,n+1).$$

Die Übersetzung von Spuren mit Zeitstempeln in zeitbehaftete Spuren geschieht mittels der Funktion mkticks: $(Act \times Nat)^\omega \to Act\surd^\infty$, die folgendermaßen definiert ist:

$$mkticks(s) = hm(s,0),$$

wobei hm: $(Act \times Nat)^\omega \times Nat \to Act\surd^\infty$ definiert ist durch:

$$hm(\varepsilon,n) = \surd^\infty,$$
$$m \geq n \Rightarrow hm(\langle a,m \rangle \circ s,n) = \surd^{m-n} \circ a \circ hm(s,m)$$

Man beachte, daß das Verhalten von hm (und damit von mkticks) nur für den Fall "korrekter" Spuren mit Zeitstempeln festgelegt ist.

Die Anforderung Fin(s) entspricht der Anforderung $\#\surd \copyright s = \infty$ bei zeitbehafteten Spuren. Eine Monoton(s) vergleichbare Anforderung ist dagegen bei zeitbehafteten Spuren nicht nötig; jedes Element aus $Act\surd^*$ entspricht dort einem "physikalisch möglichen" Teilablauf.

Formuliert man Transitionssysteme zur Spezifikation von Spuren mit Zeitstempeln, so läßt sich das Prädikat Monoton(s) durch Anforderungen an die Übergangsrelation gewährleisten, da die Monotonie lokal bzw. nach endlicher Zeit überprüfbar ist; sie ist eine Sicherheitseigenschaft.

Die Bedingung Fin(s) kann jedoch nur überprüft werden, wenn die gesamte Spur vorliegt; sie ist also eine Lebendigkeitseigenschaft.

Spuren von Multimengen von Aktionen

Mit Spuren von Multimengen von Aktionen modellieren wir explizit, wann Aktionen gleichzeitig stattfinden. Damit weichen wir vom Interleaving-Konzept des Spurformalismus ab. Technisch beschreiben wir Elemente aus $\wp_{mult}(Act)^{\omega}$, wobei $\wp_{mult}(Act)$ die Menge aller Multimengen mit Elementen aus Act bezeichnet. Eine Spur $\langle M_0,M_1...\rangle$ gibt wieder, daß zum Zeitpunkt 0 die Aktionen in M_0 gleichzeitig stattfinden, zum Zeitpunkt 1 die Aktionen aus M_1, usw. Einige der Multimengen M_i können auch leer sein. Eine Anwendung dieser Spezifikationsmethode zeigt [Weber 90b]. Diese schrittartige Modellierung findet sich auch in der Petrinetz-Theorie als eine mögliche Beschreibung von Netzabläufen (vgl. [Reisig 82]).

Mein Eindruck ist, daß Spuren von Multimengen von Aktionen weniger handlich als zeitbehaftete Spuren sind. Insbesondere lassen sich zeitfreie mit Spurformeln formulierte Anforderungen nicht auf ähnlich einfache Weise mit zeitkritischen kombinieren wie bei zeitbehafteten Spuren.

4. Komponentenorientierte Spezifikation

In diesem Kapitel erläutere ich, auf welche Weise sich die Trennung der globalen Anforderungen in Produkt- und Umgebungsanforderungen darstellen und durchführen läßt. Ich verwende hierzu das Konzept *komponentenorientierter Spezifikationen* (vgl. Abschnitt 1.2). Dies sind weiterhin spurbasierte Beschreibungen, jedoch angereichert um Strukturinformation, die die Aufteilung des Gesamtsystems in Komponenten betrifft.

Zur Beschreibung der Komponentenanforderungen werden die gleichen Techniken wie bei der globalen Spezifikation eingesetzt, die Sichtweise ist jedoch anders: das verteilte System wird nicht mehr als ein monolithischer Block gesehen, sondern in mehrere Komponenten, d.h. offene Teilsysteme des ganzen geschlossenen Systems, strukturiert. Die in der globalen Spezifikation eingeführten Aktionen werden einzelnen Komponenten als Ein- oder Ausgabeaktionen zugeordnet.

In Abschnitt 4.1 kläre ich, welche methodische Rolle der komponentenorientierten Spezifikation zukommt, und erläutere die Technik und methodische Vorgehensweise dieser Spezifikationsphase. In den Abschnitten 4.2 und 4.3 untersuche ich die Aufspaltung globaler Anforderungen in lokale. Hier ist insbesondere von Interesse, ob sich die Anforderungen an die Teilsysteme Produkt und Umgebung vollständig aus der globalen Spezifikation ableiten lassen. Während in Abschnitt 4.2 zunächst nur Sicherheitseigenschaften betrachtet werden, bezieht Abschnitt 4.3 auch Lebendigkeitseigenschaften ein. Die Ergebnisse rechtfertigen die Vorgehensweise, die ich in Abschnitt 4.1 vorschlage. Die Spezifikation von Echtzeitanforderungen an Komponenten wird schließlich in Abschnitt 4.4 behandelt.

4.1 Begriffsbestimmung und Vorgehensweise

Eine *komponentenorientierte Spezifikation* ist eine spurorientierte Beschreibung eines verteilten Systems, die im Gegensatz zu einer globalen Spezifikation Strukturinformation enthält: ein System wird hier als aus mehreren interagierenden *Komponenten* bestehend gesehen. Bei einer komponentenorientierten Spezifikation sind globale und lokale Anforderungen an das verteilte System möglich. Eine *globale* Anforderung richtet sich an mehrere, möglicherweise alle Komponenten des verteilten Systems, eine *lokale* Anforderung dagegen an eine einzelne Komponente, sie ist eine Anforderung an die Schnittstelle einer Komponente.

Die Strukturierung des Systems in gewisse Komponenten ist insofern Bestandteil der Anforderungsspezifikation, als ein Kunde fordern kann, daß das System in Form bestimmter Komponenten zu gestalten ist, insbesondere, daß einige eventuell bereits existierende Umgebungskomponenten verwendet werden. Indem der Kunde festlegt, welche Komponenten erstellt werden sollen (Produkt) und welche er bereitstellt (Umgebung), ist die Grundlage für die Aufgabenverteilung der globalen Anforderungen geschaffen. Daraus ergibt sich, daß eine globale Spezifikation die Funktion eines Kontrakts zwischen einem Kunden und einem Systementwickler nur unzureichend erfüllt. Ziel der komponentenorientierten Spezifikation ist also eine eindeutige Regelung der Zuständigkeit des Systementwicklers und des Kunden.

Nun zur Vorgehensweise bei der komponentenorientierten Spezifikation:

Der Kunde legt zunächst fest, aus welchen Komponenten das System bestehen soll. Es gibt immer mindestens zwei Komponenten[1]: eine Produkt- und eine Umgebungskomponente. Zudem gibt er an, in welcher Weise die Komponenten interagieren. Hierzu bestimmt er die Ein- und Ausgaben jeder Komponente. Nach unserer Modellierung kann eine Komponente nicht beeinflussen, wann sie Eingaben bekommt und welche Eingaben sie bekommt (asynchrone Kommunikation). Sie hat lediglich Kontrolle über ihre Ausgaben. Die Ein- und Ausgaben jeder Komponente sind Teilmengen der Aktionenmenge der globalen Spezifikation. Bei der Festlegung der Ein- und Ausgaben der Komponenten sind einige Randbedingungen zu berücksichtigen: Die Vereinigung der Ein- und Ausgabemengen aller Komponenten ergibt die Aktionenmenge der globalen Spezifikation. Jede Aktion ist Eingabeaktion[2] mindestens einer Komponente und Ausgabeaktion genau einer Komponente. Diese Bedingungen stellen sicher, daß tatsächlich alle Aktionen der globalen Spezifikation berücksichtigt werden (und nur diese), ferner daß die Zuständigkeit der Komponenten für die Aktionen genau festgelegt ist und daß insgesamt ein geschlossenes System beschrieben wird. Das Ergebnis dieser Festlegung läßt sich durch ein Datenflußdiagramm (Netz von Komponenten) darstellen, in dem die Knoten mit Komponentennamen beschriftet sind, die Kanten mit Aktionenmengen:

Beispiel (Postfachsystem: Komponentenstruktur): Eine Möglichkeit, das Postfachsystem zu strukturieren, ist die Gliederung in die drei Komponenten "sendende Prozesse", "empfangende Prozesse" und "Postfachserver". Das folgende Bild zeigt die Kommunikationsstruktur:

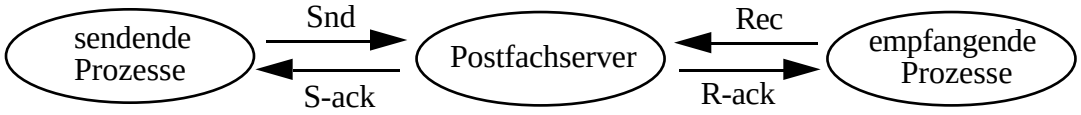


Fig. 4.1

mit Snd = {snd(ts,mb,ms) | ts∈ Tsk, mb∈ Mbx, ms∈ Msg},
S-ack = {s-ack(ts,mb,sg) | ts∈ Tsk, mb∈ Mbx, sg∈ Sig},
Rec = {rec(ts,mb) | ts∈ Tsk, mb∈ Mbx},
R-ack = {r-ack(ts,mb,ms,sg) | ts∈ Tsk, mb∈ Mbx, ms∈ Msg, sg∈ Sig}.

Die Komponente "Postfachserver" sei die Produktkomponente, die Komponenten "sendende Prozesse" und "empfangende Prozesse" seien die Umgebungskomponenten. □

Damit ist die *statische* Systemstruktur festgelegt. Die lokalen und globalen Anforderungen bestimmen das *dynamische* Verhalten.

---

[1] Trivialfälle, in denen gar keine Produktkomponente entwickelt werden soll oder das Produkt in keiner Ein-/ Ausgabebeziehung mit der Umgebung steht, betrachte ich hier nicht.

[2] Die Begriffe "Eingabe" und "Eingabeaktion" werden synonym verwendet, ebenso "Ausgabe" und "Ausgabeaktion".

Betrachten wir zunächst, wie *lokale* Anforderungen an eine einzelne Komponente durch Spuren dargestellt werden. Die zugrundeliegende Idee ist es, Anforderungen an die Ein-/ Ausgabeschnittstelle der Komponente zu richten. Das mathematische Modell einer Komponente ist ein Tripel der Form (I,O,T), wobei I und O disjunkte Mengen von Aktionen sind, die Ein- und Ausgaben der Komponente, und T eine Teilmenge von $(I \cup O)^\omega$ ist. T beschreibt die Menge der erlaubten Abläufe der Komponente. Ein Ablauf ist in dem Sinne zu verstehen, daß die Ein- und Ausgaben gemäß der Reihenfolge ihres Auftretens notiert werden.

In den späteren Phasen unserer Entwurfsmethodik entwickeln wir für jede Produktkomponente einen Agenten, der die Spurmenge T der Komponente (teilweise) *realisiert*, d.h. der nur Spuren in T erzeugt (vgl. Kap. 5). Wie wir in Abschnitt 4.3 sehen werden ist nicht jede Teilmenge von $(I \cup O)^\omega$ realisierbar; dieses Phänomen wollen wir auf der Ebene der komponentenorientierten Spezifikation nicht durch zusätzliche Anforderungen an T ausschließen. (Tatsächlich erscheint es auch schwierig, hinreichende und leicht überprüfbare Kriterien zur Realisierbarkeit anzugeben, vgl. [Broy et al. 91a]). Eine nicht realisierbare Spezifikation kann als inkonsistent gewertet werden.

Eine (lokale) Spezifikation einer Komponente hat die Form

spec Comp (I, O)
    C(t)
end

wobei Comp der Name der Komponente ist und das Prädikat C die erlaubten Spuren der Komponente angibt. C spezifizieren wir mit den in Kap. 3 beschriebenen Techniken.

Beispiel (Postfachsystem: Komponentenanforderungen): Für die Komponenten unseres Postfachsystems lassen sich leicht (ausgehend von der globalen Spezifikation in Kap. 3) lokale Anforderungen angeben. Sie lauten:

spec SENDER (S-ack, Snd)
    Snd_safe(t)
end

spec RECEIVER (R-ack, Rec)
    Rec_safe(t)
end

spec SERVER (Snd ∪ Rec, S-ack ∪ R-ack)
    t **is trace of** MAILBOX ∧ S-ack_live(t) ∧ R-ack_live(t)
end

Hierbei sind Snd_safe und S-ack_live die Prädikate aus den Beispielen zum Postfachsystem in Abschnitt 3.3. Rec_safe und R-ack_live seien die Snd_safe und S-ack_live entsprechenden Anforderung an den Empfänger, die Sendeaktionen und Senderückmeldungen seien durch Empfangsversuchaktionen und Empfangsrückmeldungen ersetzt:

$$\text{Snd\_safe}(t) \equiv \forall s \in Act^*: s \sqsubseteq t \Rightarrow \#(\{snd(ts,mb,ms) \mid ms \in Msg\} \copyright s) \leq \#(\{s\text{-}ack(ts,mb,sg) \mid sg \in Sig\} \copyright s) + 1$$

$$\text{Rec\_safe}(t) \equiv \forall s \in Act^*: s \sqsubseteq t \Rightarrow \#(\{rec(ts,mb)\} \copyright s) \leq \#(\{r\text{-}ack(ts,mb,ms,sg) \mid ms \in Msg, sg \in Sig\} \copyright s) + 1$$

$$\text{S-ack\_live}(t) \equiv \forall r \in Act^*, s \in Act^\omega: t = r \circ snd(ts,mb,ms) \circ s \Rightarrow \exists sg \in Sig: s\text{-}ack(ts,mb,sg) \textit{ in } s$$

$$\text{R-ack\_live}(t) \equiv \forall r \in Act^*, s \in Act^\omega: t = r \circ rec(ts,mb) \circ s \Rightarrow \exists sg \in Sig, ms \in Msg: r\text{-}ack(ts,mb,ms,sg) \textit{ in } s$$

MAILBOX bezeichne das in Abschnitt 3.4 angegebene Transitionssystem. Zur Schreibweise t **is trace of** MAILBOX vgl. Abschnitt 3.4.1. (Die Anforderung S-ack_safe(t) und damit auch S-ack_safe'(t) wird bereits durch das Transitionssystem erfüllt und ist in der komponentenorientierten Spezifikation daher nicht explizit aufgeführt.) □

Die globalen Anforderungen an das Postfachsystem lassen sich leicht und in natürlicher Weise den einzelnen Komponenten zuordnen, so daß dort ausschließlich lokale Anforderungen bestehen. Bei einer komponentenorientierten Spezifikation sind jedoch auch globale Anforderungen zugelassen. *Globale* Anforderungen richten sich an das gesamte verteilte System, d.h. an die Komposition aller Komponenten. Die parallele Komposition von Komponenten $Comp_1,\ldots,Comp_n$, bezeichnet mit $Comp_1 \parallel .. \parallel Comp_n$, ergibt ein Komponentennetz; dieses Netz kann ebenfalls als eine Komponente verstanden werden (mit evtl. leerer Eingabemenge). Die Semantik (I,O,T) des Netzes ergibt sich aus der Semantik der Bestandteile, $(I_1,O_1,T_1), \ldots, (I_n,O_n,T_n)$, wobei (I_i,O_i,T_i) die Semantik der Komponente $Comp_i$ sei:

$$O = O_1 \cup \ldots \cup O_n$$
$$I = (I_1 \cup \ldots \cup I_n) - O$$
$$T = \{t \in (I \cup O)^\omega \mid (I_k \cup O_k) \copyright t \in T_k \text{ für } k = 1,\ldots,n\}$$

Die parallele Komposition ist eine *partielle* Operation: sie ist nur dann definiert, wenn für alle $i \neq j$ aus $\{1,..,n\}$ gilt: $O_i \cap O_j = \emptyset$, d.h. wenn jede Ausgabeaktion eindeutig einer Komponente zugeordnet ist. Die parallele Komposition läßt sich dahingehend interpretieren, daß bei einem Netz zwischen zwei Komponenten dann eine (gerichtete) Verbindung besteht, wenn die Ausgabemenge der einen Komponente einen nichtleeren Schnitt mit der Eingabemenge der anderen hat.

Sind die einzelnen Komponentenspurmengen durch die Prädikate $C_1,\ldots,C_n$ gegeben, so ergibt sich die Anforderung an das zusammengesetzte System offensichtlich als

$$C(t) \equiv C_1((I_1 \cup O_1) \copyright t) \wedge \ldots \wedge C_n((I_n \cup O_n) \copyright t).$$

Eine Anforderung an ein Netz kann also im wesentlichen als die Konjunktion der Anforderungen an die Komponenten des Netzes verstanden werden. Dies rechtfertigt die Aufspaltung der globalen Anforderungen beim Postfachbeispiel. Der Operator $\parallel$ bietet einen

weiteren wichtigen Vorteil für unsere Methodik: Er ist die Grundlage für die kompositionale Weiterentwicklung der (Produkt-)Komponenten. Werden die Komponentenspezifikationen in Agentenbeschreibungen weiterentwickelt, so ist gewährleistet, daß im verteilten System nur Spuren aus T entstehen, wenn die Agenten nur Spuren aus $T_1, ..., T_n$ erzeugen (vgl. Kap. 5).[1]

Im Rahmen der Anforderungsspezifikation läßt sich die komponentenorientierte Spezifikation auf verschiedene Weise einsetzen. Zum einen kann der Kunde die Anforderungen an die einzelnen Komponenten explizit vorgeben. Die Spezifikation der Komponenten $C_1, ..., C_n$ ist konform zur globalen Spezifikation G, wenn gilt:

$$\forall t \in Act^{\omega}: \; C_1((I_1 \cup O_1) \copyright t) \wedge ... \wedge C_n((I_n \cup O_n) \copyright t) \Rightarrow G(t).$$

Im Postfachbeispiel wurde dieser Weg gewählt.

Zum anderen kann der Kunde neben den globalen Anforderungen allein angeben, auf welche (lokalen) Eigenschaften der Umgebungskomponenten sich der Systementwickler verlassen kann. An das Produkt werden hier also keine lokalen Anforderungen gestellt, wie der folgende Satz zeigt, sind dadurch allerdings die Anforderungen an das Produkt "in der Summe" eindeutig festgelegt. Jedoch sind noch verschiedene Lastverteilungen *zwischen* den Produktkomponenten möglich, was Raum für Entwurfsentscheidungen bietet (vgl. Kap. 5). Die Vorgabe der Umgebungsanforderungen ist deshalb nötig, da sich, wie Abschnitt 4.3 zeigen wird, die Produktanforderungen i.a. nicht allein aus den globalen Anforderungen und einer festgelegten Komponentenstruktur ableiten lassen.

Satz: Seien G (globale Anforderungen) und E (Umgebungsanforderungen) vorgegebene Prädikate über Spuren und sei $P;^{\sim}(t) \equiv \neg E(t) \vee G(t)$. Dann ist $P;^{\sim}(t)$ die größte Lösung von $P(t) \wedge E(t) \Leftrightarrow G(t)$.

Beweis: $P;^{\sim}$ ist eine Lösung: $(\neg E(t) \vee G(t)) \wedge E(t) \Leftrightarrow (\neg E(t) \wedge E(t)) \vee (G(t) \wedge E(t)) \Leftrightarrow (G(t) \wedge E(t)) \Leftrightarrow G(t)$, letzteres, da $G(t) \Rightarrow E(t)$.

"größte Lösung": Gelte $P'(t) \wedge E(t) \Leftrightarrow G(t)$ für ein P'. Zu zeigen: $P'(t) \Rightarrow P;^{\sim}(t)$. Gelte $P'(t)$. Falls außerdem $\neg E(t)$ gilt, dann auch $P;^{\sim}(t)$. Sonst gilt $E(t)$, dann gilt aber auch $G(t)$ und damit $P;^{\sim}(t)$.

□

Der Leser mag sich fragen, warum der Kunde nicht in jedem Fall gleich nur die Anforderungen an die Produktkomponenten angibt, wenn er schon lokale Anforderungen festlegen muß. Der Grund liegt darin, daß es in der Regel einfacher ist, Annahmen über die (häufig schon existierenden) Umgebungskomponenten offenzulegen, als die Schnittstelle der Produktkomponenten zu bestimmen. Ein typisches Beispiel, in dem allein globale

---

[1] In diesem Sinne kompositional ist auch der Versteckoperator *hide* (vgl. wiederum Kap. 5): Für eine Komponente C mit der Semantik (I,O,T) ist *hide* A *in* C definiert, falls $A \subseteq O$. Die Semantik von *hide* A *in* C ist dann das Tripel $(I, O - A, \{((I \cup O) - A) \copyright t \mid t \in T\})$.

Anforderungen und lokale Umgebungsanforderungen verwendet werden, um Produktanforderungen implizit festzulegen, ist die *Protokollspezifikation* (vgl. [Li, Maibaum 88]): Die *globale* Anforderung S_N ist der Dienst, den eine Schicht N in einer geschichteten Architektur eines Kommunikationssystems erbringen soll, die lokale Anforderung S_{N-1} der Dienst, den die Schicht N-1 bereitstellt. Die Produktkomponenten sind PE_1 und PE_2 *(protocol entities)*, vgl. das folgende Bild[1]:

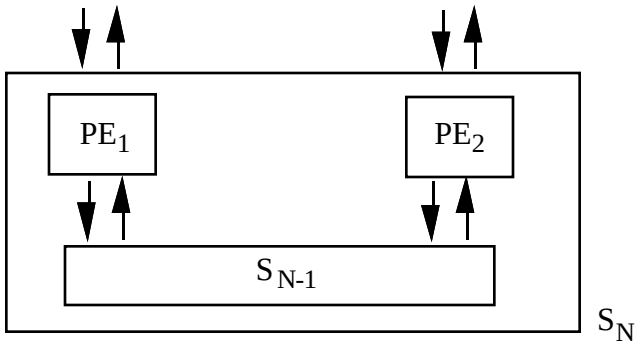


Fig. 4.2

Neben diesen beiden Extremfällen sind auch Mischformen erlaubt: neben einigen globalen Anforderungen und einigen lokalen an die Umgebungskomponenten liegen auch einige lokale Anforderungen an (einige) Produktkomponenten vor.

In jedem Fall hat eine komponentenorientierte Spezifikation also die Form

system Name
  global G(t)
  structure $Comp_1$ || ... || $Comp_n$
end

G sei die globale Anforderung für das Netz, $Comp_1$, ..., $Comp_n$ die Namen der Komponenten. Damit wird die Spurmenge $\{t \mid G(t) \wedge (I_k \cup O_k) \copyright t \in T_k$ für $k = 1,...,n\}$ beschrieben. (I_k,O_k,T_k) sei dabei die Semantik der Komponente $Comp_k$.

Die einzelnen Komponenten sind dabei spezifiziert durch:

spec $Comp_i$ (I_i,O_i)
  $C_i(t)$
end

I_i sei die Eingabemenge der Komponente $Comp_i$, O_i ihre Ausgabemenge, C_i die lokale Anforderung. Die lokalen Anforderungen an die die Umgebungskomponenten werden i.a. nicht

---

[1] Um eine vollständige formale Übereinstimmung zu meinem Vorgehensmodell zu erhalten, müßte das Bild noch um zwei Umgebungskomponenten PE_1' und PE_2' (Dienstbenutzer) ergänzt werden, die mit PE_1 bzw. PE_2 in Verbindung stehen; an PE_1' und PE_2' werden von Schicht N keine Anforderungen gestellt. Auf ähnliche Weise läßt sich jedes offene System schematisch in ein geschlossenes verwandeln.

gleich *true* sein (denn meist ist ein Produkt nicht in eine *beliebige* Umgebung eingebettet), die lokalen Anforderungen an die Produktkomponenten möglicherweise schon.

Wie ersichtlich ist, bietet die komponentenorientierte Spezifikation ein ganzes Spektrum an Möglichkeiten für die Anforderungsspezifikation. Es ist auch möglich, die Anforderungsspezifikation gleich mit einer komponentenorientierten Spezifikation zu beginnen statt mit einer globalen und gezielt Anforderungen an die einzelnen Komponenten zu richten. Der Einstiegspunkt in die Methodik kann somit flexibel gewählt werden.

Ein Ziel der Entwurfsspezifikation ist die vollständige Lokalisierung der Anforderungen an die (Produkt-) Komponenten; darauf gehe ich in Kap. 5 ein.

4.2 Aufspaltung globaler Sicherheitsanforderungen

In diesem und dem nächsten Abschnitt untersuche ich die Aufspaltung globaler Anforderungen in lokale. Zu diesem Zweck betrachte ich eine (globale) Anforderung, die über Ein- und Ausgaben einer Komponente spricht, und untersuche, welche Teilanforderung sich "in natürlicher Weise" an die *Komponente* richtet und welche an die *Umgebung* dieser Komponente, d.h. den Rest des Systems.[1] "Natürlich" heißt hier, daß die Komponente und ihre Umgebung ihre Teilanforderungen *realisieren* können sollen, d.h. die Einhaltung der Teilanforderungen garantieren können, unabhängig davon, wie sich der Partner verhält.

Diese Untersuchung läßt sich in zweierlei Weise interpretieren: Zum einen kann man sich "die Komponente" als das Produkt vorstellen, d.h. das Teilsystem aller Produktkomponenten, "die Umgebung" als das Teilsystem aller Umgebungskomponenten und untersucht wird die Frage: Läßt sich eine globale Spezifikation schematisch in Anforderungen an die Produkt- und an die Umgebungskomponenten aufspalten?

Zum anderen kann "die Komponente" eine der Komponenten des verteilten Systems sein, die bei der komponentenorientierten Spezifikation festgelegt wurden. Hier wird die Frage untersucht: Läßt sich eine globale Anforderung in eine lokale für die Komponente und eine möglicherweise immer noch globale, aber einfachere Anforderung an den Rest des verteilten Systems aufspalten?

Für diese Untersuchungen werden die Begriffe "Sicherheit" und "Lebendigkeit" für Komponenten, also offene Systeme, definiert. Diese Begriffe unterscheiden sich von denen für geschlossene Systeme und spiegeln die Ein-/Ausgabeorientierung einer Komponente wieder.

Sei im folgenden die Aktionenmenge Act die Vereinigung der Mengen Cact (*component actions, Komponentenaktionen*) und Eact (*environment actions, Umgebungsaktionen*). Cact enthalte die Ausgabeaktionen der Komponente, Eact enthält die Ausgabeaktionen der

---

[1] Auch die Umgebung läßt sich als eine "Komponente" des verteilten Systems auffassen. Der Einfachheit halber spreche ich im folgenden aber immer von *"der Komponente"* und *"der Umgebung"*.

Umgebung. Die Eingaben der Komponente seien die Ausgaben der Umgebung und umgekehrt. Dabei gelte $Cact \cap Eact = \emptyset$.

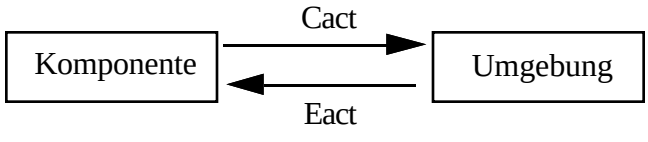


Fig. 4.3

In diesem Abschnitt betrachte ich zunächst allein Sicherheitseigenschaften, Lebendigkeitseigenschaften werden in Abschnitt 4.3 einbezogen.

4.2.1 Begriffsbildung und Aufspaltung

In Kap. 3 wurden allein *globale* Sicherheitsprädikate betrachtet. Für Komponenten, d.h. *offene* Systeme, muß die Definition eines Sicherheitsprädikats erweitert werden. Die Komponente soll ihre Sicherheitseigenschaft nur selbst, d.h. nur mit ihren eigenen Ausgaben, verletzen können. Dies läßt sich auf die folgende Weise formalisieren:

Definition (Sicherheitsprädikat für die Komponente): Ein Prädikat C über $(Eact \cup Cact)^\omega$ heißt *Sicherheitsprädikat für die Komponente*, wenn gilt:

C ist ein (globales) Sicherheitsprädikat über $(Eact \cup Cact)^\omega$ (1)

$C(t) \wedge e \in Eact \Rightarrow C(t \circ e)$ (2) □

Analog sind Sicherheitsprädikate für die Umgebung definiert.

Im folgenden interessieren wir uns für die Aufspaltung eines Sicherheitsprädikats S über $(Cact \cup Eact)^\omega$ in Sicherheitsprädikate für die Komponente C bzw. die Umgebung E. Eine *(teilweise) Aufspaltung* liegt vor, wenn gilt: $C(t) \wedge E(t) \Rightarrow S(t)$. Sie heißt *vollständig*, wenn sogar $C(t) \wedge E(t) \Leftrightarrow S(t)$ gilt. In diesem Abschnitt betrachten wir nur vollständige Aufspaltungen.

Beispiel: Sei $S(t) \equiv \forall s \in Act^*: s \sqsubseteq t \Rightarrow 0 \le \#(e \copyright s) - \#(c \copyright s) \le 1$, $Cact = \{c\}$, $Eact = \{e\}$

$C(t) \equiv \forall s \in Act^*: s \sqsubseteq t \Rightarrow \#(c \copyright s) \le \#(e \copyright s)$

$E(t) \equiv \forall s \in Act^*: s \sqsubseteq t \Rightarrow \#(e \copyright s) \le \#(c \copyright s) + 1$

Man beachte, daß diese Anforderungen den Prädikaten S-ack_safe und Snd_safe für das Postfachsystem in Abschnitt 3.3 entsprechen, wenn man von den konkreten Aktionen abstrahiert. C und E erfüllen die Definitionen von Sicherheitsprädikaten für die Komponente bzw. ihre Umgebung. Außerdem gilt $C(t) \wedge E(t) \Leftrightarrow S(t)$. □

Die Aufspaltung eines globalen Sicherheitsprädikats ist jedoch nicht eindeutig. Das zeigt das folgende Beispiel:

Beispiel: Man definiere

$C'(t) \equiv \forall s \in Act^*: s \sqsubseteq t \Rightarrow ((\forall r \in Act^*: r \sqsubset s \Rightarrow 0 \le \#(e \copyright r) - \#(c \copyright r) \le 1) \Rightarrow \#(c \copyright s) \le \#(e \copyright s))$

$E'(t) \equiv \forall s \in Act^*: s \sqsubseteq t \Rightarrow ((\forall r \in Act^*: r \sqsubset s \Rightarrow 0 \le \#(e \copyright r) - \#(c \copyright r) \le 1) \Rightarrow \#(e \copyright s) \le \#(c \copyright s) + 1)$

(Hierbei gelte: $s \sqsubset t \Leftrightarrow_{def} s \sqsubseteq t \wedge s \neq t$.) Auch C' und E' erfüllen die Definitionen von Sicherheitsprädikaten für die Komponente bzw. ihre Umgebung, sie stellen jedoch schwächere Anforderungen als C und E im obigen Beispiel dar. Die Komponente muß nur dann ihre Sicherheitsanforderung erfüllen, wenn sich ihre Umgebung bisher sicher verhalten hat, sonst darf sich die Komponente danach beliebig verhalten. Analoges gilt für die Umgebung. Es gilt jedoch:

$$C'(t) \wedge E'(t) \Leftrightarrow$$
$$\forall s \in Act^*: s \sqsubseteq t \Rightarrow ((\forall r \in Act^*: r \sqsubset s \Rightarrow 0 \leq \#(e \copyright r) - \#(c \copyright r) \leq 1) \Rightarrow$$
$$0 \leq \#(e \copyright s) - \#(c \copyright s) \leq 1)$$

und es ist leicht zu erkennen, daß dies ebenfalls zu S(t) äquivalent ist. □

In den nächsten beiden Sätzen zeige ich, daß sich globale Sicherheitsprädikate immer in solche für die Komponente und ihre Umgebung aufspalten lassen. Wie schon das obige Beispiel zeigt, gibt es i.a. nicht nur eine Dekomposition, vielmehr ergibt sich hierfür eine Bandbreite von Möglichkeiten, die durch eine "stärkste" und eine "schwächste" Aufspaltung begrenzt ist.

Satz (schwächste Aufspaltung): Sei S ein globales Sicherheitsprädikat, das nicht identisch *false* ist, und seien die Prädikate $C_{max}(S)$ und $E_{max}(S)$ folgendermaßen definiert:

$$C_{max}(S)(t) \equiv \forall s \in Act^*, c \in Cact: s \circ c \sqsubseteq t \wedge S(s) \Rightarrow S(s \circ c)$$
$$E_{max}(S)(t) \equiv \forall s \in Act^*, e \in Eact: s \circ e \sqsubseteq t \wedge S(s) \Rightarrow S(s \circ e)$$

Es gilt:

(1) $C_{max}(S)$ ist ein Sicherheitsprädikat für die Komponente,
$E_{max}(S)$ ist ein Sicherheitsprädikat für die Umgebung.

(2) $C_{max}(S)(t) \wedge E_{max}(S)(t) \Leftrightarrow S(t)$

(3) $C_{max}(S)$ und $E_{max}(S)$ sind die schwächsten Prädikate, für die (1) und (2) gilt.

Beweis: Zu (1): Es ist leicht erkennbar, daß $C_{max}(S)$ und $E_{max}(S)$ die Definition von Sicherheitsprädikaten für die Komponente bzw. ihre Umgebung erfüllen.

Zu (2): Es gilt: $C_{max}(S)(t) \wedge E_{max}(S)(t) \Leftrightarrow \forall s \in Act^*, a \in Act: s \circ a \sqsubseteq t \wedge S(s) \Rightarrow S(s \circ a)$.

$C_{max}(S)(t) \wedge E_{max}(S)(t)$ ist äquivalent zu: $S(\varepsilon) \Rightarrow \forall s \in Act^*: s \sqsubseteq t \Rightarrow S(s)$. Da $S(\varepsilon)$ gilt, ist dies äquivalent zu $\forall s \in Act^*: s \sqsubseteq t \Rightarrow S(s)$, was wiederum äquivalent zu S(t) ist, da S ein Sicherheitsprädikat ist.

Zu (3): Gelte $C(t) \wedge E(t) \Leftrightarrow S(t)$ mit Sicherheitsprädikaten C und E für die Komponente bzw. ihre Umgebung. Zu zeigen ist: $C(t) \Rightarrow C_{max}(S)(t)$

Gelte C(t). Zu zeigen ist: Es gilt $C_{max}(S)(t)$, d.h. $\forall s \in Act^*, c \in Cact: s \circ c \sqsubseteq t \wedge S(s) \Rightarrow S(s \circ c)$

Sei also $s \circ c \sqsubseteq t$ mit $s \in Act^*$ und $c \in Cact$ und gelte S(s). Falls $\neg S(s \circ c)$, dann also auch $\neg(E(s \circ c) \wedge C(s \circ c))$. Da S(s) gilt, ist auch E(s) wahr. Da $c \in Cact$ muß auch $E(s \circ c)$ gelten, daher $\neg C(s \circ c)$. Dann gilt aber auch $\neg C(t)$, da C Sicherheitsprädikat. Wir erhalten also einen Widerspruch zur Annahme. Der Beweis für E geht analog. □

Ist S identisch *false*, so gibt es mehrere unvergleichbare Aufspaltungen, nämlich alle jene, bei denen C oder E identisch *false* sind.

Neben dieser "schwächsten" Aufspaltung gibt es auch eine stärkste. Hierbei gibt die Komponente bzw. ihre Umgebung nichts mehr aus, wenn ihr Partner das globale Sicherheitsprädikat verletzt hat:

Satz (stärkste Aufspaltung): Sei S ein globales Sicherheitsprädikat und seien die Prädikate $C_{min}(S)$ und $E_{min}(S)$ folgendermaßen definiert:

$C_{min}(S)(t) \equiv t \in S \circ Eact^{\omega}$

$E_{min}(S)(t) \equiv t \in S \circ Cact^{\omega}$

Es gilt:

(1) $C_{min}(S)$ ist ein Sicherheitsprädikat für die Komponente,
$E_{min}(S)$ ist ein Sicherheitsprädikat für die Umgebung.

(2) $C_{min}(S)(t) \wedge E_{min}(S)(t) \Leftrightarrow S(t)$

(3) $C_{min}(S)$ und $E_{min}(S)$ sind die stärksten Prädikate, für die (1) und (2) gilt.

(*Anmerkung:* $\circ$ steht hier für die auf Mengen von Spuren erweiterte Konkatenation. Sie ist definiert durch: $t \in S \circ Eact^{\omega} \Leftrightarrow_{def} \exists r \in Act^{\omega}, s \in Eact^{\omega}: S(r) \wedge t = r \circ s$.)

Beweis: Zu (1): Ich zeige nur, daß $C_{min}(S)$ ein Sicherheitsprädikat für die Komponente ist, der Beweis für $E_{min}(S)$ geht analog.

Zu zeigen ist also zum einen: $C_{min}(S)(t) \Leftrightarrow (\forall s \in Act^{*}: s \sqsubseteq t \Rightarrow C_{min}(S)(s))$

"$\Rightarrow$": Gelte $C_{min}(S)(t)$. Dann gibt es ein r mit $S(r)$ und $s \in Eact^{\omega}$, so daß $t = r \circ s$. Mit $r \circ s$ ist auch jede Spur $r \circ s'$ mit $s' \sqsubseteq s$ in $S \circ Eact^{\omega}$, ebenso alle Präfixe von r, da S ein Sicherheitsprädikat ist.

"$\Leftarrow$": Seien alle endlichen Präfixe von t in $C_{min}(S)$. Falls alle Präfixe S erfüllen, so auch t, da S ein Sicherheitsprädikat ist, und daher gilt auch $C_{min}(S)(t)$. Falls ein Präfix in $(S \circ Eact^{\omega}) - S$ liegt, dann liegt auch t in $(S \circ Eact^{\omega}) - S$.

Die zweite Eigenschaft eines Sicherheitsprädikats für die Komponente, $C_{min}(S)(t) \wedge e \in Eact \Rightarrow C_{min}(S)(t \circ e)$, gilt per Definition.

Zu (2): Offensichtlich gilt $C_{min}(S) \wedge E_{min}(S) \Leftrightarrow S \circ Eact^{\omega} \wedge S \circ Cact^{\omega} \Leftrightarrow S$.

Zu (3): Nun ist noch zu zeigen: falls C ein Sicherheitsprädikat für die Komponente ist, das zusammen mit einem Sicherheitsprädikat für die Umgebung E die Gleichung $C \wedge E \Leftrightarrow S$ erfüllt, dann gilt: $C_{min}(S) \Rightarrow C$. Da wir nur Sicherheitsprädikate betrachten, müssen wir nur endliche Spuren berücksichtigen. Falls $C_{min}(S)(t)$ gilt, so gibt es $r \in Act^{*}$, $s \in Eact^{*}$ mit $r \circ s \sqsubseteq t$ und $S(r)$. Mit $S(r)$ gilt auch $C(r)$ und nach #s-maliger Anwendung der Regel $C(t) \wedge e \in Eact \Rightarrow C(t \circ e)$ auch $C(r \circ s)$. □

Der folgende Satz zeigt, daß die Konjunktion von Sicherheitsprädikaten für die Komponente und ihre Umgebung, die aus unterschiedlichen Aufspaltungen resultieren, das globale Sicherheitsprädikat ergibt:

Satz: Sei S ein globales Sicherheitsprädikat, seien C und C' Sicherheitsprädikate für die Komponente, E und E' Sicherheitsprädikate für die Umgebung; alle diese Prädikate seien nicht identisch *false*.

Wenn $C(t) \wedge E(t) \Leftrightarrow S(t)$ und $C'(t) \wedge E'(t) \Leftrightarrow S(t)$, dann auch $C(t) \wedge E'(t) \Leftrightarrow S(t)$.

Beweis: "$\Rightarrow$": Gelte $C(t)$ und $E'(t)$. Falls $\neg E(t)$, so gibt es $s \in Act^{*}$, $e \in Eact$ mit $s \circ e \sqsubseteq t$ und $E(s)$, aber $\neg E(s \circ e)$. Mit $C(t)$ und $E(s)$ gilt auch $S(s)$, also auch $C'(s)$. Da $e \in Eact$ gilt $C'(s \circ e)$, mit $E'(t)$ also $S(s \circ e)$ - Widerspruch zu $\neg E(s \circ e)$.

"$\Leftarrow$": trivial □

Es sei nochmals betont, daß sich die obigen Aussagen zur Aufspaltung einer globalen Sicherheitsanforderung auf den Sonderfall beziehen, daß nur zwei wechselseitig rückgekoppelte Komponenten betrachtet werden. Wie das folgende Beispiel zeigt, ist die Aufspaltung einer globalen Sicherheitsanforderung, die sich an mehr als zwei nicht direkt miteinander verbundene Komponenten richtet, dagegen i.a. nicht mehr auf schematische Weise durchführbar.

Beispiel: Betrachten wir ein Netz von Komponenten, für das die globale Sicherheitsanforderung $S(t) \equiv \forall s \in Act^*: s \sqsubseteq t \Rightarrow \#(D \copyright s) \leq \#(A \copyright s)$ besteht.

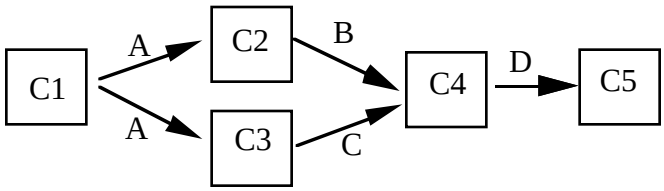


Fig. 4.4

Da zwischen den Komponenten C1 und C4 keine direkte Verbindung besteht, müssen die Komponenten C2 und/oder C3 als "Vermittler" beteiligt werden. Ob C2 oder C3 (oder beide) dazu verwendet werden, ist jedoch eine Entwurfsentscheidung:

a) C2 vermittelt: Stellt man die Anforderung $C2(t) \wedge C4(t)$ mit

$C2(t) \equiv \quad \forall s \in Act^*: s \sqsubseteq t \Rightarrow \#B \copyright s \leq \#A \copyright s$

$C4(t) \equiv \quad \forall s \in Act^*: s \sqsubseteq t \Rightarrow \#D \copyright s \leq \#B \copyright s,$

so ist $S(t)$ eine Folgerung von $C2(t) \wedge C4(t)$.

b) C3 vermittelt: Analoges gilt, wenn man $C3(t) \wedge C4'(t)$ mit

$C3(t) \equiv \quad \forall s \in Act^*: s \sqsubseteq t \Rightarrow \#C \copyright s \leq \#A \copyright s$

$C4'(t) \equiv \quad \forall s \in Act^*: s \sqsubseteq t \Rightarrow \#D \copyright s \leq \#C \copyright s$

betrachtet. □

4.2.2 Zusammenhang mit dem Annahme/Verpflichtung-Stil

In diesem Abschnitt werden Untersuchungsergebnisse des letzten Abschnitts mit dem sog. *Annahme/Verpflichtung-Stil (rely/guarantee style, assumption/commitment style)* in Verbindung gebracht. Er findet sich in den Arbeiten [Jones 83], [Misra, Chandy 81] und [Zwiers et al. 84] und erhält in der letzten Zeit wieder verstärktes Interesse (vgl. [Pandya 90], [Abadi, Lamport 90]).

Diesem Spezifikationsstil liegt die folgende Sicht einer Komponente zugrunde: Falls die Umgebung der Komponente bestimmte Bedingungen einhält *("rely")*, so garantiert die Komponente selbst bestimmte (andere) Bedingungen *("guarantee")*. Die Annahmen, die eine Komponente über ihre Umgebung macht, werden explizit modelliert. Man kann zustandsorientierte Annahme/Verpflichtung-Spezifikationen als eine Verallgemeinerung der Spezifikation mit Vor- und Nachbedingungen sehen (vgl. [Jones 83]). Eine Verbindung besteht auch zu spieltheoretischen Ansätzen der Modellierung reaktiver Systeme (vgl. Abschnitt 4.3 sowie [Abadi, Lamport 90]). Annahmen und Verpflichtungen legen in diesem Fall eine Spielregel für eine Komponente fest.

In meinem Vorgehensmodell werden auch Annahmen über die Umgebung wiedergegeben, nämlich dadurch, daß ich die Umgebung in das verteilte System einbeziehe. Ich schildere, wie man von unabhängigen Sicherheitsanforderungen an die Komponente und ihre Umgebung zu schwächeren Sicherheitsanforderungen für diese beiden Komplexe gelangt, wenn Annahmen über das Verhalten des Partners berücksichtigt werden.

Das folgende Beispiel verdeutlicht die Abhängigkeit "Die Komponente muß sich nur dann sicher verhalten (d.h. gemäß ihrem Sicherheitsprädikat), wenn sich ihre Umgebung sicher verhält":

Beispiel: Man betrachte das Sicherheitsprädikat

$$S(t) \equiv \forall s \in Act^*: s \sqsubseteq t \Rightarrow 0 \leq \#(e \copyright s) - \#(c \copyright s) \leq 1,$$

das im vorigen Abschnitt angegeben wurden. Dabei sei $Act = \{c,e\}$.
Der Sicherheitsanteil für die Komponente wurde dort als

$$C'(t) \equiv \forall s \in Act^*: s \sqsubseteq t \Rightarrow ((\forall r \in Act^*: r \sqsubset s \Rightarrow 0 \leq \#(e \copyright r) - \#(c \copyright r) \leq 1) \Rightarrow \#(c \copyright s) \leq \#(e \copyright s))$$

angegeben. Alternativ kann man jedoch auch formulieren:

$$C(t) \equiv \forall s \in Act^*: s \sqsubseteq t \Rightarrow ((\forall r \in Act^*: r \sqsubseteq s \Rightarrow \#(e \copyright r) - \#(c \copyright r) \leq 1 \Rightarrow \#(c \copyright s) \leq \#(e \copyright s))$$

Man sieht leicht, daß beide Prädikate die durch den folgenden (erweiterten) regulären Ausdruck gegebene Menge beschreiben:

$$(ec)^\omega \cup (ece)^\omega \cup (ec)^\omega ee(e \cup c)^\omega$$

In C' ist die Reaktion der Komponente von der globalen Sicherheit der Vorgeschichte abhängig, in C dagegen nur vom sicheren Verhalten der Umgebung. Man sieht jedoch leicht, daß die beiden Fälle äquivalent sind: Wenn ein Teilablauf global sicher ist, so hat sich insbesondere die Komponente sicher verhalten. Wenn sich umgekehrt die Umgebung der Komponente immer sicher verhält und daraufhin auch die Komponente selbst, so entstehen nur sichere Abläufe. Wenn sich die Umgebung nicht sicher verhält, so ist in beiden Fällen beliebiges Verhalten der Komponente erlaubt. □

Hat man ein Sicherheitsprädikat in der Form

$$\forall s \in Act^*: s \sqsubseteq t \Rightarrow C(s) \wedge E(s)$$

gegeben, wobei die Prädikate C und E die Anforderungen

$$C(t) \wedge e \in Eact \Rightarrow C(t \circ e),\ E(t) \wedge c \in Cact \Rightarrow E(t \circ c) \text{ sowie } C(\varepsilon) = E(\varepsilon) = true \qquad (*)$$

erfüllen, so sieht man leicht, daß $\forall s \in Act^*: s \sqsubseteq t \Rightarrow C(s)$ und $\forall s \in Act^*: s \sqsubseteq t \Rightarrow E(s)$ Sicherheitsprädikate für die Komponente bzw. ihre Umgebung sind. Solche Sicherheitsprädikate lassen sich unschwer in eine Annahme/Verpflichtung-Spezifikation umwandeln. Wir verstehen darunter Prädikate $C_{rg}(C,E)$ und $E_{rg}(C,E)$, die wie folgt definiert sind:

Definition (Annahme/Verpflichtung-Spezifikation): Seien C und E Prädikate über Spuren. Die Prädikate $C_{rg}(C,E)$ und $E_{rg}(C,E)$ sind folgendermaßen definiert:

$$C_{rg}(C,E)(t) \equiv \forall s \in Act^*: s \sqsubseteq t \Rightarrow ((\forall r \in Act^*: r \sqsubseteq s \Rightarrow E(r)) \Rightarrow C(s))$$
$$E_{rg}(C,E)(t) \equiv \forall s \in Act^*: s \sqsubseteq t \Rightarrow ((\forall r \in Act^*: r \sqsubseteq s \Rightarrow C(r)) \Rightarrow E(s)). \quad \square$$

Insbesondere gilt mit der Annahme (*): $C_{rg}(C,E)(\varepsilon) = \mathit{true}$ und $E_{rg}(C,E)(\varepsilon) = \mathit{true}$, da $C(\varepsilon) = E(\varepsilon) = \mathit{true}$. Man beachte, daß C und E selbst nicht unbedingt Sicherheitsprädikate sind: Zwar ist die Bedingung (2) der Charakterisierung (vgl. Abschnitt 4.2.1) erfüllt, nicht unbedingt jedoch Bedingung (1).

Man erkennt beim Prädikat $C_{rg}(C,E)(t)$ die *Annahme ("rely")* $\forall r \in Act^*: r \sqsubseteq s \Rightarrow E(r)$ und die *Verpflichtung ("guarantee")* $C(s)$. Im Gegensatz zu den am Anfang des Abschnitts erwähnten Arbeiten läßt sich hier somit ein vernünftiges semantisches Kriterium dafür angeben, wann eine Eigenschaft eine Annahme oder eine Verpflichtung darstellt (bezogen auf eine Komponente): die Verletzung einer Annahme ist nur durch die Umgebung der Komponente möglich, die Verletzung einer Verpflichtung nur von der Komponente selbst.

Der folgende Satz stellt eine Verbindung zwischen Annahme/Verpflichtung-Spezifikationen gemäß der obigen Definition und schwächsten Sicherheitsprädikaten (vgl. Abschnitt 4.2.1) für die Komponente bzw. ihre Umgebung her. Für ein globales Sicherheitsprädikat kann es mehrere Aufspaltungen der Form $\forall s \in Act^*: s \sqsubseteq t \Rightarrow C(s) \wedge E(s)$ geben (vgl. die Beispiele in Abschnitt 4.2.1). Der Satz besagt, daß man äquivalente Annahme/Verpflichtung-Spezifikationen erhält, einerlei wie man die Aufspaltung des globalen Sicherheitsprädikats wählt. Diese sind äquivalent zu den schwächsten Anforderungen an die Komponente, die sich aus dem globalen Sicherheitsprädikat ableiten lassen.

Satz: Mögen die Prädikate C und E die unter (*) angeführten Bedingungen erfüllen und seien C_{rg} und E_{rg} wie oben definiert. Sei weiterhin $S(t) \equiv \forall s \in Act^*: s \sqsubseteq t \Rightarrow C(s) \wedge E(s)$. Es gilt:

$$C_{rg}(C,E) = C_{max}(S), \quad E_{rg}(C,E) = E_{max}(S).$$

Beweis: Im Beweis wird die Abkürzung C_{rg} für $C_{rg}(C,E)$ verwendet. Er wird nur für die Behauptung $C_{rg}(C,E) = C_{max}(S)$ geführt, der zweite Teil geht analog.
"$\Rightarrow$": Gelte $C_{rg}(t)$. Zu zeigen ist $C_{max}(t)$, d.h. $\forall s \in Act^*, c \in Cact: s \circ c \sqsubseteq t \wedge S(s) \Rightarrow S(s \circ c)$. Gezeigt wird dies induktiv über die Präfixe von t mit Länge ≥ 1.
Induktionsanfang: Sei $c \sqsubseteq t$ mit $c \in Cact$ und gelte $S(\varepsilon)$. Zu zeigen ist $S(c)$. Da $c \in Cact$ gilt $E(c)$ gemäß den Anforderungen an E. Mit $C_{rg}(t)$ und $E(\varepsilon)$ und $E(c)$ ist auch $C(c)$ erfüllt, also gilt $\forall s \in Act^*: s \sqsubseteq c \Rightarrow C(s) \wedge E(s)$, was gleich $S(c)$ ist.
Induktionsschritt: Sei $q \circ b \sqsubseteq t$ mit $q \in Act^*$ und $b \in Act$ und sei die Aussage bereits für alle Präfixe von q bewiesen. Es genügt zu zeigen: $S(q) \wedge b \in Cact \Rightarrow S(q \circ b)$. $S(q \circ b) = \forall s \in Act^*: s \sqsubseteq q \circ b \Rightarrow C(s) \wedge E(s)$. Mit $S(q)$ gilt bereits $\forall s \in Act^*: s \sqsubseteq q \Rightarrow C(s) \wedge E(s)$. Es bleibt zu zeigen: $C(q \circ b) \wedge E(q \circ b)$. Aus $S(q)$ und damit $E(q)$ folgt mit $b \in Cact$ die Aussage $E(q \circ b)$. Aus $S(q)$ folgt auch $\forall r \in Act^*: r \sqsubseteq q \Rightarrow E(r)$, daher gilt mit $E(q \circ b)$ und $C_{rg}(t)$ auch $C(q \circ b)$.
"$\Leftarrow$": Gelte $C_{max}(t)$. Zu zeigen: $C_{rg}(t)$ ist auch erfüllt. Gezeigt wird dies wiederum durch Induktion über die Länge der Präfixe s von t.

Induktionsanfang: $s = \varepsilon$: $C_{rg}(\varepsilon)$ gilt.
Induktionsschritt: Gelte die Aussage für alle endlichen Präfixe von s mit $s \sqsubseteq t$. Nun soll sie für $s \circ b \sqsubseteq t$ mit $b \in Cact$ gezeigt werden. Es muß nur noch $(\forall r \in Act^*: r \sqsubseteq s \circ b \Rightarrow E(r)) \Rightarrow C(s \circ b)$ gezeigt werden. Gelte $\forall r \in Act^*: r \sqsubseteq s \circ b \Rightarrow E(r)$. Zu zeigen: $C(s \circ b)$ gilt. Da gemäß der Induktionsannahme für alle $r \sqsubseteq s$ die Aussage $C_{rg}(r)$ gilt, ist auch $\forall r \in Act^*: r \sqsubseteq s \Rightarrow C(r)$ erfüllt. Insgesamt gilt daher $\forall r \in Act^*: r \sqsubseteq s \Rightarrow C(r) \wedge E(r)$, was gleich $S(s)$ ist. Falls $b \in Cact$, so gilt mit $C_{max}(t)$ auch $C(s \circ b)$. Falls $b \in Eact$, so ist auch in diesem Fall $C(s \circ b)$ erfüllt, da $C(s)$ gilt. □

Gemäß der Definition von $C_{rg}(C,E)$ muß sich die Komponente an einer Stelle im Ablauf korrekt verhalten, wenn sich auch ihre Umgebung bis genau zu dieser Stelle korrekt verhalten hat. Tatsächlich ist dies gleichwertig zu der Forderung, daß sich die Komponente nur dann korrekt verhalten muß, wenn sich die Umgebung bis "einen Schritt vorher" (ausgedrückt durch $\sqsubset$ statt $\sqsubseteq$) korrekt verhalten hat. Man beachte, daß dies die in [Misra, Chandy 81] verwendete Modellvorstellung ist. Dort werden jedoch keine Kriterien angegeben, wann ein Prädikat als Anforderung an die Komponente bzw. ihre Umgebung zu verstehen ist.

Satz: Mögen die Prädikate C und E die unter (*) angeführten Bedingungen erfüllen und seien C_{rg} und E_{rg} wie oben definiert. Es gilt:

$C_{rg}(C,E)(t)$ ist äquivalent zu $\forall s \in Act^*: s \sqsubseteq t \Rightarrow ((\forall r \in Act^*: r \sqsubset s \Rightarrow E(r)) \Rightarrow C(s))$,
$E_{rg}(C,E)(t)$ ist äquivalent zu $\forall s \in Act^*: s \sqsubseteq t \Rightarrow ((\forall r \in Act^*: r \sqsubset s \Rightarrow C(r)) \Rightarrow E(s))$.

Beweis: Der Beweis erfolgt nur für die Anforderung an die Komponente, der zweite Teil geht analog. $\forall s \in Act^*: s \sqsubseteq t \Rightarrow ((\forall r \in Act^*: r \sqsubset s \Rightarrow E(r)) \Rightarrow C(s))$ wird mit $C'(t)$ abgekürzt.
"$\Rightarrow$": Gelte $C_{rg}(C,E)(t)$. Zu zeigen ist, daß $C'(t)$ gilt. Sei $s \in Act^*$ mit $s \sqsubseteq t$. Gelte weiterhin $\forall r \in Act^*: r \sqsubset s \Rightarrow E(r)$. Zu zeigen ist dann $C(s)$. Falls $s = \varepsilon$, so gilt $C(s)$ per Definition. Sonst hat s die Form $s = q \circ a$ mit $q \in Act^*$ und $a \in Act$.
Fall 1: $a \in Cact$. Dann gilt mit $E(q)$ (dies gilt, da $q \sqsubset s$) auch $E(q \circ a)$ nach den geforderten Eigenschaften von E und somit $\forall r \in Act^*: r \sqsubseteq s \Rightarrow E(r)$. Mit $C_{rg}(C,E)(t)$ gilt dann auch $C(s)$.
Fall 2: $a \in Eact$. Mit $\forall r \in Act^*: r \sqsubset s \Rightarrow E(r)$, was äquivalent zu $\forall r \in Act^*: r \sqsubseteq s \Rightarrow E(r)$ ist, und $C_{rg}(C,E)(t)$ ist auch $C(s)$ erfüllt. Mit $a \in Eact$ gilt auch $C(s \circ a)$.
"$\Leftarrow$": offensichtlich □

Bei Annahme/Verpflichtung-Spezifikationen ist der folgende Satz zu beachten:

Satz: Seien C und E Prädikate, die (*) erfüllen. Wenn

$$\forall s \in Act^*: s \sqsubseteq t \Rightarrow ((\forall r \in Act^*: r \sqsubseteq s \Rightarrow E(r)) \Rightarrow C(s)) \qquad (1)$$

für t gilt, dann ist auch

$$(\forall s \in Act^*: s \sqsubseteq t \Rightarrow E(s)) \Rightarrow (\forall s \in Act^*: s \sqsubseteq t \Rightarrow C(s)) \qquad (2)$$

für t erfüllt, aber i.a. nicht umgekehrt.
Beweis: offensichtlich
Gegenbeispiel für die Gegenrichtung: Sei $Cact = \{c\}$, $Eact = \{e\}$ und $C(\varepsilon) = E(\varepsilon) =$ *true*, $C(c) =$ false, $E(c) =$ *true*, $C(c \circ e) =$ *false*, $E(c \circ e) =$ *false* und $t = c \circ e$. Für $s = c$ gilt nicht $E(s)$

$\Rightarrow$ C(s). Daher ist (1) falsch, die (2) aber wahr. (1) $\Rightarrow$ (2) gilt also nicht, wenn sowohl C als auch E irgendwann *false* werden, C aber früher. □

Eine analoge Aussage gilt offensichtlich, wenn die Komponente und ihre Umgebung die Rollen tauschen.

Offensichtlich lassen sich aber (1) und (2) nur auf die gleiche Weise realisieren: Die Komponente kann nicht darauf vertrauen, daß dann, wenn sie selbst einen "Fehler" begeht, es ihr die Umgebung danach gleichtut. Dieses Phänomen eines gleichen *realisierbaren Anteils* läßt sich mit dem Begriffsapparat des folgenden Abschnitts beschreiben und analysieren.

4.3 Einbeziehung von Lebendigkeitsanforderungen

In Abschnitt 4.2 wurden allein Sicherheitseigenschaften untersucht: der Begriff der Sicherheitseigenschaft für eine Komponente wurde definiert, die Aufspaltung einer globalen Sicherheitsanforderung in solche für die Teilsysteme "Komponente" und "Umgebung" wurde angegangen. Beim Versuch, dies auch für Lebendigkeitseigenschaften zu erzielen, stößt man auf Schwierigkeiten. Diese illustriere ich zunächst anhand einiger Beispiele; sie motivieren die Analyse von Komponentenspurmengen durch Strategien, die ich anschließend durchführe.

4.3.1 Motivation für die Analyse durch Strategien

Zunächst einige Beispiele zur Aufspaltung globaler Lebendigkeitseigenschaften: Ich stelle jeweils eine globale Lebendigkeitseigenschaft L vor und zeige, welche Aufspaltung in Anforderungen an die Komponente C und die Umgebung E plausibel erscheint.

Beispiele:

a) $L(t) \equiv \#t \neq \infty$.

Diese Anforderung (Terminierung) ist äquivalent zu: $L'(t) \equiv \#(Cact \copyright t) \neq \infty \wedge \#(Eact \copyright t) \neq \infty$. Offensichtlich ist L' nur dann erfüllbar, wenn die Komponente nur endlich viele Aktionen aus Cact ausführt, ihre Umgebung nur endlich viele Aktionen aus Eact. Die Lebendigkeitsanforderungen an die Komponente bzw. ihre Umgebung lauten entsprechend:

$C(t) \equiv \#(Cact \copyright t) \neq \infty$, $E(t) \equiv \#(Eact \copyright t) \neq \infty$.

b) $L(t) \equiv \forall p \in Act^*, q \in Act^\omega: t = p \bullet e \bullet q \Rightarrow c \; in \; q$

Dabei sei angenommen, daß $e \in Eact$, $c \in Cact$. Hier läßt sich L folgendermaßen aufspalten:

$C(t) \equiv L(t)$, $E(t) \equiv true$.

c) $L(t) \equiv \#t = \infty$

Dies ist äquivalent zu: $L'(t) \equiv \#(Cact \copyright t) = \infty \vee \#(Eact \copyright t) = \infty$.

Dieser Fall ist schwieriger zu behandeln als die vorigen. Damit L' erfüllt wird, muß zumindest einer der beiden Partner unendlich viele Ausgaben machen. Betrachtet man die zu L' äquivalente Formulierung

$L''(t) \equiv (\#(Cact \copyright t) = \infty \wedge \#(Eact \copyright t) \neq \infty) \vee (\#(Cact \copyright t) \neq \infty \wedge \#(Eact \copyright t) = \infty)$
$\vee (\#(Cact \copyright t) = \infty \wedge \#(Eact \copyright t) = \infty)$,

so erkennt man, daß sich alle drei Teilprädikate aufspalten lassen. Man könnte von dem *"worst case"* ausgehen und fordern, daß *beide* Partner unendlich viele Ausgaben machen. Dies ist beim nächsten Beispiel nicht möglich:

d) $L(t) \equiv \#(Cact \copyright t) = \infty \Leftrightarrow \#(Eact \copyright t) \neq \infty$

Hier bestehen (zumindest) zwei Möglichkeiten zur Aufspaltung:

$C_1(t) \equiv \#(Cact \copyright t) = \infty$, $E_1(t) \equiv \#(Eact \copyright t) \neq \infty$

$C_2(t) \equiv \#(Cact \copyright t) \neq \infty$, $E_2(t) \equiv \#(Eact \copyright t) = \infty$

Es ergibt sich $L(t) \equiv (C_1(t) \wedge E_1(t)) \vee (C_2(t) \wedge E_2(t))$

Hier können die Komponente und ihre Umgebung das Lebendigkeitsprädikat nicht getrennt erfüllen, so daß eine zusätzliche Absprache nötig ist. □

Die Beispiele zeigen, daß es eine naheliegende Aufspaltung eines Lebendigkeitsprädikats in Lebendigkeitsprädikate für die Komponente und ihre Umgebung geben kann. Andererseits zeigen c) und d), daß manchmal eine zusätzliche Absprache nötig ist.

Welche Eigenschaft wurde bei den obigen Aufspaltungen für ein Lebendigkeitsprädikat für eine Komponente, also ein offenes Teilsystem, implizit gefordert? Die intuitive Anforderung ist, daß die Komponente jeden endlichen Ablauf zu einem lebendigen machen kann, unabhängig davon, wie sich ihre Umgebung verhält. (Zur Erinnerung: ein *globales* Lebendigkeitsprädikat forderte, daß jeder endliche Ablauf *überhaupt*, d.h. im Zusammenspiel der Komponente mit ihrer Umgebung, zu einem lebendigen erweitert werden kann.) Dabei muß das operationelle Zusammenspiel der Komponente mit ihrer Umgebung berücksichtigt werden.

Gemäß unserer Modellvorstellung kommunizieren Komponenten asynchron mittels Ein- und Ausgaben mit ihrer Umgebung. Die Eingaben einer Komponente sind kausal für ihre Ausgaben und können nicht von der Komponente beeinflußt werden. Genauer gesagt kann sie weder beeinflussen *welche* Eingaben kommen, noch *wann* Eingaben kommen. Charakterisierungen derartiger Systeme, sog. *reaktiver Systeme* (vgl. Kap. 2), und derer erzeugten Spuren sind Gegenstand mehrerer Arbeiten: Bei dem spieltheoretischen Ansatz in [Broy et al. 91a] modelliert eine Spurmenge genau dann eine Komponente, wenn sie durch eine *Strategie* bzw. eine Menge von Strategien realisiert wird. (Der Begriff der Strategie wird weiter unten definiert.) Dort wird auch eine deskriptive, explizitere Charakterisierung solcher Spurmengen versucht, die jedoch das Wesen eines reaktiven Systems nur unvollständig erfaßt. Ein ähnlicher Ansatz findet sich in [Dill 89], wo die stete Empfangsbereitschaft von Schaltungen modelliert wird. Auch in [Moschovakis 89], [Nerode et al. 90] und [Abadi, Lamport 90] werden verteilte Systeme spieltheoretisch durch Strategien beschrieben. Die ersten beiden Arbeiten sind auf die Semantik paralleler Programme ausgerichtet, die verwendete Modellierung unterscheidet sich von der hier betrachteten. Der Ansatz in [Abadi, Lamport 90] ist zustandsorientiert, die Komposition von Spezifikationen steht im Vordergrund. In [Jonsson 87] werden I/O-Automaten und ihre Spuren (vgl. Abschnitt 5.2) zur Beschreibung reaktiver Systeme vorgeschlagen.

All diesen Ansätzen ist gemein, daß operationelle Vorstellungen verwendet werden, um reaktive Systeme und ihre Spuren zu modellieren und zu analysieren. Im Gegensatz zu den erwähnten Arbeiten interessiert in der vorliegenden Arbeit die Aufspaltung globaler Anforderungen in solche an eine Komponente und ihre Umgebung sowie der (noch nicht formal definierte) Begriff der Lebendigkeitseigenschaft für eine Komponente.

Im folgenden führe ich mit dem Begriffsapparat aus [Broy et al. 91a] Untersuchungen zu diesen beiden Punkten durch. Zunächst wird eine einzelne Komponente im Zusammenspiel mit einer Umgebung, die sich beliebig verhalten darf, modelliert (Abschnitt 4.3.2), dann das Zusammenspiel mehrerer Komponenten, die sich alle an gewisse Spielregeln, verkörpert durch Strategien, halten (Abschnitt 4.3.3). In Abschnitt 4.3.4 werden die Ergebnisse zur Aufspaltung globaler Anforderungen zusammengefaßt, wobei auch die Resultate aus Abschnitt 4.2 einfließen.

4.3.2 Modellierung einer Komponente

Wie in Abschnitt 4.2 betrachten wir wieder eine Komponente mit Eingabemenge Eact und Ausgabemenge Cact. Sie bekommt fortwährend Eingaben in beliebig großen, aber endlichen "Portionen" (jede Eingabeportion wird durch ein Element aus $Eact^*$ modelliert) und erhält nach jeder Eingabeportion die Möglichkeit zu einer Ausgabe (modelliert durch ein Element aus $Cact \cup \{\varepsilon\}$). Kommt einige Male nacheinander keine Eingabe bzw. die "leere" Eingabe, modelliert durch den leeren Strom ε, so kann die Komponente mehrere Ausgaben nacheinander machen. Die Unsymmetrie zwischen Eingaben aus $Eact^*$ und Ausgaben aus $Cact \cup \{\varepsilon\}$ (und nicht $Cact^*$) modelliert, daß die Komponente nicht die Eingaberate beeinflussen kann. Die Eingaben können beliebig schnell kommen; die einzige Einschränkung ist, daß die Eingaben nicht unendlich schneller kommen als die Komponente darauf mit Ausgaben reagieren kann; hin und wieder bekommt die Komponente also die Möglichkeit zu einer Ausgabe.

Das Verhalten einer Komponente läßt sich mit dem Begriff der Strategie modellieren ([Broy et al. 91a]):

Definition (Strategie): Eine *Strategie* für eine Komponente mit Eingabemenge Eact und Ausgabemenge Cact ist eine Abbildung der Funktionalität $(Eact \cup Cact)^* \rightarrow Cact \cup \{\varepsilon\}$. □

Sei C eine Strategie; $C(w) \in Cact$ bedeutet, daß die Komponente bei der Vorgeschichte w eine Ausgabeaktion macht, $C(w) = \varepsilon$ dagegen, daß die Komponente auf w nichts ausgibt. Die Vorgeschichte ist die bisher erzeugte Teilspur (s.u.).

Die Eingabe für die Komponente wird mit dem Begriff der Eingabemodellierung formalisiert:

Definition (Eingabemodellierung): Eine *Eingabemodellierung* für die Menge Eact ist eine Abbildung der Funktionalität $Nat \rightarrow Eact^*$. □

Sei e eine Eingabemodellierung; e(n) ist die Eingabe für die Komponente beim n-ten Schritt.

Wir betrachten das Wechselspiel der Komponente mit einer beliebigen Eingabemodellierung.

Definition (Spuren einer Strategie): Die Spurmenge einer Strategie C, Traces(C), ist folgendermaßen definiert:

$$Traces(C) =_{def} \{trace(C,e) \mid e \in Nat \rightarrow Eact^*\},$$

wobei trace(C,e) definiert ist durch:

$$trace(C,e) =_{def} \bigsqcup_n ptrace(C,e,n)$$

und ptrace(C,e,n):

ptrace(C,e,n) =def \B\LC\{(\A\AL\CO1(e(0),
falls n = 0; ptrace(C,e,n-1) ◦ C(ptrace(C,e,n-1)) ◦ e(n), sonst)) □

ptrace(C,e,n) gibt die Teilspur *(partial trace)* an, die von der Strategie C bei der Eingabemodellierung e nach der n-ten Eingabe erzeugt wird. trace(C,e) ist die Spur, die mit der Strategie C bei der Eingabemodellierung e erzeugt wird. Traces(C) enthält diejenigen Spuren, die von der Strategie C bei beliebigen Eingabemodellierungen erzeugt werden.

Man beachte, daß für eine Eingabemodellierung e: Nat→Eact* mit Eact = ∅ für alle n die Gleichung e(n) = ε gilt, da Eact* = ∅* = {ε}. Analog gilt für eine Strategie C: (Eact ∪ Cact)* → Cact ∪ {ε} mit Cact = ∅ für alle w die Gleichung C(w) = ε. Ist sowohl Eact als auch Cact leer, so erzeugt eine solche Strategie die Spurmenge {ε}, was auch unserer intuitiven Vorstellung entspricht.

In [Broy et al. 91a] sowie in der vorliegenden Arbeit wird von der Zugfolge abstrahiert, wobei ein Zug der Umgebung nach unserer spieltheoretischen Vorstellung aus einem Wort aus Eact* besteht, ein Zug der Komponente aus einem Element aus Cact ∪ {ε}. Dies bedeutet, zwei Zugfolgen ‹e_1,c_1,...,e_m› (mit $e_j \in$ Eact* und $c_j \in$ Cact ∪ {ε}) und ‹e_1',c_1',...,e_n'› liefern die gleiche Vorgeschichte, d.h. die gleiche Teilspur, wenn $e_1 \circ c_1 \circ ... \circ e_m = e_1' \circ c_1' \circ ... \circ e_n'$. In [Dill 89] wird ein etwas anderer Strategiebegriff verwendet, bei dem die Zugreihenfolge berücksichtigt wird. Eine Strategie ist dort eine Abbildung der Funktionalität (Eact*)* → (Cact ∪ {ε}).

Eine Strategie beschreibt ein *deterministisches* Verhalten einer Komponente. Zur Modellierung von *nichtdeterministischem* Verhalten werden *Mengen* von Strategien verwendet. Dem liegt die Modellvorstellung zugrunde, daß die Komponente am Anfang eines Ablaufs eine Strategie auswählt und sich daraufhin fortwährend an diese hält. Die nichtdeterministische Auswahl ist also schon zu Anfang eines Ablaufs getroffen, anders als etwa bei CSP ([Hoare 85]) oder bei I/O-Automaten (vgl. Abschnitt 5.2).

Die Spuren einer nichtdeterministischen Komponente sind folgendermaßen definiert:

Definition (Spuren einer nichtdeterministischen Komponente): Sei eine nichtdeterministische Komponente durch eine Menge $\mathcal{C}$ von Strategien beschrieben. Ihre Spurmenge ist definiert durch: $Traces(\mathcal{C}) =_{def} \bigcup_{C \in \mathcal{C}} Traces(C)$. □

Definition (von einer Komponente realisierbar): Eine Spurmenge T heißt *von einer Komponente (teilweise) realisierbar*, wenn es eine Strategie C gibt mit Traces(C) ⊆ T. Eine Spurmenge T heißt *von einer Komponente vollständig realisierbar*, wenn es eine Menge $\mathcal{C}$ von Strategien gibt mit $Traces(\mathcal{C})$ = T. □

Eine Anforderung, die in Form einer Spurmenge gegeben ist, kann also von einer Komponente erfüllt werden, wenn diese Spurmenge (teilweise) von ihr realisierbar ist.

Mit diesem Begriffsapparat läßt sich auf natürliche Weise der Begriff der Sicherheitseigenschaft für eine Komponente formalisieren. Dieser ist mit der in Abschnitt 4.2.1 vorgestellten Definition verträglich:

Satz (Sicherheitseigenschaft für die Komponente): Sei eine Komponente durch eine Menge C von Strategien beschrieben. Dann ist Close(Traces(C)) eine Sicherheitseigenschaft für die Komponente.
Beweis: (1) Close(Traces(C)) ist per Definition eine Sicherheitseigenschaft.
(2) Nun ist zu zeigen: $t \in \text{Close}(\text{Traces}(C)) \wedge a \in \text{Eact} \Rightarrow t^\circ a \in \text{Close}(\text{Traces}(C))$.
Wenn $t \in \text{Act}^*$ und $t \in \text{Close}(\text{Traces}(C))$, so gibt es $s \in \text{Traces}(C)$ mit $t \sqsubseteq s$. $s = \text{trace}(C,e)$ für ein gewisses $e \in \text{Nat} \to \text{Eact}^*$ und $C \in C$. Falls $t \in \text{Eact}^*$, so gilt $t^\circ a \sqsubseteq \text{trace}(c,e')$ für ein e' mit $e'(0) = t^\circ a$. Sonst gibt es ein k mit $\text{ptrace}(C,e,k) \sqsubseteq t \sqsubseteq \text{ptrace}(C,e,k+1)$. Dann läßt sich e zu einem e' abwandeln, so daß $e'(k) = e(k)^\circ a$. Es gilt $t^\circ a \sqsubseteq \text{trace}(C,e') \in \text{Traces}(C)$, daher $t^\circ a \in \text{Close}(\text{Traces}(C))$. □

Satz: Eine Sicherheitseigenschaft für eine Komponente läßt sich von ihr vollständig realisieren.
Beweis: Sei $t \in C$. t habe die Gestalt $w_0{}^\circ c_0{}^\circ w_1{}^\circ c_1{}^\circ \ldots$ mit $w_k \in \text{Eact}^*$, $c_k \in \text{Cact} \cup \{\varepsilon\}$; außerdem seien dann, wenn zwei aufeinanderfolgende Elemente gleich ε sind, alle darauffolgenden Elemente gleich ε. Man definiere:

$$C_t(x) =_{\text{def}} \begin{cases} c_k & \text{falls } x = w_0{}^\circ c_0{}^\circ \ldots {}^\circ w_k \text{ für ein } k; \\ \varepsilon & \text{sonst} \end{cases}$$

Man sieht: $\text{trace}(C_t,e_t) = t \in C$, wenn e_t: $\text{Nat} \to \text{Eact}^*$ definiert ist durch: $e_t(k) =_{\text{def}} w_k$.
Andere Eingabemodellierungen erzeugen nur Spuren in C. □

Mit Hilfe von Strategien läßt sich nunmehr auch der Begriff der Lebendigkeitseigenschaft für eine Komponente formalisieren, der die in Abschnitt 4.3.1 geschilderte Vorstellung erfaßt:

Definition (Lebendigkeitseigenschaft für eine Komponente): Eine Menge $L \subseteq (\text{Eact} \cup \text{Cact})^\omega$ ist eine *Lebendigkeitseigenschaft für eine Komponente* mit Eingabemenge Eact und Ausgabemenge Cact, wenn gilt:

$$\forall s \in \text{Act}^*: \exists C \in ((\text{Eact} \cup \text{Cact})^* \to \text{Cact} \cup \{\varepsilon\}): \forall e \in (\text{Nat} \to \text{Eact}^*): \text{trace}(C,e,s) \in L \quad (*)$$

Hierbei ist trace(C,e,s) (ähnlich wie trace(C,e) weiter oben) definiert durch:

$$\text{trace}(C,e,s) =_{\text{def}} \bigsqcup_n \text{ptrace}(C,e,n,s)$$

und ptrace(C,e,n,s) ist definiert durch:

$$\text{ptrace}(C,e,n,s) =_{\text{def}} \text{\B\LC\{(\A\AL\CO1(s,}$$
$$\text{falls } n = 0; \text{ptrace}(C,e,n\text{-}1,s) \circ C(\text{ptrace}(C,e,n\text{-}1,s)) \circ e(n), \quad \text{sonst}) \quad \square$$

Diese Definition besagt: L ist eine Lebendigkeitseigenschaft für eine Komponente, wenn es ausgehend von einer beliebigen "Vorgabe" $s \in (\text{Eact} \cup \text{Cact})^*$ eine Berechnung mit einer Strategie C gibt, so daß die resultierende Spur in L liegt. Dabei ist eine beliebige Eingabemodellierung e erlaubt. Wesentlich ist hierbei, daß wie bei globalen Lebendigkeitseigenschaften (vgl. Abschnitt 3.5.1) *jede* endliche Spur zu einer lebendigen verlängert werden kann.

Offensichtlich ist jede Lebendigkeitseigenschaft für eine Komponente auch eine allgemeine (globale) Lebendigkeitseigenschaft.

Eine explizite Charakterisierung des Begriffs der Lebendigkeitseigenschaft für eine Komponente halte ich für schwierig (vgl. dazu die Schwierigkeiten mit einer expliziten Charakterisierung des Begriffs "Realisierbarkeit" in [Broy et al. 91a]). Die obige implizite Charakterisierung ist für die methodische Anwendung nur begrenzt tauglich; die Erfahrung zeigt jedoch auch, daß sich vielen Spurmengen ein operationelles Verhalten unschwer ablesen läßt.

Die in Abschnitt 3.5.2 gemachten Aussagen über globale Sicherheits- und Lebendigkeitseigenschaften lassen sich auf Sicherheits- und Lebendigkeitseigenschaften für Komponenten übertragen; dies liegt in einigen Fällen einfach darin, daß diese Eigenschaften spezielle (globale) Eigenschaften sind. So ist z.B. wiederum eine Eigenschaft sowohl eine Sicherheits- als auch eine Lebendigkeitseigenschaft für eine Komponente, wenn sie identisch *true* ist. Andere Eigenschaften ergeben sich leicht aus den Definitionen für Sicherheits- und Lebendigkeitseigenschaften für eine Komponente. So ist die Konjunktion zweier Sicherheitseigenschaften für eine Komponente wiederum eine Sicherheitseigenschaft für eine Komponente. Die Disjunktion einer Lebendigkeitseigenschaft für eine Komponente mit einer beliebigen anderen Eigenschaft ist wiederum eine Lebendigkeitseigenschaft für eine Komponente. Wie globale Eigenschaften lassen sich auch realisierbare Komponentenspezifikationen in Sicherheits- und Lebendigkeitsanteile aufspalten:

Satz (Aufspaltung in Sicherheits- und Lebendigkeitseigenschaften): Sei C eine Menge von Strategien und seien die Mengen S_C und L_C folgendermaßen definiert:

$S_C =_{def}$ Close(Traces(C)), $L_C =_{def} \neg S_C \cup$ Traces(C).

Dann gilt:

(1) S_C ist eine Sicherheitseigenschaft für die Komponente, L_C ist eine Lebendigkeitseigenschaft für die Komponente.

(2) $S_C \cap L_C =$ Traces(C).

Beweis: (1) Daß S_C eine Sicherheitseigenschaft für die Komponente ist, wurde schon oben gezeigt. L_C ist aus dem folgenden Grund eine Lebendigkeitseigenschaft gemäß der obigen Charakterisierung (*): Man betrachte ein $s \in Act^*$. Falls $s \notin S_C$, so ist (*) mit einer beliebigen Komponentenstrategie erfüllbar. Sonst ist s Präfix einer Spur in Traces(C). Offensichtlich läßt sich dann s im Zusammenspiel einer Komponentenstrategie aus C mit einer beliebig willkürlichen Eingabemodellierung zu einer Spur in Traces$(C) \subseteq L_C$ verlängern.

(2) $S_C \cap L_C = S_C \cap (\neg S_C \cup$ Traces$(C)) = (S_C \cap \neg S_C) \cup (S_C \cap$ Traces$(C)) =$ Close(Traces$(C)) \cap$ Traces$(C) =$ Traces(C). □

Ein realisierbarer Anteil einer Spurmenge ist also immer schematisch in einen Sicherheits- und einen Lebendigkeitsanteil aufspaltbar, wenngleich ein realisierbarer Anteil durch die Abstützung auf Strategiemengen lediglich implizit gegeben ist.

Wie im Fall globaler Eigenschaften (vgl. Abschnitt 3.5.2) läßt sich auch eine "alternative Definition" einer Lebendigkeitseigenschaft für eine Komponente angeben, die das Zusammenpassen von Sicherheit und Lebendigkeit berücksichtigt:

Lebendigkeit (alternative Definition): Sei S ein Sicherheitsprädikat für eine Komponente. L heißt *Lebendigkeitsprädikat bzgl.* S, wenn gilt:

$$\forall s \in Act^*: S(s) \Rightarrow \exists C \in ((Eact \cup Cact)^* \rightarrow Cact \cup \{\varepsilon\}): \forall e \in (Nat \rightarrow Eact^*):$$
$$S(\text{ trace}(C,e,s)\ ) \ \wedge\ L(\text{ trace}(C,e,s)\ )$$ □

(trace(C,e,s) sei dabei wie bei der ersten Definition einer Lebendigkeitseigenschaft definiert.)

Ist L ein Lebendigkeitsprädikat bzgl. S (alternative Definition) und S nicht identisch *false*, so ist $S \wedge L$ offensichtlich realisierbar; dies muß gemäß den früheren Definitionen von Sicherheit und Lebendigkeit für eine Komponente nicht unbedingt gelten:

Beispiel: Sei $S(t) \equiv \#Cact©t = 0$, $L(t) \equiv \#Cact©t = \infty$. S und L sind realisierbar, aber $S \wedge L \Leftrightarrow$ *false* und *false* ist nicht realisierbar. □

Umgekehrt ist nach dem vorigen Satz jede realisierbare Eigenschaft in Sicherheits- und Lebendigkeitsanteile aufspaltbar, die - wie man leicht sieht - die Definition von Sicherheit und die alternative Definition von Lebendigkeit erfüllen. Die alternative Lebendigkeitsdefinition bietet wie die ursprüngliche Lebendigkeitsdefinition lediglich ein implizite Charakterisierung und ist für methodische Zwecke nicht direkt verwendbar.

Bei der Konjunktion von Anforderungen ist zu berücksichtigen, daß (anders als im vorigen Beispiel) auch ein nicht leerer Schnitt von Spurmengen, die von der Komponente realisierbar sind, nicht unbedingt wiederum realisierbar sein muß. Dies zeigt das folgende Beispiel:

Beispiel: Sei Eact = {e}, Cact = {c,c'} und seien die Komponentenstrategien C und C' wie folgt definiert:

$C(x) =_{def}$ \B\LC\{(\A\AL\CO1(c falls $x \in Eact^*$; ε sonst)) $C'(x) =_{def}$ $\begin{cases} c' & \text{falls } x=\varepsilon; \\ c & \text{falls } x \in Eact^+; \\ \varepsilon & \text{sonst} \end{cases}$

Dabei sei $Eact^+ = Eact^* - \{\varepsilon\}$. $Traces(C) = e^*ce^\omega$ (in der Notation regulärer Ausdrücke), $Traces(C') = e^+ce^\omega \cup c'e^\omega$. Es ergibt sich: $Traces(C) \cap Traces(C') = e^+ce^\omega$. Diese Menge stellt jedoch eine Restriktion an die Umgebung dar, nämlich keine leere Eingabe zu machen, und kann daher nicht von der Komponente realisiert werden. □

Offensichtlich ist aber die Vereinigung von der Komponente realisierbarer Mengen wiederum von der Komponente realisierbar; die Vereinigung der entsprechenden Strategiemengen realisiert die Vereinigung der Spurmengen.

4.3.3 Modellierung des Zusammenspiels mehrerer Komponenten

Bisher wurde eine einzelne Komponente im Wechselspiel mit einer beliebigen Umgebung betrachtet. Zur Untersuchung der Aufspaltung globaler Anforderungen ist es sinnvoll, wie in Abschnitt 4.2 eine symmetrische Beziehung zu untersuchen, bei der auch der Freiheitsgrad der Umgebung eingeschränkt wird.

Hierzu sei zunächst der allgemeine Fall dargestellt: ein verteiltes System, das aus Komponenten $C_1,...,C_n$ besteht, die miteinander in Verbindung stehen, zusammen ein geschlossenes System

bilden und an die jeweils gewisse Anforderungen bestehen. Diese mögen die Eingabemengen $in(C_1),...,in(C_n)$ und die Ausgabemengen $out(C_1),...,out(C_n)$ haben. Die Ausgabemengen der Komponenten seien disjunkt. Auf diese Weise ist eindeutig festgelegt, welche Komponente für welche Ausgabeaktion zuständig ist (vgl. Abschnitt 4.1). Jede Komponente C_i wird durch eine Strategiemenge beschrieben, die ebenfalls mit C_i bezeichnet wird. Jede Strategie in C_i ist eine Abbildung der Funktionalität: $(in(C_i) \cup out(C_i))^* \rightarrow out(C_i) \cup \{\varepsilon\}$.

Das Verhalten eines derart beschriebenen verteilten System kann durch Berechnungen eines Transitionssystems und, von den Zuständen abstrahierend, durch eine Spurmenge beschrieben werden:

Definition (Verhalten eines durch Strategien beschriebenen verteilten Systems): Sei ein verteiltes System (wie oben festgelegt) durch Strategiemengen $C_1,...,C_n$ beschrieben. Das Systemverhalten für jede Auswahl von Strategien $C_1 \in C_1$, ..., $C_n \in C_n$ wird durch ein Transitionssystem $TS_{C_1,...,C_n} = (A, State, \text{—}\!\!>, init)$ festgelegt. Dabei ist (sei $\alpha C_m =_{def} in(C_m) \cup out(C_m)$ für alle m):

- die Aktionenmenge A gleich $\alpha C_1 \cup ... \cup \alpha C_n \cup \{\varepsilon\}$,
- die Zustandsmenge State gleich $\alpha C_1^* \times ... \times \alpha C_n^*$,
- die Übergangsrelation $\text{—}\!\!>\ \subseteq State \times A \times State$ definiert durch:

$$\langle w_1,...,w_n\rangle \text{—}\!\!>^{a} \langle w_1',...,w_n'\rangle \Leftrightarrow_{def}$$
$$a = C_i(w_i) \text{ für ein } i \in \{1,...,n\} \text{ und}$$
$$w_m' = \begin{cases} w_m \circ a & \text{falls } a \in \alpha C_m; \\ w_m & \text{sonst} \end{cases} \quad \text{für alle } m \in \{1,...,n\}$$

- der Anfangszustand init gleich $\langle\varepsilon,...,\varepsilon\rangle$.

Eine *Berechnung* ist eine vom Anfangszustand ausgehende Folge von Übergängen (vgl. Abschnitt 3.4), in der unendlich viele Übergänge von jeder Komponente C_i vorkommen (d.h. solche, die mit einem $C_i(w_i)$ markiert sind)[1]. Die *Spur* einer Berechnung entsteht aus der (unendlichen) Konkatenation der Übergangsbeschriftungen dieser Berechnung.
Die *Spurmenge* eines so beschriebenen verteilten Systems ist die Vereinigung der Spurmengen aller Berechnungen der Transitionssysteme $TS_{C_1,...,C_n}$ mit $C_1 \in C_1$, ..., $C_n \in C_n$. □

Man beachte, daß auf diese Weise ein de facto *geschlossenes* System modelliert wird. Zwar kann es sein, daß einige Komponenten Eingaben akzeptieren, die nicht zugleich Ausgaben einer anderen Komponente sind. Solche Eingaben kommen in Berechnungen jedoch nicht vor und die Eingabemengen der Komponenten können daher um diese Eingaben reduziert werden.

Die Situation, in der nur eine einzelne Komponente und ihre Umgebung betrachtet wird, ergibt sich als Spezialfall:

---

[1] Man beachte, daß diese Fairneßannahme der von UNITY ([Chandy, Misra 88]) entspricht.

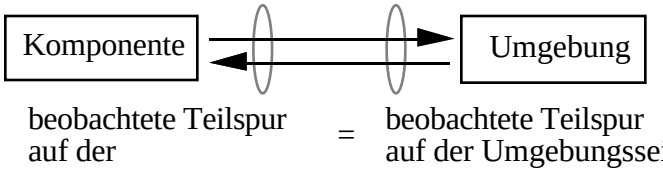


Fig. 4.5

Die Definition des Verhaltens eines durch Strategien beschriebenen verteilten Systems läßt sich als *Spielregel* auslegen. Sie lautet, daß die Komponente und ihre Umgebung fortwährend Ausgaben ausführen (evtl. auch die "leere" Ausgabe), die der Partner als Eingabe entgegennimmt. Die Komponente oder die Umgebung können auch mehrere Ausgaben in einem Stück machen, beide sind aber immer wieder am Zug. Welche Ausgabe von einem Spielpartner kommt hängt davon ab, welche Strategie dieser am Anfang des "Spiels" aus seinem Strategiereservoir auswählt und welche Eingaben von dessen Partner kommen.[1]

In diesem Abschnitt nehmen wir an, daß die Komponente und ihre Umgebung die Reaktion des Partners frühestmöglich sehen: Die Eingabe des einen ist die (nichtverzögerte) Ausgabe des anderen. Beide sehen somit zu jedem Zeitpunkt die gleiche Teilspur. Systeme, bei denen Kanallaufzeiten eine Rolle spielen, werden in Abschnitt 4.3.4 angesprochen.

Im folgenden sei eine *Komponentenstrategie* eine Abbildung der Funktionalität $(Eact \cup Cact)^* \rightarrow Cact \cup \{\varepsilon\})$, eine *Umgebungsstrategie* eine Abbildung der Funktionalität $(Eact \cup Cact)^* \rightarrow Eact \cup \{\varepsilon\})$.

Definition (durch Komponenten- und Umgebungsstrategien realisierte Spurmenge): Seien C und E Mengen von Komponenten- bzw. Umgebungsstrategien. Die durch diese Strategien realisierte Spurmenge wird mit $Traces(C,E)$ bezeichnet. Sie enthält genau die Spuren der Berechnungen derjenigen Transitionssysteme (gemäß der obigen Definition), bei denen sich die Komponente an eine Strategie aus C und ihre Umgebung an eine Strategie aus E hält. □

Definition (durch Komponenten- und Umgebungsstrategien realisierbar): Eine Spurmenge T heißt *durch Komponenten- und Umgebungsstrategien (teilweise) realisierbar*, wenn es Mengen C und E von Komponenten- bzw. Umgebungsstrategien gibt mit $Traces(C,E) \subseteq T$. T heißt *durch Komponenten- und Umgebungsstrategien vollständig realisierbar*, wenn sogar $Traces(C,E) = T$ gilt. □

Satz: Seien C und E Mengen von Komponenten- bzw. Umgebungsstrategien. Dann gilt:

$Traces(C,E) = Traces(C) \cap Traces(E)$.

Beweis: "$\subseteq$": offensichtlich

"$\supseteq$": Sei $t \in Traces(C) \cap Traces(E)$. Es gelte also: t = trace(C,e) = trace(E,c) für gewisse

---

[1] Offensichtlich läßt sich die Modellierung im vorigen Abschnitt als Wechselspiel zwischen einer Komponente und ihrer Umgebung interpretieren, bei dem das Strategiereservoir für die Umgebung *alle* Umgebungsstrategien enthält.

$C \in \mathcal{C}$, $E \in \mathcal{E}$, $e \in (Nat \to Eact^*)$, $c \in (Nat \to Cact^*)$. Gemäß der Gestalt von t können die folgenden Fälle auftreten (für jeden Fall gibt es auch einen dazu dualen Fall, wenn man Eact und Cact vertauscht; dieser Fall läßt sich jeweils analog behandeln). Für jeden Fall wird ein (fairer) Schedule angegeben, so daß die entsprechende Berechnung des Transitionssystems $TS_{C,E}$ (s.o.) die Spur t erzeugt.

Fall 1: $t = u_1 \circ v_1 \circ u_2 \circ v_2 \circ \ldots$ mit $u_i \in Eact^+$, $v_i \in Cact^+$. t kann im Zusammenspiel von C und E auf die folgende Weise erzeugt werden: Zunächst erfolgen $\#u_1$-viele E-Schritte. Da $t = trace(E,c)$ muß gelten: $E(\varepsilon) = ft(u_1)$, $E(ft(u_1)) = ft(rt(u_1))$, usw. Daher ist nach $\#u_1$ E-Schritten tatsächlich u_1 erzeugt. Anschließend geschehen $\#v_1$ C-Schritte. Da $t = trace(C,e)$ muß gelten $C(u_1) = ft(v_1)$, usw. Nach $\#u_1$ E-Schritten und $\#v_1$ C-Schritten ist dann $u_1 \circ v_1$ erzeugt. Verfolgt man den skizzierten Schedule weiter, so wird t erzeugt.

Fall 2: Irgendwann kommen nur noch Eact-Aktionen. Daher ist t von der Form $u_1 \circ v_1 \circ \ldots \circ u_n \circ v_n \circ u_{n+1} \circ u_{n+2} \circ \ldots$ mit $u_i \in Eact^+$, $v_i \in Cact^+$. Mit der Argumentation des Falls 1 kann man im Zusammenwirken von C und E die Teilspur $u_1 \circ v_1 \circ \ldots \circ u_n$ erzeugen. Da $t = trace(C,e)$ können die u_{n+i} für alle i so gewählt werden, daß $u_{n+i} = e(m+i)$ für ein gewisses m. Offensichtlich erhält man dann, wenn man jeweils nach $\#u_{n+i}$ Schritten einen C-Schritt einstreut, im Zusammenwirken von C und E wiederum t.

Fall 3: Die Spur ist endlich, d.h. t ist von der Form $u_1 \circ v_1 \circ \ldots \circ u_n \circ v_n$ oder $u_1 \circ v_1 \circ \ldots \circ u_n \circ v_n \circ u_{n+1}$. Offensichtlich gilt in beiden Fällen $C(t) = \varepsilon$ und $E(t) = \varepsilon$. Daher gibt es auch hier einen fairen Schedule, so daß durch eine Berechnung bestehend aus C- und E-Schritten t entsteht. □

Dieser Satz hat die folgende Auswirkung auf die Methodik: Man nehme an, die Anforderungen an die Komponente und ihre Umgebung seien in Form zweier Spurmengen T_C und T_E gegeben. Findet man eine Realisierung von T_C mit Strategien aus einer Menge $\mathcal{C}$ und eine Realisierung von T_E mit Strategien aus einer Menge $\mathcal{E}$, so läßt sich $T_C \cap T_E$ ebenfalls realisieren. Dies gelingt im Zusammenspiel der Strategien aus $\mathcal{C}$ und $\mathcal{E}$. Die Spurmengen T_C und T_E beschreiben damit die Anforderungen an die Komponente bzw. ihre Umgebung hinreichend genau.

Das folgende Beispiel zeigt, daß es Spurmengen gibt, die sowohl von einer Komponente als auch von ihrer Umgebung realisiert werden können.

Beispiele: a) Sei Cact = {c}, Eact = {e}. Die Anforderung: $P(t) \equiv \#e©t = \#c©t$ kann sowohl von der Komponente als auch von ihrer Umgebung allein, d.h. unabhängig vom Verhalten des Partners, erfüllt werden. Definiert man nämlich:

$C(x) =_{def}$ \B\LC\{(\A\AL\CO1(c falls $\#e©x > \#c©x$; ε sonst)) $E(x) =_{def}$ { e falls $\#c©x > \#e©x$; ε sonst

so erhält man $t \in Traces(C) \Rightarrow P(t)$, $t \in Traces(E) \Rightarrow P(t)$.

b) Sei Cact = {c}, Eact = {e}. Die Anforderung $P(t) \equiv \forall r \in Act^*, s \in Act^\omega: t = r \circ e \circ s \Rightarrow c$ *in* s. sieht auf den ersten Blick so aus, als richte sie sich allein an die Komponente. Tatsächlich erhält man mit der Komponentenstrategie C, die definiert ist durch:

$C(x) =_{def}$ { c falls e *in* x gilt; ε sonst

die Beziehung $t \in Traces(C) \Rightarrow P(t)$. P kann jedoch auch allein von der Umgebung erfüllt werden. Definiert man nämlich: $E(x) = \varepsilon$ für alle x, so erhält man $t \in Traces(E) \Rightarrow P(t)$ D.h. einerlei wie sich die Komponente verhält, es werden nur korrekte Spuren erzeugt.
Die Beispiele a) und b) bezogen sich auf Lebendigkeitseigenschaften, es gibt aber auch Sicherheitseigenschaften, die sich sowohl von einer Komponente als auch ihrer Umgebung realisieren lassen, etwa $P(t) \equiv \forall s \in Act^*: s \sqsubseteq t \Rightarrow \neg\, c \textit{ in } s \vee \neg\, e \textit{ in } s$. □

Im folgenden wird die Realisierbarkeit von Spurmengen im Zusammenwirken der Komponente mit ihrer Umgebung genauer untersucht.

Abschnitt 4.3.1 zeigte, daß es Spurmengen gibt, die nur in Absprache der Komponente mit ihrer Umgebung realisiert werden können. Ein Beispiel hierfür war die Menge $T = \{t \in (Cact \cup Eact)^\omega \mid Eact \copyright t = \infty \Leftrightarrow Cact \copyright t \neq \infty\}$, wo zwei realisierbare Anteile angegeben wurden; T wurde nur *teilweise* realisiert (vgl. die obige Definition). Dies ist aber bei jeder nichtleeren Spurmenge möglich:

Satz: Sei $T \subseteq (Cact \cup Eact)^\omega$ und $T \neq \emptyset$. Dann ist T teilweise realisierbar.
Beweis: Sei $t \in T$. t habe die Gestalt $e_1 \circ c_1 \circ e_2 \circ c_2 \circ \ldots$ mit $e_i \in Eact \cup \{\varepsilon\}$ und $c_i \in Cact \cup \{\varepsilon\}$ und zudem gelte: Falls zwei aufeinanderfolgende Elemente gleich ε sind, so sind alle darauffolgenden Elemente gleich ε. (Jede Spur aus $(Cact \cup Eact)^\omega$ ist in dieser "Normalform" darstellbar.) O.B.d.A. sei $e_1 \neq \varepsilon$.
Sei die Komponentenstrategie C folgendermaßen definiert:

$C(x) =_{def}$ { c_n falls $x = e_1 \circ c_1 \circ \ldots \circ e_n$ für ein n; ε sonst

Sei die Umgebungsstrategie E folgendermaßen definiert:

$E(x) =_{def}$ \B\LC\{(\A\AL\CO1(e_1 falls $x = \varepsilon$; e_{n+1} falls $x = e_1 \circ c_1 \circ \ldots \circ e_n \circ c_n$ für ein n; ε sonst))

Man erhält: $Traces(C,E) = \{t\}$. $\{t\} \subseteq T$ ist also realisierbar, damit ist T teilweise realisierbar. □

Definition (ohne Absprache realisierbar): Eine Spurmenge T ist *ohne Absprache (vollständig) realisierbar*, wenn gilt: T ist vollständig realisierbar und für alle Mengen C, C' von Komponentenstrategien und E, E' von Umgebungsstrategien gilt:

$Traces(C,E) = T \wedge Traces(C',E') = T \Rightarrow Traces(C \cup C', E \cup E') = T$ □

Die Bezeichnung "ohne Absprache realisierbar" ist aus dem folgenden Grund gerechtfertigt: Man stelle sich vor, eine globale Anforderung sei in Form einer Spurmenge gegeben und der Entwerfer der Komponente wisse, daß die Menge ohne Absprache realisierbar ist; zudem finde er eine Strategiemenge C, für die mit einer gewissen Strategiemenge E gilt: $Traces(C,E) = T$. In diesem Fall kann er als Komponentenanforderung $Traces(C)$ wählen und sicher sein, daß auf diese Weise nur korrekte Spuren entstehen.

Es gibt eine weitere methodische Implikation: Ist eine Spurmenge T ohne Absprache realisierbar, so gibt es größte Mengen C und E von Komponenten- bzw. Umgebungsstrategien, so daß $Traces(C,E) = T$. Die von C und E erzeugten Spurmengen

sind damit die größten (vollständig realisierbaren) Spurmengen, deren Schnitt T ergibt; sie legen die schwächsten Anforderungen an die Komponente und ihre Umgebung fest, die gewährleisten, daß die globale Anforderung (verkörpert durch T) erfüllt wird. Somit gibt es hier eine gewissermaßen "kanonische" Dekomposition einer globalen Anforderung.

Der Begriff "ohne Absprache realisierbar" ließe sich auch für teilweise realisierbare Spurmengen definieren, für die folgenden Untersuchungen spielt aber nur die vollständige Realisierbarkeit eine Rolle.

Nicht alle vollständig realisierbaren Spurmengen sind ohne Absprache realisierbar:

Beispiel (*): Sei Cact = {c}, Eact = {e}. Für $k \in Nat \cup \{\infty\}$ seien die Komponentenstrategien C_k folgendermaßen definiert:

$$C_k(x) =_{def} \begin{cases} c & \\ & \text{falls } x = (e \circ c)^i \circ e \text{ für ein } i<k; \varepsilon \\ & \text{sonst} \end{cases}$$

und die Umgebungsstrategien E_k folgendermaßen:

$$E_k(x) =_{def} \begin{cases} e & \\ & \text{falls } x = (e \circ c)^i \text{ für ein } i<k; \varepsilon \\ & \text{sonst} \end{cases}$$

Es gilt: $Traces(C_k,E_i) = \begin{cases} \{(e \circ c)^i\} & \\ \quad \text{falls } i \le k; \{(e \circ c)^k \circ e\} & \\ \quad \text{sonst} \end{cases}$

$Traces(C_k,E_\infty) = \{(e \circ c)^k \circ e\}$, $Traces(C_\infty,E_k) = \{(e \circ c)^k\}$, $Traces(C_\infty,E_\infty) = \{(e \circ c)^\infty\}$.
Sei $\mathcal{C} = \{C_k \mid k \in Nat\}$, $\mathcal{E} = \{E_k \mid k \in Nat \cup \{\infty\}\}$, $\mathcal{C}' = \{C_k \mid k \in Nat \cup \{\infty\}\}$, $\mathcal{E}' = \{E_k \mid k \in Nat\}$. Es gilt: $Traces(\mathcal{C},\mathcal{E}) = Traces(\mathcal{C}',\mathcal{E}') = \{(e \circ c)^k, (e \circ c)^k \circ e \mid k \in Nat\}$. Jedoch gilt: $(e \circ c)^\infty \in Traces(\mathcal{C} \cup \mathcal{C}', \mathcal{E} \cup \mathcal{E}')$, aber $(e \circ c)^\infty \notin Traces(\mathcal{C},\mathcal{E})$. □

Während dieses Beispiel zeigt, daß die Vereinigung von Strategiemengen, die eine Spurmenge vollständig realisieren, nicht unbedingt korrekte (d.h. sichere und lebendige) Spuren erzeugt, so erzeugt sie doch wenigstens *sichere* Spuren, d.h. solche, deren endliche Präfixe immer zu einer total korrekten Spur verlängert werden können. Dies zeigt der folgende Satz:

Satz: Gegeben seien eine Spurmenge $T \subseteq (Cact \cup Eact)^\omega$ sowie Mengen von Komponenten- bzw. Umgebungsstrategien $\mathcal{C}$ und $\mathcal{E}$ sowie $\mathcal{C}'$ und $\mathcal{E}'$ mit $T = Traces(\mathcal{C},\mathcal{E}) = Traces(\mathcal{C}',\mathcal{E}')$. Dann gilt: $Traces(\mathcal{C} \cup \mathcal{C}', \mathcal{E} \cup \mathcal{E}') \subseteq Close(T)$.

Beweis: O.B.d.A. wird nur gezeigt: $Traces(C,E') \subseteq Close(T)$, wenn $C \in \mathcal{C}$ und $E' \in \mathcal{E}'$. Betrachten wir eine Berechnung, die eine Spur in $Traces(C,E')$ erzeugt. Wiederum o.B.d.A. seien die ersten Komponenten der Spur aus Cact. Dies bedeutet, daß zunächst einige C-Schritte erfolgen; es ergibt sich eine Teilspur t. Diese ist offensichtlich aus Close(T). Daher muß t auch von einer Strategie $C' \in \mathcal{C}'$ erzeugt werden können. Bei der Berechnung folgen nun E'-Schritte. Es ergibt sich eine Teilspur t' mit $t \sqsubseteq t'$. t' ist ein Präfix einer Spur aus Traces(C',E'), also wiederum ein Element aus Close(T). Da $t \sqsubseteq t'$ gibt es eine Strategie $E \in \mathcal{E}$, so daß t' ein Präfix einer Spur in Traces(C,E) ist. Mit C-Schritten erzielt man eine Teilspur t", die wiederum in Close(T) liegt. Per Induktion ergibt sich, daß alle Präfixe einer Spur einer Berechnung in Close(T) liegen. Somit liegt auch diese Spur selbst in Close(T). □

Als Korollar ergibt sich somit:

Korollar: Sicherheitseigenschaften sind ohne Absprache vollständig realisierbar. □

Dieses Korollar ist mit der in Abschnitt 4.2.1 gemachten Beobachtung in Übereinstimmung: Ein globales Sicherheitsprädikat kann zwar auf verschiedene Weise in Sicherheitsprädikate für die Komponente und ihre Umgebung aufgespalten werden, die Unterschiede ergeben sich aber allein durch unterschiedliche Reaktionen auf Fehler des Partners.

Im folgenden wird gezeigt, daß es bei einer vollständig realisierbaren *beliebigen* Spurmenge auch einen anderen Grund für verschiedene Aufspaltungsmöglichkeiten gibt, nämlich eine unterschiedliche Verteilung der Lebendigkeitsanforderungen.

Zu diesem Zweck seien eine Spurmenge T und Strategiemengen C und C' betrachtet, für die es Strategiemengen E und E' gibt, so daß $T = Traces(C,E) = Traces(C',E')$ gilt.

Die Spurmenge Violate(T,Eact) ist definiert durch:

$$Violate(T,Eact) =_{def} \{s \circ e \circ t \mid s \in Act^* \wedge e \in Eact \wedge t \in Act^\omega \wedge s \in Close(T) \wedge s \circ e \notin Close(T)\}$$

Violate(T,Eact) ist die Menge derjenigen Spuren, bei denen die Umgebungskomponente gegen den Sicherheitsanteil von T, d.h. Close(T), verstoßen hat.

Man betrachte nun die Gleichung

$$Violate(T,Eact) \cup Traces(C) = Violate(T,Eact) \cup Traces(C') \qquad (**)$$

$Violate(T,Eact) \cup Traces(C)$ beschreibt diejenige Spurmenge, bei der sich die Komponente an ihre Spielregel hält solange sich ihre Umgebung sicher verhält. Verletzt die Umgebung die Sicherheitsanforderung, so darf sich die Komponente anschließend beliebig verhalten.

Gilt (**) für beliebige T, C und C', die die obige Bedingung erfüllen, so bedeutet dies: Sind $Traces(C)$ und $Traces(C')$ verschieden, so liegt der Grund darin, daß $Traces(C)$ und $Traces(C')$ lediglich unterschiedliche Reaktionen auf die Verletzung einer Sicherheitsanforderung durch die Umgebung wiedergeben. Ein Gegenbeispiel zeigt jedoch, daß die Gleichung (**) nicht in jedem Fall gilt:

Beispiel: Man betrachte die Strategiemengen C, C' und E' des Beispiels (*). Es gilt:

$$Traces(C,E') = Traces(C',E') = \{(e \circ c)^k, (e \circ c)^k \circ e \mid k \in Nat\}$$

Allerdings gilt: $(e \circ c)^\infty \notin Traces(C)$, aber $(e \circ c)^\infty \in Traces(C')$. Zudem gilt: $(e \circ c)^\infty \notin Violate(T,Eact)$. Die Umgebung hat sich nämlich bei dieser Spur immer sicher, wenn auch nicht lebendig verhalten; sie hätte sich nur dann lebendig verhalten, wenn sie ab einer beliebigen Stelle keine Ausgaben mehr ausgeführt hätte. □

4.3.4 Zusammenfassung und Ausblick

Ich will an dieser Stelle die Untersuchungsergebnisse der Abschnitte 4.2 und 4.3 zusammenfassen, um ein zusammenhängendes Bild der umfangreichen Untersuchungen zu präsentieren. Zudem gebe ich einen Ausblick auf eine Übertragung der Ergebnisse auf Systeme, in denen zwei Kommunikationspartner nicht die gleiche Sicht auf die bisher erfolgte Kommunikation haben.

Es wurde untersucht, wie globale Anforderungen in lokale Anforderungen an die Komponenten eines verteilten Systems überführt werden können, insbesondere, ob es dazu schematisierbare Regeln gibt. Hierzu betrachtete ich das Zusammenspiel einer Komponente mit ihrer Umgebung, dem Rest des verteilten Systems. Die Ergebnisse meiner Untersuchungen lassen sich unter zwei Blickwinkeln interpretieren: Zum einen kann die Komponente als das zu erstellende Produkt gesehen werden, die Umgebung als das Teilsystem der Umgebungskomponenten; globale Anforderungen sollen hier in lokale Anforderungen an das Produkt und seine Umgebung umgesetzt werden. Zum anderen kann die Komponente eine beliebige Komponente des verteilten Systems sein; in diesem Fall soll eine Anforderung, die über Ein- und Ausgaben dieser Komponente spricht, in eine lokale Anforderung an die Komponente und eine einfachere globale Anforderung an den Rest des Systems überführt werden.

In Abschnitt 4.2 untersuchte ich zunächst allein Sicherheitseigenschaften. Eine formale Definition einer Sicherheitseigenschaft für eine Komponente wurde angegeben. Ich zeigte, daß sich jede globale Sicherheitseigenschaft schematisch in Sicherheitsanteile für die Komponente und ihre Umgebung aufspalten läßt. Zu dieser Aufspaltung gibt es verschiedene Möglichkeiten, die sich allein durch die unterschiedliche Reaktion der Komponente bzw. der Umgebung auf die Verletzung der globalen Sicherheitsanforderung seitens des Partners unterscheiden. Da die Verletzung einer Sicherheitseigenschaft nach endlicher Zeit beobachtbar ist, sind Maßnahmen zur Fehlerbehandlung eine sinnvolle Reaktion auf die Verletzung. Dies zeigt, daß die Teilphase der komponentenorientierten Spezifikation dazu geeignet ist, auch Fehlerbehandlung zu spezifizieren, was bei der globalen Spurspezifikation als einer ersten formalen Spezifikation möglicherweise zunächst nicht berücksichtigt wurde. Maßnahmen zur Fehlerbehandlung untersuche ich in meiner Arbeit jedoch nicht weiter.

Bei der Behandlung von Sicherheitseigenschaften wurde ein kurzer Exkurs zu Annahme/Verpflichtung-Spezifikationen gemacht. Bei diesen wird die Einhaltung gewisser Bedingungen durch die Komponente von der Einhaltung gewisser anderer Bedingungen durch ihre Umgebung abhängig gemacht. Es zeigte sich, daß spezielle Spezifikationen dieses Typs zu "schwächsten" Sicherheitsprädikaten für eine Komponente äquivalent sind.

Will man Anforderungen aufspalten, die einen Lebendigkeitsanteil enthalten, so stößt man auf Schwierigkeiten: Manche Lebendigkeitsanforderungen lassen sich nicht auf "intuitive" Weise in Anforderungen an die Komponente und ihre Umgebung aufspalten. Diese Schwierigkeiten motivierten eine Analyse von Spurmengen mit einem spieltheoretischen Begriffsapparat ("Strategien"). Mit Strategien wurde der Begriff der Lebendigkeitseigenschaft für eine Komponente definiert. Der Begriff der Sicherheitseigenschaft für eine Komponente, der sich auf diese Weise ergibt, ist konform zur früheren Definition. Eine realisierbare Anforderung läßt sich ähnlich wie im Fall globaler Anforderungen in einen Sicherheits- und einen Lebendigkeitsanteil aufspalten. Die Realisierbarkeit einer Spurmenge durch Strategien diente als Leitfaden für die Untersuchung der Aufspaltung globaler Anforderungen. Bei manchen Anforderungen muß eine Absprache getroffen werden, welchen Anteil eine Komponente und welchen ihre Umgebung erfüllen soll. Andere globale Anforderungen können sowohl von der Komponente als auch von ihrer Umgebung allein realisiert werden; hier ist eine Regelung der Zuständigkeit nötig.

Somit kann es unterschiedliche Aufspaltungen globaler Anforderungen geben. Ich zeigte, daß der Grund dafür nicht unbedingt die unterschiedliche Reaktion auf die Verletzung der globalen Sicherheitseigenschaft seitens des Partners ist, sondern daß unterschiedliche Komponentenanforderungen auch unterschiedliche Lebendigkeitsanteile enthalten können.

Eine "vernünftige" Aufspaltung einer globalen Anforderung in lokale ist so geartet, daß die Komponenten die lokalen Anforderungen realisieren können. Dies ist jedoch auf der Anforderungsebene eine recht implizite Eigenschaft und methodisch nicht direkt umsetzbar, da die Realisierbarkeit erst im Laufe des Entwicklungsprozesses festgestellt wird (vgl. Kap. 5). Die lokalen Anforderungen an die Komponenten gehen i.a. nicht aus den globalen Anforderungen vollständig hervor, daher ist die Aufspaltung von globalen Anforderungen (außer eingeschränkt bei Sicherheitseigenschaften) nicht automatisierbar.

Bisher wurde die Annahme gemacht, daß die Ausgabe einer sendenden Komponente sofort von einer empfangenden Komponente als Eingabe gesehen wird und letztere sofort darauf reagieren kann. Anders als bei synchron kommunizierenden Komponenten (vgl. z.B. Hoare 85]) werden Eingaben allerdings zu jeder Zeit akzeptiert. Betrachtet man Komponenten, die über eine längere Entfernung asynchron miteinander kommunizieren, so ist die Verzögerung der asynchronen Nachrichtenübermittlung zu berücksichtigen, d.h. die Ausgabe der einen Komponente wird erst etwas später als Eingabe bei der anderen Komponente sichtbar. Die beobachteten Teilspuren müssen dann nicht mehr übereinstimmen (im Gegensatz zu Fig. 4.5). In diesem Fall sind Übertragungskanäle ebenfalls zu modellieren:

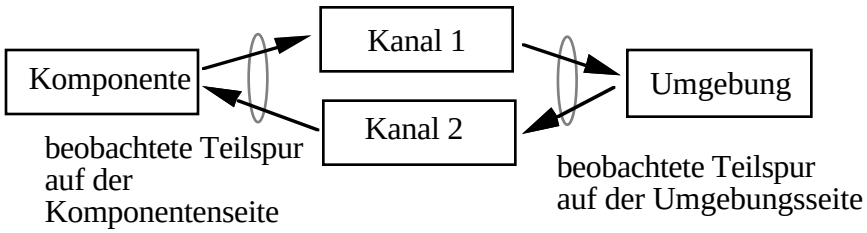


Fig. 4.6

Ein Kanal läßt sich wie jede Komponente durch Strategien oder eine Spurmenge modellieren.

Auf die Modellierung von Kanälen kann man verzichten, wenn die Komponenten eines Netzes *geschwindigkeitsunabhängig* sind, d.h. die Verzögerung durch Kanallaufzeiten keinen Einfluß auf das funktionale Verhalten hat. In diesem Fall lassen sich Komponenten mit sog. FIFO-Strategien nachbilden. FIFO-Strategien erzeugen für gleiche Eingaben gleiche Ausgaben.

Definition (FIFO-Strategie): Eine *FIFO-Strategie* für eine Komponente mit Eingabemenge Eact und Ausgabemenge Cact ist eine Funktion $C: (Eact \cup Cact)^* \rightarrow Cact \cup \{\varepsilon\}$ so daß für alle $e,e' \in Nat \rightarrow Eact^*$ gilt:

$$conc(e) = conc(e') \Rightarrow Cact \copyright trace(C,e) = Cact \copyright trace(C,e'),$$

wobei die unendliche Konkatenation conc(e) definiert ist durch: $conc(e) =_{def} e(0)^\circ e(1)^\circ \ldots$ □

In Abschnitt 5.4 wird der Zusammenhang zum verwandten Konzept der stromverarbeitenden Funktionen hergestellt.

Offensichtlich ist jede Spurmenge, die durch FIFO-Strategien realisierbar ist, auch durch (allgemeine) Strategien realisierbar. Somit wurden im vorigen Abschnitt möglichst "schwache"

Anforderungen an Spurmengen hinsichtlich ihrer Realisierbarkeit gestellt. Die Untersuchungen der vorigen Abschnitte lassen sich auf verzögerungsunabhängige Systeme übertragen, wenn man statt allgemeinen Strategien FIFO-Strategien betrachtet. Die methodische Vorgehensweise gemäß Abschnitt 4.1 bleibt davon unberührt.

4.4 Komponentenorientierte Echtzeitspezifikation

Ich skizziere nun, wie sich Komponenten durch zeitbehaftete Spuren (vgl. Abschnitt 3.6.1) modellieren lassen.

Eine Komponente mit Eingabemenge Eact und Ausgabemenge Cact wird durch Spuren aus der Menge $(\text{Eact} \cup \text{Cact} \cup \{\surd\})^{\omega}$ beschrieben, wobei Eact und Cact wiederum disjunkte Mengen von Aktionen seien und $\surd$ weder in Eact noch in Cact vorkomme. Das folgende Bild veranschaulicht den Zusammenhang:

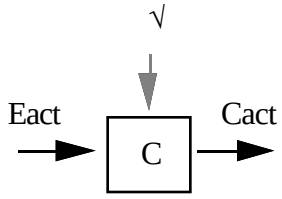


Fig. 4.7

Die Interpretation z.B. der Spur $e_1 \circ \surd \circ e_2 \circ e_3 \circ c_1 \circ \surd \circ c_2 \circ c_3 \circ \surd^{\infty}$ ist, daß im ersten Zeitschritt die Eingabeaktion e_1 stattfindet, im zweiten Zeitschritt die Eingaben e_2 und e_3 sowie die Ausgabe c_1, im dritten Zeitschritt die Ausgaben c_2 und c_3. Die $\surd$-Aktionen simulieren das Fortschalten einer Uhr. Die Interpretation gleicht also der des Abschnitts 3.6.1, nur wird jetzt zwischen Ein- und Ausgaben unterschieden. Nimmt man wiederum an, daß das Schreiben einer Nachricht auf einen Kanal von einer sendenden Komponente und das Lesen dieser Nachricht von der empfangenden Komponente gleichzeitig stattfindet, so läßt sich der ||-Operator (siehe Abschnitt 4.1) auch auf zeitbehaftete Spuren anwenden. Die $\surd$-Aktion modelliert eine *globale* Zeit und wirkt als Synchronisationsmittel; sie ist weder eine Aktion der Komponente noch ihrer Umgebung.

Beispiel: Es wird die Hintereinanderschaltung zweier Komponenten C1 und C2 betrachtet. C1 bekommt a-Aktionen als Eingaben und liefert b-Aktionen als Ausgaben. Die Eingaben von C2 sind b-Aktionen, ihre Ausgaben sind c-Aktionen (vgl. Fig. 4.8).

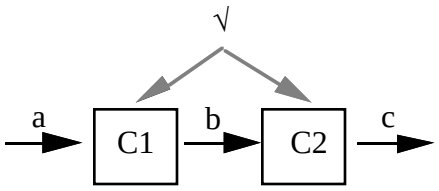


Fig. 4.8

Wenn	C1 die Spur	$a \circ \surd \circ b \circ a \circ b \circ \surd^{\infty}$	erzeugt und
	C2 die Spur	$\surd \circ b \circ c \circ b \circ \surd \circ c \circ \surd^{\infty}$,	so erzeugt die Zusammenschaltung
	C1 || C2 die Spur	$a \circ \surd \circ b \circ c \circ a \circ b \circ \surd \circ c \circ \surd^{\infty}$,	sowie die Spur
		$a \circ \surd \circ b \circ a \circ c \circ b \circ \surd \circ c \circ \surd^{\infty}$.	□

Die Modellierung wird komplizierter, wenn die √-Aktionen bei unterschiedlichen Komponenten von *lokalen* Uhren stammen, somit die √-Aktionen im obigen Beispiel unterschieden werden müssen ($\surd_1$ bzw. $\surd_2$). Dies führt auf das Problem der Uhrensynchronisation bzw. der Umrechnung von lokalen Zeiten in eine globale Zeit.

Schwieriger als bei Nicht-Echtzeitsystemen wird die Aufspaltung von globalen Sicherheitsanforderungen in lokale Anforderungen für die Komponente und ihre Umgebung. Nun gibt es nämlich drei Arten von Aktionen: Komponentenaktionen, Umgebungsaktionen und √. Für das Ticken der Uhr ist weder die Komponente noch ihre Umgebung zuständig. Das folgende Beispiel zeigt, daß die Verletzung einer Sicherheitseigenschaft nicht mehr unbedingt eindeutig der Komponente oder ihrer Umgebung zuordenbar ist:

Beispiel: In der Spurmenge $\{\surd \circ e \circ \surd^\infty, \surd \circ c \circ \surd^\infty\}$ (e sei eine Umgebungsaktion, c eine Komponentenaktion) liegt keine Spur mit Präfix $\surd \circ \surd$. Die zweite √-Aktion verletzt somit eine Sicherheitsanforderung; es hätte eine "echte" Aktion (e oder c) stattfinden müssen, die Ursache für die Verletzung der Sicherheitsanforderung ist jedoch nicht eindeutig eindeutig der Komponente oder ihrer Umgebung zuordenbar. □

Man erkennt an diesem Beispiel, daß man genau dann eine im wesentlichen eindeutige Aufspaltung einer globalen Sicherheitseigenschaft S erreichen kann, wenn gilt:

$$\begin{aligned}\mathrm{Split}(S) \equiv\ & \forall s \in \mathrm{Act}_\surd^*: (\exists t \in \mathrm{Act}_\surd^\infty: S(s \circ t) \wedge \neg \exists t \in \mathrm{Act}_\surd^\infty: S(s \circ \surd \circ t)) \Rightarrow \\ & (\forall t \in \mathrm{Act}_\surd^\infty: S(s \circ t) \Rightarrow \mathrm{ft}(t) \in \mathrm{Cact}) \vee (\forall t \in \mathrm{Act}_\surd^\infty: S(s \circ t) \Rightarrow \mathrm{ft}(t) \in \mathrm{Eact})\end{aligned}$$

Dabei sei Act wiederum die Vereinigung der disjunkten Mengen Cact und Eact. Diese Formel besagt, daß dann, wenn ein Zeitfehler auftritt, entweder nur Komponentenaktionen oder nur Umgebungsaktionen hätten stattfinden dürfen. Gilt Split(S), so ist die Fehlerursache eindeutig der Komponente oder ihrer Umgebung zuordenbar. Daher übertragen sich hier die Ergebnisse des Abschnitts 4.2.1.

Beispiel: Für ein globales Sicherheitsprädikat S mit Split(S) = *true* läßt sich ein schwächstes Sicherheitsprädikat für die Komponente (vgl. Abschnitt 4.2.1) folgendermaßen definieren:

$$\begin{aligned}C_{max}(S)(t) \equiv\ & (\forall s \in \mathrm{Act}_\surd^*, c \in \mathrm{Cact}: s \circ c \sqsubseteq t \wedge \exists u \in \mathrm{Act}_\surd^\infty: S(s \circ u) \Rightarrow \exists v \in \mathrm{Act}_\surd^\infty: S(s \circ c \circ v)) \\ & \wedge\ (\forall s \in \mathrm{Act}_\surd^*: s \circ \surd \sqsubseteq t \wedge \neg \exists u \in \mathrm{Act}_\surd^\infty: S(s \circ \surd \circ u) \Rightarrow \\ & \qquad (\forall u \in \mathrm{Act}_\surd^\infty: S(s \circ u) \Rightarrow \mathrm{ft}(u) \in \mathrm{Eact})))\end{aligned}$$

Die erste Zeile der Formel gibt die Eigenschaft wieder, die schon für Nicht-Echtzeitsysteme gefordert wurde. Die zweite Zeile berücksichtigt Zeitfehler: Entsteht nach einem endlichen Teilablauf s durch eine √-Aktion ein (Zeit-)Fehler, so hat sich die Komponente nur dann sicher verhalten, wenn nach s eine Umgebungsaktion hätte stattfinden müssen. □

5. Übergang zur Entwurfsspezifikation

In diesem Kapitel erläutere ich den Übergang von einer komponentenorientierten, spurbasierten Anforderungsspezifikation zu einer funktionalen Entwurfsspezifikation. Für die Durchgängigkeit von FOCUS ist es wichtig, daß die Beziehung zwischen diesen beiden Beschreibungsformen eines verteilten Systems geklärt ist und methodische Leitlinien für den Übergang bereitstehen. Auf die Technik der Entwurfsspezifikation gehe ich nur soweit ein, wie es für die Anbindung an die Anforderungsspezifikation nötig ist.

Zur Entwurfsspezifikation werden *stromverarbeitende Agenten* eingesetzt; dieses Konzept erläutere ich in Abschnitt 5.1, insbesondere zeige ich die Unterschiede zur komponentenorientierten Spezifikation auf. Die *Spursemantik* für stromverarbeitende Agenten ist der Anknüpfungspunkt zur Anforderungsebene; sie ist Gegenstand von Abschnitt 5.2. Ich stütze mich dabei auf bekannte Ergebnisse über I/O-Automaten. In Abschnitt 5.3 skizziere ich die methodische Vorgehensweise beim Übergang von der Anforderungs- zur Entwurfsspezifikation. In Abschnitt 5.4 setze ich das Konzept der Realisierbarkeit durch Strategien, das in Abschnitt 4.3 zentral war, zu der Realisierbarkeit durch stromverarbeitende Agenten und I/O-Automaten in Beziehung. Schließlich erfolgt in Abschnitt 5.5 ein Ausblick auf die Agentenbeschreibung von Echtzeitsystemen. Hierzu werden zeitbehaftete I/O-Automaten eingeführt, die zeitbehaftete Spuren (vgl. Abschnitte 3.6.1 und 4.4) realisieren.

5.1 Stromverarbeitende Agenten

Das Konzept der stromverarbeitenden Agenten geht auf die Arbeit [Kahn 74] zurück; verwandt damit ist die Idee der Datenflußberechnung ([Dennis 74]). Stromverarbeitende Agenten bekommen auf ihren Eingabekanälen Nachrichten zugesandt und verarbeiten diese in Nachrichten, die auf den Ausgabekanälen befördert werden. Die Nachrichten auf den Eingabekanälen lassen sich als Eingabeströme auffassen, die Nachrichten auf den Ausgabekanälen als Ausgabeströme, daher der Name "stromverarbeitender Agent". Agenten sind miteinander über gerichtete Kanäle verbunden und können auch nichtdeterministisches Verhalten zeigen.

Ein *deterministischer* stromverarbeitender Agent wird durch eine *stromverarbeitende Funktion* beschrieben. Dies ist eine Funktion, die ein Tupel von Eingabeströmen auf ein Tupel von Ausgabeströmen abbildet und bezüglich der vollständigen Halbordnung der Ströme monoton und stetig ist. Die Monotonie bedeutet, daß der Empfang von weiteren Eingaben höchstens weitere Ausgaben hervorruft. Ein Agent kann somit seine Berechnung und seine Ausgabe beginnen, wenngleich er noch nicht die gesamten Eingaben auf seinen Kanälen empfangen hat; dies ermöglicht Parallelarbeit der Agenten. Die Stetigkeit drückt aus, daß die Ausgabe auf einen unendlichen Eingabestrom durch die Ausgaben auf die endlichen Präfixe des Eingabestroms approximiert wird.

Für die Erweiterung des Ansatzes aus [Kahn 74] auf nichtdeterministische Agenten gibt es verschiedene Möglichkeiten (z.B. [Brock, Ackerman 81], [Staples, Nguyen 85], [Kok 86]). In unserer Entwurfsmethodik wird ein *nichtdeterministischer* Agent durch eine *Menge*

stromverarbeitender Funktionen beschrieben (vgl. [Keller 78], [Broy 88b]); diese Modellierung von Nichtdeterminismus entspricht der des Abschnitts 4.3.

Die Spezifikation eines stromverarbeitenden Agenten erfolgt durch ein Prädikat über stromverarbeitende Funktionen.[1] Ähnlich wie bei einer Spurspezifikation läßt sich auch hier sowohl die ablauforientierte als auch die transitionsorientierte Spezifikation einsetzen. Bei einer ablauforientierten Spezifikation von Funktionen wird über vollständige Ein- und Ausgabeströme einer Funktion gesprochen, z.B. beschreibt $Q(f) \equiv \forall x: \#x = \#f(x)$ die Menge aller Funktionen, die genausoviel ausgeben wie sie einlesen; für komplexere Beispiele vgl. [Broy 87b]. Eine transitionsorientierte Spezifikation zeigt das folgende Beispiel:

Beispiel (Postfachsystem: stromverarbeitender Agent): Die Komponente SERVER (vgl. Abschnitt 4.1) läßt sich durch die Funktion

$$\text{server}: (\text{Snd} \cup \text{Rec})^{\omega} \rightarrow (\text{S-ack} \cup \text{R-ack})^{\omega}$$

realisieren[2]. (Will man Eingaben aus Snd und Rec über getrennte Eingabekanäle empfangen, so muß vor die Funktion server noch ein Multiplexer (Mischagent) geschaltet werden. Entsprechend ließe sich die Funktion server auch derart abwandeln, daß Ausgaben aus S-ack und R-ack über getrennte Kanäle geschehen.)

Die Funktion server wird mittels einer Hilfsfunktion hs transitionssorientiert spezifiziert:

$$\text{server}(t) = \text{hs}(t,\text{init})$$

Die Funktion hs hat zusätzlich einen Zustandsparameter. Ein Zustand bestehe aus einer Warteschlange von Nachrichten für jedes Postfach, d.h. aus einer Abbildung der Funktionalität $\text{Mbx} \rightarrow \text{Msg}^{\omega}$. Der Anfangszustand init sei die Abbildung, die jedes Postfach auf ε abbildet, d.h. alle Warteschlangen sind anfangs leer. hs ist definiert durch (t sei ein Strom, s ein Zustand):

$$\neg\, \text{full}(s(mb)) \Rightarrow \text{hs}(\text{snd}(ts,mb,ms) \circ t, s) = \text{s-ack}(ts,mb,ok) \circ \text{hs}(t, s[s(mb) \circ ms \,/\, mb]),$$

$$\text{full}(s(mb)) \Rightarrow \text{hs}(\text{snd}(ts,mb,ms) \circ t, s) = \text{s-ack}(ts,mb,error) \circ \text{hs}(t,s),$$

$$\neg\, \text{empty}(s(mb)) \Rightarrow \text{hs}(\text{rec}(ts,mb) \circ t, s) = \text{r-ack}(ts,mb,\text{ft}(s(mb)),ok) \circ \text{hs}(t, s[\text{rt}(s(mb)) \,/\, mb]),$$

$$\text{empty}(s(mb)) \Rightarrow \text{hs}(\text{rec}(ts,mb) \circ t, s) = \text{r-ack}(ts,mb,nil,error) \circ \text{hs}(t,s).$$

---

[1] Ein deterministischer stromverarbeitender Agent entspricht formal einer einelementigen Menge. In der Regel wird in diesem Fall nicht diese Menge, sondern die stromverarbeitende Funktion selbst spezifiziert, vgl. das folgende Beispiel.

[2] Eine ähnliche Realisierung findet sich in [Dendorfer 91], worin eine funktionale Modellierung des (umfangreicheren) MMK-Postfachsystems ([Bemmerl et al. 90]) beschrieben ist.

s[x / mb] steht für die punktweise Änderung der Funktion s (vgl. Abschnitt 3.4.2). Man erkennt die Ähnlichkeit zu dem Transitionssystem, das für die Komponente SERVER angegeben wurde. Eine genauere Begründung, warum die Funktion server die Komponente SERVER realisiert, findet sich in Abschnitt 5.2, wo Agenten Spuren zugeordnet werden. □

Für stromverarbeitende Agenten ist eine *denotationelle Semantik (Fixpunktsemantik, mathematische Semantik)* üblich. (Es gibt jedoch auch operationelle Semantiken, vgl. [Faustini 82], [Lynch, Stark 89], [Jonsson 87].) Diese denotationelle Semantik basiert auf den Kompositionsformen für stromverarbeitende Funktionen, nämlich der sequentiellen, parallelen und Rückkopplungskomposition; die Rückkopplung wird über Fixpunktbildung behandelt (vgl. [Broy 90]). Ein wesentliches Ergebnis hierbei ist, daß ein Agentennetz wiederum als ein stromverarbeitender Agent gesehen werden kann. Dessen Semantik, eine Menge stromverarbeitender Funktionen, ergibt sich aus der Komposition der Funktionsmengen, die als Semantik den einzelnen Agenten des Netzes zugeordnet sind.

Wie bei der komponentenorientierten Spurspezifikation spielen bei der funktionalen Spezifikation Komponenten eine Rolle, es bestehen jedoch wichtige Unterschiede:

1. Die Festlegung der Produkt- und Umgebungskomponenten bei der Anforderungsspezifikation ist dem Kunden, der hierarchische Entwurf der Produktkomponenten, d.h. die Strukturierung der Produktkomponenten in weitere Teilkomponenten, dagegen dem Systementwickler überlassen; dieser muß dabei nur gewährleisten, daß die Produktkomponenten die in der Anforderungsspezifikation festgelegten Eigenschaften besitzen.

2. Während bei einer komponentenorientierten Spezifikation zumindest manche Anforderungen *global* an das zu erstellende Produkt gerichtet sein können, sind die Beschreibungen der Produktkomponenten auf der Entwurfsebene *lokal*; dies gestattet eine modulare Weiterentwicklung des zu erstellenden Produkts.

3. Ein Entwurf der Produktkomponenten wird in der Regel eine hierarchische Struktur aufweisen; diese ist ein Ergebnis von Entwurfsentscheidungen. Auf der Anforderungsebene interessieren wir uns dagegen nicht für die interne Struktur einer Komponente. Prinzipiell läßt sich jedoch ähnlich wie durch eine Spurspezifikation auch durch einen stromverarbeitenden Agenten allein die *Schnittstelle* einer Komponente spezifizieren.

4. Auf der Entwurfsebene werden stromverarbeitende Agenten zur Beschreibung von Komponenten verwendet, nicht Spuren. Die Entscheidung für stromverarbeitende Agenten ist dadurch motiviert, daß sich eine lokale und hierarchische Komponentenbeschreibung (Punkte 1 und 2) damit gut erreichen läßt und zudem der Formalismus technisch gut handhabbar ist. Zudem fällt die Anknüpfung zu Programmbeschreibungen von stromverarbeitenden Agenten aus leichter.

5. Der Freiheitsraum, den eine Anforderungsspezifikation für eine Realisierung läßt, kann durch eine Entwurfsspezifikation bereits eingeschränkt sein.

5.2 Spursemantik für Agenten

Die Verbindung zwischen einer funktionalen Beschreibung und einer Spurbeschreibung wird durch die *Spursemantik* für stromverarbeitende Agenten hergestellt, die mit der denotationellen Semantik verträglich ist. Die Spursemantik eines einzelnen Agenten ist folgendermaßen definiert:

Definition (Spursemantik eines stromverarbeitenden Agenten): Sei ein stromverarbeitender Agent durch eine Menge F von stromverarbeitenden Funktionen der Funktionalität $I_1^\omega \times ... \times I_m^\omega \rightarrow O_1^\omega \times ... \times O_n^\omega$ gegeben. Dabei seien die Mengen I_1, ..., I_m, O_1, ..., O_n paarweise disjunkt. Die *Spursemantik* dieses Agenten ist das Tripel (I,O,T) mit

$$I =_{def} I_1 \cup ... \cup I_m,$$
$$O =_{def} O_1 \cup ... \cup O_n,$$
$$T =_{def} \bigcup_{f \in F} \text{Traces}(f)$$

wobei gilt:

$$\text{Traces}(f) =_{def} \{t \in (I \cup O)^\omega \mid \langle O_1 \copyright t,...,O_n \copyright t\rangle = f(I_1 \copyright t,...,I_m \copyright t) \wedge \forall s: s \sqsubseteq t \Rightarrow \langle O_1 \copyright s,...,O_n \copyright s\rangle \sqsubseteq f(I_1 \copyright s,...,I_m \copyright s) \}. \square$$

Gemäß dieser Definition müssen die Eingaben, die für eine Ausgabe kausal sind, vor dieser Ausgabe in jeder Spur erscheinen. Die Disjunktheit der Ein- und Ausgabemengen fordere ich, um auch auf der Spurebene zwischen Ein- und Ausgaben unterscheiden zu können, genauer gesagt, um feststellen zu können, auf welchem Kanal eine Nachricht gesendet wird. Diese Anforderung stellt keine große Einschränkung dar: gleiche Nachrichten auf verschiedenen Kanälen lassen sich leicht mit einem Unterscheidungskennzeichen, z.B. dem Kanalnamen, versehen.

Bei der Spursemantik von *Agentennetzen* werden die in Abschnitt 4.1 vorgestellten Kompositionsoperatoren || und *hide* verwendet. Die Verträglichkeit mit der denotationellen Semantik eines Agentennetzes ergibt sich aus theoretischen Ergebnissen aus [Jonsson 87], wo dies für das etwas allgemeinere Konzept der I/O-Automaten gezeigt wird. Ich erläutere im folgenden, wie sich diese Ergebnisse auf stromverarbeitende Agenten übertragen lassen.

Leicht unterschiedliche Definitionen von I/O-Automaten finden sich in [Jonsson 87] und [Lynch, Stark 89]; ich verwende Jonssons Definitionen von *I/O-Automaten*, ihren *Berechnungen*, ihren *akzeptierten Spuren* und deren Komposition.

Informell gesprochen beschreiben I/O-Automaten Agenten, deren Verhalten durch zwei Bedingungen charakterisiert ist:

(1) Ein I/O-Automat akzeptiert immer alle Eingaben.
(2) Das Senden einer Nachricht durch einen I/O-Automaten und ihr Empfang von einem anderen I/O-Automaten ist eine atomare Operation, die sich gleichzeitig im sendenden und empfangenden Automaten vollzieht.

Formal ist ein I/O-Automat wie folgt definiert:

Definition (I/O-Automat): Ein *I/O-Automat* ist ein Tupel $(I,O,State,s_0,—;>,F)$. Dabei ist

I eine Menge von *Eingabeaktionen*, die nicht die stille Aktion τ enthält,

O eine Menge von *Ausgabeaktionen*, die zu I disjunkt ist und τ nicht enthält,

State eine Menge von *Zuständen*,

s_0 der Anfangszustand des Automaten,

$—;>$ eine Teilmenge von $State \times (I \cup O \cup \{\tau\}) \times State$, die Menge der *Transitionen*. Für $(s,a,s') \in —;>$ wird auch $s —;^{a}> s'$ geschrieben.

F eine endliche Menge von *Fairneßmengen*. Jede Fairneßmenge ist eine Teilmenge der Übergangsrelation $—;>$.

Zudem müssen die folgenden Bedingungen erfüllt sein:

1) Für jeden Zustand s und jede Eingabeaktion i gibt es einen Zustand s', so daß $s —;^{i}> s'$ gilt.

2) Keine Fairneßmenge $F \in F$ enthält eine Transition $s —;^{i}> s'$ mit $i \in I$.

3) Jede Transition $s —;^{a}> s'$ mit $a \in O \cup \{\tau\}$ ist Element einer Fairneßmenge in *F*.

□

1) besagt, daß der Automat immer alle Eingaben akzeptiert. Wie aus den Berechnungen des Automaten ersichtlich ist (s.u.), wird dadurch ausgeschlossen, daß Eingaben die durch die Transitionsrelation und den Anfangszustand festgelegte Sicherheitsanforderung (vgl. Abschnitt 3.4) verletzen. 2) bedeutet, daß an Eingaben keine Lebendigkeitsanforderung (Fairneß) gestellt wird. 3) drückt aus, daß der Automat alle in seinen Verantwortungsbereich fallenden Transitionen, das sind stille Transitionen und Ausgabetransitionen, fair behandelt. Stille Aktionen entstehen bei Anwendung des Versteckoperators für I/O-Automaten.

Der Unterschied zu dem in Abschnitt 3.4 vorgestellten Begriff des Transitionssystems liegt darin, daß zwischen Ein- und Ausgaben unterschieden wird und Lebendigkeitsanforderungen in I/O-Automaten in Form von Fairneßmengen verankert sind.

Das Verhalten von I/O-Automaten wird operationell beschrieben. Berechenbarkeitsannahmen werden nicht getroffen, denn die Zustände und Zustandsübergänge können beliebig gewählt werden.

Sei für die folgenden Definitionen A ein I/O-Automat $(I,O,State,s_0,—;>,F)$.

Definition (schaltbereite Transition): Eine Transition $s —;^{a}> s'$ heißt *schaltbereit* in s. Eine Fairneßmenge F ist in s schaltbereit, wenn eine Transition aus F in s schaltbereit ist.

□

Definition (Berechnung): Eine *Berechnung* von A ist eine endliche oder unendliche Sequenz der Art

$$s_0 ——;^{a1}> s_1 ——;^{a2}> \ldots ——;^{an}> s_n ———;^{an+1}> \ldots$$

(eine endliche Sequenz ende mit einem Zustand), wobei gilt:

(A) *Initialisierung:* s_0 ist der Anfangszustand von A.

(B) *Schrittfolge:* Jede Transition $s_n ——;^{an+1}> s_{n+1}$ liegt in $—;>$.

(C) *(starke) Fairneß:* Ist die Sequenz unendlich, so muß sie unendlich viele Vorkommen von Transitionen jeder Fairneßmenge $F \in F$ enthalten, die unendlich oft schaltbereit ist.

(D) *Stabilität:* Ist die Sequenz endlich, so darf weder eine Ausgabetransition noch eine stille Transition in ihrem letzten Zustand schaltbereit sein. □

Die Sicherheitseigenschaften des Automaten werden durch (A) und (B) ausgedrückt. Die Bedingungen (C) und (D) legen Lebendigkeitseigenschaften fest; zusammen bilden sie das Akzeptierkriterium für endliche und unendliche Worte (akzeptierte Spuren). (C) drückt starke Fairneß aus. Im Gegensatz dazu würde schwache Fairneß fordern, daß nur die Transitionen von jenen Fairneßmengen nicht ignoriert werden dürfen, die fortwährend schaltbereit sind. Die wesentlichen Ergebnisse über I/O-Automaten ändern sich nicht, wenn die schwache statt der starken Fairneß gewählt wird ([Jonsson 90]), insbesondere bleiben meine Ergebnisse davon unberührt.

Definition (Spursemantik eines I/O-Automaten): Eine *akzeptierte Spur* von A ist die Sequenz der Eingabe- und Ausgabeaktionen (d.h. Aktionen ungleich τ) in einer Berechnung von A. Die Menge der von A akzeptierten Spuren wird mit Traces(A) bezeichnet. Die *Spursemantik* eines I/O-Automaten A ist das Tripel (I,O,Traces(A)). □

Jonsson gibt Kompositionsoperatoren (|| und *hide*) für Automaten und Spurmengen an und zeigt, daß die Komposition von I/O-Automaten wieder einen I/O-Automaten ergibt.

Definition (Kompositionsoperator ||): Die Komposition zweier I/O-Automaten $A_1 = (I_1,O_1,State_1,s_{0_1},\text{---};_{>1},F_1)$ und $A_2 = (I_2,O_2,State_2,s_{0_2},\text{---};_{>2},F_2)$ wird mit $A_1 \parallel A_2$ bezeichnet. Sie ist definiert, wenn $O_1 \cap O_2 = \emptyset$. $A_1 \parallel A_2 =_{def} (I,O,State,s_0,—;>,F)$ mit

$O = O_1 \cup O_2$, $I = (I_1 \cup I_2) - O$, $State = State_1 \times State_2$, $s_0 = \langle s_{0_1},s_{0_2}\rangle$,
—;> ist die kleinste Relation, für die gilt:

s_1 —;^{a}>$_1$ s_1', a in s_2 nicht schaltbereit $\Rightarrow$ $\langle s_1,s_2\rangle$ —;^{a}> $\langle s_1',s_2\rangle$,
s_2 —;^{a}>$_2$ s_2', a in s_1 nicht schaltbereit $\Rightarrow$ $\langle s_1,s_2\rangle$ —;^{a}> $\langle s_1,s_2'\rangle$,
s_1 ---;^{a}>$_1$ s_1', s_2 ---;^{a}>$_2$ s_2' $\Rightarrow$ $\langle s_1,s_2\rangle$ ---;^{a}> $\langle s_1',s_2'\rangle$,
s_1 ---;$^{\tau}$>$_1$ s_1' $\Rightarrow$ $\langle s_1,s_2\rangle$ ---;$^{\tau}$> $\langle s_1',s_2\rangle$,
s_2 ---;$^{\tau}$>$_2$ s_2' $\Rightarrow$ $\langle s_1,s_2\rangle$ ---;$^{\tau}$> $\langle s_1,s_2'\rangle$,

F ist so definiert, daß es für jedes $F_i \in F_i$ es eine Fairneßmenge $F \in F$ gibt, die aus allen Übergängen besteht, die sich gemäß den obigen Regeln mit Übergängen aus F_i ergeben. □

Analog ist die n-fache Komposition von I/O-Automaten $A_1 \parallel ... \parallel A_n$ definiert (für Details siehe [Jonsson 87]). || bildet aus n Automaten ein Netz, bei dem zwischen zwei Automaten eine Verbindung besteht, wenn die Eingabemenge des einen Automaten einen nichtleeren Schnitt mit der Ausgabemenge des anderen Automaten hat.

Definition (Versteckoperator hide): Für einen I/O-Automaten $A = (I,O,State,s_0,—;>,F)$ und eine Aktionenmenge $B \subseteq O$ ist *hide* B *in* A als der Automat $(I,O - B,State,s_0,—;>',F')$ definiert. Hierbei ist —;>' als die kleinste Relation definiert, für die gilt:

$a \notin B \wedge s$ —;^{a}> $s' \Rightarrow s$ —;^{a}>' s', $\qquad a \in B \wedge s$ —;^{a}> $s' \Rightarrow s$ —;$^{\tau}$>' s'

F' ist so definiert, daß es für jede Fairneßmenge $F \in F$ es eine Fairneßmenge $F' \in F'$ gibt, die alle Transitionen enthält, die aus einer Transition aus F gemäß der Definition von $—;>'$ entstehen. □

Jonsson zeigt, daß die Spurmenge des komponierten Automaten aus der Komposition der Spurmengen der zu komponierenden Automaten resultiert, wobei die entsprechenden Kompositionsoperatoren auf der Spurebene verwendet werden (seien $A_1, ..., A_n$ und A I/O-Automaten und B eine Teilmenge der Ausgaben von A):

$$\text{Traces}(A_1 \|_a ... \|_a A_n) = \text{Traces}(A_1) \|_t ... \|_t \text{Traces}(A_n),$$
$$\text{Traces}(\mathit{hide}_a\ B\ \mathit{in}\ A) = \mathit{hide}_t\ B\ \mathit{in}\ \text{Traces}(A),$$

wobei zur Verdeutlichung Unterscheidungsindizes *a ("automaton")* und *t ("trace")* an die Kompositionsoperatoren angebracht sind.

Die Spursemantik ist also verträglich mit der operationellen Semantik.

Beispiel (Postfachsystem: I/O-Automat): Die Komponente SERVER (vgl. Abschnitt 4.1 und das entsprechende Transitionssystem in Abschnitt 3.4) läßt sich leicht durch einen I/O-Automaten realisieren: Der I/O-Automat habe die Eingabemenge $Snd \cup Rec$, die Ausgabemenge $S\text{-}ack \cup R\text{-}ack$ sowie die Zustandsmenge, den Anfangszustand und die Zustandsübergänge wie in Abschnitt 3.4 beschrieben; die Lebendigkeitsanforderungen $S\text{-}ack\_live$ und $R\text{-}ack\_live$ lassen sich durch diejenige Fairneßmenge realisieren, die alle Übergänge enthält, die mit einer Ausgabeaktion markiert sind. Wird irgendwann eine Nachricht von einem Prozeß zu einem Postfach geschickt, so wird eine entsprechende Rückmeldeaktion in der Zustandskomponente $acks$ gespeichert. Da Zustandsübergänge, die mit Rückmeldeaktionen markiert sind, immer schaltbereit sind, wird die entsprechende Rückmeldeaktion gemäß der Fairneßannahme irgendwann später auch tatsächlich ausgeführt.

Beim Postfachbeispiel sieht man relativ leicht, wie sich die globale Anforderungsspezifikation in Komponentenspezifikationen umsetzen läßt und wie die SERVER-Spezifikation durch einen Agenten realisiert werden kann. Der Grund liegt (abgesehen von der Einfachheit des Beispiels) darin, daß alle Beschreibungen transitionsorientiert sind und zudem die Zustandsräume und -übergänge gleich sind. Lediglich die Lebendigkeitsanforderungen werden unterschiedlich realisiert. Schwieriger wird der Nachweis der Implementierungsrelation, wenn die Beschreibungen stark voneinander abweichen. □

I/O-Automaten sind etwas allgemeiner als stromverarbeitende Agenten, da eine neue Eingabe eine Ausgabe, die vorher möglich war, revidieren kann:

Beispiel: Der folgende I/O-Automat

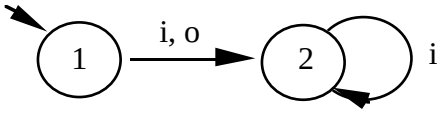


Fig. 5.1

(mit Eingabe i, Ausgabe o, Anfangszustand 1 und der Fairneßmenge, die allein den o-Übergang enthält) erzeugt die Spuren $o \circ i^\omega \cup i^\omega$ (angelehnt an die Notation regulärer

Ausdrücke), nicht jedoch eine Spur mit Präfix $i \circ o$, was bei einem stromverarbeitenden Agenten der Fall sein müßte. □

In [Jonsson 87] wird auch eine spezielle Klasse von I/O-Automaten angegeben, nämlich solche I/O-Automaten, die einen FIFO-Eingabepuffer für jeden Eingabekanal der Agenten im Zustand enthalten. Intuitiv gesehen sind dies gerade stromverarbeitende Agenten. In [Lynch, Stark 89] wird gezeigt, daß die Teilklasse der sog. "determinierten" I/O-Automaten (stetige) stromverarbeitende Funktionen beschreibt. Ich führe nun vor, wie sich eine funktionale Beschreibung eines stromverarbeitenden Agenten schematisch in eine Beschreibung eines I/O-Automaten umwandeln läßt, was weder in [Jonsson 87] noch in [Lynch, Stark 89] gezeigt wird. Mit diesen Ergebnissen läßt sich die Spursemantik von I/O-Automaten auf stromverarbeitende Agenten übertragen.

Zunächst sei eine einzelne stromverarbeitende Funktion $f: I^\omega \to O^\omega$ mit $I \cap O = \emptyset$ betrachtet. Es läßt sich ein I/O-Automat $(I, O, State, s_0, —;>, F)$ definieren, der gerade die Spuren von f akzeptiert. Die Komponenten State, s_0, $—;>$ und F sind definiert:

$State = I^* \times O^\omega$

$s_0 = \langle \varepsilon, f(\varepsilon) \rangle$

$—;>$ bestehe aus allen Übergängen der Form (1) und (2). Dabei sei $i \in I$ und $\langle x,y \rangle \in State$:

(1) $\langle x,y \rangle —;^{i}> \langle x \circ i, y \circ rt^{\#f(x)}(f(x \circ i)) \rangle$

(2) $\langle x,y \rangle ———;^{ft(y)}> \langle x, rt(y) \rangle$, falls $y \neq \varepsilon$

$F = \{\{ s —;^{a}> s' \mid s —;^{a}> s' \text{ ist Übergang vom Typ (2)} \}\}$

Die erste Zustandskomponente gibt die bisherige Eingabe an, die zweite beschreibt jene Ausgaben, die bereits festgelegt sind, aber noch nicht ausgegeben wurden. Dabei kann $rt^\infty(t)$ beliebig definiert werden, z.B. als ε.

Die Erweiterung auf Funktionen mit mehr als einen Eingabekanal und mehr als einem Ausgabekanal ist offensichtlich.

Beispiel (Postfachsystem: stromverarbeitender Agent und I/O-Automat): Gemäß der obigen Konstruktion läßt sich aus der stromverarbeitenden Funktion server (vgl. Abschnitt 5.1) ein I/O-Automat erzeugen, der dem oben angegebenen I/O-Automaten für das Postfachsystem entspricht. Dies zeigt, daß die Funktion server ebenfalls die Komponente SERVER realisiert. □

Bei der Konstruktion eines I/O-Automaten für einen nichtdeterministischen Agenten, der durch eine *Menge* stromverarbeitender Funktionen gegeben ist, müssen die "Berechnungspfade" der einzelnen Funktionen auseinandergehalten werden. Dies gelingt z.B. auf die folgende Weise: Die Zustandsmengen der I/O-Automaten für die einzelnen Funktionen der Menge werden disjunkt gemacht, ihre Anfangszustände werden identifiziert und ihre Faireßmengen vereinigt. Es ist leicht zu sehen, daß sich auf diese Weise wieder ein I/O-Automat ergibt, nämlich jener, dessen Spurmenge die Vereinigung der Spurmengen der einzelnen Automaten ist.

Die für stromverarbeitende Agenten üblichen Kompositionsoperatoren (sequentielle, parallele und Rückkopplungskomposition) lassen sich leicht durch den ||-Operator und den Versteckoperator ausdrücken, und zwar in dem Sinne, daß durch die Komposition der den Funktionen entsprechenden I/O-Automaten die gleichen Spuren erzeugt werden wie durch die Komposition der Funktionen. Dies wird im folgenden skizziert.

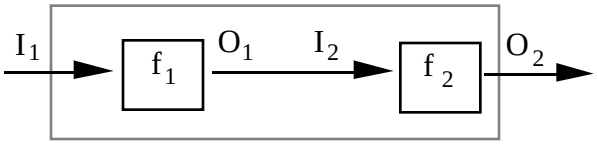
Hierzu betrachten wir Funktionen, deren Ein- und Ausgabemengen von verschiedenen Kanälen paarweise disjunkt sind, d.h. die Nachrichten sind eindeutig den verschiedenen Ein- und Ausgabekanälen zugeordnet. Formal:

Für $f: I_1^{\omega} \times ... \times I_m^{\omega} \rightarrow O_1^{\omega} \times ... \times O_n^{\omega}$ gelte $I_i \cap I_j = \emptyset$, $O_i \cap O_j = \emptyset$, $I_i \cap O_k = \emptyset$ für alle i,j,k mit $i \neq j$. (*)

Diese Annahme ist nötig, um die Beziehung zwischen Funktions- und Automatennetzen herzustellen. Anders als bei stromverarbeitenden Funktionen wird das Ein-/Ausgabeverhalten eines I/O-Automaten sowie die Verbindungsstruktur eines Automatennetzes nämlich nicht über eine Funktionalität, sondern über die Identität der Aktionen bestimmt. Durch Umbenennung von Aktionen, etwa durch die Markierung jeder Aktion mit einem Kanalnamen, läßt sich diese Bedingung immer leicht einhalten.

Im folgenden bezeichne [f] einen der Funktion f entsprechenden I/O-Automaten (s.o.). Bewiesen wird die Korrektheit der Komposition nur für den Fall der Rückkopplung; die anderen beiden Fälle verlaufen analog.

1. Sequentielle Komposition

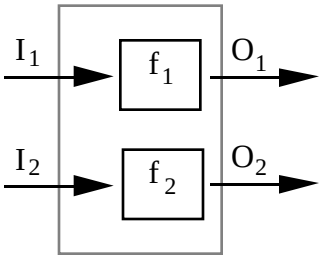


f ; g Fig. 5.2

Hierbei muß $O_1 = I_2$ und $I_1 \cap O_2 = \emptyset$ gelten; ersteres ist für die sequentielle Komposition nötig, letzteres, damit das Ergebnis die Bedingung (*) erfüllt.

Es gilt: $Traces(f_1 ; f_2) = Traces(\textit{hide}\ O_1\ \textit{in}\ ([f_1] \parallel [f_2]))$

2. Parallele Komposition



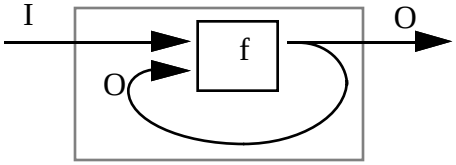
f_1 *par* f_2 Fig. 5.3

Hierbei muß $I_1 \cap I_2 = \emptyset$, $I_1 \cap O_2 = \emptyset$ und $I_2 \cap O_1 = \emptyset$ gelten, um (*) einzuhalten.

Es gilt: $\text{Traces}(f_1 \; \textit{par} \; f_2) = \text{Traces}([f_1] \parallel [f_2])$

3. Rückkopplung

Die direkte Rückkopplung ist bei I/O-Automaten nicht definiert. Im folgenden Bild gälte sonst für f nicht, daß Ein- und Ausgabemengen disjunkt sind, was die Bedingung (*) verletzt. Ein Netz der Art



μf Fig. 5.4

wird daher ersetzt durch das Netz

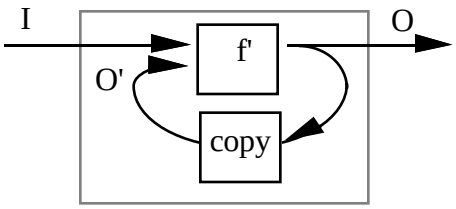


Fig. 5.5

Hierbei ist copy eine bijektive Funktion von O nach einer zu I und O disjunkten Menge O'. copy(o) wird im folgenden mit o' abgekürzt; dies wird auf Spuren erweitert: $(o_1 \circ o_2 \circ \ldots)' = o_1' \circ o_2' \circ \ldots$. Für $f: I^\omega \times O^\omega \to O^\omega$ ist $f': I^\omega \times O'^\omega \to O^\omega$ definiert $f(x,y) = z \Leftrightarrow f'(x,y') = z$.

Es gilt: $\text{Traces}(\mu f) = \text{Traces}(\textit{hide}\ O' \ \textit{in}\ ([f'] \parallel [\text{copy}]))$

Beweis: Der Beweis wird hier nur für den Fall $I = \emptyset$ geführt, d.h. f hat nur einen Eingabekanal, f ist also von der Funktionalität $O \to O$. Analog beweist man den allgemeinen Fall.

$\text{Traces}(\mu f) = \{\text{fix}(f)\}$, wobei fix(f) den kleinsten Fixpunkt der Funktion f bezeichnet. $\text{Traces}((\textit{hide}\ O' \ \textit{in}\ ([f'] \parallel [\text{copy}]))$ ist nicht leer, da jeder I/O-Automat eine nicht-leere Spurmenge hat. Man wähle ein $t \in \text{Traces}((\textit{hide}\ O' \ \textit{in}\ ([f'] \parallel [\text{copy}]))$. Es wird gezeigt, daß $t = \text{fix}(f)$ gilt. Daraus folgt sofort die Behauptung.

(Zur Verdeutlichung sind im folgenden Unterscheidungsindizes *a* für die I/O-Automaten-Operatoren und *t* für die entsprechenden Spuroperatoren angebracht.)

	$t \in \text{Traces}(\textit{hide}_a\ O' \ \textit{in}\ ([f'] \parallel_a [\text{copy}]))$		{Eigenschaft von $\parallel$ und *hide*}
$\Leftrightarrow$	$t \in \textit{hide}_t\ O' \ \textit{in}\ (\text{Traces}(f') \parallel_t \text{Traces}(\text{copy}))$		{Definition von $\textit{hide}_t$}
$\Leftrightarrow$	$\exists r: r \in \text{Traces}(f') \parallel_t \text{Traces}(\text{copy}) \wedge t = O \copyright r$		{Definition von $\parallel_t$}
$\Leftrightarrow$	$\exists r: r \in \text{Traces}(f') \wedge r \in \text{Traces}(\text{copy}) \wedge t = O \copyright r$		{Definition von Traces}
$\Leftrightarrow$	$\exists r:$	$O \copyright r = f'(O' \copyright r)$	(1)
		$\wedge\ \forall s \sqsubseteq r: O \copyright s \sqsubseteq f'(O' \copyright s)$	(2)
		$\wedge\ O' \copyright r = (O \copyright r)'$	(3)

$\wedge\ \forall s \sqsubseteq r: O'©s \sqsubseteq (O©s)'$ (4)
$\wedge\ t = O©r$

Man wähle ein solches $r \in (O \cup O')^\omega$ (Dies existiert, da Traces((*hide* O' *in* ([f'] || [copy])) $\neq$ Ø). Nach (1) und (3) gilt: $f'(O'©r) = f'((O©r)') = O©r$; also ist $O©r$ Fixpunkt von f. Nun ist zu zeigen: $O©r$ ist *kleinster* Fixpunkt von f. Sei $z \in O^\omega$ kleinster Fixpunkt von f, gelte also:

$f'(z') = z$ (5)
$\wedge\ f'(x') = x \Rightarrow z \sqsubseteq x$ (6)

Zu zeigen: $z \sqsupseteq O©r$ (*)

Mit (1) und (6) gilt $z \sqsubseteq O©r$, daher mit (3) $z' \sqsubseteq (O©r)' = O'©r$. Deshalb gibt es größte u, v $\in (O \cup O')^\omega$ mit

$u \sqsubseteq r,\ v \sqsubseteq r$ (7)
$z = O©u$ (8)
$z' = O'©v$ (9)

Behauptung: Es gilt

$u \sqsubseteq v$ (10)

Sonst gälte mit (7) $v \sqsubset u$, damit mit Monotonie $O'©v \sqsubseteq O'©u$ und $O'©v =;^{(9)} z' =;^{(8)} (O©u)' \sqsupseteq;^{(4)} O'©u$ und daher $O'©v = O'©u$. Dies wäre ein Widerspruch zur Annahme, daß v das größte Element ist, so daß (9) gilt.

$z =;^{(5)} f'(z') =;^{(9)} f'(O'©v) \sqsupseteq;^{(2)} O©v \sqsupseteq;^{(10)} O©u = z$. Daher:

$O©v = O©u$ (11)

Da u "größtmöglich" gewählt und $u \sqsubseteq v$ muß $u = v$ gelten.

Ist u echtes Präfix von r, so gilt $r = u \circ o \circ s$ mit $o \in O$ und $s \in (O \cup O')^\omega$ (sonst u nicht "größtmöglich"). Ist v echtes Präfix von r, so gilt $r = v \circ o' \circ p$ mit $o' \in O'$ und $p \in (O \cup O')^\omega$ (sonst v nicht "größtmöglich"). u (= v) kann also kein echtes Präfix von r sein, daher (mit (10)):

$u = v = r$ (12)

Damit gilt: $z =;^{(8)} O©u =;^{(12)} O©r$. Damit ist (*) gezeigt. □

Für eine weitergehende Analyse der Beziehung zwischen I/O-Automaten und stromverarbeitenden Agenten vgl. [Lynch, Stark 89].

Die Spursemantik für Agenten bietet den Vorteil, daß für unterschiedliche Komponenten eines verteilten Systems unterschiedliche Spezifikationstechniken eingesetzt werden können; der

Zusammenhalt ist durch die Spursemantik gewährleistet. Die Erfahrung zeigt nämlich, daß für manche Komponenten bereits am Anfang der Systementwicklung eine Agentenspezifikation angegeben werden kann, während bei anderen z.B. eine Beschreibung mit Spurformeln einfacher ist, die dann schrittweise in konkretere Beschreibungsformen umgesetzt wird.

5.3 Methodische Vorgehensweise bei der Entwurfsspezifikation

Die Unterschiede zwischen einer komponentenorientierten Spurspezifikation und einer funktionalen Spezifikation (vgl. Abschnitt 5.1) bestimmen die Entwicklungsrichtung beim Übergang zwischen diesen beiden Beschreibungsformen: die Anforderungen an die Produktkomponenten sind zu lokalisieren, von der Spurbeschreibung der Produktkomponenten ist zur funktionalen Beschreibung dieser Komponenten zu wechseln. Liegen die Anforderungen an die Produktkomponenten erst einmal lokal vor, so sind die Anforderungen an die Umgebungskomponenten für die weitere Systementwicklung nicht mehr von Bedeutung. Die Systementwicklung kann dann auf modulare Weise fortschreiten, basierend auf den lokalen Beschreibungen (Schnittstellen) der Produktkomponenten.

Die Lokalisierung der Anforderungen läßt sich mit der Methode der *schrittweisen Verfeinerung* durchführen: Ein System bestehe aus n Komponenten mit Ein- und Ausgaben $I_1, O_1, ..., I_n, O_n$. Auf jeder Verfeinerungsstufe gibt es eine globale Anforderung G und lokale Anforderungen $C_1, ..., C_n$. In der komponentenorientierten Anforderungsspezifikation, dem Startpunkt der schrittweisen Verfeinerung, ist G die globale Spezifikation und C_i gibt entweder die Schnittstelle einer Umgebungskomponente (falls i Index einer Umgebungskomponente) oder eine lokale (Teil-)Anforderung an eine Produktkomponente wieder (falls i Index einer Produktkomponente). Ein Verfeinerungsschritt besteht darin, eine globale Anforderung G' sowie lokale Anforderungen C_i' für die Produktkomponenten zu finden, so daß gilt:

$$\begin{aligned} & G'(t) \wedge C_1'((I_1 \cup O_1) \copyright t) \wedge ... \wedge C_n'((I_n \cup O_n) \copyright t) \\ \Rightarrow \quad & G(t) \wedge C_1((I_1 \cup O_1) \copyright t) \wedge ... \wedge C_n((I_n \cup O_n) \copyright t). \end{aligned}$$

Die Anforderungen an die Umgebungskomponenten müssen gleich bleiben, d.h. ist i Index einer Umgebungskomponente, so gilt $C_i'(t) \Leftrightarrow C_i(t)$. Die Lokalisierung der Anforderungen ist beendet, wenn der globale Anteil gleich *true* ist, d.h. es werden keine globalen Anforderungen mehr gestellt. Ein Verfeinerungsschritt ist daher zielgerichtet, wenn $G(t) \Rightarrow G'(t)$ gilt, aber nicht die Gegenrichtung; in diesem Fall wird tatsächlich ein Teil der globalen Anforderungen lokalisiert.

Die Lokalisierung ist ein Entwurfsschritt, daher nur wenig schematisierbar (vgl. Abschnitt 4.3). Ich verweise auf die Arbeiten [Li, Maibaum 88] und [Chandy, Misra 88], wo die schrittweise Verfeinerung an Beispielen demonstriert wird. Wie in unserer Methodik lassen sich dort die Verfeinerungsschritte in einem logischen Kalkül durchführen. Man sieht leicht, daß sich die prinzipielle Vorgehensweise, insbesondere Heuristiken, aus den erwähnten Arbeiten auch in unserer Entwurfsmethodik verwenden lassen.

Das Ergebnis der Verfeinerungen ist "vernünftig", wenn sich die Produktkomponenten realisieren lassen. Dies ist eine eher implizite Anforderung, die erst im Laufe der Systementwicklung festgestellt wird. Daher ist es möglich, daß Entwurfsentscheidungen rückgängig gemacht werden müssen, wenn globale Anforderungen derart den Produktkomponenten zugeordnet werden, daß sie nicht realisierbar sind (bei einem vorgegebenen Realisierungskonzept, hier stromverarbeitende Agenten, vgl. Abschnitt 5.4). Eventuell muß sogar die Komponentenstruktur des Systems geändert werden.

Die Realisierbarkeit bringt uns zur zweiten Aufgabe bei der Entwurfsspezifikation: dem Wechsel von einer Spur- zu einer funktionalen Beschreibung der Produktkomponenten, d.h. der Umsetzung eines Prädikats über Spuren in ein Prädikat über stromverarbeitende Funktionen. Unser semantisches Modell (vgl. Abschnitt 5.2) ist so gewählt, daß dieser Übergang (zumindest von der theoretischen Seite her) keine Schwierigkeit darstellt. Für ein Prädikat P über Spuren läßt sich ein Prädikat über Funktionen angeben, das den größtmöglichen realisierbaren Anteil der durch P festgelegten Spurmenge beschreibt:

$$Q_P(f) \equiv \forall t: t \in \text{Traces}(f) \Rightarrow P(t)$$

Das heißt, wir spezifizieren jene Menge stromverarbeitender Funktionen, die nur solche Spuren erzeugen, die das Prädikat P erfüllen.

Umgekehrt läßt sich auch jedes Prädikat Q über stromverarbeitende Funktionen als Prädikat über Spuren interpretieren:

$$P_Q(t) \equiv \exists f: Q(f) \wedge t \in \text{Traces}(f)$$

Auf diese Weise lassen sich Spur- und Funktionsbeschreibungen flexibel kombinieren.

Eine lokale Komponentenspezifikation P ist durch stromverarbeitende Agenten realisierbar, wenn das entsprechende Prädikat über (stetigen) Funktionen Q_P nicht identisch *false* ist. Wir erkennen, daß es (auf der Spurebene) konsistente Spezifikationen gibt, die sich (bei einer bestimmten Festlegung der Ein- und Ausgaben) nicht realisieren lassen. Ist Q_P identisch *false*, so müssen - wie oben angesprochen - Entwurfsentscheidungen revidiert werden.

5.4 Hierarchie der Realisierbarkeit

In den Abschnitten 4.3 und 5.2 lernten wir verschiedene Arten der Realisierung von Spurmengen kennen: sowohl einer Strategie, als auch einem stromverarbeitender Agenten und einem I/O-Automaten läßt sich eine Spurmenge zuordnen, die als Beschreibung des Ein-/Ausgabeverhaltens einer Komponente interpretiert werden kann. Tatsächlich gibt es eine "Hierarchie der Realisierbarkeit", nämlich

$$\text{stromverarbeitende Agenten} \subset \text{Strategien} \subset \text{I/O-Automaten} \qquad (*)$$

$A \subset B$ ist folgendermaßen zu interpretieren: für jedes $a \in A$ gibt es ein $b \in B$ (mit gleichen Ein- und Ausgaben), so daß $\text{Traces}(a) = \text{Traces}(b)$ gilt; z.B. läßt sich jede Spurmenge, die sich

durch einen stromverarbeitenden Agenten realisieren läßt, auch durch eine Strategie realisieren. Die Korrektheit der Aussage (*) zeige ich in diesem Abschnitt.[1]

Prinzipiell ließen sich zur Agentenbeschreibung auf der Entwurfsebene auch Strategien und I/O-Automaten einsetzen. Die Entscheidung für stromverarbeitende Agenten bringt jedoch zwei Vorteile: erstens lassen sich mit ihnen asynchron kommunizierende Agenten gut modellieren, ohne daß Kommunikationskanäle explizit beschrieben werden müssen. Dies ist mit Strategien und I/O-Automaten nicht (ohne eine Einschränkung auf eine Teilklasse) möglich, da in beiden Konzepten eine neue Eingabe eine bereits festgelegte Ausgabe revidieren kann. Zweitens ermöglichen sie eine funktionale Beschreibung eines verteilten Systems, was neben einem einfachen konzeptuellen Modell auch eine relativ einfache technische Handhabbarkeit bietet.

Die Einführung von Strategien und I/O-Automaten (statt allein stromverarbeitende Funktionen) im Rahmen meiner Arbeit hat folgende Gründe: Mit Strategien läßt sich das Wechselspiel zwischen einer Komponente und ihrer Umgebung anschaulicher beschreiben und analysieren als mit stromverarbeitenden Agenten; Ergebnisse aus dem Bereich der I/O-Automaten zu einer kompositionalen Spursemantik lassen sich auf stromverarbeitende Agenten übertragen. Die Hierarchie (*) rechtfertigt diese Vorgehensweise, da stromverarbeitende Agenten als Spezialfall von Strategien und I/O-Automaten verstanden werden können.

Zunächst zur Beziehung "stromverarbeitende Agenten $\subset$ Strategien". Die folgenden beiden Sätze zeigen, daß die Teilklasse der FIFO-Strategien und stromverarbeitende Funktionen (mit einem Eingabe- und einem Ausgabekanal) in Bezug auf die Spursemantik gleich mächtig sind.

Satz (FIFO-Strategien bestimmen stromverarbeitende Funktionen): Sei für jede FIFO-Strategie $\sigma: (I \cup O)^* \to O \cup \{\varepsilon\}$ mit $I \cap O = \varnothing$ die Funktion $f_\sigma: I^\omega \to O^\omega$ folgendermaßen definiert:

$$f_\sigma(w) =_{def} O \copyright \text{trace}(\sigma, h_w),$$

wobei für $w \in I^\omega$ die Funktion $h_w: \text{Nat} \to I^*$ definiert ist (sei dabei $w = i_0 \circ i_1 \circ \ldots$ mit $i_k \in I$ für alle k) durch:

$$h_w(k) =_{def} \begin{cases} i_k & \text{falls } 0 \le k < \#w; \\ \varepsilon & \text{falls } k > \#w \end{cases}$$

Dann gilt:

1. f_σ ist wohldefiniert und stetig.
2. $\text{Traces}(f_\sigma) = \text{Traces}(\sigma)$.

---

[1] Die Aussage (*) bezieht sich auf das vorgestellte Konzept stromverarbeitender Agenten. Es ist zu erwarten, daß eine abweichende funktionale Agentenmodellierung, etwa die Modellierung durch Funktionen der Funktionalität $(I^*)^\omega \to (O^*)^\omega$ statt $I^\omega \to O^\omega$, Auswirkungen auf die Realisierbarkeitsbeziehung zwischen den verschiedenen Konzepten hat. Somit spreche in nur den "Standardfall" stromverarbeitender Agenten an.

Beweis: Zu 1. a) Die Wohldefiniertheit von f_σ ist offensichtlich.
b) f_σ ist stetig. Wir spalten b) in b1) und b2) auf:
b1) f_σ ist monoton: Sei $w \sqsubseteq w'$. Zu zeigen ist: $f_\sigma(w) \sqsubseteq f_\sigma(w')$.
Nehmen wir an, daß $w \sqsubseteq w'$ und $\neg f_\sigma(w) \sqsubseteq f_\sigma(w')$ gilt. Also gibt es ein $o \in O$ und ein kleinstes $t \in O^*$, so daß $t \circ o \sqsubseteq f_\sigma(w)$, aber $\neg t \circ o \sqsubseteq f_\sigma(w')$. Da $t \circ o$ endlich ist, muß es ein k geben, so daß $O \copyright ptrace(\sigma, h_w, k) = t \circ o$ (dabei ist $k > \#w$). Man definiere h': Nat $\to I^*$ folgendermaßen:

$h'(p) =_{def} h_w(p) = h_{w'}(p)$, für $p \leq \#w$,
$h'(p) =_{def} h_w(p) = \varepsilon$, für $\#w \leq p < k$,
$h'(k-1+q) =_{def} h_{w'}(\#w+q)$, für alle $q>1$.

(d.h. man ergänze h' so, daß auch der Rest von w' eingegeben wird).
Dann gilt: $conc(h') = w' = conc(h_{w'})$. Da σ eine FIFO-Strategie ist, gilt dann: $O \copyright trace(\sigma, h') = O \copyright trace(\sigma, h_{w'})$. $O \copyright ptrace(\sigma, h', k) = t \circ o$, aber $t \circ o$ ist kein Präfix von $O \copyright trace(\sigma, h_{w'})$ und wir erhalten einen Widerspruch.
b2) $f_\sigma(\bigsqcup_n w_n) \sqsubseteq \bigsqcup_n f_\sigma(w_n)$ für jede Kette $w_0 \sqsubseteq w_1 \sqsubseteq \ldots$
Es läßt sich leicht zeigen, daß es für ein beliebiges $k \in$ Nat es ein m gibt, so daß
$ptrace(\sigma, h_{\bigsqcup_n w_n}, k) \sqsubseteq ptrace(\sigma, h_{w_m}, k)$ gilt. Daher gilt: $f_\sigma(\bigsqcup_n w_n) = \bigsqcup_k O \copyright ptrace(\sigma, h_{\bigsqcup_n w_n}, k) \sqsubseteq \bigsqcup_n \bigsqcup_k O \copyright trace(\sigma, h_{w_n}, k) = \bigsqcup_n f_\sigma(w_n)$.
Zu 2. a) $Traces(f_\sigma) \supseteq Traces(\sigma)$
Sei $t \in Traces(\sigma)$. Dann gilt $t = trace(\sigma, h)$ für ein h: Nat $\to I^*$.
i) Zunächst ist zu zeigen: $f_\sigma(I \copyright t) = O \copyright t$: Da $conc(h_{I \copyright t}) = conc(h)$, gilt nach der Definition einer FIFO-Strategie: $O \copyright t = O \copyright trace(\sigma, h) = O \copyright trace(\sigma, h_{I \copyright t}) = f_\sigma(I \copyright t)$
ii) Nun ist zu zeigen: $\forall s: s \sqsubseteq t \Rightarrow O \copyright s \sqsubseteq f_\sigma(I \copyright s)$ (*).
Sei $s \sqsubseteq t$ und s endlich (sonst gilt $s=t$ und mit i) gilt auch (*)). Dann gibt es ein kleinstes k mit $s \sqsubseteq ptrace(\sigma, h, k)$. $ptrace(\sigma, h, k)$ ist von der Gestalt $r \circ h(k)$ mit $r \in (I \cup O)^*$ (sonst wäre k nicht kleinstmöglich gewählt) und $r \sqsubseteq s$. Man definiere h': Nat $\to I^*$ folgendermaßen: $h'(p) = h(p)$ für $0 \leq p < k$, $h'(k)$ wird so definiert, daß $ptrace(\sigma, h', k) = s$ und $h'(p) = \varepsilon$ für $p > k$. Es gilt $I \copyright s = conc(h')$ und nach der Definition einer FIFO-Strategie gilt daher: $O \copyright s \sqsubseteq O \copyright trace(\sigma, h') = O \copyright trace(\sigma, h_{I \copyright s}) = f_\sigma(I \copyright s)$.
b) $Traces(f_\sigma) \subseteq Traces(\sigma)$
Sei $t \in Traces(f_\sigma)$. Dann hat t die Form $t = w_0 \circ o_0 \circ w_1 \circ o_1 \ldots$ mit $w_k \in I^*$, $o_k \in O \cup \{\varepsilon\}$, wobei gelte: Falls $o_k = \varepsilon$, dann $o_{k+1} = \varepsilon$ für alle $k \in$ Nat.
Man definiere $h(k) =_{def} w_k$ für alle k. Durch Induktion über n läßt sich zeigen: $ptrace(\sigma, h, n) = w_0 \circ o_0 \circ \ldots \circ w_n$ für alle n. Damit gilt: $t = \bigsqcup_n w_0 \circ o_0 \circ \ldots \circ w_n = \bigsqcup_n ptrace(\sigma, h, n)$. □

Satz (Stromverarbeitende Funktionen bestimmen FIFO-Strategien): Sei $f: I^\omega \to O^\omega$ eine stromverarbeitende Funktion. Die Strategie $\sigma_f: (I \cup O)^* \to O \cup \{\varepsilon\}$ ist folgendermaßen definiert:

$$\sigma_f(w) =_{def} \begin{cases} ft(\, rt^{\#O \copyright w}(\, f(I \copyright w)\,)\,) \\ \qquad \text{falls } \#\, f(I \copyright w) > \#\, O \copyright w; \varepsilon \\ \qquad\qquad\qquad \text{sonst} \end{cases}$$

Dann gilt: 1. σ_f ist eine FIFO-Strategie.
2. Traces(σ_f) = Traces(f).

Beweis: Im Beweis verwende ich die folgende Aussage, die sich durch Induktion zeigen läßt.

$\forall m$: O © ptrace(σ_f,h,m) $\sqsubseteq$ f(conc(h)) $\wedge$
$\forall k$: # f(conc(h)) $\geq$ k $\Rightarrow$ $\exists n$: # O©ptrace(σ_f,h,n) $\geq$ k. (*)

1. σ_f ist eine FIFO-Strategie: Seien h und h' Eingabemodellierungen und gelte: conc(h) = conc(h'). Dann gilt mit (*): O©trace(σ_f,h) = f(conc(h)) = f(conc(h')) = O©trace(σ_f,h').

2. a) Traces(σ_f) $\subseteq$ Traces(f):

Sei t$\in$ Traces(σ_f), d.h. t = trace(σ_f,h) für eine gewisse Eingabemodellierung h.

a1) Zunächst wird gezeigt: f(I©t) = O©t: Da I©t = I © trace(σ_f,h) = conc(h) ergibt sich mit (*) f(I©t) = f(conc(h)) = O © trace(σ_f,h) = O©t.

a2) Sei s$\sqsubseteq$ t. Es wird gezeigt: O©s $\sqsubseteq$ f(I©s):

Es ist ausreichend, s der Form s = ptrace(σ_f,h,m) $\circ$ σ_f(ptrace(σ_f,h,m)) für ein m zu betrachten. Für s$\circ$w mit w$\in I^\omega$ gilt nämlich: O©(t'$\circ$w) = O©s $\sqsubseteq$ f(I©s) $\sqsubseteq$ f(I©(s$\circ$w)).

Durch Induktion über m läßt sich zeigen, daß für alle m gilt:

O©ptrace(σ_f,h,m) $\circ$ σ_f(ptrace(σ_f,h,m)) $\sqsubseteq$ f(I© (ptrace(σ_f,h,m) $\circ$ σ_f(ptrace(σ_f,h,m))))

b) Traces(σ_f) $\supseteq$ Traces(f):

Sei t$\in$ Traces(f), d.h. f(I©t) = O©t $\wedge$ $\forall$s: s $\sqsubseteq$ t $\Rightarrow$ O©s $\sqsubseteq$ f(I©s).

Dann hat t die Form t = $w_0 \circ o_0 \circ w_1$... mit $w_k \in I^*$, $o_k \in O$ für alle k, wobei gelte: Falls o_k = ε, dann $o_{k+1} = \varepsilon$ für alle k$\in$ Nat. Man definiere h(k) =$_{def}$ w_k. Durch Induktion über m zeigt man, daß für alle m gilt: ptrace(σ_f,h,m) = $w_0 o_0 ... w_m$. Damit gilt aber auch trace(σ_f,h) = t. □

Es besteht jedoch ein Unterschied zwischen stromverarbeitenden Agenten mit mehr als einem Eingabekanal und Strategien. Mehrere Eingabekanäle werden bei Strategien dadurch modelliert, daß die Vereinigung der (disjunkten) "Kanalalphabete" als Eingabemenge der Strategie betrachtet wird. Diese Modellierungstechnik entspricht der von I/O-Automaten. Die Eingaben von verschiedenen Kanälen erscheinen also sequentiell. Daher lassen sich mit FIFO-Strategien Agenten modellieren, für die es keine Beschreibung durch eine Menge stromverarbeitender Funktionen gibt (etwa das faire nichtstrikte Mischen, vgl. [Broy 89a]). Man sieht jedoch leicht, daß sich umgekehrt jede stromverarbeitende Funktion mit mehr als einem Eingabekanal durch eine FIFO-Strategie modellieren läßt.

Betrachtet man statt FIFO-Strategien allgemeine Strategien, so findet man auch im Fall von Agenten mit nur einem Eingabekanal schnell eine Spurmenge, die sich durch eine Strategie, nicht jedoch durch einen stromverarbeitenden Agenten realisieren läßt:

Beispiel: Die Spurmenge $\{o\} \circ I^\omega \cup I^\omega$ (mit Eingabemenge I und dazu disjunkter Ausgabemenge {o}) läßt sich durch keinen stromverarbeitenden Agenten der Funktionalität $I^\omega \rightarrow \{o\}^\omega$ realisieren, aber durch die Strategie σ, die definiert ist durch:

$$\sigma(x) = \begin{cases} o & \\ & \text{falls } x = \varepsilon; \varepsilon \\ & \text{sonst} \end{cases}$$

□

Nun zum Zusammenhang zwischen I/O-Automaten und Strategie; dieser wird in [Broy et al. 91a] untersucht. Es wird gezeigt, daß sich jede Strategie durch einen I/O-Automaten modellieren läßt, nicht jedoch umgekehrt.[1] Diese Aussage ist unabhängig davon, ob der Begriff der starken oder schwachen Fairneß bei I/O-Automaten verwendet wird. Der Nichtdeterminismus bei I/O-Automaten wird dabei durch *Mengen* von Strategien modelliert. Es läßt sich ein Beispiel konstruieren, das in bestimmter Weise die Eingabe eines I/O-Automaten einschränkt, was gemäß den Anforderungen (1) und (2) an I/O-Automaten in Abschnitt 5.2 gerade ausgeschlossen sein soll. Dieser Fall einer Anomalie ist bei Strategien nicht möglich.

5.5 Echtzeitspezifikation für Agenten

Im folgenden skizziere ich, wie sich zeitbehaftete Spuren (vgl. Abschnitt 3.6.1) durch I/O-Automaten realisieren lassen. Hierzu stelle ich eine Erweiterung des Konzepts der I/O-Automaten vor.

Ein *zeitbehafteter I/O-Automat* ist wiederum ein Tupel $(I,O,State,s_0,—;>,F)$, nur ist nun die Übergangsrelation $—;>$ eine Teilmenge von $State \times (I \cup O \cup \{\tau\} \cup \{\surd\}) \times State$. $\surd$ beschreibt das Ticken einer Uhr. Von einer Berechnung wird im Unterschied zu Abschnitt 5.2 zusätzlich gefordert, daß die Anzahl der $\surd$-Übergänge unendlich ist. Insbesondere sind dann alle Berechnungen unendlich. Die $\surd$-Übergänge geben wie in Abschnitt 4.4 ein Zeitraster vor, es wird wiederum eine *globale* Zeit erfaßt.

Echtzeitanforderungen, die besagen, daß ein bestimmtes Ereignis spätestens nach einer gewissen vorgegebenen Anzahl von Zeitschritten stattgefunden haben muß, werden im I/O-Automaten dadurch repräsentiert, daß bei Überschreiten der Zeitschranke keine $\surd$-Übergänge mehr schaltbereit sind und somit die Forderung nach unendlich vielen $\surd$-Übergängen nicht mehr erfüllt werden kann.

Beispiel (zeitbehafteter I/O-Automat): Beschrieben wird ein I/O-Automat, der die Anforderung "nach spätestens N Zeiteinheiten folgt auf die Eingabe einer Aktion i die Ausgabe einer dazu korrespondierenden Aktion o" *(bounded response)* erfüllt:

Der Automat ist durch das Tupel $(I,O,State,s_0,—;>,F)$ gegeben, wobei gilt:

$I=\{i\}$, $O=\{o\}$, $State = Nat^{\omega}$, $s_0 = \varepsilon$, $—;>$ ist die Menge aller Übergänge (sei $k \geq 0$):

$$\begin{aligned}
&\langle n_1,\ldots,n_k\rangle \; —;^{i}> \; \langle n_1,\ldots,n_k,N+1\rangle, \\
&\langle n_1,\ldots,n_k\rangle \; —;^{\surd}> \; \langle n_1-1,\ldots,n_k-1\rangle \quad \text{falls } n_j \geq 0 \text{ für } j = 1,2,\ldots,k, \\
&\langle n_1,n_2,\ldots,n_k\rangle \; —;^{o}> \; \langle n_2,\ldots,n_k\rangle \quad \text{falls } n_j \geq 0 \text{ für } j = 1,2,\ldots,k,
\end{aligned}$$

---

[1] Die Modellierung des Zusammenspiels mehrerer Komponenten durch Strategien in Abschnitt 4.3.3 läßt sich als Komposition der den einzelnen Komponenten entsprechenden I/O-Automaten deuten.

mit natürlichen Zahlen $n_1,...,n_k$. Man beachte, daß der Übergang $\varepsilon \xrightarrow{\surd} \varepsilon$ ebenfalls enthalten ist. Die Fairneßmenge F ist gegeben als $\{\{s \xrightarrow{a} s' \mid a \in \{o,\surd\}\}\}$.

Die Zustände bestehen aus Tupeln von natürlichen Zahlen, die Zeitzähler modellieren. Jedes Tupel enthält die Zeitzähler derjenigen Eingabeaktionen, für die noch keine Ausgabeaktion stattfand. Ein Zeitzähler, der gleich 0 ist, verkörpert einen Zeitfehler. Erfolgt eine Eingabeaktion, so wird ein neuer Zeitzähler erzeugt und mit $N+1$ initialisiert. Eine $\surd$-Aktion erniedrigt alle vorhandenen Zeitzähler um 1; dies modelliert das Warten um eine Zeiteinheit. Eine Ausgabeaktion löscht den ersten Zeitzähler; dieser gehört zu jener Eingabe, für die noch keine Ausgabe stattfand und die darauf am längsten wartete. Allerdings geschieht dies nur, wenn vorher kein Zeitfehler stattfand, d.h. keine Eingabe länger als N Zeiteinheiten warten mußte. Man sieht: Diejenigen Spuren des Automaten, in denen unendlich viele $\surd$-Aktionen vorkommen, erfüllen die obige Anforderung. □

In [Broy 90] werden stromverarbeitende Agenten ebenfalls mit einer Aktionenmenge beschrieben, die zusätzlich die Aktion $\surd$ enthält. Die Modellierungssicht ist dort jedoch anders: Die $\surd$-Aktion modelliert eine "leere" Eingabe oder eine "leere" Ausgabe, je nachdem, ob sie auf einem Ein- oder einem Ausgabekanal erscheint. Die in diesem Abschnitt vorgestellte Modellierung mit I/O-Automaten lehnt sich dagegen an die Sichtweise der Abschnitte 3.6.1 bzw. 4.4 an.

5. Übergang zur Entwurfsspezifikation

In diesem Kapitel erläutere ich den Übergang von einer komponentenorientierten, spurbasierten Anforderungsspezifikation zu einer funktionalen Entwurfsspezifikation. Für die Durchgängigkeit von FOCUS ist es wichtig, daß die Beziehung zwischen diesen beiden Beschreibungsformen eines verteilten Systems geklärt ist und methodische Leitlinien für den Übergang bereitstehen. Auf die Technik der Entwurfsspezifikation gehe ich nur soweit ein, wie es für die Anbindung an die Anforderungsspezifikation nötig ist.

Zur Entwurfsspezifikation werden *stromverarbeitende Agenten* eingesetzt; dieses Konzept erläutere ich in Abschnitt 5.1, insbesondere zeige ich die Unterschiede zur komponentenorientierten Spezifikation auf. Die *Spursemantik* für stromverarbeitende Agenten ist der Anknüpfungspunkt zur Anforderungsebene; sie ist Gegenstand von Abschnitt 5.2. Ich stütze mich dabei auf bekannte Ergebnisse über I/O-Automaten. In Abschnitt 5.3 skizziere ich die methodische Vorgehensweise beim Übergang von der Anforderungs- zur Entwurfsspezifikation. In Abschnitt 5.4 setze ich das Konzept der Realisierbarkeit durch Strategien, das in Abschnitt 4.3 zentral war, zu der Realisierbarkeit durch stromverarbeitende Agenten und I/O-Automaten in Beziehung. Schließlich erfolgt in Abschnitt 5.5 ein Ausblick auf die Agentenbeschreibung von Echtzeitsystemen. Hierzu werden zeitbehaftete I/O-Automaten eingeführt, die zeitbehaftete Spuren (vgl. Abschnitte 3.6.1 und 4.4) realisieren.

5.1 Stromverarbeitende Agenten

Das Konzept der stromverarbeitenden Agenten geht auf die Arbeit [Kahn 74] zurück; verwandt damit ist die Idee der Datenflußberechnung ([Dennis 74]). Stromverarbeitende Agenten bekommen auf ihren Eingabekanälen Nachrichten zugesandt und verarbeiten diese in Nachrichten, die auf den Ausgabekanälen befördert werden. Die Nachrichten auf den Eingabekanälen lassen sich als Eingabeströme auffassen, die Nachrichten auf den Ausgabekanälen als Ausgabeströme, daher der Name "stromverarbeitender Agent". Agenten sind miteinander über gerichtete Kanäle verbunden und können auch nichtdeterministisches Verhalten zeigen.

Ein *deterministischer* stromverarbeitender Agent wird durch eine *stromverarbeitende Funktion* beschrieben. Dies ist eine Funktion, die ein Tupel von Eingabeströmen auf ein Tupel von Ausgabeströmen abbildet und bezüglich der vollständigen Halbordnung der Ströme monoton und stetig ist. Die Monotonie bedeutet, daß der Empfang von weiteren Eingaben höchstens weitere Ausgaben hervorruft. Ein Agent kann somit seine Berechnung und seine Ausgabe beginnen, wenngleich er noch nicht die gesamten Eingaben auf seinen Kanälen empfangen hat; dies ermöglicht Parallelarbeit der Agenten. Die Stetigkeit drückt aus, daß die Ausgabe auf einen unendlichen Eingabestrom durch die Ausgaben auf die endlichen Präfixe des Eingabestroms approximiert wird.

Für die Erweiterung des Ansatzes aus [Kahn 74] auf nichtdeterministische Agenten gibt es verschiedene Möglichkeiten (z.B. [Brock, Ackerman 81], [Staples, Nguyen 85], [Kok 86]). In unserer Entwurfsmethodik wird ein *nichtdeterministischer* Agent durch eine *Menge*

stromverarbeitender Funktionen beschrieben (vgl. [Keller 78], [Broy 88b]); diese Modellierung von Nichtdeterminismus entspricht der des Abschnitts 4.3.

Die Spezifikation eines stromverarbeitenden Agenten erfolgt durch ein Prädikat über stromverarbeitende Funktionen.[1] Ähnlich wie bei einer Spurspezifikation läßt sich auch hier sowohl die ablauforientierte als auch die transitionsorientierte Spezifikation einsetzen. Bei einer ablauforientierten Spezifikation von Funktionen wird über vollständige Ein- und Ausgabeströme einer Funktion gesprochen, z.B. beschreibt $Q(f) \equiv \forall x: \#x = \#f(x)$ die Menge aller Funktionen, die genausoviel ausgeben wie sie einlesen; für komplexere Beispiele vgl. [Broy 87b]. Eine transitionsorientierte Spezifikation zeigt das folgende Beispiel:

Beispiel (Postfachsystem: stromverarbeitender Agent): Die Komponente SERVER (vgl. Abschnitt 4.1) läßt sich durch die Funktion

$$\text{server}: (\text{Snd} \cup \text{Rec})^{\omega} \rightarrow (\text{S-ack} \cup \text{R-ack})^{\omega}$$

realisieren[2]. (Will man Eingaben aus Snd und Rec über getrennte Eingabekanäle empfangen, so muß vor die Funktion server noch ein Multiplexer (Mischagent) geschaltet werden. Entsprechend ließe sich die Funktion server auch derart abwandeln, daß Ausgaben aus S-ack und R-ack über getrennte Kanäle geschehen.)

Die Funktion server wird mittels einer Hilfsfunktion hs transitionssorientiert spezifiziert:

$$\text{server}(t) = \text{hs}(t,\text{init})$$

Die Funktion hs hat zusätzlich einen Zustandsparameter. Ein Zustand bestehe aus einer Warteschlange von Nachrichten für jedes Postfach, d.h. aus einer Abbildung der Funktionalität $\text{Mbx} \rightarrow \text{Msg}^{\omega}$. Der Anfangszustand init sei die Abbildung, die jedes Postfach auf ε abbildet, d.h. alle Warteschlangen sind anfangs leer. hs ist definiert durch (t sei ein Strom, s ein Zustand):

$$\neg\, \text{full}(s(mb)) \Rightarrow \text{hs}(\text{snd}(ts,mb,ms) \circ t, s) = \text{s-ack}(ts,mb,ok) \circ \text{hs}(t, s[s(mb) \circ ms \,/\, mb]),$$

$$\text{full}(s(mb)) \Rightarrow \text{hs}(\text{snd}(ts,mb,ms) \circ t, s) = \text{s-ack}(ts,mb,error) \circ \text{hs}(t,s),$$

$$\neg\, \text{empty}(s(mb)) \Rightarrow \text{hs}(\text{rec}(ts,mb) \circ t, s) = \text{r-ack}(ts,mb,\text{ft}(s(mb)),ok) \circ \text{hs}(t, s[\text{rt}(s(mb)) \,/\, mb]),$$

$$\text{empty}(s(mb)) \Rightarrow \text{hs}(\text{rec}(ts,mb) \circ t, s) = \text{r-ack}(ts,mb,nil,error) \circ \text{hs}(t,s).$$

---

[1] Ein deterministischer stromverarbeitender Agent entspricht formal einer einelementigen Menge. In der Regel wird in diesem Fall nicht diese Menge, sondern die stromverarbeitende Funktion selbst spezifiziert, vgl. das folgende Beispiel.

[2] Eine ähnliche Realisierung findet sich in [Dendorfer 91], worin eine funktionale Modellierung des (umfangreicheren) MMK-Postfachsystems ([Bemmerl et al. 90]) beschrieben ist.

s[x / mb] steht für die punktweise Änderung der Funktion s (vgl. Abschnitt 3.4.2). Man erkennt die Ähnlichkeit zu dem Transitionssystem, das für die Komponente SERVER angegeben wurde. Eine genauere Begründung, warum die Funktion server die Komponente SERVER realisiert, findet sich in Abschnitt 5.2, wo Agenten Spuren zugeordnet werden. □

Für stromverarbeitende Agenten ist eine *denotationelle Semantik (Fixpunktsemantik, mathematische Semantik)* üblich. (Es gibt jedoch auch operationelle Semantiken, vgl. [Faustini 82], [Lynch, Stark 89], [Jonsson 87].) Diese denotationelle Semantik basiert auf den Kompositionsformen für stromverarbeitende Funktionen, nämlich der sequentiellen, parallelen und Rückkopplungskomposition; die Rückkopplung wird über Fixpunktbildung behandelt (vgl. [Broy 90]). Ein wesentliches Ergebnis hierbei ist, daß ein Agentennetz wiederum als ein stromverarbeitender Agent gesehen werden kann. Dessen Semantik, eine Menge stromverarbeitender Funktionen, ergibt sich aus der Komposition der Funktionsmengen, die als Semantik den einzelnen Agenten des Netzes zugeordnet sind.

Wie bei der komponentenorientierten Spurspezifikation spielen bei der funktionalen Spezifikation Komponenten eine Rolle, es bestehen jedoch wichtige Unterschiede:

1. Die Festlegung der Produkt- und Umgebungskomponenten bei der Anforderungsspezifikation ist dem Kunden, der hierarchische Entwurf der Produktkomponenten, d.h. die Strukturierung der Produktkomponenten in weitere Teilkomponenten, dagegen dem Systementwickler überlassen; dieser muß dabei nur gewährleisten, daß die Produktkomponenten die in der Anforderungsspezifikation festgelegten Eigenschaften besitzen.

2. Während bei einer komponentenorientierten Spezifikation zumindest manche Anforderungen *global* an das zu erstellende Produkt gerichtet sein können, sind die Beschreibungen der Produktkomponenten auf der Entwurfsebene *lokal*; dies gestattet eine modulare Weiterentwicklung des zu erstellenden Produkts.

3. Ein Entwurf der Produktkomponenten wird in der Regel eine hierarchische Struktur aufweisen; diese ist ein Ergebnis von Entwurfsentscheidungen. Auf der Anforderungsebene interessieren wir uns dagegen nicht für die interne Struktur einer Komponente. Prinzipiell läßt sich jedoch ähnlich wie durch eine Spurspezifikation auch durch einen stromverarbeitenden Agenten allein die *Schnittstelle* einer Komponente spezifizieren.

4. Auf der Entwurfsebene werden stromverarbeitende Agenten zur Beschreibung von Komponenten verwendet, nicht Spuren. Die Entscheidung für stromverarbeitende Agenten ist dadurch motiviert, daß sich eine lokale und hierarchische Komponentenbeschreibung (Punkte 1 und 2) damit gut erreichen läßt und zudem der Formalismus technisch gut handhabbar ist. Zudem fällt die Anknüpfung zu Programmbeschreibungen von stromverarbeitenden Agenten aus leichter.

5. Der Freiheitsraum, den eine Anforderungsspezifikation für eine Realisierung läßt, kann durch eine Entwurfsspezifikation bereits eingeschränkt sein.

5.2 Spursemantik für Agenten

Die Verbindung zwischen einer funktionalen Beschreibung und einer Spurbeschreibung wird durch die *Spursemantik* für stromverarbeitende Agenten hergestellt, die mit der denotationellen Semantik verträglich ist. Die Spursemantik eines einzelnen Agenten ist folgendermaßen definiert:

Definition (Spursemantik eines stromverarbeitenden Agenten): Sei ein stromverarbeitender Agent durch eine Menge F von stromverarbeitenden Funktionen der Funktionalität $I_1^\omega \times ... \times I_m^\omega \rightarrow O_1^\omega \times ... \times O_n^\omega$ gegeben. Dabei seien die Mengen I_1, ..., I_m, O_1, ..., O_n paarweise disjunkt. Die *Spursemantik* dieses Agenten ist das Tripel (I,O,T) mit

$$I =_{def} I_1 \cup ... \cup I_m,$$
$$O =_{def} O_1 \cup ... \cup O_n,$$
$$T =_{def} \bigcup_{f \in F} \text{Traces}(f)$$

wobei gilt:

$$\text{Traces}(f) =_{def} \{t \in (I \cup O)^\omega \mid \langle O_1 \copyright t,...,O_n \copyright t\rangle = f(I_1 \copyright t,...,I_m \copyright t) \wedge \forall s: s \sqsubseteq t \Rightarrow \langle O_1 \copyright s,...,O_n \copyright s\rangle \sqsubseteq f(I_1 \copyright s,...,I_m \copyright s) \}. \square$$

Gemäß dieser Definition müssen die Eingaben, die für eine Ausgabe kausal sind, vor dieser Ausgabe in jeder Spur erscheinen. Die Disjunktheit der Ein- und Ausgabemengen fordere ich, um auch auf der Spurebene zwischen Ein- und Ausgaben unterscheiden zu können, genauer gesagt, um feststellen zu können, auf welchem Kanal eine Nachricht gesendet wird. Diese Anforderung stellt keine große Einschränkung dar: gleiche Nachrichten auf verschiedenen Kanälen lassen sich leicht mit einem Unterscheidungskennzeichen, z.B. dem Kanalnamen, versehen.

Bei der Spursemantik von *Agentennetzen* werden die in Abschnitt 4.1 vorgestellten Kompositionsoperatoren || und *hide* verwendet. Die Verträglichkeit mit der denotationellen Semantik eines Agentennetzes ergibt sich aus theoretischen Ergebnissen aus [Jonsson 87], wo dies für das etwas allgemeinere Konzept der I/O-Automaten gezeigt wird. Ich erläutere im folgenden, wie sich diese Ergebnisse auf stromverarbeitende Agenten übertragen lassen.

Leicht unterschiedliche Definitionen von I/O-Automaten finden sich in [Jonsson 87] und [Lynch, Stark 89]; ich verwende Jonssons Definitionen von *I/O-Automaten*, ihren *Berechnungen*, ihren *akzeptierten Spuren* und deren Komposition.

Informell gesprochen beschreiben I/O-Automaten Agenten, deren Verhalten durch zwei Bedingungen charakterisiert ist:

(1) Ein I/O-Automat akzeptiert immer alle Eingaben.
(2) Das Senden einer Nachricht durch einen I/O-Automaten und ihr Empfang von einem anderen I/O-Automaten ist eine atomare Operation, die sich gleichzeitig im sendenden und empfangenden Automaten vollzieht.

Formal ist ein I/O-Automat wie folgt definiert:

Definition (I/O-Automat): Ein *I/O-Automat* ist ein Tupel $(I,O,State,s_0,—;>,F)$. Dabei ist

I eine Menge von *Eingabeaktionen*, die nicht die stille Aktion τ enthält,

O eine Menge von *Ausgabeaktionen*, die zu I disjunkt ist und τ nicht enthält,

State eine Menge von *Zuständen*,

s_0 der Anfangszustand des Automaten,

$—;>$ eine Teilmenge von $State \times (I \cup O \cup \{\tau\}) \times State$, die Menge der *Transitionen*. Für $(s,a,s') \in —;>$ wird auch $s —;^{a}> s'$ geschrieben.

F eine endliche Menge von *Fairneßmengen*. Jede Fairneßmenge ist eine Teilmenge der Übergangsrelation $—;>$.

Zudem müssen die folgenden Bedingungen erfüllt sein:

1) Für jeden Zustand s und jede Eingabeaktion i gibt es einen Zustand s', so daß $s —;^{i}> s'$ gilt.

2) Keine Fairneßmenge $F \in F$ enthält eine Transition $s —;^{i}> s'$ mit $i \in I$.

3) Jede Transition $s —;^{a}> s'$ mit $a \in O \cup \{\tau\}$ ist Element einer Fairneßmenge in *F*.

□

1) besagt, daß der Automat immer alle Eingaben akzeptiert. Wie aus den Berechnungen des Automaten ersichtlich ist (s.u.), wird dadurch ausgeschlossen, daß Eingaben die durch die Transitionsrelation und den Anfangszustand festgelegte Sicherheitsanforderung (vgl. Abschnitt 3.4) verletzen. 2) bedeutet, daß an Eingaben keine Lebendigkeitsanforderung (Fairneß) gestellt wird. 3) drückt aus, daß der Automat alle in seinen Verantwortungsbereich fallenden Transitionen, das sind stille Transitionen und Ausgabetransitionen, fair behandelt. Stille Aktionen entstehen bei Anwendung des Versteckoperators für I/O-Automaten.

Der Unterschied zu dem in Abschnitt 3.4 vorgestellten Begriff des Transitionssystems liegt darin, daß zwischen Ein- und Ausgaben unterschieden wird und Lebendigkeitsanforderungen in I/O-Automaten in Form von Fairneßmengen verankert sind.

Das Verhalten von I/O-Automaten wird operationell beschrieben. Berechenbarkeitsannahmen werden nicht getroffen, denn die Zustände und Zustandsübergänge können beliebig gewählt werden.

Sei für die folgenden Definitionen A ein I/O-Automat $(I,O,State,s_0,—;>,F)$.

Definition (schaltbereite Transition): Eine Transition $s —;^{a}> s'$ heißt *schaltbereit* in s. Eine Fairneßmenge F ist in s schaltbereit, wenn eine Transition aus F in s schaltbereit ist.

□

Definition (Berechnung): Eine *Berechnung* von A ist eine endliche oder unendliche Sequenz der Art

$$s_0 ——;^{a1}> s_1 ——;^{a2}> \ldots ——;^{an}> s_n ———;^{an+1}> \ldots$$

(eine endliche Sequenz ende mit einem Zustand), wobei gilt:

(A) *Initialisierung:* s_0 ist der Anfangszustand von A.

(B) *Schrittfolge:* Jede Transition $s_n ——;^{an+1}> s_{n+1}$ liegt in $—;>$.

(C) *(starke) Fairneß:* Ist die Sequenz unendlich, so muß sie unendlich viele Vorkommen von Transitionen jeder Fairneßmenge $F \in F$ enthalten, die unendlich oft schaltbereit ist.

(D) *Stabilität:* Ist die Sequenz endlich, so darf weder eine Ausgabetransition noch eine stille Transition in ihrem letzten Zustand schaltbereit sein. □

Die Sicherheitseigenschaften des Automaten werden durch (A) und (B) ausgedrückt. Die Bedingungen (C) und (D) legen Lebendigkeitseigenschaften fest; zusammen bilden sie das Akzeptierkriterium für endliche und unendliche Worte (akzeptierte Spuren). (C) drückt starke Fairneß aus. Im Gegensatz dazu würde schwache Fairneß fordern, daß nur die Transitionen von jenen Fairneßmengen nicht ignoriert werden dürfen, die fortwährend schaltbereit sind. Die wesentlichen Ergebnisse über I/O-Automaten ändern sich nicht, wenn die schwache statt der starken Fairneß gewählt wird ([Jonsson 90]), insbesondere bleiben meine Ergebnisse davon unberührt.

Definition (Spursemantik eines I/O-Automaten): Eine *akzeptierte Spur* von A ist die Sequenz der Eingabe- und Ausgabeaktionen (d.h. Aktionen ungleich τ) in einer Berechnung von A. Die Menge der von A akzeptierten Spuren wird mit Traces(A) bezeichnet. Die *Spursemantik* eines I/O-Automaten A ist das Tripel (I,O,Traces(A)). □

Jonsson gibt Kompositionsoperatoren (|| und *hide*) für Automaten und Spurmengen an und zeigt, daß die Komposition von I/O-Automaten wieder einen I/O-Automaten ergibt.

Definition (Kompositionsoperator ||): Die Komposition zweier I/O-Automaten $A_1 = (I_1,O_1,State_1,s_{0_1},\text{---};_{>1},F_1)$ und $A_2 = (I_2,O_2,State_2,s_{0_2},\text{---};_{>2},F_2)$ wird mit $A_1 \parallel A_2$ bezeichnet. Sie ist definiert, wenn $O_1 \cap O_2 = \emptyset$. $A_1 \parallel A_2 =_{def} (I,O,State,s_0,—;>,F)$ mit

$O = O_1 \cup O_2$, $I = (I_1 \cup I_2) - O$, $State = State_1 \times State_2$, $s_0 = \langle s_{0_1},s_{0_2}\rangle$,
—;> ist die kleinste Relation, für die gilt:

s_1 —;^{a}>$_1$ s_1', a in s_2 nicht schaltbereit $\Rightarrow$ $\langle s_1,s_2\rangle$ —;^{a}> $\langle s_1',s_2\rangle$,
s_2 —;^{a}>$_2$ s_2', a in s_1 nicht schaltbereit $\Rightarrow$ $\langle s_1,s_2\rangle$ —;^{a}> $\langle s_1,s_2'\rangle$,
s_1 ---;^{a}>$_1$ s_1', s_2 ---;^{a}>$_2$ s_2' $\Rightarrow$ $\langle s_1,s_2\rangle$ ---;^{a}> $\langle s_1',s_2'\rangle$,
s_1 ---;$^{\tau}$>$_1$ s_1' $\Rightarrow$ $\langle s_1,s_2\rangle$ ---;$^{\tau}$> $\langle s_1',s_2\rangle$,
s_2 ---;$^{\tau}$>$_2$ s_2' $\Rightarrow$ $\langle s_1,s_2\rangle$ ---;$^{\tau}$> $\langle s_1,s_2'\rangle$,

F ist so definiert, daß es für jedes $F_i \in F_i$ es eine Fairneßmenge $F \in F$ gibt, die aus allen Übergängen besteht, die sich gemäß den obigen Regeln mit Übergängen aus F_i ergeben. □

Analog ist die n-fache Komposition von I/O-Automaten $A_1 \parallel ... \parallel A_n$ definiert (für Details siehe [Jonsson 87]). || bildet aus n Automaten ein Netz, bei dem zwischen zwei Automaten eine Verbindung besteht, wenn die Eingabemenge des einen Automaten einen nichtleeren Schnitt mit der Ausgabemenge des anderen Automaten hat.

Definition (Versteckoperator hide): Für einen I/O-Automaten $A = (I,O,State,s_0,—;>,F)$ und eine Aktionenmenge $B \subseteq O$ ist *hide* B *in* A als der Automat $(I,O - B,State,s_0,—;>',F')$ definiert. Hierbei ist —;>' als die kleinste Relation definiert, für die gilt:

$a \notin B \wedge s$ —;^{a}> $s' \Rightarrow s$ —;^{a}>' s', $\qquad a \in B \wedge s$ —;^{a}> $s' \Rightarrow s$ —;$^{\tau}$>' s'

F' ist so definiert, daß es für jede Fairneßmenge $F \in F$ es eine Fairneßmenge $F' \in F'$ gibt, die alle Transitionen enthält, die aus einer Transition aus F gemäß der Definition von $—;>'$ entstehen. □

Jonsson zeigt, daß die Spurmenge des komponierten Automaten aus der Komposition der Spurmengen der zu komponierenden Automaten resultiert, wobei die entsprechenden Kompositionsoperatoren auf der Spurebene verwendet werden (seien $A_1, ..., A_n$ und A I/O-Automaten und B eine Teilmenge der Ausgaben von A):

$$\text{Traces}(A_1 \|_a ... \|_a A_n) = \text{Traces}(A_1) \|_t ... \|_t \text{Traces}(A_n),$$
$$\text{Traces}(\mathit{hide}_a\ B\ \mathit{in}\ A) = \mathit{hide}_t\ B\ \mathit{in}\ \text{Traces}(A),$$

wobei zur Verdeutlichung Unterscheidungsindizes *a ("automaton")* und *t ("trace")* an die Kompositionsoperatoren angebracht sind.

Die Spursemantik ist also verträglich mit der operationellen Semantik.

Beispiel (Postfachsystem: I/O-Automat): Die Komponente SERVER (vgl. Abschnitt 4.1 und das entsprechende Transitionssystem in Abschnitt 3.4) läßt sich leicht durch einen I/O-Automaten realisieren: Der I/O-Automat habe die Eingabemenge $Snd \cup Rec$, die Ausgabemenge $S\text{-}ack \cup R\text{-}ack$ sowie die Zustandsmenge, den Anfangszustand und die Zustandsübergänge wie in Abschnitt 3.4 beschrieben; die Lebendigkeitsanforderungen $S\text{-}ack\_live$ und $R\text{-}ack\_live$ lassen sich durch diejenige Fairneßmenge realisieren, die alle Übergänge enthält, die mit einer Ausgabeaktion markiert sind. Wird irgendwann eine Nachricht von einem Prozeß zu einem Postfach geschickt, so wird eine entsprechende Rückmeldeaktion in der Zustandskomponente $acks$ gespeichert. Da Zustandsübergänge, die mit Rückmeldeaktionen markiert sind, immer schaltbereit sind, wird die entsprechende Rückmeldeaktion gemäß der Fairneßannahme irgendwann später auch tatsächlich ausgeführt.

Beim Postfachbeispiel sieht man relativ leicht, wie sich die globale Anforderungsspezifikation in Komponentenspezifikationen umsetzen läßt und wie die SERVER-Spezifikation durch einen Agenten realisiert werden kann. Der Grund liegt (abgesehen von der Einfachheit des Beispiels) darin, daß alle Beschreibungen transitionsorientiert sind und zudem die Zustandsräume und -übergänge gleich sind. Lediglich die Lebendigkeitsanforderungen werden unterschiedlich realisiert. Schwieriger wird der Nachweis der Implementierungsrelation, wenn die Beschreibungen stark voneinander abweichen. □

I/O-Automaten sind etwas allgemeiner als stromverarbeitende Agenten, da eine neue Eingabe eine Ausgabe, die vorher möglich war, revidieren kann:

Beispiel: Der folgende I/O-Automat

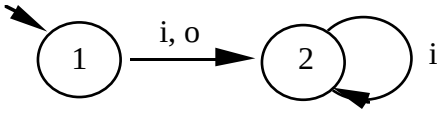


Fig. 5.1

(mit Eingabe i, Ausgabe o, Anfangszustand 1 und der Fairneßmenge, die allein den o-Übergang enthält) erzeugt die Spuren $o \circ i^\omega \cup i^\omega$ (angelehnt an die Notation regulärer

Ausdrücke), nicht jedoch eine Spur mit Präfix $i \circ o$, was bei einem stromverarbeitenden Agenten der Fall sein müßte. □

In [Jonsson 87] wird auch eine spezielle Klasse von I/O-Automaten angegeben, nämlich solche I/O-Automaten, die einen FIFO-Eingabepuffer für jeden Eingabekanal der Agenten im Zustand enthalten. Intuitiv gesehen sind dies gerade stromverarbeitende Agenten. In [Lynch, Stark 89] wird gezeigt, daß die Teilklasse der sog. "determinierten" I/O-Automaten (stetige) stromverarbeitende Funktionen beschreibt. Ich führe nun vor, wie sich eine funktionale Beschreibung eines stromverarbeitenden Agenten schematisch in eine Beschreibung eines I/O-Automaten umwandeln läßt, was weder in [Jonsson 87] noch in [Lynch, Stark 89] gezeigt wird. Mit diesen Ergebnissen läßt sich die Spursemantik von I/O-Automaten auf stromverarbeitende Agenten übertragen.

Zunächst sei eine einzelne stromverarbeitende Funktion $f: I^\omega \to O^\omega$ mit $I \cap O = \emptyset$ betrachtet. Es läßt sich ein I/O-Automat $(I, O, State, s_0, —;>, F)$ definieren, der gerade die Spuren von f akzeptiert. Die Komponenten State, s_0, $—;>$ und F sind definiert:

$State = I^* \times O^\omega$

$s_0 = \langle \varepsilon, f(\varepsilon) \rangle$

$—;>$ bestehe aus allen Übergängen der Form (1) und (2). Dabei sei $i \in I$ und $\langle x,y \rangle \in State$:

(1) $\langle x,y \rangle —;^{i}> \langle x \circ i, y \circ rt^{\#f(x)}(f(x \circ i)) \rangle$

(2) $\langle x,y \rangle ———;^{ft(y)}> \langle x, rt(y) \rangle$, falls $y \neq \varepsilon$

$F = \{\{ s —;^{a}> s' \mid s —;^{a}> s' \text{ ist Übergang vom Typ (2)} \}\}$

Die erste Zustandskomponente gibt die bisherige Eingabe an, die zweite beschreibt jene Ausgaben, die bereits festgelegt sind, aber noch nicht ausgegeben wurden. Dabei kann $rt^\infty(t)$ beliebig definiert werden, z.B. als ε.

Die Erweiterung auf Funktionen mit mehr als einen Eingabekanal und mehr als einem Ausgabekanal ist offensichtlich.

Beispiel (Postfachsystem: stromverarbeitender Agent und I/O-Automat): Gemäß der obigen Konstruktion läßt sich aus der stromverarbeitenden Funktion server (vgl. Abschnitt 5.1) ein I/O-Automat erzeugen, der dem oben angegebenen I/O-Automaten für das Postfachsystem entspricht. Dies zeigt, daß die Funktion server ebenfalls die Komponente SERVER realisiert. □

Bei der Konstruktion eines I/O-Automaten für einen nichtdeterministischen Agenten, der durch eine *Menge* stromverarbeitender Funktionen gegeben ist, müssen die "Berechnungspfade" der einzelnen Funktionen auseinandergehalten werden. Dies gelingt z.B. auf die folgende Weise: Die Zustandsmengen der I/O-Automaten für die einzelnen Funktionen der Menge werden disjunkt gemacht, ihre Anfangszustände werden identifiziert und ihre Faireßmengen vereinigt. Es ist leicht zu sehen, daß sich auf diese Weise wieder ein I/O-Automat ergibt, nämlich jener, dessen Spurmenge die Vereinigung der Spurmengen der einzelnen Automaten ist.

Die für stromverarbeitende Agenten üblichen Kompositionsoperatoren (sequentielle, parallele und Rückkopplungskomposition) lassen sich leicht durch den ||-Operator und den Versteckoperator ausdrücken, und zwar in dem Sinne, daß durch die Komposition der den Funktionen entsprechenden I/O-Automaten die gleichen Spuren erzeugt werden wie durch die Komposition der Funktionen. Dies wird im folgenden skizziert.

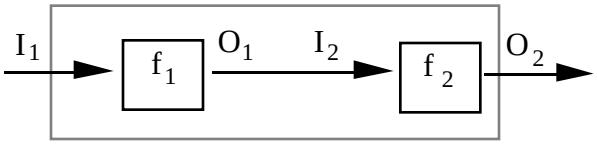
Hierzu betrachten wir Funktionen, deren Ein- und Ausgabemengen von verschiedenen Kanälen paarweise disjunkt sind, d.h. die Nachrichten sind eindeutig den verschiedenen Ein- und Ausgabekanälen zugeordnet. Formal:

Für $f: I_1^{\omega} \times ... \times I_m^{\omega} \rightarrow O_1^{\omega} \times ... \times O_n^{\omega}$ gelte $I_i \cap I_j = \emptyset$, $O_i \cap O_j = \emptyset$, $I_i \cap O_k = \emptyset$ für alle i,j,k mit $i \neq j$. (*)

Diese Annahme ist nötig, um die Beziehung zwischen Funktions- und Automatennetzen herzustellen. Anders als bei stromverarbeitenden Funktionen wird das Ein-/Ausgabeverhalten eines I/O-Automaten sowie die Verbindungsstruktur eines Automatennetzes nämlich nicht über eine Funktionalität, sondern über die Identität der Aktionen bestimmt. Durch Umbenennung von Aktionen, etwa durch die Markierung jeder Aktion mit einem Kanalnamen, läßt sich diese Bedingung immer leicht einhalten.

Im folgenden bezeichne [f] einen der Funktion f entsprechenden I/O-Automaten (s.o.). Bewiesen wird die Korrektheit der Komposition nur für den Fall der Rückkopplung; die anderen beiden Fälle verlaufen analog.

1. Sequentielle Komposition

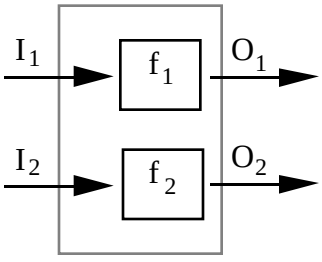


f ; g Fig. 5.2

Hierbei muß $O_1 = I_2$ und $I_1 \cap O_2 = \emptyset$ gelten; ersteres ist für die sequentielle Komposition nötig, letzteres, damit das Ergebnis die Bedingung (*) erfüllt.

Es gilt: $Traces(f_1 ; f_2) = Traces(\textit{hide}\ O_1\ \textit{in}\ ([f_1] \parallel [f_2]))$

2. Parallele Komposition



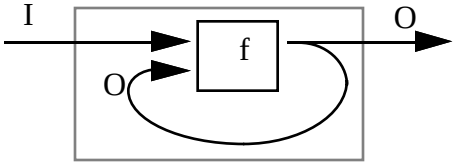
f_1 *par* f_2 Fig. 5.3

Hierbei muß $I_1 \cap I_2 = \emptyset$, $I_1 \cap O_2 = \emptyset$ und $I_2 \cap O_1 = \emptyset$ gelten, um (*) einzuhalten.

Es gilt: $\text{Traces}(f_1 \; \textit{par} \; f_2) = \text{Traces}([f_1] \parallel [f_2])$

3. Rückkopplung

Die direkte Rückkopplung ist bei I/O-Automaten nicht definiert. Im folgenden Bild gälte sonst für f nicht, daß Ein- und Ausgabemengen disjunkt sind, was die Bedingung (*) verletzt. Ein Netz der Art



μf Fig. 5.4

wird daher ersetzt durch das Netz

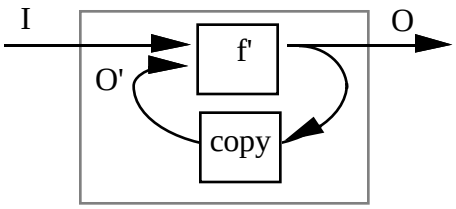


Fig. 5.5

Hierbei ist copy eine bijektive Funktion von O nach einer zu I und O disjunkten Menge O'. copy(o) wird im folgenden mit o' abgekürzt; dies wird auf Spuren erweitert: $(o_1 \circ o_2 \circ \ldots)' = o_1' \circ o_2' \circ \ldots$. Für $f: I^\omega \times O^\omega \to O^\omega$ ist $f': I^\omega \times O'^\omega \to O^\omega$ definiert $f(x,y) = z \Leftrightarrow f'(x,y') = z$.

Es gilt: $\text{Traces}(\mu f) = \text{Traces}(\textit{hide}\ O' \ \textit{in}\ ([f'] \parallel [\text{copy}]))$

Beweis: Der Beweis wird hier nur für den Fall $I = \emptyset$ geführt, d.h. f hat nur einen Eingabekanal, f ist also von der Funktionalität $O \to O$. Analog beweist man den allgemeinen Fall.

$\text{Traces}(\mu f) = \{\text{fix}(f)\}$, wobei fix(f) den kleinsten Fixpunkt der Funktion f bezeichnet. $\text{Traces}((\textit{hide}\ O' \ \textit{in}\ ([f'] \parallel [\text{copy}]))$ ist nicht leer, da jeder I/O-Automat eine nicht-leere Spurmenge hat. Man wähle ein $t \in \text{Traces}((\textit{hide}\ O' \ \textit{in}\ ([f'] \parallel [\text{copy}]))$. Es wird gezeigt, daß $t = \text{fix}(f)$ gilt. Daraus folgt sofort die Behauptung.

(Zur Verdeutlichung sind im folgenden Unterscheidungsindizes *a* für die I/O-Automaten-Operatoren und *t* für die entsprechenden Spuroperatoren angebracht.)

	$t \in \text{Traces}(\textit{hide}_a\ O' \ \textit{in}\ ([f'] \parallel_a [\text{copy}]))$		{Eigenschaft von $\parallel$ und *hide*}
$\Leftrightarrow$	$t \in \textit{hide}_t\ O' \ \textit{in}\ (\text{Traces}(f') \parallel_t \text{Traces}(\text{copy}))$		{Definition von $\textit{hide}_t$}
$\Leftrightarrow$	$\exists r: r \in \text{Traces}(f') \parallel_t \text{Traces}(\text{copy}) \wedge t = O \copyright r$		{Definition von $\parallel_t$}
$\Leftrightarrow$	$\exists r: r \in \text{Traces}(f') \wedge r \in \text{Traces}(\text{copy}) \wedge t = O \copyright r$		{Definition von Traces}
$\Leftrightarrow$	$\exists r:$	$O \copyright r = f'(O' \copyright r)$	(1)
		$\wedge\ \forall s \sqsubseteq r: O \copyright s \sqsubseteq f'(O' \copyright s)$	(2)
		$\wedge\ O' \copyright r = (O \copyright r)'$	(3)

$\wedge\ \forall s \sqsubseteq r: O'©s \sqsubseteq (O©s)'$ (4)
$\wedge\ t = O©r$

Man wähle ein solches $r \in (O \cup O')^\omega$ (Dies existiert, da Traces((*hide* O' *in* ([f'] || [copy])) $\neq$ Ø). Nach (1) und (3) gilt: $f'(O'©r) = f'((O©r)') = O©r$; also ist $O©r$ Fixpunkt von f. Nun ist zu zeigen: $O©r$ ist *kleinster* Fixpunkt von f. Sei $z \in O^\omega$ kleinster Fixpunkt von f, gelte also:

$f'(z') = z$ (5)
$\wedge\ f'(x') = x \Rightarrow z \sqsubseteq x$ (6)

Zu zeigen: $z \sqsupseteq O©r$ (*)

Mit (1) und (6) gilt $z \sqsubseteq O©r$, daher mit (3) $z' \sqsubseteq (O©r)' = O'©r$. Deshalb gibt es größte u, v $\in (O \cup O')^\omega$ mit

$u \sqsubseteq r,\ v \sqsubseteq r$ (7)
$z = O©u$ (8)
$z' = O'©v$ (9)

Behauptung: Es gilt

$u \sqsubseteq v$ (10)

Sonst gälte mit (7) $v \sqsubset u$, damit mit Monotonie $O'©v \sqsubseteq O'©u$ und $O'©v =;^{(9)} z' =;^{(8)} (O©u)' \sqsupseteq;^{(4)} O'©u$ und daher $O'©v = O'©u$. Dies wäre ein Widerspruch zur Annahme, daß v das größte Element ist, so daß (9) gilt.

$z =;^{(5)} f'(z') =;^{(9)} f'(O'©v) \sqsupseteq;^{(2)} O©v \sqsupseteq;^{(10)} O©u = z$. Daher:

$O©v = O©u$ (11)

Da u "größtmöglich" gewählt und $u \sqsubseteq v$ muß $u = v$ gelten.

Ist u echtes Präfix von r, so gilt $r = u \circ o \circ s$ mit $o \in O$ und $s \in (O \cup O')^\omega$ (sonst u nicht "größtmöglich"). Ist v echtes Präfix von r, so gilt $r = v \circ o' \circ p$ mit $o' \in O'$ und $p \in (O \cup O')^\omega$ (sonst v nicht "größtmöglich"). u (= v) kann also kein echtes Präfix von r sein, daher (mit (10)):

$u = v = r$ (12)

Damit gilt: $z =;^{(8)} O©u =;^{(12)} O©r$. Damit ist (*) gezeigt. □

Für eine weitergehende Analyse der Beziehung zwischen I/O-Automaten und stromverarbeitenden Agenten vgl. [Lynch, Stark 89].

Die Spursemantik für Agenten bietet den Vorteil, daß für unterschiedliche Komponenten eines verteilten Systems unterschiedliche Spezifikationstechniken eingesetzt werden können; der

Zusammenhalt ist durch die Spursemantik gewährleistet. Die Erfahrung zeigt nämlich, daß für manche Komponenten bereits am Anfang der Systementwicklung eine Agentenspezifikation angegeben werden kann, während bei anderen z.B. eine Beschreibung mit Spurformeln einfacher ist, die dann schrittweise in konkretere Beschreibungsformen umgesetzt wird.

5.3 Methodische Vorgehensweise bei der Entwurfsspezifikation

Die Unterschiede zwischen einer komponentenorientierten Spurspezifikation und einer funktionalen Spezifikation (vgl. Abschnitt 5.1) bestimmen die Entwicklungsrichtung beim Übergang zwischen diesen beiden Beschreibungsformen: die Anforderungen an die Produktkomponenten sind zu lokalisieren, von der Spurbeschreibung der Produktkomponenten ist zur funktionalen Beschreibung dieser Komponenten zu wechseln. Liegen die Anforderungen an die Produktkomponenten erst einmal lokal vor, so sind die Anforderungen an die Umgebungskomponenten für die weitere Systementwicklung nicht mehr von Bedeutung. Die Systementwicklung kann dann auf modulare Weise fortschreiten, basierend auf den lokalen Beschreibungen (Schnittstellen) der Produktkomponenten.

Die Lokalisierung der Anforderungen läßt sich mit der Methode der *schrittweisen Verfeinerung* durchführen: Ein System bestehe aus n Komponenten mit Ein- und Ausgaben $I_1, O_1, ..., I_n, O_n$. Auf jeder Verfeinerungsstufe gibt es eine globale Anforderung G und lokale Anforderungen $C_1, ..., C_n$. In der komponentenorientierten Anforderungsspezifikation, dem Startpunkt der schrittweisen Verfeinerung, ist G die globale Spezifikation und C_i gibt entweder die Schnittstelle einer Umgebungskomponente (falls i Index einer Umgebungskomponente) oder eine lokale (Teil-)Anforderung an eine Produktkomponente wieder (falls i Index einer Produktkomponente). Ein Verfeinerungsschritt besteht darin, eine globale Anforderung G' sowie lokale Anforderungen C_i' für die Produktkomponenten zu finden, so daß gilt:

$$\begin{aligned} & G'(t) \wedge C_1'((I_1 \cup O_1) \copyright t) \wedge ... \wedge C_n'((I_n \cup O_n) \copyright t) \\ \Rightarrow \quad & G(t) \wedge C_1((I_1 \cup O_1) \copyright t) \wedge ... \wedge C_n((I_n \cup O_n) \copyright t). \end{aligned}$$

Die Anforderungen an die Umgebungskomponenten müssen gleich bleiben, d.h. ist i Index einer Umgebungskomponente, so gilt $C_i'(t) \Leftrightarrow C_i(t)$. Die Lokalisierung der Anforderungen ist beendet, wenn der globale Anteil gleich *true* ist, d.h. es werden keine globalen Anforderungen mehr gestellt. Ein Verfeinerungsschritt ist daher zielgerichtet, wenn $G(t) \Rightarrow G'(t)$ gilt, aber nicht die Gegenrichtung; in diesem Fall wird tatsächlich ein Teil der globalen Anforderungen lokalisiert.

Die Lokalisierung ist ein Entwurfsschritt, daher nur wenig schematisierbar (vgl. Abschnitt 4.3). Ich verweise auf die Arbeiten [Li, Maibaum 88] und [Chandy, Misra 88], wo die schrittweise Verfeinerung an Beispielen demonstriert wird. Wie in unserer Methodik lassen sich dort die Verfeinerungsschritte in einem logischen Kalkül durchführen. Man sieht leicht, daß sich die prinzipielle Vorgehensweise, insbesondere Heuristiken, aus den erwähnten Arbeiten auch in unserer Entwurfsmethodik verwenden lassen.

Das Ergebnis der Verfeinerungen ist "vernünftig", wenn sich die Produktkomponenten realisieren lassen. Dies ist eine eher implizite Anforderung, die erst im Laufe der Systementwicklung festgestellt wird. Daher ist es möglich, daß Entwurfsentscheidungen rückgängig gemacht werden müssen, wenn globale Anforderungen derart den Produktkomponenten zugeordnet werden, daß sie nicht realisierbar sind (bei einem vorgegebenen Realisierungskonzept, hier stromverarbeitende Agenten, vgl. Abschnitt 5.4). Eventuell muß sogar die Komponentenstruktur des Systems geändert werden.

Die Realisierbarkeit bringt uns zur zweiten Aufgabe bei der Entwurfsspezifikation: dem Wechsel von einer Spur- zu einer funktionalen Beschreibung der Produktkomponenten, d.h. der Umsetzung eines Prädikats über Spuren in ein Prädikat über stromverarbeitende Funktionen. Unser semantisches Modell (vgl. Abschnitt 5.2) ist so gewählt, daß dieser Übergang (zumindest von der theoretischen Seite her) keine Schwierigkeit darstellt. Für ein Prädikat P über Spuren läßt sich ein Prädikat über Funktionen angeben, das den größtmöglichen realisierbaren Anteil der durch P festgelegten Spurmenge beschreibt:

$$Q_P(f) \equiv \forall t: t \in \text{Traces}(f) \Rightarrow P(t)$$

Das heißt, wir spezifizieren jene Menge stromverarbeitender Funktionen, die nur solche Spuren erzeugen, die das Prädikat P erfüllen.

Umgekehrt läßt sich auch jedes Prädikat Q über stromverarbeitende Funktionen als Prädikat über Spuren interpretieren:

$$P_Q(t) \equiv \exists f: Q(f) \wedge t \in \text{Traces}(f)$$

Auf diese Weise lassen sich Spur- und Funktionsbeschreibungen flexibel kombinieren.

Eine lokale Komponentenspezifikation P ist durch stromverarbeitende Agenten realisierbar, wenn das entsprechende Prädikat über (stetigen) Funktionen Q_P nicht identisch *false* ist. Wir erkennen, daß es (auf der Spurebene) konsistente Spezifikationen gibt, die sich (bei einer bestimmten Festlegung der Ein- und Ausgaben) nicht realisieren lassen. Ist Q_P identisch *false*, so müssen - wie oben angesprochen - Entwurfsentscheidungen revidiert werden.

5.4 Hierarchie der Realisierbarkeit

In den Abschnitten 4.3 und 5.2 lernten wir verschiedene Arten der Realisierung von Spurmengen kennen: sowohl einer Strategie, als auch einem stromverarbeitender Agenten und einem I/O-Automaten läßt sich eine Spurmenge zuordnen, die als Beschreibung des Ein-/Ausgabeverhaltens einer Komponente interpretiert werden kann. Tatsächlich gibt es eine "Hierarchie der Realisierbarkeit", nämlich

$$\text{stromverarbeitende Agenten} \subset \text{Strategien} \subset \text{I/O-Automaten} \qquad (*)$$

$A \subset B$ ist folgendermaßen zu interpretieren: für jedes $a \in A$ gibt es ein $b \in B$ (mit gleichen Ein- und Ausgaben), so daß $\text{Traces}(a) = \text{Traces}(b)$ gilt; z.B. läßt sich jede Spurmenge, die sich

durch einen stromverarbeitenden Agenten realisieren läßt, auch durch eine Strategie realisieren. Die Korrektheit der Aussage (*) zeige ich in diesem Abschnitt.[1]

Prinzipiell ließen sich zur Agentenbeschreibung auf der Entwurfsebene auch Strategien und I/O-Automaten einsetzen. Die Entscheidung für stromverarbeitende Agenten bringt jedoch zwei Vorteile: erstens lassen sich mit ihnen asynchron kommunizierende Agenten gut modellieren, ohne daß Kommunikationskanäle explizit beschrieben werden müssen. Dies ist mit Strategien und I/O-Automaten nicht (ohne eine Einschränkung auf eine Teilklasse) möglich, da in beiden Konzepten eine neue Eingabe eine bereits festgelegte Ausgabe revidieren kann. Zweitens ermöglichen sie eine funktionale Beschreibung eines verteilten Systems, was neben einem einfachen konzeptuellen Modell auch eine relativ einfache technische Handhabbarkeit bietet.

Die Einführung von Strategien und I/O-Automaten (statt allein stromverarbeitende Funktionen) im Rahmen meiner Arbeit hat folgende Gründe: Mit Strategien läßt sich das Wechselspiel zwischen einer Komponente und ihrer Umgebung anschaulicher beschreiben und analysieren als mit stromverarbeitenden Agenten; Ergebnisse aus dem Bereich der I/O-Automaten zu einer kompositionalen Spursemantik lassen sich auf stromverarbeitende Agenten übertragen. Die Hierarchie (*) rechtfertigt diese Vorgehensweise, da stromverarbeitende Agenten als Spezialfall von Strategien und I/O-Automaten verstanden werden können.

Zunächst zur Beziehung "stromverarbeitende Agenten $\subset$ Strategien". Die folgenden beiden Sätze zeigen, daß die Teilklasse der FIFO-Strategien und stromverarbeitende Funktionen (mit einem Eingabe- und einem Ausgabekanal) in Bezug auf die Spursemantik gleich mächtig sind.

Satz (FIFO-Strategien bestimmen stromverarbeitende Funktionen): Sei für jede FIFO-Strategie $\sigma: (I \cup O)^* \to O \cup \{\varepsilon\}$ mit $I \cap O = \varnothing$ die Funktion $f_\sigma: I^\omega \to O^\omega$ folgendermaßen definiert:

$$f_\sigma(w) =_{def} O \copyright \text{trace}(\sigma, h_w),$$

wobei für $w \in I^\omega$ die Funktion $h_w: \text{Nat} \to I^*$ definiert ist (sei dabei $w = i_0 \circ i_1 \circ \ldots$ mit $i_k \in I$ für alle k) durch:

$$h_w(k) =_{def} \begin{cases} i_k & \text{falls } 0 \le k < \#w; \\ \varepsilon & \text{falls } k > \#w \end{cases}$$

Dann gilt:

1. f_σ ist wohldefiniert und stetig.
2. $\text{Traces}(f_\sigma) = \text{Traces}(\sigma)$.

---

[1] Die Aussage (*) bezieht sich auf das vorgestellte Konzept stromverarbeitender Agenten. Es ist zu erwarten, daß eine abweichende funktionale Agentenmodellierung, etwa die Modellierung durch Funktionen der Funktionalität $(I^*)^\omega \to (O^*)^\omega$ statt $I^\omega \to O^\omega$, Auswirkungen auf die Realisierbarkeitsbeziehung zwischen den verschiedenen Konzepten hat. Somit spreche in nur den "Standardfall" stromverarbeitender Agenten an.

Beweis: Zu 1. a) Die Wohldefiniertheit von f_σ ist offensichtlich.
b) f_σ ist stetig. Wir spalten b) in b1) und b2) auf:
b1) f_σ ist monoton: Sei $w \sqsubseteq w'$. Zu zeigen ist: $f_\sigma(w) \sqsubseteq f_\sigma(w')$.
Nehmen wir an, daß $w \sqsubseteq w'$ und $\neg f_\sigma(w) \sqsubseteq f_\sigma(w')$ gilt. Also gibt es ein $o \in O$ und ein kleinstes $t \in O^*$, so daß $t \circ o \sqsubseteq f_\sigma(w)$, aber $\neg t \circ o \sqsubseteq f_\sigma(w')$. Da $t \circ o$ endlich ist, muß es ein k geben, so daß $O \copyright ptrace(\sigma, h_w, k) = t \circ o$ (dabei ist $k > \#w$). Man definiere h': Nat $\to I^*$ folgendermaßen:

$h'(p) =_{def} h_w(p) = h_{w'}(p)$, für $p \leq \#w$,
$h'(p) =_{def} h_w(p) = \varepsilon$, für $\#w \leq p < k$,
$h'(k-1+q) =_{def} h_{w'}(\#w+q)$, für alle $q>1$.

(d.h. man ergänze h' so, daß auch der Rest von w' eingegeben wird).
Dann gilt: $conc(h') = w' = conc(h_{w'})$. Da σ eine FIFO-Strategie ist, gilt dann: $O \copyright trace(\sigma, h') = O \copyright trace(\sigma, h_{w'})$. $O \copyright ptrace(\sigma, h', k) = t \circ o$, aber $t \circ o$ ist kein Präfix von $O \copyright trace(\sigma, h_{w'})$ und wir erhalten einen Widerspruch.
b2) $f_\sigma(\bigsqcup_n w_n) \sqsubseteq \bigsqcup_n f_\sigma(w_n)$ für jede Kette $w_0 \sqsubseteq w_1 \sqsubseteq \ldots$
Es läßt sich leicht zeigen, daß es für ein beliebiges $k \in$ Nat es ein m gibt, so daß
$ptrace(\sigma, h_{\bigsqcup_n w_n}, k) \sqsubseteq ptrace(\sigma, h_{w_m}, k)$ gilt. Daher gilt: $f_\sigma(\bigsqcup_n w_n) = \bigsqcup_k O \copyright ptrace(\sigma, h_{\bigsqcup_n w_n}, k) \sqsubseteq \bigsqcup_n \bigsqcup_k O \copyright trace(\sigma, h_{w_n}, k) = \bigsqcup_n f_\sigma(w_n)$.
Zu 2. a) $Traces(f_\sigma) \supseteq Traces(\sigma)$
Sei $t \in Traces(\sigma)$. Dann gilt $t = trace(\sigma, h)$ für ein h: Nat $\to I^*$.
i) Zunächst ist zu zeigen: $f_\sigma(I \copyright t) = O \copyright t$: Da $conc(h_{I \copyright t}) = conc(h)$, gilt nach der Definition einer FIFO-Strategie: $O \copyright t = O \copyright trace(\sigma, h) = O \copyright trace(\sigma, h_{I \copyright t}) = f_\sigma(I \copyright t)$
ii) Nun ist zu zeigen: $\forall s: s \sqsubseteq t \Rightarrow O \copyright s \sqsubseteq f_\sigma(I \copyright s)$ (*).
Sei $s \sqsubseteq t$ und s endlich (sonst gilt $s=t$ und mit i) gilt auch (*)). Dann gibt es ein kleinstes k mit $s \sqsubseteq ptrace(\sigma, h, k)$. $ptrace(\sigma, h, k)$ ist von der Gestalt $r \circ h(k)$ mit $r \in (I \cup O)^*$ (sonst wäre k nicht kleinstmöglich gewählt) und $r \sqsubseteq s$. Man definiere h': Nat $\to I^*$ folgendermaßen: $h'(p) = h(p)$ für $0 \leq p < k$, $h'(k)$ wird so definiert, daß $ptrace(\sigma, h', k) = s$ und $h'(p) = \varepsilon$ für $p > k$. Es gilt $I \copyright s = conc(h')$ und nach der Definition einer FIFO-Strategie gilt daher: $O \copyright s \sqsubseteq O \copyright trace(\sigma, h') = O \copyright trace(\sigma, h_{I \copyright s}) = f_\sigma(I \copyright s)$.
b) $Traces(f_\sigma) \subseteq Traces(\sigma)$
Sei $t \in Traces(f_\sigma)$. Dann hat t die Form $t = w_0 \circ o_0 \circ w_1 \circ o_1 \ldots$ mit $w_k \in I^*$, $o_k \in O \cup \{\varepsilon\}$, wobei gelte: Falls $o_k = \varepsilon$, dann $o_{k+1} = \varepsilon$ für alle $k \in$ Nat.
Man definiere $h(k) =_{def} w_k$ für alle k. Durch Induktion über n läßt sich zeigen: $ptrace(\sigma, h, n) = w_0 \circ o_0 \circ \ldots \circ w_n$ für alle n. Damit gilt: $t = \bigsqcup_n w_0 \circ o_0 \circ \ldots \circ w_n = \bigsqcup_n ptrace(\sigma, h, n)$. □

Satz (Stromverarbeitende Funktionen bestimmen FIFO-Strategien): Sei $f: I^\omega \to O^\omega$ eine stromverarbeitende Funktion. Die Strategie $\sigma_f: (I \cup O)^* \to O \cup \{\varepsilon\}$ ist folgendermaßen definiert:

$$\sigma_f(w) =_{def} \begin{cases} ft(\, rt^{\#O \copyright w}(\, f(I \copyright w)\,)\,) \\ \qquad \text{falls } \#\, f(I \copyright w) > \#\, O \copyright w; \varepsilon \\ \qquad\qquad\qquad \text{sonst} \end{cases}$$

Dann gilt: 1. σ_f ist eine FIFO-Strategie.
2. Traces(σ_f) = Traces(f).

Beweis: Im Beweis verwende ich die folgende Aussage, die sich durch Induktion zeigen läßt.

$\forall m$: O © ptrace(σ_f,h,m) $\sqsubseteq$ f(conc(h)) $\wedge$
$\forall k$: # f(conc(h)) $\geq$ k $\Rightarrow$ $\exists n$: # O©ptrace(σ_f,h,n) $\geq$ k. (*)

1. σ_f ist eine FIFO-Strategie: Seien h und h' Eingabemodellierungen und gelte: conc(h) = conc(h'). Dann gilt mit (*): O©trace(σ_f,h) = f(conc(h)) = f(conc(h')) = O©trace(σ_f,h').

2. a) Traces(σ_f) $\subseteq$ Traces(f):

Sei t$\in$ Traces(σ_f), d.h. t = trace(σ_f,h) für eine gewisse Eingabemodellierung h.

a1) Zunächst wird gezeigt: f(I©t) = O©t: Da I©t = I © trace(σ_f,h) = conc(h) ergibt sich mit (*) f(I©t) = f(conc(h)) = O © trace(σ_f,h) = O©t.

a2) Sei s$\sqsubseteq$ t. Es wird gezeigt: O©s $\sqsubseteq$ f(I©s):

Es ist ausreichend, s der Form s = ptrace(σ_f,h,m) $\circ$ σ_f(ptrace(σ_f,h,m)) für ein m zu betrachten. Für s$\circ$w mit w$\in I^\omega$ gilt nämlich: O©(t'$\circ$w) = O©s $\sqsubseteq$ f(I©s) $\sqsubseteq$ f(I©(s$\circ$w)).

Durch Induktion über m läßt sich zeigen, daß für alle m gilt:

O©ptrace(σ_f,h,m) $\circ$ σ_f(ptrace(σ_f,h,m)) $\sqsubseteq$ f(I© (ptrace(σ_f,h,m) $\circ$ σ_f(ptrace(σ_f,h,m))))

b) Traces(σ_f) $\supseteq$ Traces(f):

Sei t$\in$ Traces(f), d.h. f(I©t) = O©t $\wedge$ $\forall$s: s $\sqsubseteq$ t $\Rightarrow$ O©s $\sqsubseteq$ f(I©s).

Dann hat t die Form t = $w_0 \circ o_0 \circ w_1$... mit $w_k \in I^*$, $o_k \in O$ für alle k, wobei gelte: Falls o_k = ε, dann $o_{k+1} = \varepsilon$ für alle k$\in$ Nat. Man definiere h(k) =$_{def}$ w_k. Durch Induktion über m zeigt man, daß für alle m gilt: ptrace(σ_f,h,m) = $w_0 o_0 ... w_m$. Damit gilt aber auch trace(σ_f,h) = t. □

Es besteht jedoch ein Unterschied zwischen stromverarbeitenden Agenten mit mehr als einem Eingabekanal und Strategien. Mehrere Eingabekanäle werden bei Strategien dadurch modelliert, daß die Vereinigung der (disjunkten) "Kanalalphabete" als Eingabemenge der Strategie betrachtet wird. Diese Modellierungstechnik entspricht der von I/O-Automaten. Die Eingaben von verschiedenen Kanälen erscheinen also sequentiell. Daher lassen sich mit FIFO-Strategien Agenten modellieren, für die es keine Beschreibung durch eine Menge stromverarbeitender Funktionen gibt (etwa das faire nichtstrikte Mischen, vgl. [Broy 89a]). Man sieht jedoch leicht, daß sich umgekehrt jede stromverarbeitende Funktion mit mehr als einem Eingabekanal durch eine FIFO-Strategie modellieren läßt.

Betrachtet man statt FIFO-Strategien allgemeine Strategien, so findet man auch im Fall von Agenten mit nur einem Eingabekanal schnell eine Spurmenge, die sich durch eine Strategie, nicht jedoch durch einen stromverarbeitenden Agenten realisieren läßt:

Beispiel: Die Spurmenge $\{o\} \circ I^\omega \cup I^\omega$ (mit Eingabemenge I und dazu disjunkter Ausgabemenge {o}) läßt sich durch keinen stromverarbeitenden Agenten der Funktionalität $I^\omega \rightarrow \{o\}^\omega$ realisieren, aber durch die Strategie σ, die definiert ist durch:

$$\sigma(x) = \begin{cases} o & \\ & \text{falls } x = \varepsilon; \varepsilon \\ & \text{sonst} \end{cases}$$

□

Nun zum Zusammenhang zwischen I/O-Automaten und Strategie; dieser wird in [Broy et al. 91a] untersucht. Es wird gezeigt, daß sich jede Strategie durch einen I/O-Automaten modellieren läßt, nicht jedoch umgekehrt.[1] Diese Aussage ist unabhängig davon, ob der Begriff der starken oder schwachen Fairneß bei I/O-Automaten verwendet wird. Der Nichtdeterminismus bei I/O-Automaten wird dabei durch *Mengen* von Strategien modelliert. Es läßt sich ein Beispiel konstruieren, das in bestimmter Weise die Eingabe eines I/O-Automaten einschränkt, was gemäß den Anforderungen (1) und (2) an I/O-Automaten in Abschnitt 5.2 gerade ausgeschlossen sein soll. Dieser Fall einer Anomalie ist bei Strategien nicht möglich.

5.5 Echtzeitspezifikation für Agenten

Im folgenden skizziere ich, wie sich zeitbehaftete Spuren (vgl. Abschnitt 3.6.1) durch I/O-Automaten realisieren lassen. Hierzu stelle ich eine Erweiterung des Konzepts der I/O-Automaten vor.

Ein *zeitbehafteter I/O-Automat* ist wiederum ein Tupel $(I,O,State,s_0,—;>,F)$, nur ist nun die Übergangsrelation $—;>$ eine Teilmenge von $State \times (I \cup O \cup \{\tau\} \cup \{\surd\}) \times State$. $\surd$ beschreibt das Ticken einer Uhr. Von einer Berechnung wird im Unterschied zu Abschnitt 5.2 zusätzlich gefordert, daß die Anzahl der $\surd$-Übergänge unendlich ist. Insbesondere sind dann alle Berechnungen unendlich. Die $\surd$-Übergänge geben wie in Abschnitt 4.4 ein Zeitraster vor, es wird wiederum eine *globale* Zeit erfaßt.

Echtzeitanforderungen, die besagen, daß ein bestimmtes Ereignis spätestens nach einer gewissen vorgegebenen Anzahl von Zeitschritten stattgefunden haben muß, werden im I/O-Automaten dadurch repräsentiert, daß bei Überschreiten der Zeitschranke keine $\surd$-Übergänge mehr schaltbereit sind und somit die Forderung nach unendlich vielen $\surd$-Übergängen nicht mehr erfüllt werden kann.

Beispiel (zeitbehafteter I/O-Automat): Beschrieben wird ein I/O-Automat, der die Anforderung "nach spätestens N Zeiteinheiten folgt auf die Eingabe einer Aktion i die Ausgabe einer dazu korrespondierenden Aktion o" *(bounded response)* erfüllt:

Der Automat ist durch das Tupel $(I,O,State,s_0,—;>,F)$ gegeben, wobei gilt:

$I=\{i\}$, $O=\{o\}$, $State = Nat^{\omega}$, $s_0 = \varepsilon$, $—;>$ ist die Menge aller Übergänge (sei $k \geq 0$):

$$\begin{aligned}
&\langle n_1,\ldots,n_k\rangle \; —;^{i}> \; \langle n_1,\ldots,n_k,N+1\rangle, \\
&\langle n_1,\ldots,n_k\rangle \; —;^{\surd}> \; \langle n_1-1,\ldots,n_k-1\rangle \quad \text{falls } n_j \geq 0 \text{ für } j = 1,2,\ldots,k, \\
&\langle n_1,n_2,\ldots,n_k\rangle \; —;^{o}> \; \langle n_2,\ldots,n_k\rangle \quad \text{falls } n_j \geq 0 \text{ für } j = 1,2,\ldots,k,
\end{aligned}$$

---

[1] Die Modellierung des Zusammenspiels mehrerer Komponenten durch Strategien in Abschnitt 4.3.3 läßt sich als Komposition der den einzelnen Komponenten entsprechenden I/O-Automaten deuten.

mit natürlichen Zahlen $n_1,...,n_k$. Man beachte, daß der Übergang $\varepsilon \xrightarrow{\surd} \varepsilon$ ebenfalls enthalten ist. Die Fairneßmenge F ist gegeben als $\{\{s \xrightarrow{a} s' \mid a \in \{o,\surd\}\}\}$.

Die Zustände bestehen aus Tupeln von natürlichen Zahlen, die Zeitzähler modellieren. Jedes Tupel enthält die Zeitzähler derjenigen Eingabeaktionen, für die noch keine Ausgabeaktion stattfand. Ein Zeitzähler, der gleich 0 ist, verkörpert einen Zeitfehler. Erfolgt eine Eingabeaktion, so wird ein neuer Zeitzähler erzeugt und mit $N+1$ initialisiert. Eine $\surd$-Aktion erniedrigt alle vorhandenen Zeitzähler um 1; dies modelliert das Warten um eine Zeiteinheit. Eine Ausgabeaktion löscht den ersten Zeitzähler; dieser gehört zu jener Eingabe, für die noch keine Ausgabe stattfand und die darauf am längsten wartete. Allerdings geschieht dies nur, wenn vorher kein Zeitfehler stattfand, d.h. keine Eingabe länger als N Zeiteinheiten warten mußte. Man sieht: Diejenigen Spuren des Automaten, in denen unendlich viele $\surd$-Aktionen vorkommen, erfüllen die obige Anforderung. □

In [Broy 90] werden stromverarbeitende Agenten ebenfalls mit einer Aktionenmenge beschrieben, die zusätzlich die Aktion $\surd$ enthält. Die Modellierungssicht ist dort jedoch anders: Die $\surd$-Aktion modelliert eine "leere" Eingabe oder eine "leere" Ausgabe, je nachdem, ob sie auf einem Ein- oder einem Ausgabekanal erscheint. Die in diesem Abschnitt vorgestellte Modellierung mit I/O-Automaten lehnt sich dagegen an die Sichtweise der Abschnitte 3.6.1 bzw. 4.4 an.

6. Ausblick

Dieses Kapitel enthält Hinweise auf Forschungsgegenstände, die im Umfeld meiner Arbeit liegen, aber in ihr nicht oder nur ansatzweise behandelt wurden.

1. Gewinnung von Anforderungen

Meine Arbeit setzt bei vorhandenen informellen Anforderungen an, die in formale abzubilden sind. Von großer Bedeutung sind aber auch die vielfältigen personenbezogenen Probleme (soziologische, psychologische und Managementprobleme) bei der Gewinnung der Anforderungen aus ersten informellen Vorstellungen. Dieser Gesichtspunkt wurde in meiner Arbeit nur angerissen (siehe Abschnitt 1.2), für die Akzeptanz einer Entwurfsmethodik ist es jedoch wichtig, daß auch hierfür methodische Leitlinien bereitstehen.

2. Spezielle Anwendungsbereiche

Zu prüfen wäre, ob sich die in meiner Arbeit vorgestellte Methodik für gewisse Anwendungsbereiche spezialisieren läßt. Ein Beispiel ist der Schaltungsentwurf. Aufgrund der speziellen Struktur von Schaltungen (endlicher Zustandsraum, regulärer Aufbau) konnten in der Spurtheorie für den Schaltungsentwurf (vgl. Kap. 2) wesentliche Ergebnisse zur Verifikation asynchroner Schaltungen erzielt werden (siehe [Dill 89]). Zu untersuchen wäre, wie sich diese Ergebnisse in eine spezialisierte Entwurfsmethodik für Schaltungen einbringen lassen. Deren Zielsprache wäre eine Hardware-Beschreibungssprache, z.B. VHDL.

Ein anderer Bereich ist die Spezifikation und Verifikation von Kommunikationsprotokollen. Die im Spurformalismus verwendeten Techniken, die ablauforientierte und transitionsorientierte Spezifikation, ähneln typischen Techniken der Protokollbeschreibung: In Zeitablaufdiagrammen werden Abläufe beispielhaft und ausschnittsweise erfaßt; mit Spurformeln ist dagegen eine formale und vollständige Beschreibung von Abläufen möglich; Protokollautomaten ähneln den Transitionssystemen meiner Arbeit. Meine Methodik der Anforderungsspezifikation bietet weitere Vorteile: Protokolle und Dienste lassen sich durch Spuren auf abstraktere Weise beschreiben als mit den für die Telekommunikation entwickelten formalen Beschreibungstechniken (siehe [Hogrefe 89]) (vgl. Abschnitt 4.1). Zudem ist der Spurformalismus anders als diese Beschreibungstechniken in eine durchgängige Entwurfsmethodik eingebettet (vgl. auch Punkt 4).

3. Echtzeit

In der vorliegenden Arbeit wurden Erweiterungsmöglichkeiten des Spurformalismus zur Echtzeitmodellierung skizziert. Der Schwerpunkt lag bei der globalen Spezifikation; die komponentenorientierte und Agentenspezifikation von Echtzeitsystemen wurden nur kurz behandelt. Bei komponentenorientierten Spezifikationen stellt sich das Problem, lokale Zeiten in eine globale umzurechnen (vgl. Abschnitt 4.4). Für die Agentenspezifikation wurden zeitbehaftete I/O-Automaten als ein mögliches Echtzeitmodell vorgestellt (siehe Abschnitt 5.5); zu prüfen wäre, ob es komfortablere Beschreibungsmöglichkeiten für Agenten gibt, die sich in einfacher Weise zu Spurspezifikationen in Beziehung setzen lassen.

4. Einbeziehung anderer Agentenarten

Lediglich die globale Spezifikation (Kap. 3) ist unabhängig davon, auf welche Agentenart die Entwurfsmethodik ausgerichtet ist. Schon die komponentenorientierte Spezifikation (Kap. 4) ist auf asynchron kommunizierende Agenten abgestimmt. Zu untersuchen wäre, wie synchron kommunizierende Agenten in die Entwurfsmethodik einbezogen werden können und ob auch eine Mischform (manche Agenten kommunizieren synchron, manche asynchron) sinnvoll ist. Wichtige Ergebnisse zum Übergang von Spuren zu synchron kommunizierenden Agenten sind in der Habilitationsschrift von Olderog ([Olderog 88]) zu finden, in der jedoch nur eine spezielle Klasse von Lebendigkeitseigenschaften behandelt wird (vgl. Abschnitt 2.3).

Für Anwendungen im Bereich der Telekommunikation ist es lohnenswert, nach einer Schnittstelle zu den normierten formalen Beschreibungstechniken LOTOS, Estelle und SDL (vgl. [Hogrefe 89]) zu suchen. Für LOTOS, eine Erweiterung von CCS ([Milner 80]), gelten die für synchron kommunizierende Agenten gemachten Aussagen. Um die Sprache SDL in eine formale Entwurfsmethodik einbetten zu können, ist zunächst die Definition einer formalen Semantik nötig; Ansätze dazu sind in [Broy 89b] zu finden. Da sich in SDL Echtzeiteigenschaften ausdrücken lassen, besteht auch eine Verbindung zu Punkt 3. LOTOS, Estelle und SDL sind gemäß der Begriffswahl von FOCUS (vgl. Kap. 1) auf der Ebene der abstrakten Implementierung anzusiedeln. Daher ergäbe sich sich durch ihre Einbettung in FOCUS der Nutzeffekt, daß ein methodischer Weg von abstrakten, anwendungsorientierten Beschreibungen (Spurspezifikationen) zu Programmbeschreibungen in diesen normierten Sprachen vorgezeichnet ist.

Literaturverzeichnis

[Abadi et al. 89] M. Abadi, L. Lamport, P. Wolper: Realizable and Unrealizable Specifications of Reactive Systems. In: G. Ausiello, M. Dezani-Ciancaglini, S. Ronchi Della Rocca (Hrsg.): Automata, Languages and Programming, 16th Colloquium. LNCS 372, Berlin Heidelberg New York (Springer), 1989, S. 1-17

[Abadi, Lamport 90] M. Abadi, L. Lamport: Composing Specifications. In: J.W. de Bakker, W.-P. de Roever, G. Rozenberg (Hrsg.): Stepwise Refinement of Distributed Systems. LNCS 430, Berlin Heidelberg New York (Springer), 1990, S. 1-41

[Abadi, Plotkin 91] M. Abadi, G.D. Plotkin: A Logical View of Composition and Refinement. Erscheint in den "Proc. of POPL '91", 1991

[Alpern, Schneider 85] B. Alpern, F.B. Schneider: Defining Liveness. Information Processing Letters, Vol. 21, 1985, S. 181-185

[Bartussek, Parnas 77] A.W. Bartussek, D.L. Parnas: Using Traces to Write Abstract Specifications for Software Modules. Report TR 77-012, Univ. North Carolina, Chapel Hill, 1977

[Bemmerl et al. 91a] Th. Bemmerl, A. Bode, Th. Ludwig, S. Tritscher: MMK - Multiprocessor Multitasking Kernel (User's Guide and User's Reference Manual). SFB-Bericht Nr. 342/26/90 A, Technische Universität München, Institut für Informatik, TUM-I9103, August 1990

[Bjørner, Jones 82] D. Bjørner, C.B. Jones: Formal Specification and Software Development. Englewood Cliffs, New Jersey (Prentice-Hall), 1982

[Brock, Ackerman 81] J.D. Brock, W.B. Ackerman: Scenarios: A Model of Non-deterministic Computation. In: J. Diaz, I. Ramos (Hrsg.): Foundations of Programming Concepts, Intern. Coll. Peniscola, Spain, 1981. LNCS 107, Berlin Heidelberg New York (Springer), 1981, S. 252-259

[Broy 87a] M. Broy: Specification of a Railway System. Technische Berichte der Fakultät für Mathematik und Informatik, Universität Passau, MIP-8715, 1987

[Broy 87b] M. Broy: Algebraic and Functional Specification of a Serializable Database Interface. Technische Berichte der Fakultät für Mathematik und Informatik, Universität Passau, MIP-8718, 1987

[Broy 88a] M. Broy: An Example for the Design of Distributed Systems in a Formal Setting: The Lift Problem. Technische Berichte der Fakultät für Mathematik und Informatik, Universität Passau, MIP-8802, 1988

[Broy 88b] M. Broy: Nondeterministic Data Flow Programs: How to Avoid the Merge Anomaly. Science of Computer Programming, Vol. 10, 1988, S. 65-85

[Broy 89a] M. Broy: Towards a Design Methodology for Distributed Systems. In: M. Broy (Hrsg.): Constructive Methods in Computing Science. Berlin Heidelberg New York (Springer), 1989, S. 311-364

[Broy 89b] M. Broy: Towards a Formal Foundation of the Specification and Description Language SDL. Technische Berichte der Fakultät für Mathematik und Informatik, Universität Passau, MIP-8923, 1989

[Broy 90] M. Broy: Functional Specification of Time Sensitive Communicating Systems. In: J.W. de Bakker, W.-P. de Roever, G. Rozenberg (Hrsg.): Stepwise Refinement of Distributed Systems. LNCS 430, Berlin Heidelberg New York (Springer), 1990, S. 153-179

[Broy 91] M. Broy: Composition and Refinement of Functional System Specifications. Interner Vortrag, Technische Universität München, September 1991

[Broy, Streicher 87] M. Broy, Th. Streicher: Specification and Design of Shared Resource Arbitration. Technische Berichte der Fakultät für Mathematik und Informatik, Universität Passau, MIP-8721, 1987

[Broy et al. 91a] M. Broy, F. Dederichs, C. Dendorfer, R. Weber: Characterizing the Behaviour of Reactive Systems by Trace Sets. SFB-Bericht Nr. 342/3/91 A, Technische Universität München, Institut für Informatik, TUM-I9103, Februar 1991

[Broy et al. 91b] M. Broy, F. Dederichs, C. Dendorfer, M. Fuchs, Th. Gritzner, R. Weber: The Design of Distributed Systems - An Introduction. Vorversion, 1991

[Chandy, Misra 88] K.M. Chandy, J. Misra: Parallel Program Design. Wokingham, England, u.a. (Addison-Wesley), 1988

[Dasarathy 85] B. Dasarathy: Timing Constraints of Real-Time Systems: Constructs for Expressing Them, Methods of Validating Them. IEEE Trans. on Software Engineering, Vol. SE-11, No. 1, 1985, S. 80-86

[Davis 90] A.M. Davis: Software Requirements, Analysis and Specification. Englewood Cliffs, New Jersey (Prentice-Hall), 1990

[Dederichs, Weber 90] F. Dederichs, R. Weber: Safety and Liveness from a Methodological Point of View. Information Processing Letters, Vol. 36, No. 1, 1990, S. 25-30

[Dendorfer 91] C. Dendorfer: Funktionale Modellierung eines Postsystems. Entwurfsversion, Juli 1991

[Denert 91] E. Denert: Software Engineering. Berlin Heidelberg New York (Springer), 1991

[Dennis 74] J.B. Dennis: First Version of a Data Flow Procedure Language. In: B. Robinet (Hrsg.): Colloque sur la Programmation. LNCS 19, Berlin Heidelberg New York (Springer), 1974, S. 362-367

[Dill 89] D.L. Dill: Trace Theory for Automatic Hierarchical Verification of Speed-independent Circuits. Cambridge, Massachusetts (MIT Press), 1989

[Drost 90] N.J. Drost: Algebraic Formulations of Trace Theory. Technical Report P9004, University of Amsterdam, Department of Mathematics and Computer Science, Programming Research Group, Juli 1990

[Dubois, Hagelstein 87] E. Dubois, J. Hagelstein: Reasoning on Formal Requirements: A Lift Control System. Proc. of the 4th International Workshop on Software Specification and Design. Silver Spring, MD (IEEE Computer Society Press), 1987, S. 161-168

[Dubois et al. 88] E. Dubois, J. Hagelstein, A. Rifaut: Formal Requirements Engineering with ERAE. Philips Journal of Research, Oktober 1988

[Dubois et al. 90] E. Dubois, J. Hagelstein, A. Rifaut: ERAE: A Formal Language for Expressing and Structuring Real-Time Requirements. Entwurfsversion, Juni 1990

[Emerson 90] E.A. Emerson: Temporal and Modal Logic. In: J. van Leeuwen (Hrsg.): Handbook of Theoretical Computer Science. Volume B. Formal Models and Semantics. Amsterdam (Elsevier), 1990, S. 995-1072

[Emerson, Halpern 86] E.A. Emerson, J.Y. Halpern: "Sometimes" and "Not Never" Revisited: On Branching versus Linear Time Temporal Logic. Journal of the ACM, Vol. 33, No. 1, Januar 1986, S. 151-178

[Faustini 82] A.A. Faustini: An Operational Semantics for Pure Dataflow. In: M. Nielsen, E.M. Schmidt (Hrsg.): Automata, Languages and Programming, 9th Colloquium. LNCS 140, Berlin Heidelberg New York (Springer), 1982, S. 212-224

[Finkelstein et al. 91] A. Finkelstein, M. Gödicke, J. Kramer, C. Niskier: ViewPoint Oriented Software Development: Methods and Viewpoints in Requirements Engineering. In: J.A. Bergstra, L.M.G. Feijs (Hrsg.): Algebraic Methods II: Theory, Tools and Applications. LNCS 490, Berlin Heidelberg New York (Springer), 1991, S. 29-54

[Gries 81] D. Gries: Science of Programming. Berlin Heidelberg New York (Springer), 1981

[Hall 90] A. Hall: Seven Myths of Formal Methods. IEEE Software, September 1990, S. 11-19

[Harel, Pnueli 85] D. Harel, A. Pnueli: On the Development of Reactive Systems. In: K.R. Apt (Hrsg.): Logics and Models of Concurrent Systems. Berlin Heidelberg New York (Springer), 1985, S. 477-498

[Hoare 85] C.A.R. Hoare: Communicating Sequential Processes. Englewood Cliffs, New Jersey (Prentice Hall), 1985

[Hogrefe 89] D. Hogrefe: Estelle, LOTOS und SDL: Standard-Spezifikationssprachen für verteilte Systeme. Berlin Heidelberg New York (Springer), 1989

[IEEE 83] IEEE Standard Glossary of Software Engineering Terminology. IEEE Standard 729, 1983

[Jeremaes et al. 86] P. Jeremaes, S. Khosla, T.S.E. Maibaum: A Modal (Action) Logic for Requirements Specification. In: D. Barnes, P. Brown (Hrsg.): Software Engineering 86. London (Peter Peregrinus), 1986, S. 278-294

[Jones 83] C.B. Jones: Tentative Steps Toward a Development Method for Interfering Programs. ACM Trans. on Programming Languages and Systems, Vol. 5, No. 4, 1983, S. 596-619

[Jonsson 87] B. Jonsson: Compositional Verification of Distributed Systems. Ph.D. Thesis, Department of Computer Systems, Uppsala University, Uppsala, Schweden, DoCS 87/09, 1987

[Jonsson 90] B. Jonsson: A Hierarchy of Fully Abstract Models of I/O-Automata. In: B. Rovan (Hrsg.) Mathematical Foundations of Computer Science. LNCS 452, Berlin Heidelberg New York (Springer), 1990, S. 347-354

[Kahn 74] G. Kahn: The Semantics of a Simple Language for Parallel Programming. In: Information Processing 74. Amsterdam (North Holland), 1974. S. 471-475

[Keller 78] R.M. Keller: Denotational Models for Parallel Programs with Indeterminate Operators. In: E.J. Neuhold (Hrsg.): Formal Description of Programming Concepts. Amsterdam (North Holland), 1978, S. 337-366

[Kok 86] J.N. Kok: Denotational Semantics of Nets with Nondeterminism. In: B. Robinet, R. Wilhelm (Hrsg.): ESOP 86. LNCS 213, Berlin Heidelberg New York (Springer), 1986, S. 237-249

[Lam, Shankar 90] S.S. Lam, A.U. Shankar: A Relational Notation for State Transition Systems. IEEE Trans. on Software Engineering, Vol. SE-16, No. 7, 1990, S. 755-775

[Lamport 80] L. Lamport: "Sometime" is Sometimes "Not Never". In: Proc. of the 7th Annual ACM Sigact-Sigplan Symposium on Principles of Programming Languages. New York (ACM Press), 1980, S. 174-185

[Lamport 83a] L. Lamport: Specifying Concurrent Program Modules. ACM Trans. on Programming Languages and Systems, Vol. 5, No. 2, 1983, S. 190-222

[Lamport 83b] L. Lamport: What Good is Temporal Logic? In: Information Processing 83. Amsterdam (North Holland), 1983, S. 657-668

[Lamport 89] L. Lamport: A Simple Approach to Specifying Concurrent Systems. Comm. of the ACM, Vol. 32, No. 1, 1989, S. 32-45

[Lamport 90] L. Lamport: A Temporal Logic of Actions. Digital Systems Research Center, Palo Alto, Kalifornien, Technical Report 57, April 1990

[Lamport, Lynch 90] L. Lamport, N. Lynch: Distributed Computing: Models and Methods. In: J. van Leeuwen (Hrsg.): Handbook of Theoretical Computer Science. Volume B. Formal Models and Semantics. Amsterdam (Elsevier), 1990, S. 1157-1200

[Li, Maibaum 88] D.-H. Li, T.S.E. Maibaum: A Top-down Step-wise Refinement Methodology for Protocol Specification. In: F.H. Vogt (Hrsg.): Concurrency 88. LNCS 335, Berlin Heidelberg New York (Springer), 1988, S. 197-221

[Lichtenstein et al. 85] O. Lichtenstein, A. Pnueli, L. Zuck: The Glory of the Past. In: Proc. Conf. on Logics of Programs. LNCS 131, Berlin Heidelberg New York (Springer), 1985, S. 200-252

[Loeckx, Sieber 87] J. Loeckx, K. Sieber: The Foundations of Program Verification. Stuttgart, Chichester u.a. (Wiley-Teubner), 2. Auflage, 1987

[Lynch, Stark 89] N.A. Lynch, E.W. Stark: A Proof of the Kahn Principle for Input/Output Automata. Information and Computation, Vol. 82, 1989, S. 81-92

[Mazurkiewicz 86] A. Mazurkiewicz: Trace Theory. In: W. Brauer, W. Reisig, G. Rozenberg (Hrsg.): Petri Nets: Applications and Relationship to Other Models of Concurrency. LNCS 255, Berlin Heidelberg New York (Springer), 1986, S. 279-304

[Meyer 83] M. Meyer: Operations Research/Systemforschung. Stuttgart (Gustav Fischer), 1983

[Milner 80] R. Milner: A Calculus for Communicating Systems. LNCS 92, Berlin Heidelberg New York (Springer), 1980

[Misra, Chandy 81] J. Misra, K.M. Chandy: Proofs of Networks of Processes. IEEE Trans. on Software Engineering, Vol. SE-7, No. 4, 1981, S. 417-426

[Moschovakis 89] Y.N. Moschovakis: A Game-theoretic Modelling of Concurrency. Fourth Annual Symposium on Logic in Computer Science. Silver Spring, MD (IEEE Computer Society Press), 1989, S. 154-163

[Nerode et al. 90] A. Nerode, A. Yakhnis, V. Yakhnis: Concurrent Programs as Strategies in Games. Technical Report '90-78, Cornell University, 1990

[Olderog 88] E.-R. Olderog: Nets, Terms and Formulas: Three Views of Concurrent Processes and Their Relationship. Habilitationsschrift, Universität Kiel, Dezember 1988

[Pandya 90] P. Pandya: Some Comments on the Assumption-commitment Framework for Compositional Verification of Distributed Programs. In: J.W. de Bakker, W.-P. de Roever, G. Rozenberg (Hrsg.): Stepwise Refinement of Distributed Systems. LNCS 430, Berlin Heidelberg New York (Springer), 1990, S. 622-640

[Parnas et al. 90] D.L. Parnas, G.J.K. Asmis, J. Madey: Assessment of Safety-critical Software. Queen's University at Kingston, Department of Computing and Information Science, Technical Report 90-295, Kingston, Ontario, Kanada, 1990

[Pnueli 86] A. Pnueli: Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends. In: J.W. de Bakker, W.-P. de Roever, G. Rozenberg (Hrsg.): Current Trends in Concurrency. Overviews and Tutorials. LNCS 224, Berlin Heidelberg New York (Springer), 1986, S. 510-584

[Pratt 86] V. Pratt: Modeling Concurrency with Partial Orders. International Journal of Parallel Programming, Vol. 15, No. 1, 1986, S. 33-71

[Reed, Roscoe 86] G.M. Reed, A.W. Roscoe: A Timed Model for Communicating Sequential Processes. In: L. Kott (Hrsg.): Automata, Languages, and Programming. LNCS 226, Berlin Heidelberg New York (Springer), 1986, S.314-323

[Reisig 82] W. Reisig: Petrinetze. Eine Einführung. Berlin Heidelberg New York (Springer), 1982

[Rem 85] M. Rem: Concurrent Computations and VLSI Circuits. In: M. Broy (Hrsg.): Control Flow and Data Flow: Concepts of Distributed Programming. Berlin Heidelberg New York (Springer), 1985, S.399-437

[Sanders 91] B.A. Sanders: Eliminating the Substitution Axiom from UNITY Logic. Formal Aspects of Computing, Vol. 3, No. 2, 1991, S. 189-205

[Sanella 88] D. Sanella: A Survey of Formal Software Development Methods. University of Edinburgh, Technical Report ECS-LFCS-88-56, 1988

[Snepscheut 85] J.L.A. van de Snepscheut: Trace Theory and VLSI Design. LNCS 200, Berlin Heidelberg New York (Springer), 1985

[Schneider 87] F.B. Schneider: Decomposing Properties into Safety and Liveness using Predicate Logic. Technical Report, Cornell University, Oktober 1987

[Staples, Nguyen 85] J. Staples, V.L. Nguyen: A Fixpoint Semantics for Nondeterministic Data Flow. Journal of the ACM, Vol. 32, No. 2, 1985, S. 411-444

[Tanenbaum 81] A.S. Tanenbaum: Computer Networks. Englewood Cliffs, New Jersey (Prentice-Hall), 1981

[Thomas 90] W. Thomas: Automata on Infinite Objects. In: J. van Leeuwen (Hrsg.): Handbook of Theoretical Computer Science. Volume B. Formal Models and Semantics. Amsterdam (Elsevier), 1990, S. 135-192

[Vogler 91] W. Vogler: Is Partial Order Semantics Necessary for Action Refinement?. SFB-Bericht Nr. 342/1/91 A, Technische Universität München, Institut für Informatik, TUM-I9103, Januar 1991

[Weber 90a] R. Weber: Specifying Distributed Systems in the PROSPECTRA Project - An Introduction. PROSPECTRA Report M.2.3.C1-R-4.0, Universität Passau, 1990

[Weber 90b] R. Weber: Echtzeit-Anforderungsspezifikationen und ihre Anwendung auf die Beschreibung eines Postfachsystems. Entwurfsversion, 1990

[Weber 91] R. Weber: Where can I get gas round here? - An Application of a Design Methodology for Distributed Systems. In: J.A. Bergstra, L.M.G. Feijs (Hrsg.): Algebraic Methods II: Theory, Tools and Applications. LNCS 490, Berlin Heidelberg New York (Springer), 1991, S. 143-166

[Wirsing 90] M. Wirsing: Algebraic Specification. In: J. van Leeuwen (Hrsg.): Handbook of Theoretical Computer Science. Volume B. Formal Models and Semantics. Amsterdam (Elsevier), 1990, S. 675-788

[Zave, Schell 83] P. Zave, W. Schell: Salient Features of an Executable Specification Language and Its Environment. IEEE Trans. on Software Engineering, Vol. SE-12, No. 2, 1986, S. 312-325

[Zwiers et al. 84] J. Zwiers, A. de Bruin, W.-P. de Roever: A Proof System for Partial Correctness of Dynamic Networks of Processes. In: E. Clarke, D. Kozen (Hrsg.): Logics of Programs. LNCS 164, Berlin Heidelberg New York (Springer), 1984, S. 513-527

Verzeichnis der Textstellen zum Postfachbeispiel

Stichwortverzeichnis