T U M

I N S T I T U T F Ü R I N F O R M A T I K

Sprachkonzepte zur Konstruktion verteilter Systeme

P. P. Spies
C. Eckert, M. Lange, D. Marek
R. Radermacher, F. Weimer und H.-M. Windisch

TUM-I9618
März 1996

TECHNISCHE UNIVERSITÄT MÜNCHEN

TUM-INFO-03-1996-I9618-100/1.-FI

Typescript: ---

Druck: Mathematisches Institut und
Institut für Informatik der
Technischen Universität München

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Dieser Bericht ist ein Beitrag zu einer nach wie vor hochaktuellen Aufgabe der Informatik: Die Entwicklung verteilter und paralleler Rechensysteme — kurz VP–Systeme — als leistungsfähige Hilfsmittel für den Umgang mit dem ständig wichtiger werdenden Gut Information.

Die von den benötigten Systemen geforderten Eigenschaften haben zunächst anwendungsbezogene, intuitiv verständliche Bedeutungen. Die Systeme sollen ihre Speicher– und Rechenfähigkeiten jeweils dort zur Verfügung stellen, wo sie benötigt werden, nämlich physisch/räumlich verteilt an den Arbeitsplätzen der Anwender. Sie sollen den Gegebenheiten und Anforderungen in den Institutionen, die auf den Umgang mit Information angewiesen sind, entsprechend gemeinsam und gleichzeitig von vielen Anwendern für viele Anwendungen nutzbar sein und kooperative Problemlösungen ermöglichen. Sie sollen leistungsfähige Hilfsmittel sein, die dabei helfen, Verfahren für Problemlösungen effizient zu entwickeln, und Berechnungen von Lösungen effizient ausführen. Dazu gehört, daß Anwender ihre Aufgaben frei von technischen Details auf hohem Abstraktionsniveau mit angemessenen Konzepten bearbeiten und dabei auf ein reichhaltiges Diensteangebot zurückgreifen können; dazu gehört weiter, daß die Wirkungen eingesetzter Konzepte und Dienste für Anwender verständlich sind und notwendige Berechnungen insbesondere durch Nutzung von Parallelität mit angemessen kurzen Antwortzeiten ausgeführt werden.

Der erreichte Entwicklungsstand der Hardwaretechnologien schafft gute Voraussetzungen für Realisierungen von VP–Systemen. Geeignet sind die verbreiteten Hardwarekonfigurationen mit vernetzten Stellen(rechnern): Die Stellen — Workstations und zunehmend auch PCs — liefern die hohen Speicher– und Rechenkapazitäten, die benötigt werden; für den Nachrichtenaustausch zwischen den Stellen stehen leistungsfähige Nachrichtenkommunikationssysteme zur Verfügung.

Der Bericht befaßt sich mit der Konstruktion von VP–Systemen nach einem **sprachbasierten Gesamtsystem–Ansatz**, was bedeutet, daß ein VP–System mit dem Konzeptevorrat einer Programmiersprache als Programm konstruiert wird und dann die Eigenschaften hat, die sich mit der Ausführung dieses Programms ergeben.

Die Wahl eines sprachbasierten Ansatzes entspricht in erster Linie einer Entscheidung über das **Abstraktionsniveau**, auf dem die Eigenschaften eines VP–Systems festgelegt, beschrieben und benutzt werden sollen, nämlich das Niveau für algorithmische Anwendungsproblemlösungen. Verfahren für die Lösung von Anwendungsproblemen sollen in der gewählten Sprache frei von technischen Details und frei von realisierungsbedingten Beschränkungen formuliert werden; Dienste, die zur Verfügung stehen, sollen entsprechend definiert werden und benutzbar sein; Berechnungen von Problemlösungen sollen auf diesem Niveau ausgeführt werden und beobachtbar sein. Der sprachbasierte Ansatz hat für ein entsprechend konstruiertes System **konzeptionelle Homogenität** zur Folge: Die Eigenschaften, die das System zur Verfügung stellt oder konstruktiv erhält, basieren auf einem festgelegten Konzeptevorrat, der freie Kombinierbarkeit der Eigenschaften und damit ein hohes Maß an Flexibilität ermöglicht.

Der durch konzeptionelle Homogenität geschaffene Rahmen für Kombinierbarkeit und Flexibilität wird eingeschränkt durch **Abhängigkeiten** zwischen den Komponenten, aus denen ein System besteht, und durch die Konsequenzen, die sich aus diesen Abhängigkeiten für die Nutzungsmöglichkeiten der Komponenten und ihrer Eigenschaften ergeben. Als wichtige Forderung an die Konzepte der gewählten Sprache ergibt sich, differenziert abgestufte Abhängigkeiten zwischen den Komponenten eines Systems ausdrücken und gezielt konstruktiv festlegen zu können, also die jeweiligen Komponentenmengen des Systems den Abhängigkeiten entsprechend strukturieren zu können. Diese **Strukturierung** ist zugleich das Instrumentarium, das geeignet und notwendig dazu ist, ein großes System mit vielen Komponenten überschaubar, beherrschbar und damit nutzbar zu gestalten.

Auf dem gewählten Abstraktionsniveau liefern die Abhängigkeiten zwischen den (abstrakten) Komponenten eines Systems die Kriterien für **abstrakte Verteiltheit**. Dabei sind die Komponenten, die Berechnungen ausführen, von primärem Interesse. Ein Extremfall ist Unabhängigkeit; wenn er vorliegt, können die entsprechenden Berechnungen parallel ausgeführt werden. Daraus ergibt sich die Forderung nach **Separation**, was bedeutet, daß möglichst weitgehend unabhängige Komponenten konstruiert werden sollen. Am anderen Ende der Skala liegt der Extremfall determinierter Abhängigkeit zwischen Berechnungen und entsprechenden Komponenten, der für kooperative Problemlösungen nötig sein kann. Daraus ergibt sich die Forderung nach **Integration**, was bedeutet, daß kooperativ nutzbare Komponenten konstruiert werden sollen, soweit dies erforderlich ist. Für die Konstruktion eines Systems ergibt sich, daß diese beiden gegeneinander gerichteten Anforderungen gemeinsam zu erfüllen sind; es sind anwendungsspezifische Synthesen von Separation und Integration nötig, und diese charakterisieren die abstrakte Verteiltheit eines Systems.

Aus dem bisher Gesagten ergibt sich, daß einerseits Konzepte für die Konstruktion von Systemkomponenten, und zwar von speicherfähigen, passiven und von rechenfähigen, aktiven Komponenten, sowie andererseits Konzepte für die Strukturierung der jeweiligen Komponentenmengen eines Systems benötigt werden. Der sprachbasierte **Gesamtsystem–Ansatz** dient dazu, diese Anforderungen gemeinsam und konstruktiv zu erfüllen: Ein VP–System wird konstruiert, indem mit dem festgelegten Konzeptevorrat ein Programm konstruiert und dann ausgeführt wird. Die Eigenschaften des Systems ergeben sich mit der Dynamik der Berechnungen, die das System ausführt, wobei Komponenten erzeugt und aufgelöst werden. Für jede Komponente, die erzeugt wird, sind konzeptionell die individuellen Eigenschaften und die Abhängigkeiten von anderen Komponenten, also die strukturellen Eigenschaften, festgelegt. Eine Komponente wird in die Struktur des Systems eingeordnet erzeugt; als solche leistet sie ihren Beitrag zu den Eigenschaften des Systems bis zu ihrer Auflösung. Der Konzeptevorrat, von dem ausgegangen wird, legt die Regeln für das Verhalten des Systems mit seiner Dynamik im Großen und im Kleinen fest.

Das mit dem jetzt motivierten, sprachbasierten Gesamtsystem–Ansatz gewählte Abstraktionsniveau ordnet entsprechende Systeme in mittlere Schichten der gesamten für VP–Systeme relevanten **Abstraktionshierarchie** ein. In obere Schichten gehören Konzepte und Konstruktionen, mit denen von nichtalgorithmischen Anforderungsspezifikationen ausgehend algorithmische Anwendungsproblemlösungen entwickelt werden. In untere Schichten gehören Konzepte und Konstruktionen, mit denen Systeme, die mit den hier gewählten Konzepten entwickelt sind, auf Hardwarekonfigurationen mit vernetzten Stellen realisiert werden. In diesen unteren Schichten sind die Betriebssystemaufgaben zu lösen; benötigt werden anwendungsangepaßte und –anpaßbare verteilte Betriebssysteme, die das Potential, das die Hardwarekonfigurationen bieten, ausschöpfen. Auf entsprechende laufende Arbeiten und auf bereits realisierte Experimentalsysteme wird im Ausblick des vorliegenden Berichts kurz eingegangen. Die Arbeiten zeigen mit den vorliegenden Ergebnissen, daß der verfolgte Ansatz erfolgversprechend ist.

Der vorliegende Bericht erklärt die Konzepte, die für den verfolgten sprachbasierten Gesamtsystem–Ansatz festgelegt sind, die Einsatzmöglichkeiten dieser Konzepte und die Eigenschaften der Systeme, die sich mit ihnen konstruieren lassen. Die Konzepte legen Rahmeneigenschaften für die Komponenten und die strukturierten Komponentenmengen eines Systems fest. Diese Rahmen werden mit der imperativen Programmiersprache INSEL (INtegration and SEparation supporting Language) ausgefüllt. INSEL ist jedoch nicht Gegenstand dieses Berichts; die Sprache ist in einem weiteren Bericht [RW94] erklärt.

Kapitel 2

Konzepte

In diesem Kapitel werden Konzepte besprochen, welche für die Konstruktion von VP-Systemen benutzt werden sollen; diese Konzepte finden eine konkrete Ausprägung in der imperativen experimentellen Programmiersprache INSEL. Ergänzend zu dem vorliegenden Sprachbericht enthält der Syntaxbericht [RW94] eine ausführliche Darstellung der Syntax der Sprache INSEL zusammen mit vielen Beispielen, die die Nutzung der einzelnen Sprachkonzepte illustrieren.
Im vorliegenden Bericht wird zunächst angegeben, nach welchen Prinzipien und unter welchen Aspekten die Konzepte ausgewählt wurden. Die Sprachkonzepten umfassen zum einen Konzepte zur Konstruktion von Komponenten und zum anderen Strukturierungskonzepte, die Abhängigkeiten zwischen den konstruierten Komponenten erfassen und beschreiben. Die Komponentenkonzepte werden im Abschnitt 2.2 vorgestellt. Zur Beschreibung der Strukturen von Systemen, die mit den eingeführten Konzepten konstruiert werden können, werden im letzten Abschnitt des Kapitels Struktur-Relationen eingeführt.

2.1 Grundlagen

In diesem Abschnitt werden einige für das Verständnis des Folgenden grundlegende Begriffe erklärt. Zudem werden die Prinzipien und Aspekte angegeben, die für die Auswahl der Konzepte für die Konstruktion von Systemen maßgeblich waren.

Ein **System** ist eine zusammengesetzte Einheit, für die „innen“ und „außen“ definiert ist. Ein System ist aus Komponenten zusammengesetzt. Eine Komponente ist ihrerseits eine Einheit, für die „innen“ und „außen“ definiert ist und die aus Komponenten zusammengesetzt sein kann. Was außen zu einem System ist, wird **Umwelt** des Systems genannt. Die Umwelt eines Systems gehört nicht zu diesem; es sind jedoch Wechselwirkungen zwischen einem System und seiner Umwelt möglich. Ein System ist **offen**, wenn Wechselwirkungen zwischen ihm und seiner Umwelt möglich sind; sonst ist ein System abgeschlossen. Ein System ist **dynamisch**, wenn sich seine Eigenschaften mit der Zeit verändern können; sonst ist ein System statisch. In einem dynamischen System können sich die Eigenschaften seiner Komponenten und die Menge der Komponenten, aus denen es zusammengesetzt ist, mit der Zeit verändern.

Ein **Rechensystem** ist ein offenes dynamisches System mit Fähigkeiten zur Speicherung und zur Verarbeitung von Information. Die Speicherfähigkeit eines Systems ergibt sich aus

der Speicherfähigkeit seiner Komponenten. Die Komponenten des Systems können zudem zur Durchführung von Berechnungen fähig sein und bestimmen damit die Fähigkeit des Systems zur Verarbeitung von Information. Die Ausführung von Operationen durch rechenfähige Komponenten bewirken Veränderungen der Eigenschaften eines Rechensystems.
Die Komponenten eines Systems werden gemäß des **Objektprinzips** konstruiert. Es besagt, daß die Eigenschaften eines Objekts durch Operationen zu definieren sind, daß die inneren und die äußeren Eigenschaften eines Objekts unterschieden werden, und daß ein Objekt von außen allein durch die Ausführung seiner äußeren Operationen benutzbar ist. Die hier betrachteten Systeme sind **Objekt-basiert** [Weg87]; das System und seine Komponenten haben die Eigenschaften von Objekten, d.h. Komponenten und ihre äußeren Operationen sind mit Objekten und deren Methoden in Objekt-orientierten Ansätzen vergleichbar. Der verfolgte Ansatz ist ein Objekt-basierter und kein Objekt-orientierter, weil in dem Konzepterepertoire kein Vererbungskonzept vorhanden ist, das es erlaubt, Klassen als Unterklassen von bereits definierten Klassen festzulegen. Klassenverfeinerungen sind nur durch Subtyp-Bildung möglich.
Wegen der Objekteigenschaften gilt für eine Komponente:

- ihre inneren und äußeren Eigenschaften werden durch innere und äußere Operationen definiert; bei einfachen Komponenten, wie z.B. Feldern oder Records, wird zwischen diesen Eigenschaften nicht unterschieden;
- die äußeren Operationen definieren die Schnittstelle der Komponente zu ihrer Umwelt. Die einzige Möglichkeit der Umwelt zur Benutzung der Komponente besteht in der Ausführung einer der äußeren Operationen der Komponente;
- die Ausführung einer äußeren Operation auf einer Komponente bewirkt insbesondere den (kontrollierten) Übergang von außen nach innen zur Komponente;
- die Beendigung der Ausführung einer der äußeren Operationen der Komponente bewirkt den kontrollierten Übergang von innen nach außen zurück zur aufrufenden Komponente.

Die Eigenschaften von Systemen und Komponenten, die durch das Objektprinzip präzisiert werden, werden durch das **Schachtelungsprinzip** verschärft. Es besagt, daß für jede Komponente eines Systems deren Einordnung innen bzw. außen zu anderen Komponenten des Systems für die Dauer ihrer Existenz festgelegt ist. Die hier betrachteten Systeme entsprechen dem Schachtelungsprinzip. Die konsequente Anwendung des Schachtelungsprinzips hat zur Folge, daß für jede Komponente eines Systems für die Dauer ihrer Existenz die Komponenten festgelegt sind, zu denen sie innen ist. Die konsequente Anwendung des Schachtelungsprinzips hat weiter zur Folge, daß ein System anfangs aus einer Komponente besteht, von der ausgehend weitere Komponenten erzeugt werden, die jeweils innen zu den bereits existierenden sind.

Das **Prinzip der Klassenbildung** besagt, daß die Komponenten, aus denen ein System zusammengesetzt ist, als Instanzen von Komponentenklassen erzeugt werden. Das Prinzip der Klassenbildung ordnet die Vorgehensweise bei der Festlegung von Komponenteneigenschaften. Zunächst werden mit der Klassendefinition die Klasseneigenschaften festgelegt. Bei der Erzeugung einer Instanz der Klasse werden dann spezifische Eigenschaften der Instanz festgelegt. Die Systeme, die wir betrachten, entsprechen dem Prinzip der Klassenbildung. Für die Anwendung dieses Prinzips ist es erforderlich, daß es in einem System Komponenten gibt, mit denen Komponentenklassen definiert werden. Das Prinzip der Klassenbildung zusammen mit dem Prinzip der Schachtelung führt dazu, daß über die Einordnung von Komponenten in die in Abschnitt 2.3 einzuführenden System-Strukturen bereits bei ihrer Klassendefinition entschieden werden kann.

Ein System, das den oben genannten Prinzipien entsprechend konstruiert wird, besitzt einige charakteristische Merkmale. Aus dem Objektprinzip ergibt sich, daß die zur Durchführung der von einem System auszuführenden Berechnung notwendige Kooperation der Komponenten des Systems durch Ausführen äußerer Operationen erreicht wird. Die Eigenschaften einer Komponente ergeben sich gemäß dem Schachtelungsprinzip aus den Eigenschaften der Komponenten, aus denen sie zusammengesetzt ist. Dem Prinzip der Klassenbildung entsprechend werden die grundlegende Komponenteneigenschaften durch die Eigenschaften der Komponentenklassen festgelegt.

Aus dem Objektprinzip und dem Prinzip der Klassenbildung folgt weiter, daß Komponenten als Instanzen von Komponentenklassen erzeugt werden. Ein wesentlicher Unterschied zwischen den hier vorzustellenden Sprachkonzepten und den Konzepten bekannter Objektorientierter Sprachen besteht darin, daß die Menge der Komponenten, aus denen ein System besteht, konzeptionell strukturiert ist. Die vielfältigen Abhängigkeiten zwischen Komponenten können damit differenziert erfaßt werden. Die Möglichkeit, Komponenten beliebig tief zu schachteln, liefert die Basis für die Festlegung unterschiedlicher Abhängigkeiten. Demgegenüber ist eine flache Struktur basierend auf Benutzungsabhängigkeiten zwischen Objekten für Objekt-orientierte Sprachen charakteristisch. Über die Nutzungs- sowie Vererbungsabhängigkeiten hinausgehende Abhängigkeiten lassen sich auf Grund der flachen Struktur nicht angeben. Zur Beschreibung der Strukturen werden in Abschnitt 2.3 entsprechende Relationen eingeführt.

2.2 Komponentenarten

Nachdem im vorangehenden Abschnitt grundlegende Begriffe und Prinzipien erklärt wurden, sollen im folgenden Konzepte für die Konstruktion von Systemen besprochen werden. Wegen des Zusammenhangs zwischen einem System und seinen Komponenten werden Konzepte für die Konstruktion von Komponenten benötigt. Den einzelnen Konzepten entsprechen Komponentenarten. Jedes Konzept legt Eigenschaften für die Komponenten seiner Art, also für die Komponenten, die in Anwendung des Konzepts konstruiert sind, fest.

Von den Komponenten eines Systems ist verlangt, daß ihre inneren und äußeren Eigenschaften definiert werden. Diese Eigenschaften, also die entsprechenden Operationen, können explizit definiert oder als implizit definiert (vordefiniert) angenommen werden. Im einfachsten Fall kann man festlegen, daß die inneren und die äußeren Eigenschaften einer Komponente gleich sind. Wesentlich sind Konzepte für Komponenten, deren innere und äußere Eigenschaften verschieden sind. Sie ermöglichen eine flexible Anpassung der zu konstruierenden Komponenten an die Anforderungen einer Problemlösung. Komponenten, für die konzeptionell zwischen inneren und äußeren Eigenschaften unterschieden wird, werden wesentliche oder **DA-Komponenten** (bestehend aus <u>D</u>eklarations- und <u>A</u>nweisungsteil) genannt. Komponenten, für welche dies nicht der Fall ist, heißen einfache oder **DE-Komponenten** (<u>E</u>infache Komponenten bestehend aus einem <u>D</u>eklarationsteil).

Die Menge der DE-Komponenten kann entsprechend ihrer Verwendung klassifiziert werden nach **Wert-orientierte Komponenten** und nach **Generatoren**. Wert-orientierte Komponenten, wie z.B. Variablen, stellen elementare Speicher dar. Mit einem Generator werden die Eigenschaften einer Komponentenklasse festgelegt. Generatoren für einfache Komponentenklassen sind vergleichbar mit Typen in konventionellen prozeduralen Programmiersprachen. Eine Übersicht über die DE-Komponentenarten ist in Abbildung 2.1 angegeben.

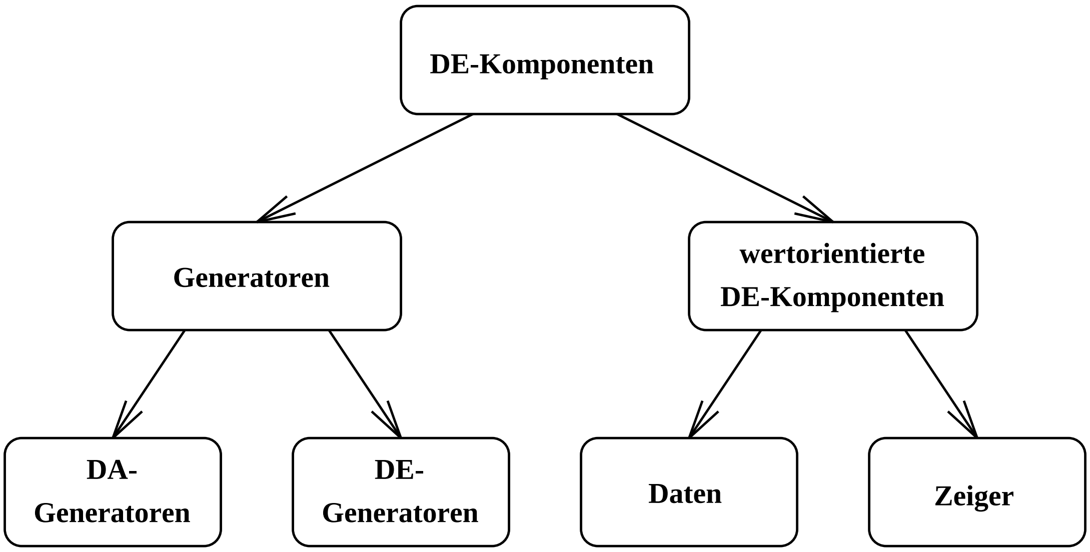


Abbildung 2.1: Klassifikation der DE-Komponenten

Für die Benutzung einer Komponente ist es erforderlich, daß die Komponente identifiziert werden kann. Bezüglich der Identifikation von Komponenten werden zwei Arten, nämlich **benannte** und **anonyme** Komponenten, unterschieden. Für benannte Komponenten, auch **N-Komponenten** genannt, ist für die Dauer ihrer Existenz ein Name festgelegt, der in einem jeweils definierten Kontext eindeutig ist. N-Komponenten werden als Instanzen von Komponentenklassen erzeugt, indem entsprechende Deklarationen erarbeitet werden. Eine anonyme Komponente, auch **Z-Komponente** genannt, wird erzeugt, indem eine Erzeuge-Operation auf dem entsprechenden Generator ausgeführt wird. Eine anonyme Komponente wird durch einen Zeigerwert identifiziert. Eine Komponente ist Z-Komponente für die Dauer ihrer Existenz. DE- und DA-Komponenten können N- oder Z-Komponenten sein.

Die Menge der von einer DA-Komponente benutzbaren Komponenten setzt sich dem oben Gesagten entsprechend aus benannten und anonymen Komponenten zusammen. Diese Menge heißt **Ausführungsumgebung** der DA-Komponente. Auf die Ausführungsumgebung einer Komponente wird in Kapitel 4.1 genauer eingegangen.

2.2.1 DA-Komponenten

Es gibt mehrere Arten von DA-Komponenten. Zunächst werden die allen DA-Komponenten gemeinsamen Eigenschaften angegeben.[1]

Jede DA-Komponente k hat folgende Eigenschaften:

- sie ist zusammengesetzt und fähig, Information zu speichern. Die Komponenten, aus denen sie sich zusammensetzt, können DA- und DE-Komponenten sein;

[1] Die Eigenschaften einer DA-Komponente werden in (3.29) präzisiert.

- sie besteht aus einem Deklarationsteil und einem Anweisungsteil;
- für sie ist implizit genau eine ausgezeichnete äußere Operation definiert; abhängig von der Komponentenart ist dies die Operation „führe_aus“, „initialisiere“ oder „starte“;
- für sie ist explizit eine ausgezeichnete innere Operation definiert, die als **kanonische Operation** von k mit $op(k)$ bezeichnet wird;
- sie befindet sich zu jedem Zeitpunkt ihrer Existenz in einem der **Zustände** V ($\equiv$ vorbereitet), A ($\equiv$ in Ausführung), R ($\equiv$ rechnend), W ($\equiv$ wartend) oder T ($\equiv$ terminiert). Ist k eine DA-Komponente, so wird ihr Zustand zum Zeitpunkt t mit $\mu_t(k)$ bezeichnet. Unmittelbar nach der Erzeugung gilt für jede DA-Komponente k: $\mu_t(k) = V$. Dies ist die Voraussetzung für die Ausführung der implizit definierten äußeren Operation auf k.

Eigenschaften einer DA-Komponente k:

- k setzt sich zusammen aus
  - den durch Deklarationen im Deklarationsteil von k definierten N-Komponenten $L_0(k)$ und
  - dem Anweisungsteil $L_a(k)$ der kanonischen Operation von k.
- der Komponente k können anonyme Komponenten zugeordnet sein[2].

Die Gesamtheit der Komponenten, aus denen k zusammengesetzt ist, und die k zugeordnet sind, wird mit $\widetilde{L}(k)$ bezeichnet. Mit $L(k) = L_0(k) \cup L_a(k)$ wird die Menge aller Komponenten bezeichnet, die der Komponente k statisch zugeordnet sind. Die Menge $L(k)$ ist Bestandteil der Menge $\widetilde{L}(k)$. Über die Zuordnung einer Komponente zu $\widetilde{L}(k)$ kann im Falle der benannten Komponenten zu jedem Zeitpunkt eindeutig entschieden werden. Die Ausführung der kanonischen Operation $op(k)$ besteht aus der Erarbeitung des Deklarationsteils und der Ausführung des Anweisungsteils[3]. Nach der Erarbeitung des Deklarationsteils sind die benannten Komponenten von k und damit $L_0(k)$ für die gesamte Lebenszeit von k festgelegt. $\widetilde{L}(k)$ kann darüberhinaus anonyme Komponenten enthalten, die bei der Erarbeitung des Deklarationsteils oder im Laufe der Ausführung des Anweisungsteils der Komponente k dynamisch erzeugt wurden, oder die von anderen Komponenten im Verlauf der Ausführung des Systems erzeugt wurden. Die Regeln für die Zuordnung von Komponenten ergeben sich aus den konzeptionell festgelegten Lebenszeiten von Komponenten und werden in Abschnitt 3.3 erklärt.

Die Ausführung der implizit auf k definierten äußeren Operation bewirkt von Details abgesehen

- den Übergang der Operationsausführung von außen nach innen zu k,
- die Ausführung der kanonischen Operation $op(k)$ und
- nach Beendigung der Ausführung der Operation den Übergang zurück von innen nach außen.

[2] Eine detaillierte Beschreibung der Regeln, nach denen anonyme Komponenten einer anderen Komponente zugeordnet werden, erfolgt in (3.29).

[3] Die Ausführungsphasen einer kanonischen Operation werden in 3.3 beschrieben.

DA-Komponenten werden dynamisch erzeugt und aufgelöst[4]. Sei t_1 der Zeitpunkt, zu dem die DA-Komponente k erzeugt und t_2 der Zeitpunkt, zu dem k aufgelöst wird. Dann ist das Zeitintervall $\Lambda(k) \triangleq [t_1, t_2]$ die **Lebenszeit** der DA-Komponente k.

Neben den angegebenen Eigenschaften der Speicherfähigkeit können DA-Komponenten weitere Eigenschaften besitzen. DA-Komponenten können zusätzlich Rechenfähigkeit haben, woraus sich eine Einteilung der DA-Komponenten in **aktive** und **passive** ergibt. Passive DA-Komponenten sind Order oder Depots und aktive DA-Komponenten sind Akteure. Weiterhin können für DA-Komponenten explizit äußere Operationen definiert sein, die neben der vordefinierten äußeren Operation weitere Möglichkeiten zum Übergang von außen nach innen und damit zur Nutzung der Komponente bieten.

Order:
Eine Order ist eine Operations-definierende Komponente. Sie ist mit einer Inkarnation eines Unterprogramms und mit einer Inkarnation eines Blocks in imperativen Programmiersprachen vergleichbar. Eine Order definiert im wesentlichen ihre kanonische Operation. Ihre lokalen N-Komponenten sind Hilfsmittel für diese. Für jede Order sind die **Zustände** V, A und T definiert, wobei der wesentliche Zustand A ist. Für jede Order ist implizit eine äußere Operation definiert, die mit „führe_aus“ bezeichnet wird. Für Order können keine äußeren Operationen explizit definiert werden. Alle Order sind anonym, also Z-Komponenten. Da für eine Order genau eine äußere Operation definiert ist und nach deren Ausführung keine weitere Nutzung von außen mehr möglich ist, müssen Ordern nicht über explizite Zeiger identifizierbar sein. Sei x eine Order. Die Ausführung von „führe_aus“ auf x bewirkt die Überführung von x vom Zustand V in den Zustand A und die Ausführung von $op(x)$. Die Ausführung von $op(x)$ wird mit der Überführung von x in den Zustand T abgeschlossen. Order werden in zwei Arten unterteilt – in S-Order und K-Order. **K-Order** sind Kommunikationsoperationen, die von aktiven Komponenten, den sogenannten K-Akteuren, definiert werden können. K-Order werden im Zusammenhang mit der Erklärung des Akteur-Konzepts erläutert. Eine **S-Order** kann eine **PS-Order**, eine **FS-Order** oder eine **BS-Order** mit den Eigenschaften einer Prozedur-, Funktions- bzw. Block-Inkarnation, die aus imperativen Sprachen bekannt sind, sein.

Depots:
Ein Depot ist eine Speicher-definierende Komponente. Ein Depot definiert mit seinen lokalen N-Komponenten Speicher und Operationen für deren Benutzung[5]. Für jedes Depot sind die **Zustände** V, A und T definiert, wobei der wesentliche Zustand T ist. Ein Depot ist eine passive Komponente, die von anderen Komponenten genutzt werden kann; deshalb ist der Zustand T sein wesentlicher Zustand. Für jedes Depot ist implizit eine äußere Operation definiert, die mit „initialisiere“ bezeichnet wird; weitere äußere Operationen – die **Zugriffsoperationen** des Depots – sind explizit zu definieren. Depots können N- oder Z-Komponenten sein. Sei x ein Depot. Die Ausführung von „initialisiere“ auf x bewirkt die Überführung von x vom Zustand V in den Zustand A und die Ausführung von $op(x)$. Die Ausführung der kanonischen Operation $op(x)$ wird mit der Überführung von x in den Zustand T abgeschlossen. Wie der Name der Operation es bereits suggeriert, dient die Operation „initialisiere“ dazu, die passive Komponente zu initialisieren. Ausführungen von Zugriffsoperationen auf x sind genau dann zulässig, wenn x im Zustand T existiert.
In weiterführenden Arbeiten ist das Depot-Konzept zu differnzieren, z.B. durch die

[4] Die Erzeugung und Auflösung von Komponenten wird in Kapitel 3 erklärt.
[5] Vgl. Package-Konstrukt in Ada [Ada83].

Einführung eines Monitor-Depot-Konzepts, so daß die Zugriffsoperationen eines Depots konzeptionell synchronisiert werden.

Akteure:
Ein Akteur ist eine aktive DA-Komponente und damit eine Komponente, die zusätzlich zur Speicherfähigkeit auch Rechenfähigkeit besitzt. Ein Akteur führt Berechnungen durch, indem er Operationen ausführt. Die Erzeugung[6] eines Akteurs erzeugt einen zusätzlichen Kontrollfluß parallel zum Kontrollfluß der erzeugenden Komponente. Akteure werden während der Erarbeitung der kanonischen Operation von DA-Komponenten durch Ausführen entsprechender Erzeuge-Operationen erzeugt und mit der Komponente, die ihre Lebenszeit[7] bestimmt, aufgelöst. Somit ermöglichen sie explizite Parallelverarbeitung mit dynamischem Parallelitätsgrad.
Für jeden Akteur sind die **Zustände** V, R ($\equiv$ rechnend), W ($\equiv$ wartend) und T definiert, wobei die wesentlichen Zustände R und W sind. Die für jeden Akteur implizit definierte äußere Operation „starte“ bewirkt bei ihrer Ausführung die Überführung des Akteurs x vom Zustand V in den Zustand R. Damit führt x seine kanonische Operation $op(x)$ aus. Im Verlauf der Ausführung von $op(x)$ können Akteure, Order oder Depots erzeugt werden. Nach der Erzeugung einer Order oder eines Depots durch einen Akteur führt dieser die kanonische Operation der erzeugten Komponente aus, d.h. diese Operationen werden durch den Akteur gemäß des Subordinationsprinzips sequentiell eingeordnet in die kanonische Operation des Akteurs ausgeführt.
Die Komponente, deren kanonische Operation zum Zeitpunkt t durch den Akteur x ausgeführt wird, wird **Ausführungskomponente** von x genannt und mit $\varphi_t(x)$ bezeichnet. Zu Beginn der Ausführung der Operation „starte“ zum Zeitpunkt t_1 gilt $\varphi_{t_1}(x) = x$, d.h. der Akteur x führt seine eigene kanonische Operation aus. Erzeugt x zum Zeitpunkt $t_2 > t_1$ eine Order bzw. ein Depot k, so wechselt die Ausführungskomponente, es gilt $\varphi_{t_2}(x) = k$. Wird die Komponente k nach der Ausführung ihrer kanonischen Operation zum Zeitpunkt $t_3 > t_2$ in den Zustand T überführt, so gilt $\varphi_{t_3}(x) = x$. Die Erzeugung weiterer Order bzw. Depots während der Ausführung der kanonischen Operation einer Order bzw. eines Depots bewirkt einen entsprechenden Wechsel der Ausführungskomponente. Diese Vorgehensweise ist der in imperativen Programmiersprachen üblichen Wechsel der Ausführungsumgebungen bei Unterprogrammaufrufen vergleichbar.

Zur Konstruktion aktiver Komponenten werden zwei Akteur-Konzepte festgelegt. Mit dem Konzept des **M-Akteurs** werden aktive Komponenten festgelegt, die außer der vordefinierten Operation „starte“ keine anderen äußeren Operationen besitzen können. M-Akteure sind <u>m</u>ono-operational, d.h. sie besitzen genau eine äußere Operation und führen bei deren Ausführung ihre kanonische Operation ohne unmittelbare Beeinflussung durch andere Akteure aus. M-Akteure sind anonyme Komponenten. Analog zu den Ordern ist auch ein M-Akteur nicht explizit über Zeiger identifizier- und von außen nutzbar, weil eine Nutzung von außen nach der Erzeugung und dem darauffolgenden Aufruf der Operation „starte“ nicht möglich ist. M-Akteure werden durch M-Akteur-Aufrufe in den Anweisungsteilen von DA-Komponenten erzeugt.

Mit dem Konzept des **K-Akteurs** werden aktive Komponenten festgelegt, für die explizit äußere Operationen, sogenannte **Kommunikationsoperationen**, definiert werden können. Ein K-Akteur kann N- oder Z-Komponente sein. Wenn x ein K-Akteur ist, dann kann ein

[6] Siehe Abschnitt 2.2.4 zur Erzeugung von DA-Komponenten.
[7] Die Lebenszeitabhängigkeit von Komponenten wird in Abschnitt 2.3.5 geklärt.

anderer Akteur y mittels der für x explizit definierten Operationen höchstens dann mit x kommunizieren, wenn x in einem der Zustände R oder W existiert und x in der Ausführungsumgebung von y liegt. Die Kommunikation erfolgt nach dem Operationen-orientierten Rendezvous-Konzept. Die Kommunikationsoperationen eines K-Akteurs, die von anderen Komponenten aufgerufen werden, werden von dem anbietenden K-Akteur mittels spezieller Anweisungen zur Ausführung angenommen und als **K-Order** eingebettet in die Ausführung der kanonischen Operation des K-Akteurs ausgeführt. Über die Ausführung seiner Kommunikationsoperationen ist ein K-Akteur während der Ausführung seiner kanonischen Operation von außen beeinflußbar. Ein K-Akteur ist vergleichbar mit einer Task in der Programmiersprache Ada [Ada83]. Das Operationen-orientierte Rendezvous-Konzept unterscheidet sich jedoch wesentlich von dem Rendezvous-Konzept der Ada-Tasks. Das Operationen-orientierte Rendezvous-Konzept wird detailliert in Abschnitt 3.2.5 erklärt.

Die Klassifikation der Menge der DA-Komponenten ist in Abbildung 2.2 zusammengefaßt.

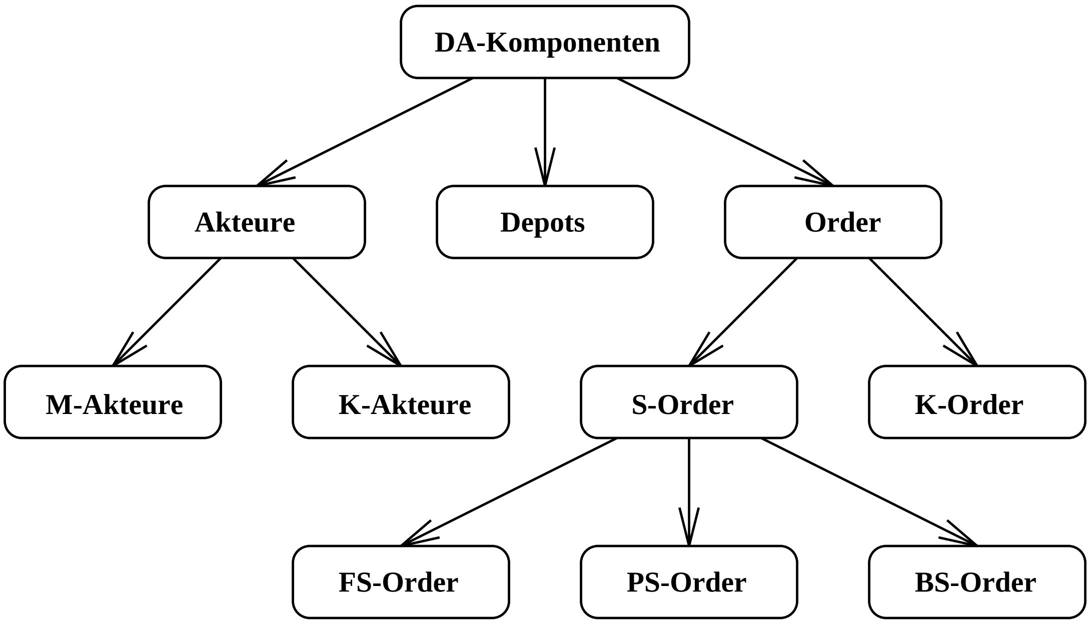


Abbildung 2.2: Klassifikation der DA-Komponenten

Das bisher Erklärte gibt eine Übersicht über die Arten von DA-Komponenten, aus denen ein System besteht. Für Depots und K-Akteure gilt, daß diese mit äußeren Operationen, den Zugriffs- bzw. Kommunikationsoperationen, definiert werden können. Im folgenden soll die Definition von äußeren Operationen näher erläutert werden.

Äußere Operationen einer DA-Komponente x werden explizit folgendermaßen festgelegt:

- Die Elemente von $L_0(x)$ können mit dem **Attribut** E (Export) definiert werden.
- Ist $y \in L_0(x)$ mit dem Attribut E definiert, dann sind alle für y definierten äußeren Operationen auch äußere Operationen von x.

Wenn y mit E definiert ist, dann sind alle implizit und explizit definierten äußeren Operationen von y auch äußere Operationen von x. Wird eine Komponente mit dem Attribut E exportiert, so ist es nicht möglich, für die einzelnen äußeren Operationen der Komponente differenziert festzulegen, ob sie exportiert werden sollen oder nicht, da konzeptionell alle

äußeren Operationen exportiert werden.
Die Gesamtheit der äußeren Operationen von x ergibt sich entsprechend aus den äußeren Operationen aller y, die als Elemente von $L_0(x)$ mit E definiert sind.

Dem Prinzip der Klassenbildung entsprechend werden Komponenteneigenschaften festgelegt, indem entsprechende Generatoren definiert werden. Die äußeren Operationen von DA-Komponenten sind eine Eigenschaft der Komponentenklasse, der sie angehören. Folglich wird die Entscheidung, ob eine lokale N-Komponente mit dem Attribut E definiert wird, bei der Definition des entsprechenden Generators getroffen.

Für die Festlegung von äußeren Operationen gelten folgende Beschränkungen:

(2.1)

- Für **Order und M-Akteure** können keine äußeren Operationen explizit definiert werden.
- Für **Depots** gilt, daß alle Arten von Komponenten, die als lokale N-Komponenten zugelassen sind, mit dem Attribut E definiert werden können.
- Für **K-Akteure** gilt, daß die K-Order-Generatoren konzeptionell mit dem Attribut E definiert sind. Weitere mit E attributierte Komponenten sind für K-Akteure nicht zugelassen.

2.2.2 Wert–orientierte Komponenten

Wert-orientierte Komponenten können skalar oder strukturiert sein. Sie stellen elementare Speicher dar und entsprechen den Konstanten oder Variablen in bekannten imperativen Programmiersprachen.

Eine **skalare Wert-orientierte Komponente** ist ein Datum oder ein Zeiger. Sie besitzt Speicherfähigkeiten, die durch ihre Wertmenge festgelegt sind. Für jede skalare Wert-orientierte Komponente sind die Operationen zum Lesen des Wertes, zum Schreiben eines Elements der Wertmenge und zu Wertvergleichen vordefiniert.

Daten : Für ein Datum ist die Wertmenge mit der Definition des Generators explizit zu definieren.

Zeiger : Zeiger dienen zur Identifikation anonymer Komponenten, d.h sie werden für den Zugriff auf anonyme Komponenten benötigt. Jeder Zeiger ist mit einer Komponentenklasse qualifiziert. Die Wertmenge eines Zeigers ist implizit definiert. Der Wert eines Zeigers identifiziert eine anonyme Komponente und der Zeiger verweist damit auf diese Komponente.

Eine **strukturierte Wert-orientierte Komponente** ist ein Feld oder ein Record. Sie wird nach den in Programmiersprachen üblichen Regeln definiert.

2.2.3 Generatoren

Generatoren sind Komponenten, die Komponentenklassen definieren. Generatoren sind die einzigen Komponenten eines Systems, die per se, d.h. durch Erarbeitung ihrer Definition, erzeugt werden können. Auf jedem Generator ist eine äußere Operation „erzeuge“ vordefiniert.

Ihre Ausführung bewirkt, daß eine Inkarnation als Element der durch den Generator definierten Klasse erzeugt wird. Das erzeugte Element kann eine Wert-orientierte-Komponente, eine DA-Komponente oder wiederum ein Generator sein. Letzteres besagt, daß ein Generator auch als eine Inkarnation bezüglich eines Generators erzeugt werden kann. Es handelt sich dann um einen Generator zweiter Ordnung, auf den in 2.2.3.2 eingegangen wird. Alle übrigen Komponenten werden erzeugt, indem die Operation „erzeuge“ auf einem Generator, der bereits als Komponente existiert, ausgeführt wird.

2.2.3.1 Generatoren erster Ordnung

Ein Generator, der eine Klasse von Komponenten definiert, die keine Generatoren sind, heißt **Generator erster Ordnung**. Generatoren erster Ordnung sind vergleichbar mit Typ- bzw. Klassenvereinbarungen bekannter imperativer bzw. Objekt-orientierter Programmiersprachen[8]. Generatorarten werden den Komponentenarten, die mit ihnen definiert werden, entsprechend bezeichnet. Ein Generator erster Ordnung ist demnach ein Daten-, ein Zeiger- oder ein DA-Generator.

Ein **Daten-Generator** definiert eine Klasse von Daten mit einer für alle Elemente der Klasse festgelegten Wertmenge. Die Wertmenge wird mit dem entsprechenden Generator explizit definiert. In INSEL gibt es mehrere Unterarten von Daten-Generatoren, die in [RW94] ausführlich beschrieben werden.

Ein **DA-Generator** ist ein Akteur-, ein Order- oder ein Depot-Generator. Die verschiedenen Unterarten von DA-Generatoren und die Klassen von Komponenten, die mit ihnen definiert werden können, werden in Abschnitt 2.2.4 besprochen.

Ein **Zeiger-Generator** definiert eine Klasse von Zeigern. Die Komponentenklasse, auf deren Elemente die Zeiger-Inkarnationen bezüglich des Generators zeigen können, wird bei der Definition des Zeiger-Generators unter Bezugnahme auf die entsprechende Komponentenklasse festgelegt.

Für die Komponentenklasse, mit der ein Zeiger-Generator qualifiziert ist, gilt:

(2.2) Für die **Qualifikation eines Zeiger-Generators** sind Daten-, Zeiger-, K-Akteur- und Depot-Generatoren zugelassen.

Ein Zeiger-Generator, der mit einem K-Akteur- oder mit einem Depot-Generator qualifiziert ist, heißt **DA-Zeiger-Generator**. Ein Zeiger, der Element einer durch einen DA-Zeiger-Generator definierten Klasse ist, ist ein DA-Zeiger.

2.2.3.2 Generatoren zweiter Ordnung

Ein Generator, der eine Klasse von Generatoren definiert, heißt **Generator zweiter Ordnung**. Generatoren zweiter Ordnung stellen eine weitere Abstraktionsstufe für die Festlegung von Klasseneigenschaften dar und erlauben das Zusammenfassen gemeinsamer Eigenschaften von Generatoren erster Ordnung.

[8] vgl. Typvereinbarungen in Ada [Ada83] bzw. Klassendefinitionen in C++ [Str91].

Jeder Generator zweiter Ordnung ist als lokale N-Komponente einer DA-Komponente durch eine entsprechende Deklaration zu definieren. Die Ausführung von „erzeuge“ auf einem Generator zweiter Ordnung bewirkt, daß eine Inkarnation als Element der durch den Generator definierten Klasse erzeugt wird. Diese Inkarnation ist ein Generator erster Ordnung.

Die folgende Regel gibt an, welche Klassen von Komponenten durch Generatoren zweiter Ordnung definiert werden können.

(2.3) Generatoren zweiter Ordnung sind zur Definition von K-Akteur- und von Depot-Generatoren zugelassen.

2.2.4 Parametrisierung

Nach der Erklärung der Generator-Arten werden nun die Sprachkonzepte zur Parametrisierung von DA-Generatoren vorgestellt. Dabei wird festgelegt, wie Generatoren definiert und Inkarnationen erzeugt werden können.

Generatoren werden parametrisiert, indem sie mit formalen Parametern definiert werden. Bei der Erzeugung von Inkarnationen bzgl. parametrisierter Generatoren wird jedem formalen Parameter ein aktueller Parameter assoziiert.

Generatoren erster Ordnung werden parametrisiert, indem sie mit **Parametern erster Ordnung** definiert werden. Alle DA-Generatoren mit Ausnahme der BS-Order-Generatoren können parametrisiert werden. Bei der Definition ist für jeden formalen Parameter der Name, der Generator-Name und der Übergabe-Modus festzulegen.

Zunächst wird angegeben, welche Arten von Komponenten als Parameter für Generatoren erster Ordnung möglich sind.

(2.4) Als **Parameter für M-Akteure, S-Order und K-Order** sind die Wert-orientierten DE-Komponenten zugelassen.

(2.5) Als **Parameter erster Ordnung** sind für **K-Akteure und Depots** Daten und DA-Zeiger zugelassen.

Neben der Festlegung des Namens des Generators muß für jeden Parameter ein Übergabe-Modus angegeben werden. Es sind Eingabe-, Ausgabe- und Ein-Ausgabeparameter zugelassen.

(2.6) Der Übergabe-Modus für **Eingabeparameter** ist **call by value** .

Für eine DA-Komponente, die Inkarnation bzgl. eines Generators erster Ordnung mit dem formalen Eingabeparameter x ist, ist x eine Konstante mit dem Wert des aktuellen Parameters der Inkarnation.

(2.7) Der Übergabe-Modus für **Ausgabeparameter** ist **call by result**.

Für eine DA-Komponente, die Inkarnation bzgl. eines Generators erster Ordnung mit dem formalen Ausgabeparameter x ist, ist x eine Variable. Der Anfangswert von x ist undefiniert. Bei Abschluß der Ausführung der kanonischen Operation der DA-Komponente wird der aktuelle Wert von x dem entsprechenden aktuellen Parameter der Inkarnation zugewiesen. [9]

(2.8) Der Übergabe-Modus für **Ein-Ausgabeparameter** ist **call by value-result**.

Für eine DA-Komponente, die Inkarnation bzgl. eines Generators erster Ordnung mit dem formalen Ein-Ausgabeparameter x ist, ist x eine Variable. Der Anfangswert von x ist der des aktuellen Parameters der Inkarnation. Bei Abschluß der Ausführung der kanonischen Operation der DA-Komponente wird der aktuelle Wert von x dem entsprechenden aktuellen Parameter der Inkarnation zugewiesen.

Im folgenden werden die zugelassenen Parameter-Übergabe-Modi für die hier betrachteten DA-Komponenten festgelegt.

(2.9) Für **M-Akteure**, **PS-Order** und **K-Order** sind Eingabe-, Ausgabe- und Ein-Ausgabeparameter zugelassen.

(2.10) Für **FS-Order** sind nur Eingabeparameter zugelassen.

Jede FS-Order liefert einen **Ergebnis-Wert**. Durch die Festlegung eines Generators werden die Eigenschaften der Ergebnis-Komponente festgelegt. Für die Definition der kanonischen Operation einer FS-Order wird festgelegt, daß deren dynamisch letzte Operation durch eine **Return-Anweisung mit Ergebnis-Ausdruck** definiert werden muß. Eine FS-Order liefert dann den erarbeiteten Wert des Ergebnis-Ausdrucks der Return-Anweisung, mit der die Ausführung der kanonischen Operation abschließt.

Analog zu den M-Akteuren und Order sind mit der Definition eines K-Akteur- oder Depot-Generators die formalen Parameter mit ihren Übergabe-Modi festzulegen. Für K-Akteure und Depots ist zu berücksichtigen, daß diese im wesentlichen durch ihre äußeren Operationen genutzt werden und dementsprechend auch Ergebnisse nur bei Ausführung ihrer äußeren Operationen erarbeiten sollen. Deshalb gilt:

(2.11) Für **K-Akteure** und **Depots** sind nur Eingabeparameter zugelassen.

Mit den oben angegebenen Regeln (2.4) bis (2.11) sind die Möglichkeiten zur Parametrisierung von DA-Generatoren mit Parametern erster Ordnung festgelegt. Sie sind in Tabelle 2.1 zusammengefaßt.

Als nächstes werden Sprachkonzepte zur **Definition von Generatoren erster Ordnung** und zur Erzeugung von Inkarnationen bzgl. Generatoren erster Ordnung vorgestellt.

Klassen der eingeführten Arten von DA-Komponenten werden durch **Art-spezifische Generatoren** definiert, nämlich durch

[9] Der genaue Zeitpunkt der Parameterübergabe wird in Abschnitt 3.3 festgelegt.

	Parameter erster Ordnung	
DA-Komponenten	Komponentenarten	Übergabe-Modi
M-Akteur	Daten Zeiger	Eingabe Ein-Ausgabe Ausgabe
K-Akteur	Daten DA-Zeiger	Eingabe
FS-Order	Daten Zeiger	Eingabe
PS-Order	Daten Zeiger	Eingabe Ein-Ausgabe Ausgabe
BS-Order	keine	—
K-Order	Daten Zeiger	Eingabe Ein-Ausgabe Ausgabe
Depot	Daten DA-Zeiger	Eingabe

Tabelle 2.1: Parametrisierung von DA-Generatoren mit Parametern erster Ordnung

- M-Akteur-Generatoren,
- S-Order-Generatoren und deren Unterarten,
- K-Order-Generatoren,
- Depot-Generatoren bzw.
- K-Akteur-Generatoren.

Alle Inkarnationen bzgl. eines dieser Generatoren sind DA-Komponenten der festgelegten Art. Alle Generatoren mit Ausnahme der BS-Order-Generatoren sind als lokale N-Komponenten von DA-Komponenten zu definieren. BS-Order-Generatoren können implizit als anonyme Komponenten oder als benannte Komponenten in Anweisungsteilen von DA-Komponenten definiert werden.

Die Sprachkonzepte, mit denen Inkarnationen bzgl. M-Akteur-, S-Order-, und K-Order-Generatoren erzeugt werden können, heißen **Aufrufe**. Alle Aufrufe sind Art-spezifisch für die Art der DA-Komponenten, die mit ihnen erzeugt werden. Demnach sind M-Akteur-Aufrufe und Order-Aufrufe zu unterscheiden. M-Akteur-, PS-Order-, BS-Order- und K-Order-Aufrufe sind **Anweisungen** und können daher nur in Anweisungsteilen verwendet werden. FS-Order-Aufrufe sind **Ausdrücke** und können daher in Deklarationen und in Anweisungen verwendet werden.

Eine Anweisung, die ein **BS-Order-Aufruf** ist, definiert einen BS-Order-Generator. Jede Ausführung der Anweisung bewirkt die Inkarnierung einer entsprechenden BS-Order und die Ausführung ihrer kanonischen Operation.

M-Akteur-, PS-Order-, FS-Order- und K-Order-Aufrufe nehmen Bezug auf Generatoren, die als N-Komponenten definiert sein müssen. Der Generator, auf den ein FS- bzw. K-Order-

Aufruf Bezug nimmt, muß Art-spezifisch, also ein FS-bzw. K-Order-Generator sein. Tabelle 2.2 gibt eine Übersicht über die besprochenen Sprachkonzepte.

Aufrufe		Generatoren				
		M-Akteur-	PS-Order-	FS-Order-	K-Order-	BS-Order-
M-Akteur-	Anweisung	×	—	—	—	—
PS-Order-	Anweisung	—	×	—	—	—
FS-Order-	Ausdruck	—	—	×	—	—
K-Order-	Anweisung	—	—	—	×	—
BS-Order-	Anweisung	—	—	—	—	×

Tabelle 2.2: Sprachkonzepte zur Definition von Komponentenklassen und zur Inkarnierung von M-Akteuren, S-Order und K-Order

Abschließend werden die Inkarnierungsmöglichkeiten bezüglich Depot- und K-Akteur-Generatoren erster Ordnung erklärt.

Benannte Depots bzw. benannte K-Akteure werden durch Erarbeitung ihrer Deklaration erzeugt. Dies ist dann der Fall, wenn die kanonische Operation einer DA-Komponente s ausgeführt wird, und im Deklarationsteil von s eine entsprechende Deklaration auftritt. Anonyme Depots bzw. anonyme K-Akteure werden durch Erarbeitung eines Generierungsausdrucks (siehe Abschnitt 3.2.2.2) erzeugt. Generierungsausdrücke können sowohl im Deklarations- als auch im Anweisungsteil einer DA-Komponente auftreten.

Generatoren zweiter Ordnung werden parametrisiert, indem sie mit formalen Parametern erster und zweiter Ordnung definiert werden. Parameter erster Ordnung entsprechen den Parametern für Generatoren erster Ordnung, die zugelassenen Komponentenarten und Übergabe-Modi ergeben sich deshalb aus den entsprechenden Regeln für Depot- und K-Akteur-Generatoren erster Ordnung. Die Regeln für Parameter zweiter Ordnung werden im folgenden angegeben.

(2.12) Als Parameter zweiter Ordnung sind

- für K-Akteur-Generatoren zweiter Ordnung Daten- und Zeiger-Generatoren sowie
- für Depot-Generatoren zweiter Ordnung Daten-, Zeiger-, FS-Order und PS-Order-Generatoren

zugelassen.

Die Parameter zweiter Ordnung, mit denen ein Generator zweiter Ordnung definiert werden kann, sind formale Generatorparameter der mit (2.12) festgelegten Arten. Bei der Erzeugung einer Generator-Inkarnation bezüglich des Generators zweiter Ordnung sind für die formalen Generatorparameter, wie bei anderen Parametern üblich, aktuelle Generatoren anzugeben. Die Parameter erster Ordnung, mit denen ein Generator zweiter Ordnung definiert werden kann, sind formale Eingabeparameter der durch (2.5) festgelegten Arten, deren Werte bei der Inkarnierung anzugeben sind.

(2.13) Der Übergabe-Modus für alle Parameter zweiter Ordnung ist **call by name**.

Abschliessend wird nun noch erklärt, aus welchen Komponenten sich die Ausführungsumgebung einer Komponente u zusammensetzt, die eine Inkarnation eines Generators erster Ordnung h ist, der seinerseits eine Inkarnation eines Generators zweiter Ordnung H ist. Sei y die DA-Komponente, die den Generator h als lokale Komponente enthält und sei x die DA-Komponente, die den Generator H als lokale Komponente enthält. Die Ausführungsumgebung einer Inkarnation u bzgl. h ergibt sich aus den lokalen N-Komponenten von u, der Ausführungsumgebung der DA-Komponente y, zu der der Generator erster Ordnung h lokale N-Komponente ist, und der Ausführungsumgebung der DA-Komponente x, zu der der Generator zweiter Ordnung H lokale N-Komponente ist.

2.3 Die System-Strukturen

In diesem Abschnitt werden die System-Strukturen, die unterschiedliche Abhängigkeiten zwischen den Komponenten eines Rechensystems zu einem betrachteten Zeitpunkt charakterisieren, vorgestellt. Durch die Ausführung von Operationen und die Erzeugung und Auflösung von Komponenten verändern sich die Menge der existierenden Komponenten und die Abhängigkeiten zwischen den Komponenten. Diese Veränderungen werden in Kapitel 3 beschrieben.

Die angegebenen Konzepte für DA-Komponenten und das Schachtelungsprinzip haben zur Folge, daß ein System, das mit diesen Konzepten konstruiert wird, im wesentlichen aus einer DA-Komponente besteht, und zwar einem M-Akteur. Dieser M-Akteur heißt **Hauptkomponente** des Systems. Bei der Ausführung der kanonischen Operation dieses M-Akteurs können DA-Komponenten erzeugt und aufgelöst werden. In jedem Zeitpunkt seiner Existenz besteht das System aus einer Menge von DA-Komponenten, die voneinander abhängig sind. Diese Abhängigkeiten werden durch die System-Strukturen erfaßt. Während der Lebenszeit eines Systems können Wechselwirkungen zwischen diesem und seiner Umwelt in Form von Eingaben an bzw. Ausgaben durch „Benutzer“ bestehen, die Systeme sind daher offen.

Ein System besteht aus seiner Hauptkomponente und einer Menge von weiteren DA-Komponenten, welche bei der Ausführung der kanonischen Operation der Hauptkomponente bzw. anderer Komponenten erzeugt wurden. Alle diese Komponenten sind Teil der Hauptkomponente. Damit ergibt sich, daß alle Eigenschaften, die ein System zu einem Zeitpunkt t seiner Existenz hat, Eigenschaften der DA-Komponenten sind, aus denen das System zum Zeitpunkt t besteht. Die Entwicklungsmöglichkeiten dieser DA-Komponenten legen die weiteren Entwicklungsmöglichkeiten des Systems fest. Daraus folgt einerseits, daß man die Eigenschaften eines Systems auf der Grundlage der Generatoren, bzgl. derer die DA-Komponenten des Systems inkarniert wurden, beschreiben kann. Andererseits ergibt sich die Notwendigkeit, die Komponenten-Menge eines Systems so zu ordnen, daß das System auch dann durchschaubar und nutzbar ist, wenn die Anzahl der Komponenten groß ist. Diese Ordnung wird durch Strukturierung erreicht.

Sei $\mathcal{S}$ ein System. $\mathcal{S}$ werde vom Beginn seiner Existenz zum Zeitpunkt $t = 0$ an betrachtet; die benutzte Zeitachse[10] sei $I\!R_+$. Zu jedem Zeitpunkt $t \in I\!R_+$ besteht $\mathcal{S}$ aus einer Menge X_t von DA-Komponenten. Wenn $|X_t| \geq 2$ gilt, dann sind die Elemente von X_t voneinander

[10] Im folgenden wird die Zeit als eine physikalische Zeit verwendet, die eine Linearisierung der für das System relevanten Ereignissen festlegt, die konsistent ist mit der für das System festgelegten partiellen Ordnung über der Menge dieser Ereignisse.

abhängig. Diese Abhängigkeiten ergeben sich aus den Eigenschaften der Komponentenkonzepte und aus der dynamischen Entwicklung des Systems durch Operationsausführungen. Die Eigenschaften der Elemente von X_t zusammen mit den Abhängigkeiten zwischen diesen ergeben die Eigenschaften von $\mathcal{S}$ zum Zeitpunkt t. Diese Abhängigkeiten werden durch Struktur-Relationen über X_t beschrieben; dazu werden fünf Struktur-Relationen eingeführt.

2.3.1 Die Definitions-Struktur

Die DA-Komponente, die einen Generator als lokale N-Komponente enthält, bestimmt die Menge der Komponenten, die von Inkarnationen bzgl. dieses Generators potentiell benutzbar ist. Diesen Zusammenhang zwischen einer DA-Komponente und der Komponente, die ihren Generator enthält, stellt die Struktur-Relation *unmittelbar-δ-innen* her. Die Relation *unmittelbar-δ-innen* ist eine zweistellige Relation auf der Menge der DA-Komponenten. Die Relation *δ-innen*, kurz δ, ist die transitive Hülle der Relation *unmittelbar-δ-innen*, die folgendermaßen definiert ist:

Definition 2.14.:

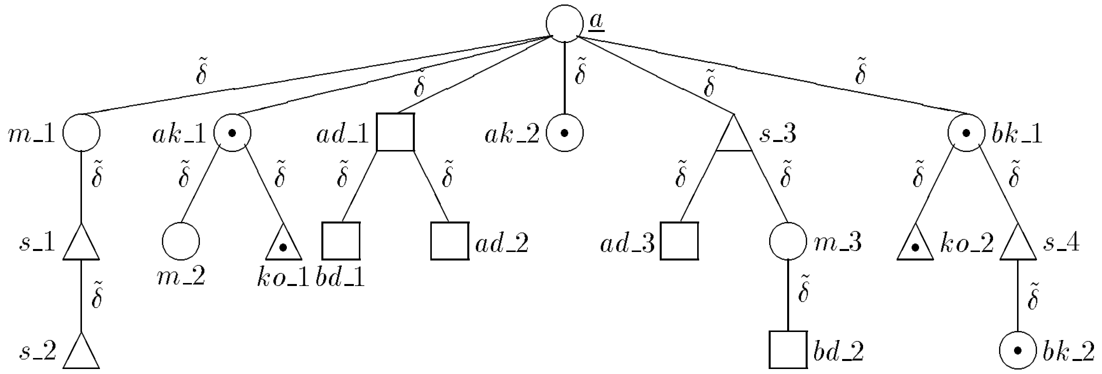
Seien $a, b \in X_t$. a ist *unmittelbar-δ-innen* zu b ($\tilde{\delta}_t$-innen) $-\ (a,b) \in \tilde{\delta}_t\ -$ $\overset{\triangle}{\iff}$ der Generator G, bzgl. dessen a Inkarnation ist, eine lokale N-Komponente von b ist, d.h. es gilt: $G \in L_0(b)$.
δ_t ist die transitive Hülle von $\tilde{\delta}_t$.

□

Die durch Definition (2.14) auf X_t definierte Relation δ_t-innen stellt den Zusammenhang zwischen den DA-Komponenten, den Generatoren und den DA-Komponenten, zu denen die Generatoren lokale N-Komponenten sind, her.
Die **Definitions-Struktur**, beschrieben durch das Paar (X_t, δ_t), beschreibt diesen Zusammenhang für ein System $\mathcal{S}$ zur Zeit t. Jede DA-Komponente ist für ihre gesamte Lebenszeit in die Definitions-Struktur eingeordnet, d.h. eine DA-Komponente wird mit ihrer Erzeugung gemäß Definition (2.14) in δ eingeordnet und mit ihrer Auflösung aus δ ausgeordnet. Durch die Erzeugung bzw. Auflösung von DA-Komponenten[11] verändert sich die Menge X_t über die Zeit. Die Relation δ_t beschreibt die zum betrachteten Zeitpunkt t zwischen den existierenden Komponenten bestehenden definitorischen Abhängigkeiten. Aus den in 2.2.3 eingeführten Regeln zur Inkarnierung von DA-Komponenten folgt, daß jede Komponente Inkarnation bzgl. genau eines Generators ist. Daher gibt es für jede DA-Komponente $a \in X_t$, mit Ausnahme der Hauptkomponente $\underline{a}$, genau eine DA-Komponente $b \in X_t$ mit $(a,b) \in \tilde{\delta}_t$. Interpretiert man X_t als Knoten- und $\tilde{\delta}_t$ als Kantenmenge eines Graphen, so ergibt sich, daß $(X_t, \tilde{\delta}_t)$ ein Baum mit der Hauptkomponente $\underline{a}$ als Wurzel ist. Abbildung 2.3 zeigt ein Beispiel für eine Definitions-Struktur.

Da die bei der Definition eines Generators sichtbaren[12] Komponenten im wesentlichen die Komponenten sind, die von Inkarnationen bzgl. des Generators benutzt werden können, ist die Definitions-Struktur die Grundlage für die Bestimmung der Ausführungsumgebung einer

[11] In Kapitel 3 wird die Erzeugung und Auflösung von DA-Komponenten detailliert beschrieben.
[12] Es werden die aus imperativen Programmiersprachen bekannten Sichtbarkeitsregeln zugrundegelegt.



Komponenten: ≡ Depot

≡ M–Akteur ≡ K–Akteur

≡ S–Order ≡ K–Order

Die Bezeichner der Komponenten ergeben sich aus der Komponenten-Art (m, k, ko, s bzw. d), einer nachgestellten Nummer und einem vorangestellten a bzw. b bei anonymen bzw. benannten K-Akteuren und Depots. $\underline{a}$ bezeichnet die Hauptkomponente des Systems.

Abbildung 2.3: Der Baum einer Definitions-Struktur

DA-Komponente. Die Ausführungsumgebung einer Komponente k ist die Menge von Komponenten, die potentiell bei Ausführung der kanonischen Operation von k benutzbar ist. Diese Menge ergibt sich nach festgelegten Regeln. Auf die Ausführungsumgebung wird in Kapitel 4.1 ausführlich eingegangen.

2.3.2 Die Ausführungs-Struktur

Ein mit den vorgestellten Konzepten konstruiertes System besteht aus einer Menge von parallelen Kontrollflüssen. Jeder Kontrollfluß beschreibt die Berechnung eines Akteurs und besteht seinerseit aus einer Folge von Anweisungen, die durch die kanonischen Operationen von Ordern, Depots oder des Akteurs definiert werden. Die **Ausführungs-Struktur** *α-innen* beschreibt die Abhängigkeiten zwischen den parallelen Kontrollflüssen sowie die Einbettung von sequentiellen Operationen in den Kontrollfluß. α ist die transitive Hülle der Relation *unmittelbar-α-innen*. Diese umfaßt parallele, sequentielle sowie Kommunikations-Einordnungen und setzt sich dementsprechend aus drei Relationen zusammen.

Sequentielle Bestandteile der Ausführungs-Struktur:
Die Relation *unmittelbar-σ-innen* beschreibt die sequentielle Einordnung von DA-Komponenten. Sie ist folgendermaßen definiert:

Definition 2.15.:

Seien $a, b \in X_t$; a sei S-Order, K-Order im Zustand A oder Depot im Zustand A.
a ist *unmittelbar-σ-innen* zu b ($\tilde{\sigma}_t$-innen) $- (a,b) \in \tilde{\sigma}_t -$ $\stackrel{\triangle}{\Longleftrightarrow}$ a eine S-Order ist und bei Ausführung der kanonischen Operation $op(b)$ erzeugt worden ist bzw. a eine K-Order des K-Akteurs b mit der kanonischen Operation $op(b)$ ist, und die kanonische Operation von a sequentiell eingeordnet in $op(b)$ ausgeführt wird.
σ_t ist die transitive Hülle von $\tilde{\sigma}_t$.

□

S-Order sind für ihre gesamte Lebenszeit in die σ-Relation eingeordnet. Bei einer K-Order fallen im Gegensatz zu einer S-Order das Erzeugen der Order und der Beginn der Ausführung der kanonischen Operation nicht zusammen. Dieser Unterschied ist durch die für K-Order charakteristische Rendezvous-Semantik begründet. Auch Depots sind nicht für ihre gesamte Lebenszeit in die σ-Relation eingeordnet, da sie nach der Ausführung ihrer kanonischen Operation in den Zustand T übergehen und damit keinen Beitrag mehr zur Ausführungs-Struktur leisten. Deshalb werden Depots beim Übergang in den Zustand T aus der σ-Relation ausgeordnet.

Parallele Bestandteile der Ausführungs-Struktur:
Zur Beschreibung der Einordnungen der parallelen Kontrollflüsse von Akteuren wird die Relation *unmittelbar-π-innen* definiert. Dabei ist zwischen M-Akteuren und K-Akteuren zu unterscheiden.

M-Akteure werden folgendermaßen eingeordnet:

Definition 2.16.:

Seien $m, b \in X_t$; m sei M-Akteur.
m ist *unmittelbar-π-innen* zu b ($\tilde{\pi}_t$-innen) $- (m,b) \in \tilde{\pi}_t -$ $\stackrel{\triangle}{\Longleftrightarrow}$ m bei Ausführung der kanonischen Operation $op(b)$ erzeugt worden ist und $op(m)$ parallel in $op(b)$ eingeordnet ausgeführt wird.
π_t ist die transitive Hülle von $\tilde{\pi}_t$.

□

K-Akteure werden im Unterschied zu M-Akteuren nicht immer π-innen zu der Komponente eingeordnet, in deren kanonischer Operation sie erzeugt wurden. Das liegt daran, daß K-Akteure über ihre Kommunikationsoperationen von außen nutzbar sind. Gegeben sei ein K-Akteur k, der bei der Ausführung der kanonischen Operation $op(d)$ eines Depots d erzeugt wurde. Das Depot kann erst über seine Zugriffsoperationen von außen genutzt werden, wenn es sich im Zustand T befindet. Das Depot kann aber erst dann in den Zustand terminiert übergehen, wenn es keine 'innere' Aktivität mehr besitzt, d.h. wenn alle Akteure, die bei Ausführung der kanonischen Operation des Depots erzeugt worden sind, bereits terminiert sind. Befindet sich das Depot im Zustand T, so kann der K-Akteur, wenn er analog zu Definition (2.16) π-innen zu d eingeordnet würde, nicht mehr von außen genutzt werden, da er sich für die Ausführung seiner Kommunikationsoperationen in einem der Zustände R oder W befinden muß.
Um zu erreichen, daß der K-Akteur auch dann von außen nutzbar ist, wenn sich das Depot im Zustand T befindet, muß er π-innen zu einer anderen DA-Komponente, die kein Depot ist,

eingeordnet werden. Die Grundlage für die entsprechende Einordnung bildet die nachfolgend definierte Abbildung η_t. Die Abbildung η_t ordnet jeder DA-Komponente einen Akteur oder eine Order zu. Zur Vorbereitung dieser Definition wird die Abbildung γ_t für anonyme K-Akteure und anonyme Depots definiert. γ_t ordnet jeder anonymen DA-Komponente x, die über einen DA-Zeiger v identifizierbar ist, die DA-Komponente zu, die den mit dem Generator von x qualifizierten Zeigergenerator für den Zeiger v definiert. Es können mehrere derart qualifizierte Zeigergeneratoren in einem System existieren.

Definition 2.17.:

Sei $x \in X_t$ ein anonymer K-Akteur oder ein anonymes Depot und seien X der Generator für x, V ein mit X qualifizierter DA-Zeiger-Generator und v eine Inkarnation bezüglich V. Der Zeigerwert, den der Generierungsausdruck, durch den x erzeugt wird, liefert, werde v zugewiesen. Dann ist $\gamma_t(x)$ die DA-Komponente, die den Zeiger-Generator V als lokale N-Komponente enthält, d.h. für die gilt: $V \in L_0(\gamma_t(x))$.

□

Definition 2.18.:

A_t sei die Menge der Akteure und O_t die Menge der Order von X_t.
Die linkstotale Abbildung $\eta_t : X_t \to A_t \cup O_t$ ist wie folgt definiert:

Sei $x \in X_t$.

- falls $x \in A_t \cup O_t$ ist, gilt: $\eta_t(x) = x$;
- falls x benanntes Depot im Zustand A ist, gilt: $\eta_t(x) = \eta_t(y)$, wobei y die DA-Komponente mit $(x, y) \in \tilde{\sigma}_t$ ist;
- falls x benanntes Depot im Zustand T ist, gilt: $\eta_t(x) = \eta_t(y)$, wobei y die DA-Komponente ist, die x als lokale N-Komponente enthält, $y \in L_0(y)$;
- falls x anonymes Depot ist, gilt: $\eta_t(x) = \eta_t(\gamma_t(x))$, wobei $\gamma_t(x)$ die mit (2.17) definierte Komponente ist.

□

Mit Hilfe der Abbildung η_t können nun die Regeln für die Einordnung von K-Akteuren in die Ausführungs-Struktur angegeben werden. Dabei ist zwischen benannten und anonymen K-Akteuren zu unterscheiden.

Benannte K-Akteure werden wie folgt eingeordnet:

Definition 2.19.:

Seien $k, b \in X_t$; k sei benannter K-Akteur. k ist *unmittelbar-π-innen* zu b – $(k, b) \in \tilde{\pi}_t$ – $\overset{\triangle}{\iff}$ $b = \eta_t(x)$ gilt, wobei x die DA-Komponente ist, in deren Deklarationsteil die entsprechende K-Akteur-Deklaration für k auftritt, d.h. die Komponente, zu der k lokale N-Komponente ist, $k \in L_0(x)$.

□

Bei anonymen K-Akteuren ist zusätzlich zu berücksichtigen, daß diese über entsprechende Zeiger von allen DA-Komponenten benutzt werden können, die Zeigerinkarnationen bzgl. des entsprechenden Zeigergenerators erzeugen können. Diesem Sachverhalt wird in der nachfolgend angegebenen Definition der π-Einordnung anonymer K-Akteure mit Hilfe der Abbildung γ_t Rechnung getragen.

Definition 2.20.:

Seien $k, b \in X_t$; k sei anonymer K-Akteur. k ist *unmittelbar-π-innen* zu b – $(k, b) \in \tilde{\pi}_t$ – $\overset{\triangle}{\Longleftrightarrow}$ $b = \eta_t(\gamma_t(k))$ gilt.

□

Mit (2.16) bis (2.20) ist die Einordnung von parallelen Kontrollflüssen von Akteuren erklärt.

Kommunikations-Bestandteile der Ausführungs-Struktur:
Mit dem Aufruf einer Kommunikationsoperation eines K-Akteurs k wird eine K-Order durch einen Auftraggeber erzeugt. Während der Ausführung der kanonischen Operation der K-Order durch den Auftragnehmer k, ist die K-Order σ-innen zu k eingeordnet. Um den Zusammenhang zwischen dem Auftraggeber und dem Auftragnehmer der Kommunikation gemäß des Rendezvous-Konzeptes zu erfassen, wird die Relation *unmittelbar-κ-innen* eingeführt.

Definition 2.21.:

Seien $c, x \in X_t$; c sei K-Order. c ist *unmittelbar-κ-innen* zu x ($\tilde{\kappa}_t$-innen) – $(c, x) \in \tilde{\kappa}_t$ – $\overset{\triangle}{\Longleftrightarrow}$ es einen Akteur $a \in X_t$ gibt, der Auftraggeber für c ist und dessen Ausführungskomponente x ist, d.h. $\varphi_t(a) = x$.

□

Mit Hilfe der beiden Relationen unmittelbar-κ-innen und unmittelbar-σ-innen ist es möglich, die für das Operationen-orientierte Rendezvous typische Auftraggeber-Auftragnehmer-Beziehung sowohl kausal als auch zeitlich zu beschreiben: die Relation unmittelbar-κ-innen stellt dabei den Bezug zum Auftraggeber her. Durch die Einordnung in die Relation unmittelbar-σ-innen wird die Ausführung der Kommunikationsoperation durch den Auftragnehmer angezeigt.

Mit den durch (2.15) – (2.21) definierten Relationen wird nun die Relation *unmittelbar-α-innen* definiert.

Definition 2.22.:

Seien $x, y \in X_t$. x ist *unmittelbar-α-innen* zu y – $(x, y) \in \tilde{\alpha}_t$ – $\overset{\triangle}{\Longleftrightarrow}$
$(x, y) \in \tilde{\sigma}_t \;\vee\; (x, y) \in \tilde{\pi}_t \;\vee\; (x, y) \in \tilde{\kappa}_t$.
α_t ist die transitive Hülle von $\tilde{\alpha}_t$.

□

Die **Ausführungs-Struktur** des Systems wird in jedem Zeitpunkt t durch das Paar (X_t, α_t) beschrieben. $\mathcal{S}$ zum Zeitpunkt t. Interpretiert man wie bei der Definitions-Struktur die Elemente von X_t als Knoten und die Elemente von $\tilde{\alpha}_t$ als Kanten eines Graphen, so erhält man

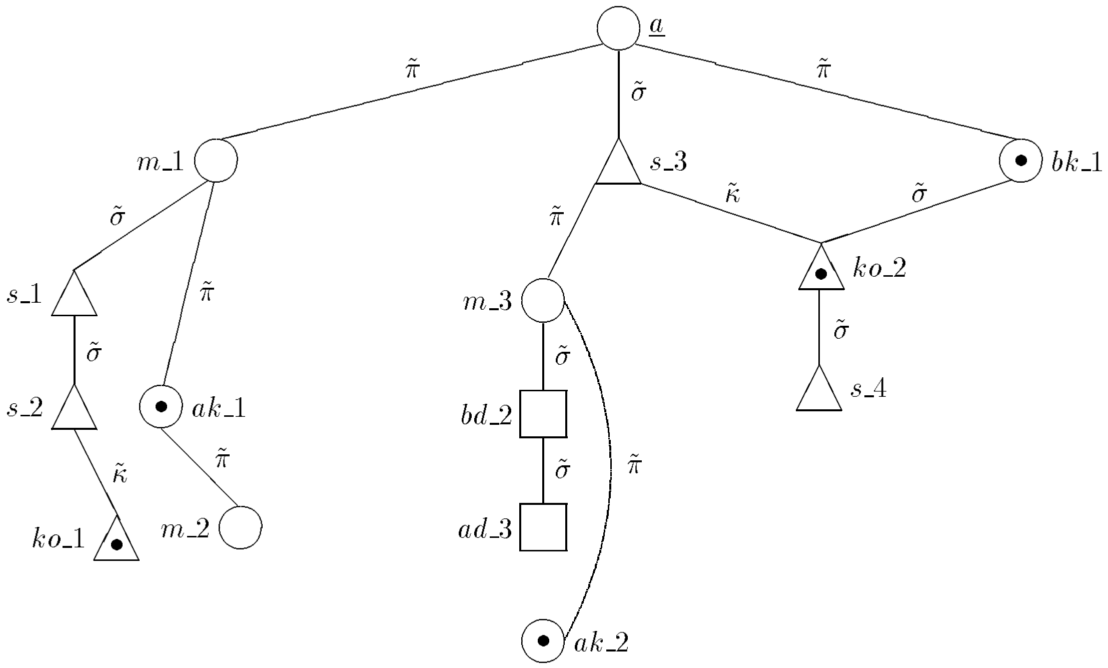


Abbildung 2.4: Beispiel für eine Ausführungs-Struktur

eine graphische Darstellung der Ausführungs-Struktur. Der sich ergebende Graph ist im allgemeinen kein Baum, da K-Order im Zustand A nach (2.15) bzw. (2.21) sowohl unmittelbar-σ-innen als auch unmittelbar-κ-innen eingeordnet sind. Sieht man jedoch von K-Ordern im Zustand A sowie der Hauptkomponente $\underline{a}$ des Systems $\mathcal{S}$ ab, gibt es zu jeder DA-Komponente $x \in X_t$ genau eine DA-Komponente $y \in X_t$ mit $(x, y) \in \tilde{\alpha}_t$.

Beispiel:
Die Abbildung 2.4 zeigt ein Beispiel für eine allgemeine Ausführungs-Struktur (X_t, α_t); dieser liegt die Definitions-Struktur aus Abbildung 2.3 zugrunde. Die in Abbildung 2.3 angegebenen Depots *ad_1*, *ad_2* und *bd_1* seien im Zustand T und der benannte K-Akteur *bk_2* sei bereits erzeugt, jedoch noch nicht gestartet. Dementsprechend sind *ad_1*, *ad_2*, *bd_1* sowie *bk_2* nicht in die Relation $\tilde{\alpha}_t$ eingeordnet. Für die anonymen DA-Komponenten aus Abbildung 2.3 sei die in (2.17) definierte Abbildung γ_t wie folgt definiert:

$$\gamma_t(ak\_1) = m\_1;\ \gamma_t(ak\_2) = ad\_3;\ \gamma_t(ad\_1) = \underline{a};\ \gamma_t(ad\_2) = bd\_1;\ \gamma_t(ad\_3) = bd\_2$$

Dementsprechend wird der anonyme K-Akteur *ak_1* *unmittelbar-π-innen* zu dem M-Akteur *m_1* und der anonyme K-Akteur *ak_2* *unmittelbar-π-innen* zu dem M-Akteur *m_3* eingeordnet. Die Abbildung 2.4 zeigt den M-Akteur $\underline{a}$ in der kanonischen Operation von s_3 im Rendezvous mit dem benannten K-Akteur *bk_1*. Der Akteur *m_1* ist in der kanonischen Operation s_2 der Auftraggeber für die K-Order *ko_1*, er wartet auf das Rendezvous mit dem anonymen K-Akteur *ak_1*, der den Generator für *ko_1* exportiert. $\Diamond$

2.3.3 Die Lokalitäts-Struktur

Ein benanntes Depot bzw. ein benannter K-Akteur wird durch Erarbeitung einer entsprechenden Deklaration im Rahmen der Ausführung der kanonischen Operation einer DA-Komponente erzeugt. Zu jedem benannten Depot bzw. K-Akteur x gibt es folglich genau eine DA-Komponente y, so daß $x \in L_0(y)$ gilt. Benannte Depots bzw. K-Akteure sind als lokale N-Komponenten aller Arten von DA-Komponenten zugelassen. Die eindeutige Zuordnung eines benannten Depots bzw. eines benannten K-Akteure zu der DA-Komponente, die deren Deklaration enthält, wird in der Struktur-Relation *λ-innen*, kurz λ, erfaßt. Diese ist die transitive Hülle der Relation *unmittelbar-λ-innen*, die wie folgt definiert ist.

Definition 2.23.:

Seien $a, b \in X_t$; a sei ein benanntes Depot im Zustand T oder ein benannter K-Akteur. a ist *unmittelbar-λ-innen* zu b ($\tilde{\lambda}_t$-innen) – $(a, b) \in \tilde{\lambda}_t$ – $\overset{\triangle}{\Longleftrightarrow}$ a bei Ausführung der kanonischen Operation $op(b)$ erzeugt worden ist und a lokale N-Komponente von b ist, d.h., es gilt: $a \in L_0(b)$.
λ_t ist die transitive Hülle von $\tilde{\lambda}_t$.

□

Die **Lokalitäts-Struktur** des Systems $\mathcal{S}$ zum Zeitpunkt t wird durch das Paar (X_t, λ_t) beschrieben. Der Graph der Struktur $(X_t, \tilde{\lambda}_t)$ ist im allgemeinen nicht zusammenhängend, da nur benannte DA-Komponenten in die Lokalitäts-Struktur eingeordnet werden. Faßt man jedoch eine DA-Komponente $x \in X_t$ mit genau den benannten DA-Komponenten von X_t zusammen, die *λ-innen* zu x sind, so ergibt sich, daß die Menge $L_{lokal}(x) \triangleq \{x\} \cup \{y \in X_t : (y, x) \in \lambda_t\}$ zusammen mit der Einschränkung von $\tilde{\lambda}_t$ auf $L_{lokal}(x)$, die mit $\tilde{\lambda}_t(x)$ bezeichnet wird, ein Baum mit der Wurzel x ist. Das Paar $(L_{lokal}(x), \tilde{\lambda}_t(x))$ beschreibt die Lokalitäts-Struktur der DA-Komponente x zum Zeitpunkt t. Dabei kann die Menge $L_{lokal}(x) - \{x\}$ leer sein.

Beispiel:
Die Abbildung 2.5 zeigt ein Beispiel für eine mögliche Lokalitäts-Struktur des M-Akteurs m. Für die benannten K-Akteure ist neben der Einordnung in die Relation $\tilde{\lambda}$ auch die Einordnung in die Relation $\tilde{\pi}$ angegeben. Das Beispiel verdeutlicht darüberhinaus die parallele Einordnung von Kontrollflüssen von K-Akteuren, die in der kanonischen Operation eines Depots erzeugt werden. Für den K-Akteur *bk_1* gilt, daß er bei Ausführung der kanonischen Operation des benannten Depots *bd_5* erzeugt wird, das seinerseits lokale Komponente des benannten Depots *bd_3* ist. Der K-Akteur *bk_1* wird entsprechend der festgelegten Regeln *unmittelbar-π-innen* zu dem Akteur m eingeordnet, von dem die Depotkette ausgeht. ◊

2.3.4 Die Zeiger-Struktur

Anonyme Depots und anonyme K-Akteure werden durch die Erarbeitung von Generierungsausdrücken erzeugt[13]. Über den bei der Erarbeitung des entsprechenden Generierungsausdrucks erzeugten Zeigerwert ist die Komponente von außen nutzbar. Da Zeigerwerte an Zeiger zugewiesen werden können, kann es keinen, einen oder mehrere Zeiger geben, die auf ein

[13] Generierungsausdrücke werden in 3.2.2.2 detailliert beschrieben.

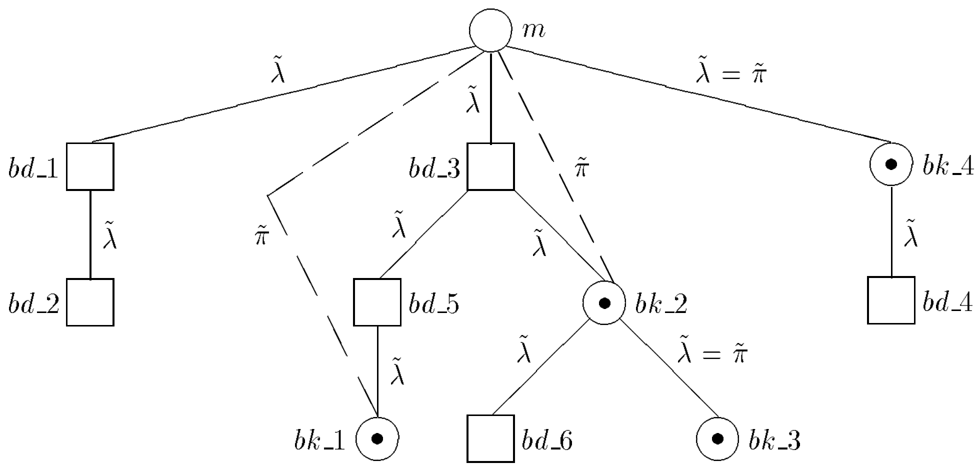


Abbildung 2.5: Der Baum der Lokalitäts-Struktur eines M-Akteurs m

anonymes Depot oder einen anonymen K-Akteur zeigen. Die Möglichkeit, eine Komponente über einen Zeiger identifizieren und nutzen zu können, wird durch die Struktur-Relation ζ-*innen*, kurz ζ, erfaßt. ζ ist die transitive Hülle der Relation *unmittelbar-ζ-innen*, die wie folgt definiert ist.

Definition 2.24.:

> Seien $a, b \in X_t$; a sei ein anonymes Depot im Zustand T oder ein anonymer K-Akteur. a ist *unmittelbar-ζ-innen* zu b ($\tilde{\zeta}_t$-innen) $-$ $(a, b) \in \tilde{\zeta}_t$ $- \overset{\triangle}{\Longleftrightarrow}$ es einen Zeiger v gibt, der auf a zeigt und der lokale N-Komponente von b ist, d.h. es gilt: $v \in L_0(b)$.
> ζ_t ist die transitive Hülle von $\tilde{\zeta}_t$.

□

Die Relation ζ beschreibt die Einordnung anonymer Depots und anonymer K-Akteure durch Zeiger des Systems. Sie setzt dabei ein anonymes Depot bzw. einen anonymen K-Akteur x mit den DA-Komponenten in Beziehung, die lokale Zeiger-Komponenten besitzen, die auf x zeigen.

Das Paar (X_t, ζ_t) beschreibt die **Zeiger-Struktur** des Systems $\mathcal{S}$ zum Zeitpunkt t. Da nur anonyme Depots bzw. anonyme K-Akteure unmittelbar-ζ-innen zu anderen DA-Komponenten eingeordnet werden, ist der Graph der Zeiger-Struktur im allgemeinen nicht zusammenhängend. Faßt man jedoch eine DA-Komponente $x \in X_t$ mit den anonymen Depots im Zustand T und anonymen K-Akteuren zusammen, die ζ-*innen* zu x sind, so ergibt sich, daß die Menge $Z_{lokal}(x) \triangleq \{x\} \cup \{y \in X_t : (y, x) \in \zeta_t\}$ zusammen mit der Einschränkung von $\tilde{\zeta}_t$ auf $Z_{lokal}(x)$, die mit $\tilde{\zeta}_t(x)$ bezeichnet wird, ein zusammenhängender Graph ist. Dieser Graph kann aufgrund der möglichen Wertzuweisungen für DA-Zeiger und der Definition der Relation ζ-*innen* Zyklen enthalten. Das Paar $(Z_{lokal}(x), \tilde{\zeta}_t(x))$ beschreibt die Zeiger-Struktur der DA-Komponente x zum Zeitpunkt t. Dabei kann die Menge $Z_{lokal}(x) - \{x\}$ leer sein.

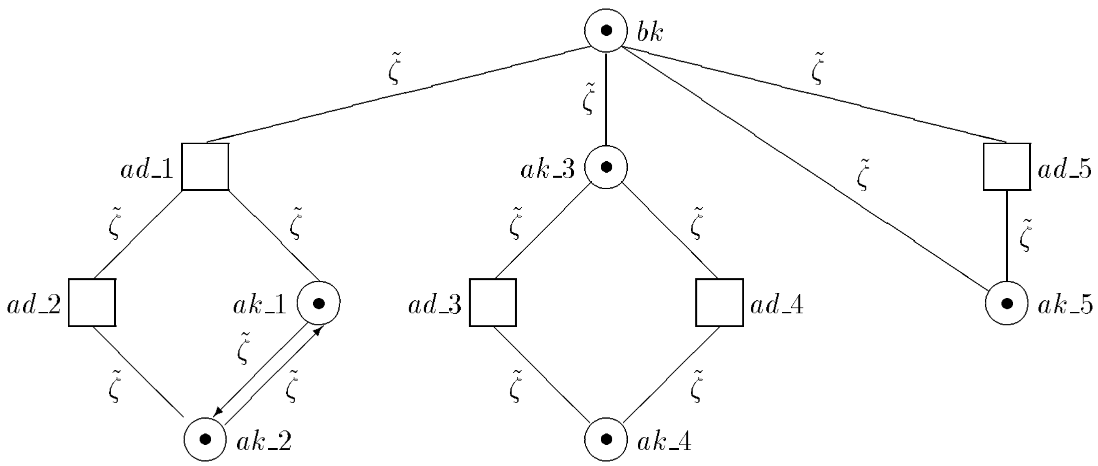


Abbildung 2.6: Die Zeiger-Struktur eines K-Akteurs *bk*

Beispiel:
Die Abbildung 2.6 zeigt ein Beispiel für eine mögliche Zeiger-Struktur eines benannten K-Akteurs *bk*. ◊

2.3.5 Die Lebenszeit-Struktur

Für ein System, dessen Komponenten dynamisch erzeugt und aufgelöst werden, muß sichergestellt sein, daß aus der Existenz einer Komponente x, die eine Komponente y benutzen kann, die Existenz der Komponente y folgt. Das bedeutet, daß konzeptionell die Existenz der DA-Komponenten, die bei Ausführung der kanonischen Operation $op(x)$ einer DA-Komponente x gemäß der Ausführungsumgebung von x benutzt werden können, für die Lebenszeit von x sicherzustellen ist. Für jede DA-Komponente ist konzeptionell festgelegt, mit welcher DA-Komponente sie wieder aufzulösen ist. Bei der Erzeugung einer DA-Komponente wird nach den unten angegebenen Regeln die erzeugte Komponenten lebenszeitmäßig genau einer anderen DA-Komponente zugeordnet. Die Lebenszeitabhängigkeiten zwischen den DA-Komponenten eines Systems werden durch die **Lebenszeit-Struktur** beschrieben. Die Lebenszeit-Struktur wird durch die Relation *ε-innen*, kurz ε (existenziell), definiert. ε ist die transitive Hülle der Relation *unmittelbar-ε-innen*. Deren Definition wird wesentlich von den in Abschnitt 3.4 angegebenen Regeln zur Auflösung von Komponenten bestimmt. Diese wurden unter Berücksichtigung der Art des Beitrags, den eine Komponente zur Funktionalität eines Systems leistet, sowie der Forderung nach der Existenz benutzbarer Komponenten, festgelegt. Zur Motivation der Relation ε werden charakteristische Eigenschaften der Komponentenarten und deren Bedeutung für die Lebenszeiteinordnung von Komponenten im folgenden zusammengefaßt.

- **S-Order und Depots im Zustand** A bzw. **M-Akteure** leisten einen sequentiellen bzw. parallelen Beitrag zur Funktionalität der Komponente, in deren kanonische Operation sie eingeordnet ausgeführt werden. Die σ- bzw. π-Einordnung dieser Komponenten ist deshalb auch für die Lebenszeiteinordnung maßgeblich.

- **Benannte Depots im Zustand** T oder **benannte K-Akteure** werden der Komponente zugeordnet, zu der sie lokal sind. Diese Festlegung ergibt sich daraus, daß benannte Depots im Zustand T oder benannte K-Akteure nur nutzbar sind, solange die Komponente existiert, zu der sie lokal sind.

- **Anonyme Depots** und **anonyme K-Akteure** werden über Zeiger identifiziert und genutzt. Solange die DA-Komponente y, die den Zeigergenerator, der in dem Generierungsausdruck für die anonyme DA-Komponente x auftritt, definiert, noch existiert, können Zeiger bezüglich dieses Generators deklariert werden und die Komponente x kann potentiell über diese Zeiger genutzt werden. Das bedeutet, daß die Lebenszeit von x an die Lebenszeit von y zu binden ist. Zur Erfüllung der Forderung nach der Existenz aller konzeptionell benutzbaren Komponenten werden anonyme Depots und anonyme K-Akteure der Komponente y zugeordnet.

- **K-Order-Generatoren** sind Kommunikationsoperationen, die von K-Akteure nach außen angeboten werden. Die Lebenszeit einer K-Order kommt in der in Abschnitt 2.3.2 definierten Relation κ zum Ausdruck, da die κ-Einordnung für K-Order von der Erzeugung bis zum Zeitpunkt der Auflösung besteht. Das bedeutet, daß eine K-Order lebenszeitmäßig an den Auftraggeber der Kommunikation gebunden wird.

Im folgenden wird die Relation *unmittelbar-ε-innen*, kurz $\tilde{\varepsilon}$, die die existenzielle Einordnung von DA-Komponenten beschreibt und damit die Lebenszeitabhängigkeiten zwischen den DA-Komponenten eines Systems erfaßt, definiert.

Definition 2.25.:

Seien $a, b \in X_t$ und γ_t die mit (2.17) definierte Abbildung.
a ist *unmittelbar-ε-innen* zu b ($\tilde{\varepsilon}_t$-innen) – $(a, b) \in \tilde{\varepsilon}_t$ – $\stackrel{\triangle}{\Longleftrightarrow}$

- $(a, b) \in \tilde{\sigma}_t$ falls a S-Order oder benanntes Depot im Zustand A ist;
- $(a, b) \in \tilde{\pi}_t$ falls a M-Akteur ist;
- $(a, b) \in \tilde{\lambda}_t$ falls a benanntes Depot im Zustand T oder benannter K-Akteur ist;
- $b = \gamma_t(a)$ falls a anonymes Depot oder anonymer K-Akteur ist;
- $(a, b) \in \tilde{\kappa}_t$ falls a K-Order ist.

ε_t ist die transitive Hülle von $\tilde{\varepsilon}_t$.

□

Das Paar (X_t, ε_t) beschreibt die **Lebenszeit-Struktur** des Systems $\mathcal{S}$ zum Zeitpunkt t. Für jede DA-Komponente gilt, daß sie für ihre gesamte Lebenszeit in die Lebenszeit-Struktur eingeordnet ist, d.h. sie wird mit ihrer Erzeugung gemäß (2.25) in ε eingeordnet und mit ihrer Auflösung aus ε ausgeordnet. Weiter folgt aus Definition (2.25), daß es für jede DA-Komponente $a \in X_t$, mit Ausnahme der Hauptkomponente, genau eine DA-Komponente $b \in X_t$ mit $(a, b) \in \tilde{\varepsilon}_t$ gibt. Damit ergibt sich also, daß $(X_t, \tilde{\varepsilon}_t)$ ein Baum mit der Hauptkomponente als Wurzel ist.

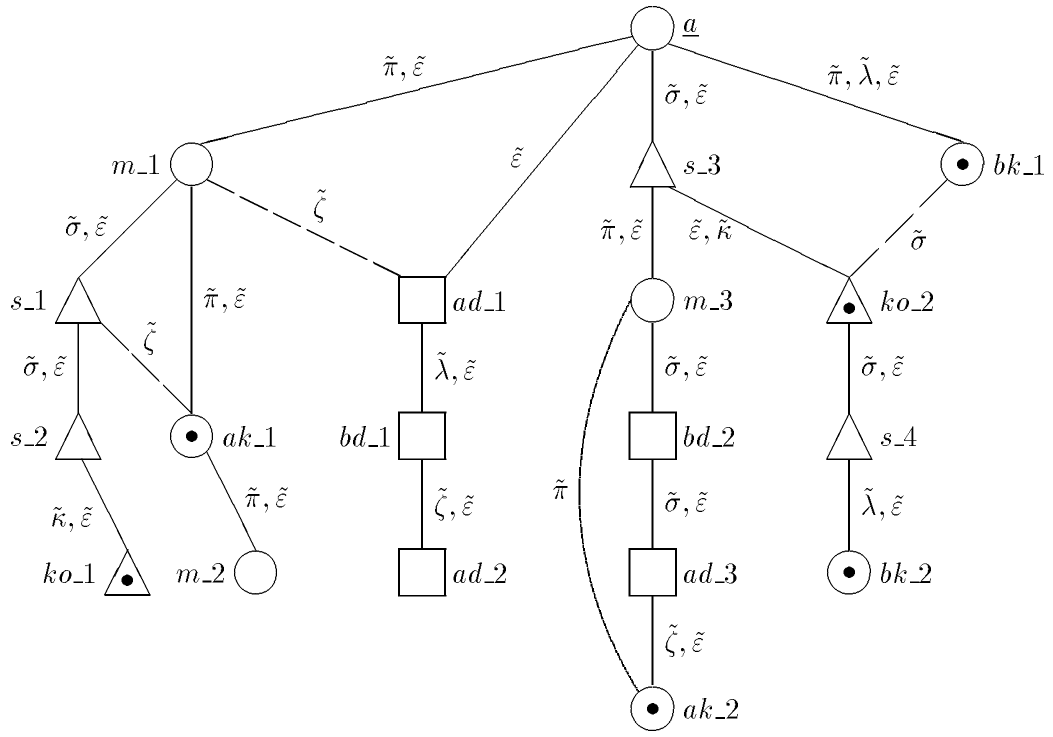


Abbildung 2.7: Beispiel für eine Lebenszeit-Struktur

Beispiel:
Die Abbildung 2.7 zeigt auf Grundlage der Ausführungs-Struktur aus Abbildung 2.4 und der Definitions-Struktur aus Abbildung 2.3 ein Beispiel für eine Lebenszeit-Struktur. Dabei sind neben der Relation ε auch die Relationen, über die ε definiert ist, angegeben. Weiterhin ist für die anonymen Akteure und Depots die Einordnung in die Relation $\tilde{\zeta}$ dargestellt. $\Diamond$

Kapitel 3

Dynamische Systeme

In dem vorhergehenden Kapitel wurden die Konzepte für Komponenten erklärt, aus denen ein System bestehen kann, und die System-Strukturen definiert, mit denen die unterschiedlichen Arten von Abhängigkeiten zwischen den DA-Komponenten eines Systems erfaßt werden. Die mit diesen Konzepten konstruierten Systeme sind dynamisch, d.h. daß ausgehend von der Ausführung der kanonischen Operation der Hauptkomponente weitere DA-Komponenten erzeugt und aufgelöst werden können. Die Erzeugung und Auflösung von DA-Komponenten sowie Zustandsübergänge existierender DA-Komponenten bewirken Veränderungen der System-Strukturen. Um diese Veränderungen zu erfassen und damit die Dynamik eines Systems beschreiben zu können, werden im folgenden Abschnitt System-Konfigurationen definiert. Eine System-Konfiguration beschreibt ein System zu einem Zeitpunkt t als die Menge der zu dem Zeitpunkt t existierenden DA-Komponenten zusammen mit den Strukturen über diesen. Die Dynamik eines Systems im Großen, d.h. auf Ebene der DA-Komponenten, läßt sich durch eine Familie von Konfigurationen beschreiben.

In den weiteren Abschnitten dieses Kapitels werden die Entwicklungsmöglichkeiten eines Systems durch die Erzeugung und Auflösung von DA-Komponenten bei der Ausführung kanonischer Operationen von DA-Komponenten detailliert erklärt. Dabei wird angegeben, wie sich die Entwicklung eines Systems in Konfigurationsveränderungen niederschlägt.

3.1 System-Konfigurationen

Sei $\mathcal{S}$ ein System. In Kapitel 2 wurde festgelegt, daß das System $\mathcal{S}$ zu Beginn seiner Existenz aus einem isolierten M-Akteur besteht. Dieser M-Akteur ist die **Hauptkomponente** des Systems $\mathcal{S}$ und wird im weiteren mit $\underline{a}$ bezeichnet. Alle Eigenschaften des Systems $\mathcal{S}$ entwickeln sich dynamisch aus den Berechnungen seiner Hauptkomponente $\underline{a}$, indem bei der Ausführung der kanonischen Operation $op(\underline{a})$ einerseits DA-Komponenten erzeugt und aufgelöst werden und andererseits die Eigenschaften existierender DA-Komponenten verändert werden können. Die Eigenschaften, die das Systen $\mathcal{S}$ in einem Zeitpunkt t seiner Existenz hat, ergeben sich somit aus den Eigenschaften der DA-Komponenten, aus denen $\mathcal{S}$ zum Zeitpunkt t besteht und aus den Abhängigkeiten zwischen diesen. Dies führt dazu, daß das System $\mathcal{S}$ in jedem Zeitpunkt t seiner Existenz durch eine Konfiguration $\mathcal{S}_t$ beschrieben werden kann.

Eine **System-Konfiguration** $\mathcal{S}_t$ ist ein 6-Tupel mit der Menge der zum Zeitpunkt t existierenden DA-Komponenten X_t und den Struktur-Relationen, welche die Abhängigkeiten zwischen den Elementen von X_t beschreiben. Zur Beschreibung der unterschiedlichen Arten von Abhängigkeiten zwischen den DA-Komponenten eines Systems sind in Abschnitt 2.3 die fünf Struktur-Relationen δ (Definitions-Struktur), α (Ausführungs-Struktur), λ (Lokalitäts-Struktur), ζ (Zeiger-Struktur) und ε (Lebenszeit-Struktur) definiert worden. Mit diesen ergibt sich, daß das System $\mathcal{S}$ zum Zeitpunkt t durch die System-Konfiguration

$$\textbf{(3.1)} \qquad \mathcal{S}_t = (X_t, \delta_t, \alpha_t, \lambda_t, \zeta_t, \varepsilon_t) \quad \text{mit } t \in I\!R_+$$

beschrieben werden kann.
Wie bereits erklärt, besteht das System $\mathcal{S}$ zu Beginn seiner Existenz zum Zeitpunkt $t = 0$ aus dem isolierten M-Akteur $\underline{a}$ als Hauptkomponente. Die **Anfangskonfiguration** des Systems $\mathcal{S}$ ist demnach:

$$\textbf{(3.2)} \qquad \mathcal{S}_0 = (X_0, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset) \quad \text{mit } X_0 = \{\underline{a}\}.$$

Für die Ausführungskomponente $\varphi_0(\underline{a})$ und den Zustand $\mu_0(\underline{a})$ von $\underline{a}$ zum Zeitpunkt $t = 0$ gilt:

$$\textbf{(3.3)} \qquad \varphi_0(\underline{a}) = \underline{a} \quad \text{und} \quad \mu_0(\underline{a}) = R.$$

Vom Zeitpunkt $t = 0$ an führt also der M-Akteur $\underline{a}$ seine kanonische Operation $op(\underline{a})$ aus. Die dynamische Entwicklung von $\mathcal{S}$ für $t > 0$ ergibt sich aus der Ausführung von $op(\underline{a})$. Alle Entwicklungsmöglichkeiten von $\mathcal{S}$ sind durch $\underline{a}$ festgelegt. Bei der Ausführung von $op(\underline{a})$ können alle Arten von DA-Komponenten erzeugt und aufgelöst werden. Die Erzeugung bzw. Auflösung einer DA-Komponente x zur Zeit t bewirkt eine Veränderung der Konfiguration $\mathcal{S}_t$ dahingehend, daß die DA-Komponenten-Menge X_t um x erweitert bzw. verkleinert wird und x Komponentenart-spezifisch in die Struktur-Relationen ein- bzw. ausgeordnet wird. Neben der Erzeugung und Auflösung von DA-Komponenten und den damit verbundenen Zustandsübergängen kann die Überführung einer DA-Komponente x in den Zustand T, die im allgemeinen nicht mit der Auflösung von x zusammenfällt (siehe Abschnitt 3.4), eine Konfigurationsveränderung bewirken. Die Entwicklung des Systems $\mathcal{S}$ im Großen (d.h. auf Ebene der DA-Komponenten) wird somit durch eine **Familie von Konfigurationen** beschrieben. Für das Weitere wird postuliert, daß bei der Ausführung von $op(\underline{a})$ nur endlich viele DA-Komponenten erzeugt werden. Weiterhin wird festgelegt, daß der M-Akteur $\underline{a}$ mit seiner Überführung in den Zustand T aufgelöst wird. Ist $t_e \in I\!R_+$ dieser Zeitpunkt, so ist das Zeitintervall $\Lambda(\mathcal{S}) \triangleq [0, t_e]$ die **Lebenszeit des Systems $\mathcal{S}$**. Die dynamische Entwicklung von $\mathcal{S}$ während seiner Lebenszeit wird durch eine Familie von Konfigurationen $(\mathcal{S}_t : t \in \Lambda(\mathcal{S}))$ mit $\mathcal{S}_t$ gemäß (3.1) und $\mathcal{S}_0$ gemäß (3.2) beschrieben.

In den folgenden Abschnitten werden die Dynamik eines Systems $\mathcal{S}$ und die daraus resultierenden Veränderungen der Konfigurationen detailliert erklärt. In Abschnitt 3.2 werden zunächst die dynamsichen Veränderungen von $\mathcal{S}$ durch die Erzeugung von DA-Komponenten angegeben. Anschließend werden in Abschnitt 3.3 die Phasen, aus denen die Ausführung der kanonischen Operation einer DA-Komponente besteht, sowie die Regeln für die Terminierung einer DA-Komponente erklärt. In Abschnitt 3.4 werden schließlich die Regeln für die Auflösung von DA-Komponenten angegeben.

3.2 Erzeugung von DA-Komponenten

In diesem Abschnitt werden die Möglichkeiten für dynamische Entwicklungen eines Systems $\mathcal{S}$ durch die Erzeugung von DA-Komponenten erklärt. Wie in Kapitel 2 angegeben, wird eine DA-Komponente unter Bezug auf den entsprechenden Generator erzeugt. Bei der Ausführung der kanonischen Operation $op(x)$ einer DA-Komponente x wird genau dann eine DA-Komponente erzeugt, wenn in der Definition von $op(x)$ ein entsprechendes Erzeugungs-Konstrukt auftritt und ausgeführt bzw. erarbeitet wird. Dieses Erzeugungs-Konstrukt ist Art-spezifisch für die Art der DA-Komponente, die erzeugt werden soll, entweder ein spezieller **Aufruf**, eine **Deklaration** oder ein **Generierungsausdruck** . In dem Aufruf bzw. in der Deklaration oder dem Generierungsausdruck steht der Name des Generators, bzgl. dessen bei Ausführung des Aufrufs bzw. bei Erarbeitung der Deklaration oder des Generierungsausdrucks eine Inkarnation erzeugt werden soll. Dabei ist folgendes festgelegt:

(3.4) Voraussetzung dafür, daß in der Definition von x ein Aufruf, eine Deklaration bzw. ein Generierungsausdruck auftritt, ist die Existenz des entsprechenden Generators, auf den in dem Aufruf bzw. in der Deklaration oder dem Generierungsausdruck Bezug genommen wird, in der Ausführungsumgebung von x.[1]

Die Ausführung eines Aufrufs bzw. die Erarbeitung einer Deklaration oder eines Generierungsausdrucks bei Ausführung von $op(x)$ bewirkt dann, daß eine DA-Komponente als Element der durch den Generator definierten Komponentenklasse erzeugt wird. Die Einordnung der Aufrufe, der Deklarationen und der Generierungsausdrücke in die Ausführung der kanonischen Operation $op(x)$ sowie die funktionalen Beiträge, die sie zur Komponente x liefern, werden in Abschnitt 3.3 erklärt.

In den folgenden Unterabschnitten wird Komponentenart-spezifisch erklärt, welche Konfigurationsveränderungen sich durch die Erzeugung von DA-Komponenten ergeben. Dazu werden zunächst einige Festlegungen getroffen und Bezeichnungen eingeführt, die für den Rest dieses Abschnitts gültig sind.

Festlegungen und Bezeichnungen für diesen Abschnitt:

Sei $\mathcal{S}_t = (X_t, \delta_t, \alpha_t, \lambda_t, \zeta_t, \varepsilon_t)$ gemäß (3.1) die Konfiguration des Systems $\mathcal{S}$ zum Zeitpunkt $t > 0$. $a \in X_t$ sei ein Akteur, dessen Ausführungskomponente x ist, d.h. es gelte: $\varphi_t(a) = x$ mit $x \in X_t$. a führt also zum Zeitpunkt t die kanonische Operation $op(x)$ von x aus. Die DA-Komponente x kann der Akteur a selbst (dann gilt $a = x$), eine S-Order oder ein Depot sein. In x trete ein Erzeugungs-Konstrukt für eine DA-Komponente auf. Die Voraussetzung (3.4) sei erfüllt. G sei der entsprechende Generator und $y \in X_t$ die DA-Komponente aus der Ausführungs-Umgebung von x, zu der G lokale N-Komponente ist, d.h. für die gilt: $G \in L_0(y)$. y ist damit eindeutig festgelegt.

3.2.1 S-Order

In diesem Abschnitt werden die Konfigurationsveränderungen beschrieben, die sich durch die Erzeugung von S-Order ergeben. S-Order werden durch die Ausführung von **S-Order-**

[1] Neben dieser Voraussetzung bestehen für die Erzeugung anonymer Depots und anonymer K-Akteure weitere Voraussetzungen. Darauf wird bei der Erklärung der Erzeugung anonymer Depots und anonymer K-Akteure eingegangen.

Aufrufen erzeugt. Die Ausführung eines S-Order-Aufrufs bezüglich des S-Order-Generators G bei Ausführung der kanonischen Operation $op(x)$ durch a bewirkt folgendes:

(3.5)

- Erzeugung einer S-Order s im Zustand V zum Zeitpunkt t in der Konfiguration $\mathcal{S}_t$ als Inkarnation bzgl. G mit der Folgekonfiguration $\mathcal{S}_{t'}$ zum Zeitpunkt $t' > t$ und Zuweisung der Werte der aktuellen Eingabe- und Ein-Ausgabeparameter;
- Einordnung von s in die Struktur-Relationen:
  $\widetilde{\delta}_{t'} \triangleq \widetilde{\delta}_t \cup \{ (s, y) \}$,
  $\widetilde{\sigma}_{t'} \triangleq \widetilde{\sigma}_t \cup \{ (s, x) \}$,
  $\widetilde{\varepsilon}_{t'} \triangleq \widetilde{\varepsilon}_t \cup \{ (s, x) \}$;
- Wechsel der Ausführungskomponente des Akteurs a: $\varphi_{t'}(a) \triangleq s$;
- Ausführung der implizit definierten äußeren Operation „führe_aus" auf s und Überführung von s in den Zustand A, $\mu_{t'}(s) = A$.

Mit (3.5) sind die Veränderungen der Konfiguration $\mathcal{S}_t$ des Systems $\mathcal{S}$ durch Ausführung des S-Order-Aufrufs beschrieben; die Ergebnisse der Veränderungen sind durch den Zeitpunkt t' gekennzeichnet. Mit der Erzeugung von s wird die Menge der DA-Komponenten X_t um s erweitert; für $X_{t'}$ gilt: $X_{t'} \triangleq X_t \cup \{s\}$. Die Relationen δ_t, α_t und ε_t werden auf $X_{t'}$ erweitert; die Relationen λ_t und ζ_t bleiben unverändert.
s wird unmittelbar σ*-innen* zu x eingeordnet, was bedeutet, daß die kanonische Operation $op(s)$ sequentiell in $op(x)$ eingeordnet ausgeführt wird. Dementsprechend wird s die Ausführungskomponente des Akteurs a, der den S-Order-Aufruf ausführt. Die Ausführung von „führe_aus" durch a auf s bewirkt den Übergang vom Zustand V in den Zustand A für s und wird als die **Anfangssynchronisation** der Ausführung von $op(s)$ mit der ausführenden Komponente a bezeichnet. Für sequentiell eingeordnete Komponenten entspricht dies einem Umgebungswechsel, der von Unterprogrammaufrufen imperativer Programmiersprachen bekannt ist.

3.2.2 Depots

In diesem Abschnitt werden die Konfigurationsveränderungen beschrieben, die sich durch die Erzeugung von Depots ergeben; dabei ist zwischen benannten und anonymen Depots zu unterscheiden.

3.2.2.1 Benannte Depots

Benannte Depots werden durch die Erarbeitung von **Depot-Deklarationen** erzeugt. Ein benanntes Depot d ist lokale N-Komponente der DA-Komponente, in deren Deklarationsteil die entsprechende Depot-Deklaration von d auftritt.
Die Erarbeitung einer Depot-Deklaration bezüglich des Depot-Generators G bei Ausführung der kanonischen Operation $op(x)$ durch a in der Konfiguration $\mathcal{S}_t$ bewirkt zunächst folgendes:

(3.6)

- Erzeugung eines benannten Depots d imZustand V als Inkarnation bzgl. G und Zuweisung der Werte der aktuellen Eingabeparameter;

- Einordnung von d in die Struktur-Relationen:
  $\widetilde{\delta}_{t'} \triangleq \widetilde{\delta}_t \cup \{(d,y)\}$,
  $\widetilde{\sigma}_{t'} \triangleq \widetilde{\sigma}_t \cup \{(d,x)\}$,
  $\widetilde{\varepsilon}_{t'} \triangleq \widetilde{\varepsilon}_t \cup \{(d,x)\}$;
- Wechsel der Ausführungskomponente des Akteurs a: $\varphi_{t'}(a) \triangleq d$;
- Ausführung der implizit definierten äußeren Operation „initialisiere" auf d und Überführung von d in den Zustand A, $\mu_{t'}(d) = A$.

Durch (3.6) sind die Veränderungen der Konfiguration $\mathcal{S}_t$ des Systems $\mathcal{S}$ beschrieben, mit denen die Erarbeitung der Depot-Deklaration als Bestandteil von $op(x)$ beginnt. Mit der Erzeugung von d wird die Menge der DA-Komponenten X_t um d erweitert; für $X_{t'}$ gilt: $X_{t'} \triangleq X_t \cup \{d\}$. Die Relationen δ_t, α_t und ε_t werden auf $X_{t'}$ erweitert; die Relationen λ_t und ζ_t bleiben zunächst unverändert. d wird unmittelbar σ-*innen* zu x eingeordnet, was bedeutet, daß die kanonische Operation $op(d)$ sequentiell in $op(x)$ eingeordnet ausgeführt wird. Dementsprechend wird d die Ausführungskomponente des Akteurs a, der die Depot-Deklaration erarbeitet. Die Ausführung von „initialisiere" durch a auf d ist die **Anfangssynchronisation** der Ausführung von $op(d)$ mit der ausführenden Komponente. Sie bewirkt die Überführung von d in den Zustand A und die Ausführung von $op(d)$ durch a.

In Kapitel 2 wurde angegeben, daß der wesentliche Zustand eines Depots d der Zustand T ist, da allein in diesem Zustand Ausführungen der für d definierten Zugriffsoperationen zulässig sind. Sei $\tau > t$ der Zeitpunkt, zu dem das Depot d in den Zustand T, $\mu_{\tau'}(d) = T$, überführt wird[2]. Die entsprechende Konfiguration des Systems $\mathcal{S}$ zum Zeitpunkt τ' sei durch $\mathcal{S}_{\tau'}$ gegeben. Die Überführung von d in T bewirkt folgendes:

(3.7)
- Umordnung von d:
  $\widetilde{\sigma}_{\tau'} \triangleq \widetilde{\sigma}_\tau - \{(d,x)\}$;
  $\widetilde{\lambda}_{\tau'} \triangleq \widetilde{\lambda}_\tau \cup \{(d,x)\}$;
- Wechsel der Ausführungskomponente des Akteurs a: $\varphi_{\tau'}(a) \triangleq x$.

Durch (3.7) sind die Veränderungen der Konfiguration $\mathcal{S}_\tau$ bei Überführung des benannten Depots d in den Zustand T beschrieben. Da mit Überführung von d in T die Ausführung der kanonischen Operation $op(d)$ abgeschlossen wird, wird d aus der Relation σ und damit aus der Ausführungs-Relation ausgeordnet. Die DA-Komponente x, in der voraussetzungsgemäß die Depot-Deklaration für d auftritt und in die $op(d)$ sequentiell eingeordnet ausgeführt wurde, wird wieder zur Ausführungskomponente des Akteurs a. d existiert im Zustand T weiter, und zwar als lokale N-Komponente von x. Dementsprechend wird d unmittelbar λ-*innen* zu x eingeordnet. Der Name von d ist durch die Deklaration festgelegt. Die Relationen δ_τ, ζ_τ und ε_τ bleiben unverändert.

Mit (3.7) ist die Erarbeitung der Depot-Deklaration für d in x abgeschlossen. d kann nun durch Ausführung seiner Zugriffsoperationen genutzt werden.

[2] Die Regeln für die Überführung einer DA-Komponente in den Zustand T werden detailliert in Abschnitt 3.3 behandelt.

3.2.2.2 Anonyme Depots

Anonyme Depots werden durch Erarbeitung von **Generierungsausdrücken** als Bestandteile von Anweisungen mit Wertzuweisungen an Zeiger erzeugt. Es sei vorausgesetzt, daß der Generator G ein Depot-Generator und Z ein mit G qualifizierter Zeiger-Generator ist. Dann ist

(3.8) $\mathtt{new}(Z, G)$ ein **Generierungsausdruck** bezüglich des Generators G.

Die Erarbeitung von (3.8) bewirkt, daß ein anonymes Depot d als Inkarnation bzgl. G und ein eindeutiger Zeigerwert $\zeta(d)$ für d erzeugt werden. Der Generierungsausdruck liefert den Zeigerwert $\zeta(d)$ als Ergebnis. Ist v eine Zeiger-Inkarnation bezüglich des Zeiger-Generators Z, der mit G qualifiziert ist, und ist $\zeta(d)$ der Wert von v, so **zeigt** v auf d. Die Existenz eines mit G qualifizierten Zeigers v, der auf d zeigt, ist Voraussetzung dafür, daß d von außen benutzbar ist. Deshalb wird ein Generierungsausdruck immer als Teil eines Konstrukts mit Wertzuweisung an einen Zeiger betrachtet. Es werden zwei solcher Konstrukte unterschieden, nämlich eine Zeigerdeklaration mit Generierungsausdruck und eine Generierungsanweisung , die aus einer Zeigerzuweisung mit einem Generierungsausdruck besteht.

(3.9) Voraussetzung dafür, daß eine Deklaration mit Generierungsausdruck in der Definition von x auftreten darf, ist neben der Existenz des Depot-Generators G in der Ausführungsumgebung von x die Existenz des Zeiger-Generators Z in der Ausführungsumgebung von x.
Voraussetzung dafür, daß eine Generierungsanweisung in der Definition von x auftreten darf, ist neben der Existenz des Depot-Generators G in der Ausführungsumgebung von x die Existenz des Zeigers v in der Ausführungsumgebung von x.

Dann ist

(3.10) $v : Z := \mathtt{new}(Z, G);$ eine **Zeigerdeklaration mit Generierungsausdruck**.

Eine Zeigerdeklaration mit Generierungsausdruck kann im Deklarationsteil einer DA-Komponente auftreten. Sei nun weiter v ein Zeiger, der Inkarnation bzgl. des Zeiger-Generators Z ist. Dann ist

(3.11) $v := \mathtt{new}(Z, G);$ eine **Generierungsanweisung: Zeigerzuweisung mit Generierungsausdruck**.

Eine Generierungsanweisung kann im Anweisungsteil von DA-Komponente auftreten.

Bei Ausführung der kanonischen Operation $op(x)$ durch a wird also genau dann ein anonymes Depot erzeugt, wenn in der Definition von x eine Zeigerdeklaration mit Generierungsausdruck gemäß (3.10) oder eine Generierungsanweisung gemäß (3.11) auftritt. Sei $z \in X_t$ die DA-Komponente aus der Ausführungsumgebung von x, die den Zeiger-Generator Z als lokale N-Komponente enthält, d.h. für die gilt: $Z \in L_0(z)$.
Im folgenden wird die Wirkung der Erarbeitung des Generierungsausdrucks und der anschließenden Zeigerwertzuweisung bei Erarbeitung von (3.10) bzw. Ausführung von (3.11) durch a erklärt. Die Erarbeitung des Generierungsausdrucks $\mathtt{new}(Z, G)$ bewirkt zunächst folgendes:

(3.12)

- Erzeugung eines anonymen Depots d im Zustand V als Inkarnation bzgl. G und Zuweisung der Werte der aktuellen Eingabeparameter;
- Einordnung von d in die Struktur-Relationen:
  $\widetilde{\delta}_{t'} \triangleq \widetilde{\delta}_t \cup \{\,(d,y)\,\}$;
  $\widetilde{\sigma}_{t'} \triangleq \widetilde{\sigma}_t \cup \{\,(d,x)\,\}$
  $\widetilde{\varepsilon}_{t'} \triangleq \widetilde{\varepsilon}_t \cup \{\,(d,z))\,\}$, mit $Z \in L_0(z)$;
- Wechsel der Ausführungskomponente des Akteurs a: $\varphi_{t'}(a) \triangleq d$;
- Ausführung der implizit definierten äußeren Operation „initialisiere“ auf d und Überführung von d in den Zustand A, $\mu_{t'}(d) = A$.

Durch (3.12) sind die Veränderungen der Konfiguration $\mathcal{S}_t$ des Systems $\mathcal{S}$ beschrieben, mit denen die Erarbeitung des Generierungsausdrucks $\texttt{new}(Z,G)$ beginnt. Mit der Erzeugung von d wird die Menge der DA-Komponenten X_t um d erweitert; für $X_{t'}$ gilt: $X_{t'} \triangleq X_t \cup \{d\}$. Die Relationen δ_t, α_t und ε_t werden auf $X_{t'}$ erweitert; die Relationen λ_t und ζ_t bleiben zunächst unverändert.
d wird unmittelbar σ-*innen* zu x eingeordnet, was bedeutet, daß die kanonische Operation $op(d)$ sequentiell in $op(x)$ eingeordnet ausgeführt wird. Dementsprechend wird d die Ausführungskomponente des Akteurs a, der den Generierungsausdruck erarbeitet. Für die Einordnung von d in die Relation ε und damit in die Lebenszeit-Struktur ist die DA-Komponente z, die den Zeiger-Generator Z als lokale N-Komponente enthält, ausschlaggebend. Durch die Einordnung von d unmittelbar ε-*innen* zu z und die noch anzugebenen Regeln für die Auflösung von DA-Komponenten (siehe Abschnitt 3.4) wird konzeptionell sichergestellt, daß es nach der Auflösung von d keine Zeiger mehr geben kann, die noch auf d zeigen könnten. Das bedeutet, daß keine Komponenten existieren können, die d konzeptionell noch über Zeiger nutzen könnten, obwohl d bereits aufgelöst ist.
Die Ausführung von „initialisiere“ durch a auf d ist die **Anfangssynchronisation** der Ausführung von $op(d)$ mit der ausführenden Komponente a. Sie bewirkt die Überführung von d in den Zustand A und die Ausführung von $op(d)$ durch a.

Wie für benannte Depots ist der wesentliche Zustand eines anonymen Depots der Zustand T. Sei $\tau > t$ der Zeitpunkt, zu dem das anonyme Depot d in den Zustand T überführt wird, mit $\mu_{\tau'}(d) = T$. Die entsprechende Konfiguration des Systems $\mathcal{S}$ zum Zeitpunkt τ' sei durch $\mathcal{S}_{\tau'}$ gegeben. Die Überführung von d in T bewirkt folgendes:

(3.13)

- Ausordnung von d aus α:
  $\widetilde{\sigma}_{\tau'} \triangleq \widetilde{\sigma}_\tau - \{\,(d,x)\,\}$;
- Wechsel der Ausführungskomponente des Akteurs a: $\varphi_{\tau'}(a) \triangleq x$;
- Erzeugung des Zeigerwerts $\zeta(d)$ für d.

Durch (3.13) sind die Veränderungen der Konfiguration $\mathcal{S}_\tau$ bei Überführung des anonymen Depots d in den Zustand T beschrieben. Mit Überführung von d in T wird die Ausführung der kanonischen Operation $op(d)$ abgeschlossen; dementsprechend wird d aus der Relation σ und damit aus der Ausführungs-Relation ausgeordnet. Alle anderen Relationen bleiben zunächst unverändert. Die DA-Komponente x, in der voraussetzungsgemäß der Generierungsausdruck für d auftritt und in die $op(d)$ sequentiell eingeordnet ausgeführt wurde, wird wieder zur Ausführungskomponente des Akteurs a. Mit (3.12) und (3.13) ist die Erarbeitung des Generierungsausdrucks $\texttt{new}(Z,G)$ erklärt. Der Generierungsausdruck liefert gemäß (3.13) den

Zeigerwert $\zeta(d)$ als Ergebnis.
$\zeta(d)$ wird nun dem Zeiger v zugewiesen; dabei ist zu unterscheiden, ob der Generierungsausdruck $\mathtt{new}(Z,G)$ Bestandteil einer Zeigerdeklaration mit Generierungsausdruck gemäß (3.10) oder einer Generierungsanweisung gemäß (3.11) ist. Im ersten Fall wird zunächst der Zeiger v als Inkarnation bzgl. des Zeiger-Generators Z erzeugt; v ist dann lokale N-Komponente von x. Im zweiten Fall existiert der Zeiger v bereits. Er ist lokale N-Komponente einer DA-Komponente $c \in X_\tau$, wobei $c = x$ gelten kann. Die Zuweisung des Zeigerwertes $\zeta(d)$ an den Zeiger v bewirkt, daß zwischen c und d eine Zeigerabhängigkeit in der Zeiger-Relation festgehalten wird und eine zum Zeitpunkt τ eventuell bestehende Zeigerabhängigkeit zwischen der Komponente c und einer Komponente d' wegfällt, wenn v vor der Zuweisung auf die Komponente d' gezeigt hat und dies die einzige Zeigerabhängigkeit zwischen d' und der Komponente c ist. Es gilt:

(3.14)

- Einordnung von d in die Zeiger-Relation:
  $\widetilde{\zeta}_{\tau'} \triangleq \widetilde{\zeta}_\tau \cup \{\,(d,c)\,\}$; dabei gilt $c = x$ im ersten Fall.
- Falls v zum Zeitpunkt τ den Wert $\zeta(d')$ besitzt und falls über keinen anderen Zeiger v' zwischen d' und c eine Zeigerabhängigkeit existiert, dann wird das Paar (d',c) aus der Zeiger-Relation entfernt:
  $\widetilde{\zeta}_{t'} \triangleq \widetilde{\zeta}_t - \{\,(d',c)\,\}$.

Mit (3.12) bis (3.14) ist die Erarbeitung einer Zeigerdeklaration mit Generierungsausdruck (3.10) bzw. die Ausführung einer Generierungsanweisung (3.11) für Depots vollständig erklärt. Das anonyme Depot d existiert im Zustand T weiter und kann mittels des Zeigers v durch Ausführung seiner Zugriffsoperationen von außen genutzt werden. Solange die Komponente d existiert, kann es weitere Zeiger geben, die auf d zeigen und über die d benutzbar ist. Ebenso ist es möglich, daß der Zeiger v im Laufe seiner Lebenszeit auf andere anonyme Depots zeigt, die Inkarnationen bzgl. des Generators G sind, und die mit einem Generierungsausdruck $\mathtt{new}(Z,G)$ erzeugt wurden. Wenn es keinen Zeiger mehr gibt, der auf d zeigt, kann d nicht mehr von außen genutzt werden. Dies hat im allgemeinen jedoch nicht die unmittelbare Auflösung von d zur Konsequenz, da Zeiger dynamisch erzeugt werden können, solange deren Zeiger-Generator noch existiert. Vielmehr ist die Auflösung von d, wie für alle anonymen Komponenten, konzeptionell so festgelegt, daß sie unabhängig von der Existenz einer oder mehrere Zeiger ist, die auf d zeigen (siehe dazu Abschnitt 3.4).

3.2.3 M-Akteure

In diesem Abschnitt werden die Konfigurationsveränderungen beschrieben, die sich durch die Erzeugung von M-Akteuren ergeben. M-Akteure werden durch die Ausführung von **M-Akteur-Aufrufen**, die in Anweisungsteilen von DA-Komponenten auftreten können, erzeugt. Die Ausführung eines M-Akteur-Aufrufs bezüglich des M-Akteur-Generators G bei Ausführung der kanonischen Operation $op(x)$ durch a in der Konfiguration $\mathcal{S}_t$ bewirkt folgendes:

(3.15)

- Erzeugung eines M-Akteurs m im Zustand V als Inkarnation bzgl. G mit $\varphi_{t'}(m) = m$ und Zuweisung der Werte der aktuellen Eingabe- und Ein-Ausgabeparameter;

- Einordnung von m in die Struktur-Relationen:
  $\widetilde{\delta}_{t'} \triangleq \widetilde{\delta}_t \cup \{ (m, y) \}$,
  $\widetilde{\pi}_{t'} \triangleq \widetilde{\pi}_t \cup \{ (m, x) \}$,
  $\widetilde{\varepsilon}_{t'} \triangleq \widetilde{\varepsilon}_t \cup \{ (m, x) \}$;
- Starten des Akteurs m:
  Ausführung der implizit definierten äußeren Operation „starte" auf m und die Überführung von m in den Zustand R, $\mu_{t'}(m) = R$.

Mit (3.15) sind die Veränderungen der Konfiguration $\mathcal{S}_t$ des Systems $\mathcal{S}$ durch Ausführung des M-Akteur-Aufrufs beschrieben. Mit der Erzeugung von m wird die Menge der DA-Komponenten X_t um m erweitert; für $X_{t'}$ gilt: $X_{t'} \triangleq X_t \cup \{m\}$. Die Relationen δ_t, α_t und ε_t werden auf $X_{t'}$ erweitert; die Relationen λ_t und ζ_t bleiben unverändert.
m wird unmittelbar π-*innen* zu x eingeordnet, was bedeutet, daß die kanonische Operation *op*(m) parallel in *op*(x) eingeordnet ausgeführt wird. Dementsprechend bleibt die Ausführungskomponente des Akteurs a nach wie vor x. Die Ausführung von „starte" durch a auf m ist die **Anfangssynchronisation** des erzeugenden Akteurs a mit dem erzeugten Akteur m. Sie bewirkt die Überführung von m vom Zustand V in den Zustand R. Damit beginnt m mit der Ausführung seiner kanonischen Operation *op*(m). Der erzeugende Akteur a setzt die Ausführung der kanonischen Operation *op*(x), in der der M-Akteur-Aufruf auftritt, fort. In der Folgezeit führen die Akteure a und m ihre Berechnungen parallel aus.

3.2.4 K-Akteure

In diesem Abschnitt werden die Konfigurationsveränderungen beschrieben, die sich durch die Erzeugung von K-Akteuren ergeben; dabei ist zwischen benannten und anonymen K-Akteuren zu unterscheiden.

3.2.4.1 Benannte K-Akteure

Benannte K-Akteure werden durch die Erarbeitung von **K-Akteur-Deklarationen**, die in Deklarationsteilen von DA-Komponenten auftreten können, erzeugt. Ein benannter K-Akteur ist lokale N-Komponente der DA-Komponente, in deren Deklarationsteil die entsprechende K-Akteur-Deklaration auftritt.
Die Erarbeitung einer K-Akteur-Deklaration bezüglich des K-Akteur-Generators G bei Ausführung der kanonischen Operation *op*(x) durch a in der Konfiguration $\mathcal{S}_t$ bewirkt folgendes:

(3.16)

- Erzeugung eines benannten K-Akteurs k im Zustand V als Inkarnation bzgl. G mit $\varphi_{t'}(k) = k$ und Zuweisung der Werte der aktuellen Eingabeparameter;
- Erarbeiten der K-Order-Generatoren von k;
- Einordnung von k in die Relationen $\widetilde{\delta}, \widetilde{\lambda}, \widetilde{\varepsilon}$:
  $\widetilde{\delta}_{t'} \triangleq \widetilde{\delta}_t \cup \{ (k, y) \}$,
  $\widetilde{\lambda}_{t'} \triangleq \widetilde{\lambda}_t \cup \{ (k, x) \}$,
  $\widetilde{\varepsilon}_{t'} \triangleq \widetilde{\varepsilon}_t \cup \{ (k, x) \}$;

- Einordnung von k in die Ausführungs-Relation:
  $\widetilde{\pi}_{t'} \triangleq \widetilde{\pi}_t \cup \{\,(k, \eta_t(x))\,\}$;
- Starten des Akteurs k durch die erzeugende Komponente a:
  Ausführung der implizit definierten äußeren Operation „starte" auf k und die Überführung von k in den Zustand R, $\mu_{t'}(k) = R$.

Mit (3.16) sind die Veränderungen der Konfiguration $\mathcal{S}_t$ des Systems $\mathcal{S}$, die durch die Erarbeitung der K-Akteur-Deklaration bewirkt werden, beschrieben. Mit der Erzeugung von k wird die Menge der DA-Komponenten X_t um k erweitert; für $X_{t'}$ gilt: $X_{t'} \triangleq X_t \cup \{k\}$. Die Relationen δ_t, λ_t, α_t und ε_t werden auf $X_{t'}$ erweitert. Mit (3.16) ist k als lokale N-Komponente von x erarbeitet; dementsprechend wird k unmittelbar λ-*innen* zu x eingeordnet. Der Name von k ist durch die Deklaration festgelegt und der Zustand von k ist R. Der erzeugende Akteur a setzt die Erarbeitung des Deklarationsteils der DA-Komponente x fort.

In Kapitel 2 wurde angegeben, daß die wesentlichen Zustände eines K-Akteurs die Zustände R und W sind. Genau in diesen Zuständen können die Kommunikationsoperationen, die ein K-Akteur anbietet, von außen genutzt werden.

Bei der Einordnung des Akteurs k in die Ausführungs-Relation bezeichnet η_t die in Definition (2.18) definierte Abbildung. Ist x kein Depot, so gilt $\eta_t(x) = x$, und damit wird k unmittelbarπ-*innen* zu x eingeordnet, was bedeutet, daß die kanonische Operation $op(k)$ parallel in $op(x)$ eingeordnet ausgeführt wird. Wenn x Depot ist, wird k, wie bei Erklärung der Ausführungs-Struktur in Abschnitt 2.3.2 motiviert, unmittelbar π-*innen* zu der Order oder dem Akteur eingeordnet, die bzw. der mit $\eta_t(x)$ festgelegt ist. Die Ausführung von „starte" durch a auf k ist die **Anfangssynchronisation** des erzeugenden Akteurs a mit dem erzeugten Akteur k. Sie bewirkt die Überführung von k vom Zustand V in den Zustand R. Damit beginnt k mit der Ausführung seiner kanonischen Operation $op(k)$. In der Folgezeit führen die Akteure a und k ihre Berechnungen parallel aus.

3.2.4.2 Anonyme K-Akteure

Für die Erzeugung anonymer K-Akteure gilt das in Abschnitt 3.2.2.2 zur Erzeugung anonymer Depots Gesagte entsprechend. Dies bedeutet, daß anonyme K-Akteure durch Erarbeitung von Generierungsausdrücken erzeugt werden. Ein Generierungsausdruck $\texttt{new}(Z, G)$, wobei G nun ein K-Akteur-Generator ist, kann Bestandteil einer Zeigerdeklaration mit Generierungsausdruck gemäß (3.10) oder einer Generierungsanweisung gemäß (3.11) sein.

Im folgenden wird zunächst der Fall betrachtet, daß im Deklarationsteil der DA-Komponente x eine Zeigerdeklaration mit Generierungsausdruck auftritt. Diese habe die Form $v : Z := \texttt{new}(Z, G)$; G sei ein K-Akteur-Generator und Z sei ein Generator für Zeiger mit der Qualifikation G. Voraussetzung dafür, daß die Deklaration mit Generierungsausdruck in der Definition von x auftreten darf, ist neben der Existenz des K-Akteur-Generators G in der Ausführungsumgebung von x die Existenz des Zeiger-Generators Z in der Ausführungsumgebung von x. Die Voraussetzungen seien für x erfüllt.
Sei $z \in X_t$ die DA-Komponente aus der Ausführungsumgebung von x, die den Zeiger-Generator Z als lokale N-Komponente enthält, d.h. für die gilt: $Z \in L_0(z)$. Die Erarbeitung von $v : Z := \texttt{new}(Z, G)$ durch den Akteur a beginnt mit der Erarbeitung des Generierungsausdrucks $\texttt{new}(Z, G)$. Diese bewirkt in der Konfiguration $\mathcal{S}_t$ folgendes:

(3.17)
- Erzeugung eines anonymen K-Akteurs k im Zustand V als Inkarnation bzgl. G mit $\varphi_{t'}(k) = k$ und Zuweisung der Werte der aktuellen Eingabeparameter;
- Erarbeiten der K-Order-Generatoren von k;
- Einordnung von k in die Relationen $\widetilde{\delta}, \widetilde{\varepsilon}, \widetilde{\zeta}$:
  $\widetilde{\delta}_{t'} \triangleq \widetilde{\delta}_t \cup \{ (k, y) \}$,
  $\widetilde{\varepsilon}_{t'} \triangleq \widetilde{\varepsilon}_t \cup \{ (k, z)) \}$;
  $\widetilde{\zeta}_{t'} \triangleq \widetilde{\zeta}_t \; ; \cup \{ (k, x) \}$
- Erzeugung des eindeutigen Zeigerwerts $\zeta(k)$ für k;
- Einordnen des erzeugten anonyme K-Akteurs k in die Ausführungs-Relation:
  $\widetilde{\pi}_{t'} \triangleq \widetilde{\pi}_t \cup \{ (k, \eta_t(z)) \}$, mit $Z \in L_0(z)$;
- Starten des Akteurs k:
  Ausführen der implizit definierten äußere Operation „starte“ auf k und Überführung von k in den Zustand R, $\mu_{t'}(k) = R$.

Durch (3.17) sind die Veränderungen der Konfiguration $\mathcal{S}_t$ des Systems $\mathcal{S}$ beschrieben, die durch die Erarbeitung des Generierungsausdrucks `new`(Z, G) bewirkt werden. Mit der Erzeugung von k wird die Menge der DA-Komponenten X_t um k erweitert; für $X_{t'}$ gilt: $X_{t'} \triangleq X_t \cup \{k\}$. Die Relationen δ_t, ε_t, ζ_t werden auf $X_{t'}$ erweitert.
Zur Einordnung von k in die Relation ε und damit in die Lebenszeit-Relation gilt das in Abschnitt 3.2.2.2 für anonyme Depots Gesagte analog. Die Erarbeitung des Generierungsausdrucks liefert gemäß (3.17) den Zeigerwert $\zeta(k)$ als Ergebnis; dieser ist einem mit G qualifizierten Zeiger, der eine Inkarnation des Zeiger-Generators Z ist, zuzuweisen. Dazu wird zunächst der Zeiger v als Inkarnation bzgl. des Zeiger-Generators Z erzeugt. Anschließend wird der Zeigerwert $\zeta(k)$ dem so erzeugten Zeiger v zugewiesen. Der Zeiger v ist lokale N-Komponente von x; dementsprechend bewirkt die Zuweisung von $\zeta(k)$ an v die angegebene Einordnung von k in die Zeiger-Struktur. Der K-Akteur k wird, wie bei Erklärung der Ausführungs-Relation in Abschnitt 2.3.2 motiviert, unmittelbar π-*innen* zu der Order oder dem Akteur eingeordnet, die bzw. der mit $\eta_t(z)$ festgelegt ist. Ausschlaggebend für die Einordnung von k in die Ausführungs-Relation ist also die DA-Komponente z, die den Zeiger-Generator Z als lokale N-Komponente enthält.
Das für benannte K-Akteure zur Anfangssynchronisation des erzeugenden Akteurs a mit dem erzeugten Akteur k Gesagte gilt für anonyme K-Akteure entsprechend.
Der erzeugende Akteur a setzt die Erarbeitung des Deklarationsteils der DA-Komponente x parallel zu k, der seine kanonische Operation ausführt, fort.

Im folgenden wird der Fall behandelt, daß in der Definition von x eine Generierungsanweisung der Form $v :=$ `new`(Z, G); auftritt. Die Wirkung der Ausführung der Generierungsanweisung durch den Akteur a unterscheidet sich von der eben erklärten Erarbeitung der Zeigerdeklaration mit Generierungsausdruck allein in folgendem Punkt:

(3.18) Der durch die Erarbeitung des Generierungsausdrucks `new`(Z, G) für k erzeugte eindeutige Zeigerwert $\zeta(k)$ wird unmittelbar nach seiner Erzeugung dem bereits existierenden Zeiger v zugewiesen. Ist $c \in X_t$ die DA-Komponente, die den Zeiger v als lokale N-Komponente enthält, $v \in L_0(c)$, so bewirkt die Zuweisung von $\zeta(k)$ an v die Einordnung von k in die Zeiger-Relation und eventuell eine Ausordnung einer Komponente k', falls v zum Zeitpunkt t auf die Komponente k' verweist:

- Einordnung von k in die Zeiger-Relation:
  $\widetilde{\zeta}_{t'} \triangleq \widetilde{\zeta}_t \cup \{\,(k, c)\,\}$;
- Falls v zum Zeitpunkt t den Wert $\zeta(k')$ besitzt und falls über keinen anderen Zeiger v' zwischen k' und der DA-Komponente c eine Zeigerabhängigkeit existiert, dann wird das Paar (k', c) aus der Zeiger-Relation entfernt:
  $\widetilde{\zeta}_{t'} \triangleq \widetilde{\zeta}_t - \{\,(k', c)\,\}$;

Mit (3.17) bis (3.18) sind die Möglichkeiten zur Erzeugung anonymer K-Akteure und die sich damit ergebenden Konfigurationsveränderungen erklärt. Nach der Erzeugung und dem Start eines K-Akteurs k können andere Akteure mit k über dessen Kommunikationsoperationen kommunizieren. Die Kommunikationsoperationen von K-Akteuren werden durch K-Order-Generatoren definiert. Im folgenden Abschnitt wird die Kommunikation mit K-Akteuren nach dem Operationen-orientierten Rendezvous-Konzept beschrieben, indem die Erzeugung, Ausführung und Auflösung von K-Order erklärt wird.

3.2.5 K-Order und Kommunikation mit K-Akteuren

In den vorangehenden Abschnitten wurde erklärt, wie K-Akteure erzeugt werden und wie der dadurch verursachte Konfigurationsübergang festgelegt ist. Für einen K-Akteur k können äußere Operationen explizit definiert sein; dies sind die **Kommunikationsoperationen**, die der Akteur k anbietet. Die Kommunikationsoperationen dürfen nicht von k bei Ausführung seiner kanonischen Operation aufgerufen werden, da dadurch eine Verklemmung auftreten würde. Ein anderer Akteur k' kann mit k über dessen Kommunikationsoperationen kommunizieren, wenn k in einem der Zustände R oder W existiert und k in der Ausführungsumgebung von k' liegt. Die Kommunikation erfolgt nach dem **Operationen-orientierten Rendezvous-Konzept**, das im folgenden informell erklärt wird.

Sei k ein K-Akteur des Systems $\mathcal{S}$ zum Zeitpunkt $t > 0$ mit der Konfiguration $\mathcal{S}_t$. k ist **Auftragnehmer** für die von ihm angebotenen Kommunikationsoperationen. Andere Akteure können **Aufträge** an k zur Ausführung von dessen Kommunikationsoperationen erteilen. Solche Akteure sind dann **Auftraggeber** für k. Die Kommunikationsoperationen, die k anbietet, werden durch die **K-Order-Generatoren** definiert; sie gehören zu $L_0(k)$. Da ein K-Akteur höchstens Kommunikationsoperationen nach außen anbieten kann, sind diese K-Order-Generatoren und nur diese mit dem Attribut E definiert. Ein Auftraggeber a erteilt einen Auftrag an k, indem er einen **K-Order-Aufruf** bezüglich eines K-Order-Generators ko von k ausführt und eine K-Order erzeugt. Der K-Akteur k nimmt einen bezüglich ko erteilten Auftrag an, indem er eine **K-Order-Annahme** für ko-Aufträge ausführt. Mit der Annahme des Auftrages wird die vom Auftraggeber bereits erzeugte K-Order ausgeführt.
Ein **Rendezvous** zwischen dem Auftraggeber a und dem Auftragnehmer k findet statt, wenn a einen Auftrag erteilt hat und k diesen Auftrag annimmt. Auftraggeber und Auftragnehmer synchronisieren sich zur Ausführung von Kommunikationsoperationen. Ein Auftrag wird nach seiner Erteilung durch einen Auftraggeber in eine Warteschlange für Aufträge bezüglich des aufgerufenen K-Order-Generators eingeordnet und der Auftraggeber wird in den Zustand W überführt. Er bleibt in diesem Zustand solange, bis die dem Auftrag entsprechende K-Order von dem Auftragnehmer ausgeführt ist. Ein Auftragnehmer wird bei Ausführung einer K-Order-Annahme in den Zustand W überführt, falls kein entsprechender Auftrag vorliegt, d.h. wenn die Warteschlange für Aufträge des zugehörigen K-Order-Generators leer ist. Der

Auftragnehmer wartet dann auf Erteilung eines solchen Auftrags. Liegen bei Ausführung einer Annahme bezüglich eines K-Order-Generators ko mehrere Aufträge vor, so wählt der Auftragnehmer aus der Warteschlange den nächsten Auftrag gemäß der „First In First Out"-Strategie aus. Nach Ausführung einer Kommunikationsoperation führen sowohl der Auftragnehmer als auch der Auftraggeber ihre jeweiligen Berechnungen parallel fort.

Das skizzierte Operationen-orientierte Rendezvous-Konzept ist eine Ausprägung des Konzepts der synchronen bidirektionalen Nachrichtenkommunikation, bei dem die Auftragsausführung, d.h. die Berechnung der Ergebnisse durch den Auftragnehmer, im Vordergrund steht. Die für die Auftragserteilung und die Ergebnisübergabe notwendige Nachrichten-Kommunikation ist von nachgeordnetem Interesse. Das Charakteristische des Operationen-orientierten Rendezvous-Konzepts ist, daß der Auftragnehmer die Funktionalität jeder Kommunikationsoperation eindeutig definiert. Diese Kommunikationsoperationen werden durch die K-Order-Generatoren des Auftragnehmers vollständig definiert. Jeder K-Order-Generator spezifiziert die Anzahl, Typen und Übergabemodi der Parameter, die bei einer Auftragserteilung bezüglich des K-Order-Generators zu übergeben sind, sowie die Berechnung, die die Funktionalität der entsprechenden Kommunikationsoperation definiert. Die Wirkungen der Ausführungen von Kommunikationsoperationen, die Inkarnationen bzgl. desselben K-Order-Generators sind, werden durch die aktuellen Eingabeparameter des entsprechenden Aufrufs beeinflußt. Im Vergleich dazu werden bei dem Rendezvous-Konzept wie es zum Beispiel in der imperativen Programmiersprache *Ada* [Ada83] verwendet wird, im Spezifikationsteil eines Auftragnehmers für jede seiner Kommunikationsoperation lediglich ihre Signatur festgelegt. Für jede Kommunikationsoperation muß es im Anweisungssteil des Auftragnehmers wenigstens eine Annahmeanweisung geben. Eine Annahmeanweisung ist ein Block, der durch die Festlegung der Folge der bei der Annahme auszuführenden Anweisungen die Funktionalität der Kommunikationsoperation definiert. Für eine Kommunikationsoperation sind mehrere Annahmeanweisungen im Anweisungssteil des Auftragnehmers zulässig, die **verschiedene** Anweisungsfolgen enthalten können. Für einen Auftraggeber einer Kommunikationsoperation ist damit, im Gegensatz zum Operationen-orientierten Rendezvous, nicht konzeptionell sichergestellt, daß die Funktionalität einer Kommunikationsoperation eindeutig festgelegt ist. Das *Ada*-Rendezvous-Konzept ist eine Mischform zwischen dem Operationen-orientierten Rendezvous und dem reinen Nachrichten-orientierten Rendezvous-Konzept, auf das hier nicht weiter eingegangen wird.

Nach dieser Übersicht und Einordnung des Operationen-orientierten Rendezvous-Konzepts wird die Kommunikation mit K-Akteuren durch die Erzeugung, Ausführung und Auflösung von K-Order erklärt. Jeder in einem System $\mathcal{S}$ mit der Konfiguration $\mathcal{S}_t$ definierte K-Order-Generator ist lokale N-Komponente eines K-Akteurs von X_t, weil K-Order-Generatoren allein in Deklarationsteilen von K-Akteuren erlaubt sind. Sei $k \in X_t$ ein K-Akteur des Systems $\mathcal{S}$ zum Zeitpunkt t, in dessen Deklarationsteil der K-Order-Generator C definiert ist, also $C \in L_0(k)$. Dann sind

(3.19) K-Order-Annahmen bzgl. C sind nur im Anweisungsteil der kanonischen Operation von k zulässig.

K-Order-Annahmen sind Anweisungen. Alle K-Order-Annahmen bzgl. C sind also gemäß (3.19) in die Ausführung der kanonischen Operation $op(k)$ eingeordnet.
Als Hilfsmittel für die Synchronisation von Auftraggebern mit dem Auftragnehmer k ist

(3.20) mit dem K-Order-Generator C implizit ein Warteraum $w(C)$ für K-Order bzgl. C definiert; mit $w(C)$ werden für Auftraggeber bzgl. C und für k die Zustände W ($\equiv$ wartend) bzgl. C realisiert.

Sei $a \in X_t$ ein Akteur mit $a \neq k$; a kann M-Akteur oder K-Akteur sein. Die Ausführungskomponente von a sei x, d.h. es gilt: $\varphi_t(a) = x$. Der Akteur a erteilt einen Auftrag bzgl. C an k, indem a einen K-Order-Aufruf bzgl. C ausführt. K-Order-Aufrufe sind Anweisungen, die in Anweisungsteilen von DA-Komponenten auftreten können. Die Voraussetzung für das Auftreten eines K-Order-Aufrufs bzgl. C in $op(x)$ ist die Existenz des K-Akteurs k in der Ausführungsumgebung von x. Diese Voraussetzung sei für $op(x)$ erfüllt. Die **Ausführung des K-Order-Aufrufs** bzgl. C durch a in der Konfiguration $\mathcal{S}_t$ bewirkt folgendes:

(3.21)

- Erzeugung einer K-Order c im Zustand V als Inkarnation bzgl. C und Zuweisung der Werte der aktuellen Eingabe- und Ein-Ausgabeparameter;
- Einordnung von c in die Struktur-Relationen:
  $\widetilde{\delta}_{t'} \triangleq \widetilde{\delta}_t \cup \{\,(c,k)\,\}$,
  $\widetilde{\kappa}_{t'} \triangleq \widetilde{\kappa}_t \cup \{\,(c,x)\,\}$,
  $\widetilde{\varepsilon}_{t'} \triangleq \widetilde{\varepsilon}_t \cup \{\,(c,x)\,\}$;
- Einfügen von c in den Warteraum $w(C)$ und Überführung von k in den Zustand R, $\mu_{t'}(k) = R$, falls k zum Zeitpunkt t im Zustand W bzgl. C ist;
- Überführung von a in den Zustand W bzgl. C, $\mu_{t'}(a) = W$.

Mit (3.21) sind die Veränderungen der Konfiguration $\mathcal{S}_t$ des Systems $\mathcal{S}$ durch Ausführung des K-Order-Aufrufs beschrieben. Mit der Erzeugung von c wird die Menge der DA-Komponenten X_t um c erweitert; für $X_{t'}$ gilt: $X_{t'} \triangleq X_t \cup \{c\}$. Die Relationen δ_t, α_t und ε_t werden auf $X_{t'}$ erweitert; die Relationen λ_t und ζ_t bleiben zunächst unverändert. Mit der κ-Einordnung von c wird der Zusammenhang zwischen der K-Order c und ihrem Auftraggeber a in der Ausführungs-Relation erfaßt. Der durch die K-Order c definierte Auftrag an k wird in dem Warteraum $w(C)$ bereitgestellt. Wenn sich der Auftragnehmer k bei der Auftragserteilung im Zustand W bzgl. C befindet, wird er in den Zustand R überführt. Der Auftraggeber a überführt sich in den Zustand W bzgl. C und wartet damit auf den Abschluß der Auftragsausführung durch k. Damit ist der Auftrag c von a an k erteilt.

In folgenden wird die **Ausführung einer K-Order-Annahme** bzgl. C durch den K-Akteur k erklärt. Dazu sei $\mathcal{S}_\tau$ die Konfiguration des Systems $\mathcal{S}$ zum Zeitpunkt τ. k führe zum Zeitpunkt τ eine K-Order-Annahme bzgl. C aus. Gemäß (3.19) gilt dann für die Ausführungskomponente von k: $\varphi_\tau(k) \triangleq k$. Die Ausführung der K-Order-Annahme bzgl. C bewirkt zunächst folgendes:

(3.22)

- Überführung von k in den Zustand W bzgl. C, $\mu_{\tau'}(k) = W$, falls der Warteraum $w(C)$ leer ist oder
- Auswahl einer K-Order $\widetilde{c}$ aus dem Warteraum $w(C)$, falls $w(C)$ nicht leer ist. Enthält $w(C)$ mehrere K-Order, wird die nächste K-Order nach dem nach FIFO (First In – First Out) Kriterium ausgewählt.

Wenn $w(C)$ keine K-Order enthält, überführt sich k in den Zustand W bzgl. C; er wartet dann auf Erteilung eines Auftrags bzgl. C. In diesem Fall wird k bei Erteilung des nächsten

Auftrags bzgl. C von dem entsprechenden Auftraggeber in den Zustand R überführt; dies ist in (3.21) beschrieben.

Im weiteren sei angenommen, daß $\tau > t$ gilt, wobei t der Zeitpunkt der Erzeugung der K-Order gemäß (3.21) ist, und k zum Zeitpunkt τ die Annahme gemäß (3.22) ausführt und c auswählt. Dann ist k im Zustand R. Die mit (3.22) begonnene Ausführung der K-Order-Annahme bzgl. C bewirkt dann weiter folgendes:

(3.23)
- Einordnung von c in σ:
  $\widetilde{\sigma}_{\tau'} \triangleq \widetilde{\sigma}_{\tau} \cup \{ (c,k) \}$;
- Wechsel der Ausführungskomponente des K-Akteurs k: $\varphi_{\tau'}(k) = c$;
- Ausführung der implizit definierten äußeren Operation „führe_aus“ auf c und Überführung der K-Order in den Zustand A, $\mu_{\tau'}(c) = A$.

Der Wechsel der Ausführungskomponente des K-Akteurs k von k zu c bedeutet, daß k in der Folgezeit die kanonische Operation $op(c)$, d.h. die dem Auftrag entsprechende Kommunikationsoperation, ausführt. $op(c)$ wird sequentiell in $op(k)$ eingeordnet ausgeführt. Für die Ausführung von $op(c)$ gilt das in Abschnitt 3.2.1 für S-Order Gesagte entsprechend. Sei $\theta > \tau$ der Zeitpunkt, zu dem die K-Order c in den Zustand T überführt wird. Die entsprechende Konfiguration des Systems $\mathcal{S}$ zum Zeitpunkt θ sei durch $\mathcal{S}_\theta$ gegeben. Die Überführung von c in T durch k bewirkt folgendes:

(3.24)
- Ausordnung von c aus σ:
  $\widetilde{\sigma}_{\theta'} \triangleq \widetilde{\sigma}_{\theta} - \{ (c,k) \}$;
- Wechsel der Ausführungskomponente des K-Akteurs k: $\varphi_{\theta'}(k) \triangleq k$;
- Entnahme von c aus dem Warteraum $w(C)$;
- Überführung des Akteurs a in den Zustand R, $\mu_{\theta'}(a) = R$.

Mit der Überführung von c in T wird die Ausführung der kanonischen Operation $op(c)$ abgeschlossen; dementsprechend wird c aus der Relation σ ausgeordnet; der K-Akteur k setzt in der Folgezeit die Ausführung seiner kanonischen Operation $op(k)$ fort. Wesentlich ist, daß die K-Order mit den Übergang nach T noch nicht aufgelöst wird, da die Ausgabeparameter noch an den Auftraggeber zu übergeben sind. Die Überführung des Akteurs a in den Zustand R bewirkt folgendes:

(3.25)
- Ergebnisübernahme: Zuweisung der Ausgabe und Ein-Ausgabeparameterwerte von c;
- Ausordnung von c aus den Struktur-Relationen:
  $\widetilde{\kappa}_{\theta'} \triangleq \widetilde{\kappa}_{\theta} - \{ (c,x) \}$,
  $\widetilde{\delta}_{\theta'} \triangleq \widetilde{\delta}_{\theta} - \{ (c,k) \}$,
  $\widetilde{\varepsilon}_{\theta'} \triangleq \widetilde{\varepsilon}_{\theta} - \{ (c,x) \}$;
- Auflösung von c.

Mit (3.24) ist die K-Order-Annahme bzgl. C durch den K-Akteur k und mit (3.25) der K-Order-Aufruf bzgl. C durch den Akteur a ausgeführt. Damit ist die Ausführung einer Kommunikationsoperation bzgl. C vollständig erklärt.

Die in diesem Abschnitt erklärten Sprachkonzepte, die für die Erteilung und für die Annahme von Aufträgen zur Verfügung stehen, sind in der Sprache INSEL um ein nichtdeterministisches Annahme-Konstrukt, die **auswählende Annahmeanweisung**, erweitert. Darauf wird in diesem Bericht jedoch nicht weiter eingegangen; verwiesen sei auf die entsprechende Erklärungen in [RW94].

3.3 Ausführungsphasen einer kanonischen Operation

In diesem Abschnitt werden die Ausführungsphasen einer kanonischen Operation einer DA-Komponente erklärt. Dazu wird eine DA-Komponente x eines Systems $\mathcal{S}$ mit der Konfiguration $\mathcal{S}_t$ zum Zeitpunkt $t > 0$ betrachtet. Das im folgenden für x Gesagte gilt für alle DA-Komponenten des Systems $\mathcal{S}$.
Nach der in Abschnitt 3.2 erklärten Anfangssynchronisation durch Ausführung der implizit definierten äußeren Operation auf x beginnt die Ausführung der kanonischen Operation $op(x)$. Die kanonische Operation von x wird in zwei aufeinanderfolgenden **Phasen**, der **Aufbau-** und der **Berechnungsphase** ausgeführt. Die Berechnungsphase zerfällt wiederum in aufeinanderfolgende Phasen, nämlich in die **Haupt-**, die **Synchronisations-** und für FS-Order noch zusätzlich in die **Ergebnisphase**.

(3.26) In der **Aufbauphase** der kanonischen Operation von x werden die lokalen N-Komponenten von x, die nicht Eingabe- oder Ein-Ausgabeparametern entsprechen, erarbeitet. Die Aufbauphase entspricht also der Erarbeitung des Deklarationsteils von x. Sie beginnt mit der Anfangssynchronisation und endet mit dem Abschluß der Erarbeitung des Deklarationsteils von x.

(3.27) Die Berechnungsphase besteht aus der **Hauptphase**, die explizit durch Anweisungen zu definieren ist, wobei die Return-Anweisung von FS-Ordern[3] nicht zur Hauptphase zählt, gefolgt von der **Synchronisationsphase**, die implizit definiert ist und in der die **Abschlußsynchronisation** mit allen $\tilde{\pi}$-inneren Komponenten durchgeführt wird. Die Ausführung der Anweisungen der Synchronisationsphase bewirkt für alle $a \in X_t$ mit $(a, x) \in \tilde{\pi}_t$

- die Zuweisung der Ausgabe- und der Ein-Ausgabeparameter-Werte von a sowie die Auflösung von a, falls a M-Akteur ist, und seine Ausordnung aus der Ausführungs-Relation, $\tilde{\pi}_{t'} \triangleq \tilde{\pi}_t - \{(a, x)\}$;
- die Ausordnung von a aus der Ausführungs-Relation, falls a K-Akteur ist: $\tilde{\pi}_{t'} \triangleq \tilde{\pi}_t - \{(a, x)\}$.

Die Synchronisationsphase wird für FS-Order gefolgt von einer **Ergebnisphase**. In der Ergebnisphase der kanonischen Operation von x wird

- die Return-Anweisung mit Ergebnisausdruck ausgeführt, falls x FS-Order ist.

Die **Regel für die Abschlußsynchronisation** einer Komponente x ist wie folgt festgelegt:

[3] Zur Erinnerung: FS-Order sind eine Unterart der S-Order; sie haben die Eigenschaften von Funktions-Inkarnationen (siehe Kapitel 2).

(3.28) Voraussetzung für die Ausführung der implizit festgelegten Anweisungen der Synchronisationsphase ist, daß sich alle Akteure $a \in X_t$ mit $(a, x) \in \tilde{\pi}_t$ im Zustand T befinden.

Die Phasen der kanonischen Operation $op(x)$ einer DA-Komponente x sind in Abbildung 3.1 skizziert.

Aus dem bisher in diesem Kapitel Gesagten ergeben sich folgende **Eigenschaften** für die DA-Komponente x:

(3.29) (a) x ist eine Einheit und ist aus Komponenten zusammengesetzt.

(b) x besteht aus den durch den Deklarationsteil definierten lokalen N-Komponenten $L_0(x)$ und aus dem Programm $L_a(x)$ der explizit für x definierten inneren Operation, der kanonischen Operation $op(x)$. Das Programm enthält insbesondere den durch den Anweisungsteil von x definierten Teil.

(c) x können weitere Komponenten zugeordnet sein (vgl. Abbildung 3.2):

- einfache anonyme Komponenten, deren Zeiger-Generator in $L_0(x)$ enthalten ist;
- anonyme Depots und K-Akteure auf die ein Zeiger verweist, dessen Zeiger-Generator in $L_0(x)$ enthalten ist;
- weitere anonyme DA-Komponenten ohne explizite Zeiger:
  - Order, die unmittelbar σ-innen bzw. unmittelbar κ-innen zu x eingeordnet werden;
  - M-Akteure, die π-innen zu x eingeordnet werden.

(d) Für x sind globale Zustände definiert: V, A, T; mit $A = \{R, W\}$ für Akteure.

(e) x wird im Zustand V erzeugt.
Für x in V gilt:

- x besteht aus lokalen N-Komponenten, die den Eingabe- und Ein-Ausgabeparametern entsprechen, aus den K-Order-Generatoren von x, falls x ein K-Akteur ist, und aus dem Programm $L_a(x)$ der kanonischen Operation $op(x)$.
- Auf x ist die implizite, ausgezeichnete äußere Operation „führe_aus“, bzw. „initialisiere“ bzw. „starte“ definiert. Das Erzeugen von x endet mit der Ausführung der impliziten, ausgezeichneten äußeren Operation, die zunächst den Übergang von x in den Zustand A bewirkt, und mit der die Ausführung von $op(x)$ beginnt.
- Die kanonische Operation $op(x)$ besteht aus dem **sequentiellen Kern**, der durch σ-, π- und κ-eingeordnete Komponenten erweitert werden kann.
  Die kanonische Operation $op(x)$ wird von einem Akteur k ausgeführt, wenn gilt: $\varphi_t(k) = x$. Dann führt k den sequentiellen Kern von $op(x)$ aus. Der sequentielle Kern von $op(x)$ gehört nicht zum sequentiellen Kern von $op(k)$, aber ist in diesen eingeordnet.
- Die Ausführung des sequentiellen Kerns von $op(x)$ besteht in der Ausführung der Aufbauphase gefolgt von der Ausführung der Berechnungsphase von x.
- Die Ausführung von $op(x)$ endet mit der Überführung von x in den Zustand T.

Neben M-Akteur-Generatoren und K-Order-Generatoren können PS-Order-Generatoren mit Ausgabe- und Ein-Ausgabeparametern parametrisiert werden. Die Zuweisung der entsprechenden Parameter-Werte erfolgt bei M-Akteuren bei Ausführung der entsprechenden Operation zur Abschlußsynchronisation. Für K-Order wurde die Zuweisung der Ausgabe- und Ein-Ausgabewerte in Abschnitt 3.2.5 erklärt. Für eine PS-Order x wird festgelegt, daß mit der Überführung von x in den Zustand T die Ausgabe- und Ein-Ausgabeparameterwerte von x den entsprechenden aktuellen Parametern zugewiesen werden.

Die Ausführung eines Akteur-Aufrufs in der kanonischen Operation $op(x)$ besteht aus einem sequentiellen Anteil (Erzeugen und Starten eines M-Akteurs) und aus einem parallelen Anteil, der in der Ausführung der kanonischen Operation des erzeugten Akteurs besteht und von dem erzeugten Akteur selbst ausgeführt wird. Wie in Abschnitt 3.2.3 angegeben, wird ein erzeugter M-Akteur unmittelbar π-innen zu x eingeordnet; er leistet also einen parallelen Beitrag zu $op(x)$. Wie in Abschnitt 3.2.4.2 erklärt, wird ein erzeugter K-Akteur nicht notwendigerweise unmittelbar π-innen zur erzeugenden Komponente eingeordnet. Er leistet einerseits einen parallelen Beitrag zur kanonischen Operation der DA-Komponente, zu der er unmittelbarπ-innen eingeordnet wird, andererseits leistet er Beiträge zu den kanonischen Operationen, bei deren Ausführung eine seiner Kommunikationsoperationen aufgerufen wird.

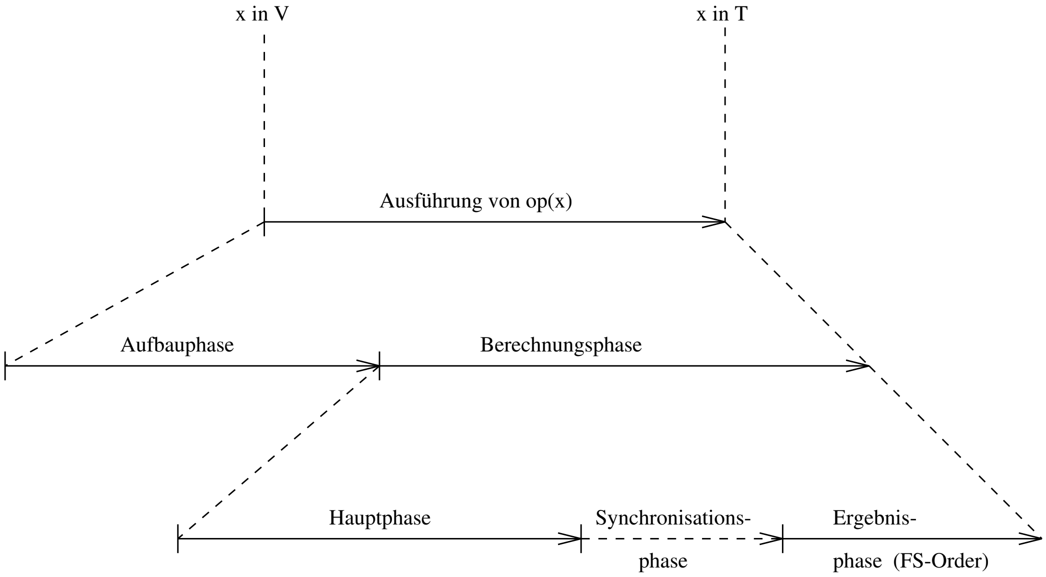


Abbildung 3.1: Phasen einer kanonischen Operation

Die Erweiterung einer DA-Komponente x um ε-, π-, σ-, und κ-eingeordnete Komponenten ist in Abbildung 3.2 skizziert. Für die ε-Erweiterungen wird mit dem mit (*) markierten Pfeil die Erweiterung der Komponente um anonyme Komponenten, die von anderen Komponenten erzeugt worden sind, und deren Generator im Deklarationsteil der betrachteten Komponente definiert ist, angedeutet.

Mit der Überführung von x in den Zustand T wird die Ausführung der kanonischen Operation $op(x)$ abgeschlossen. Aus dem bisher Erklärten ergibt sich folgende **Terminierungsregel** für DA-Komponenten:

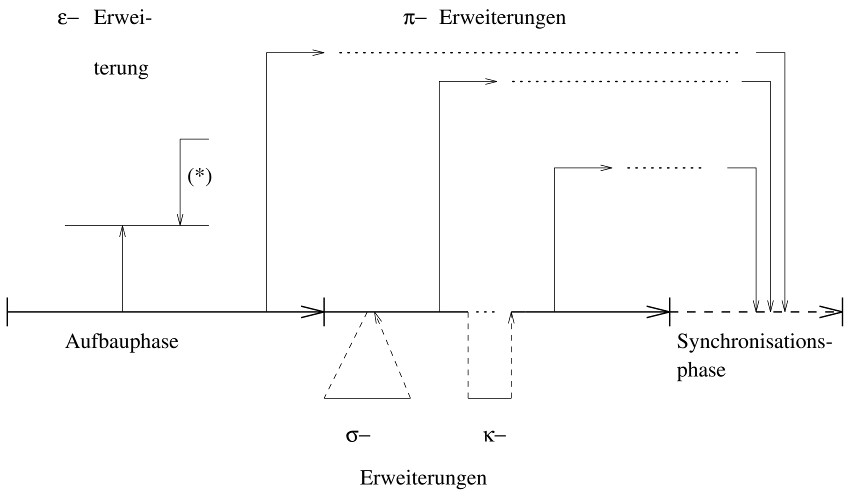


Abbildung 3.2: Dynamische Erweiterungen einer DA-Komponente

(3.30) Eine DA-Komponente $x \in X_t$ wird in den Zustand T überführt, wenn

- sich alle Akteure $a \in X_t$ mit $(a, x) \in \tilde{\pi}_t$ im Zustand T befinden und
- das Ende der Berechnungsphase der kanonischen Operation $op(x)$ erreicht ist.

3.4 Auflösung von DA-Komponenten

In diesem Abschnitt werden die Regeln für die Auflösung von DA-Komponenten erklärt. Für ein System, dessen Komponenten dynamisch erzeugt und aufgelöst werden, ist zu fordern, daß konzeptionell die Existenz der DA-Komponenten, die bei Ausführung der kanonischen Operation $op(x)$ einer DA-Komponente x gemäß der Ausführungsumgebung von x benutzt werden können, für die Lebenszeit von x sichergestellt ist. Die entsprechenden Lebenszeit-Abhängigkeiten der DA-Komponenten eines Systems werden durch die in Abschnitt 2.7 definierte Lebenszeit-Struktur beschrieben, deren Definition durch die im folgenden angegebenen Auflösungsregeln für DA-Komponenten bestimmt ist.

Die Regeln, nach denen die DA-Komponenten eines Systems aufgelöst werden, sind Komponentenart-spezifisch festgelegt. Sie sind gerade so gewählt, daß die obige Forderung nach der Existenz der DA-Komponenten, die gemäß der Ausführungsumgebung benutzbar sind, konzeptionell erfüllt ist. Zur Angabe der Regeln wird das System $\mathcal{S}$ zum Zeitpunkt $t > 0$ mit der Konfiguration $\mathcal{S}_t$ betrachtet.

Für alle Arten von DA-Komponenten gilt:

(3.31) Voraussetzung für die Auflösung einer DA-Komponente $x \in X_t$ ist, daß x sich im Zustand T befindet.

Die Voraussetzungen für die Überführung einer DA-Komponente in den Zustand T werden durch die Terminierungsregel (3.30) für DA-Komponenten festgelegt. Mit der Terminierung von x, $\mu_t(x) = T$, ist die Berechnung von x abgeschlossen. Bei der Durchführung der Berechnung werden im allgemeinen von x Ergebnisse berechnet, die vor der Auflösung von x

zu übergeben sind. Die Übernahme der Ergebniswerte ist nicht mehr Bestandteil der kanonischen Operation von x. Da die Ergenisse noch nach der Terminierung übergeben werden müssen, fallen die Zeitpunkte der Terminierung und der Auflösung einer DA-Komponente $x \in X_t$ im allgemeinen nicht zusammen. Als Beispiel hierfür sind die M-Akteure zu nennen. Die Übernahme der Ausgabe- und Ein-Ausgabeparameter eines zu x $\widetilde{\pi}$-*inneren* M-Akteurs erfolgt erst in der Synchronisationsphase von $op(x)$. Dementsprechend kann ein M-Akteur erst nach der Parameterübergabe aufgelöst werden. Wesentlich ist, daß die erarbeiteten Ergebnisse von einem Akteur a an den Akteur, in den er unmittelbar π-*innen* eingeordnet ist, an festgelegten Synchronisationspunkten übergeben werden.

Für die Auflösung von DA-Komponenten gelten folgende **Regeln**:

- Eine **S-Order** $s \in X_t$ wird mit ihrer Überführung in den Zustand T aufgelöst.
  Ist der S-Order-Generator für s mit Ausgabe- bzw. Ein-Ausgabeparametern definiert, erfolgt die entsprechende Parameterrückgabe mit der Überführung von s in den Zustand T unmittelbar vor der Auflösung von s. Die Auflösungsregel für S-Order ergibt sich daraus, daß mit Terminierung der kanonischen Operation einer S-Order s und der dabei möglicherweise stattfindenden Parameterrückgabe der sequentielle Beitrag von s zu dem Akteur a, der s erzeugt und damit $op(s)$ ausgeführt hat, abgeschlossen ist. Wenn s zum Zeitpunkt t in den Zustand T überführt wird, gilt wegen der zweiten in der Terminierungsregel (3.30) gestellten Bedingung $\varphi_t(a) = s$. Zudem gibt es eine DA-Komponente $x \in X_t$ mit $(s, x) \in \widetilde{\sigma}_t$. Mit der Auflösung von s wird x zur Ausführungskomponente von a, die Komponente s wird aus der Menge X_t der DA-Komponenten entfernt und s wird aus den Struktur-Relationen δ_t, σ_t und ε_t ausgeordnet.

- Eine **K-Order** $c \in X_t$ wird unmittelbar nach Rückgabe ihrer Ausgabe- bzw. Ein-Ausgabeparameter von dem Auftraggeber für c aufgelöst (siehe dazu Abschnitt 3.2.5; insbesondere (3.25)). Mit der Auflösung von c wird c aus den Struktur-Relationen δ_t, κ_t und ε_t ausgeordnet; c wird aus der Menge X_t der DA-Komponenten entfernt.

- Ein **M-Akteur** $m \in X_t$ wird mit Ausführung der Operation zur Abschlußsynchronisation von m in der Synchronisationsphase der kanonischen Operation der DA-Komponente $x \in X_t$ mit $(m, x) \in \widetilde{\pi}_t$, in die er also parallel eingeordnet ist, aufgelöst.
  Wenn die Operation zur Abschlußsynchronisation von m in der Synchronisationsphase von $op(x)$ zum Zeitpunkt t ausgeführt wird, gilt nach Regel (3.27): $\mu_t(m) = T$. Die Ausführung der Operation zur Abschlußsynchronisation von m bewirkt zunächst die Rückgabe der Ausgabe- und Ein-Ausgabeparameter von m. Damit ist der parallele Beitrag, den der Akteur m zur kanonischen Operation $op(x)$ leistet, abgeschlossen. m kann somit aufgelöst werden. Mit der Auflösung von m wird m aus den Struktur-Relationen δ_t, π_t und ε_t ausgeordnet; die Menge X_t der DA-Komponenten wird um m verkleinert.

  Für die Hauptkomponente $\underline{a}$, die gemäß Festlegung ein M-Akteur ist, ist obige Regel nicht anwendbar. Für die Hauptkomponente ist festgelegt, daß sie mit ihrer Überführung in den Zustand T aufgelöst wird.

- Ein **benanntes Depot** $d \in X_t$ wird mit Auflösung der DA-Komponente $x \in X_t$, die d als lokale N-Komponente enthält, d.h. für die gilt $(d, x) \in \widetilde{\lambda}_t$, aufgelöst.
  Die Auflösungsregel für Depots ergibt sich aus der Festlegung, daß die Elemente der Menge $L_0(x)$ der lokalen N-Komponenten einer DA-Komponente x mit x aufgelöst werden. Diese Festlegung ist dadurch motiviert, daß eine lokale N-Komponente $y \in$

$L_0(x)$ "unmittelbarer" Bestandteil von x ist und somit zeitlich nicht vor x aufgelöst werden darf. Mit den Auflösungsregeln für die weiteren Komponentenarten und die noch folgende Definition der Ausführungsumgebung ist gewährleistet, daß nach Auflösung eines Depots d alle DA-Komponenten, die Zugriffsoperationen auf d ausführen könnten, ebenfalls aufgelöst sind. Mit seiner Auflösung wird d aus den Struktur-Relationen δ_t, λ_t und ε_t ausgeordnet; d wird aus der Menge X_t der DA-Komponenten entfernt.

- Ein **anonymes Depot** $d \in X_t$ wird mit Auflösung der DA-Komponente $x \in X_t$, die durch die Abbildung γ_t gemäß (2.17) mit $\gamma_t(d)$ festgelegt ist, aufgelöst. d sei mit dem Generierungsausdruck $\texttt{new}(Z, G)$ erzeugt worden.
  $x \triangleq \gamma_t(d)$ ist die DA-Komponente, die den Zeiger-Generator Z für die Zeiger, die auf d zeigen können, als lokale N-Komponente enthält. Alle Zeiger, die den Wert $\zeta(d)$ haben können, werden spätestens mit x aufgelöst. Damit ist d nicht mehr von außen über seine Zugriffsoperationen benutzbar und kann somit aufgelöst werden. Mit seiner Auflösung wird d aus den Struktur-Relationen δ_t und ε_t ausgeordnet; die Menge X_t der DA-Komponenten wird um d verkleinert.

- Ein **benannter K-Akteur** $k \in X_t$ wird mit Auflösung der DA-Komponente $x \in X_t$, zu der k lokale N-Komponente ist, $(k, x) \in \widetilde{\lambda}_t$, aufgelöst.
  Diese Auflösungsregel läßt sich analog zur Regel für die Auflösung benannter Depots motivieren. Es ist gewährleistet, daß alle DA-Komponenten die Kommunikationsoperationen auf dem K-Akteur k aufrufen können, spätestens mit k aufgelöst werden. Ist x ein Depot, so folgt aus (3.16), daß x und die DA-Komponente y, zu der k unmittelbar π-*innen* eingeordnet ist, verschieden sind. k wird mit Ausführung der Operation zur Abschlußsynchronisation von k in der Synchronisationsphase der kanonischen Operation $op(y)$ aus $\widetilde{\pi}$ ausgeordnet. Mit Regel (3.27) für die Abschlußsynchronisation und den Auflösungsregeln ist gewährleistet, daß die Operation zur Abschlußsynchronisation von k zeitlich vor seiner Auflösung ausgeführt wird und k damit vor seiner Auflösung aus der Ausführungs-Relation ausgeordnet wird. Mit seiner Auflösung wird k aus den Struktur-Relationen δ_t, λ_t und ε_t ausgeordnet; die Menge X_t der DA-Komponenten wird um k verkleinert.

- Ein **anonymer K-Akteur** $k \in X_t$ wird mit Auflösung der DA-Komponente $x \in X_t$, die durch die Abbildung γ_t gemäß (2.17) mit $\gamma_t(k)$ festgelegt ist, aufgelöst.
  k wird mit Ausführung der Operation zur Abschlußsynchronisation von k in der Synchronisationsphase der kanonischen Operation $op(y)$, mit $(k, y) \in \widetilde{\pi}$, aus $\widetilde{\pi}$ ausgeordnet. Die Auflösungsregel läßt sich analog zur Regel für die Auflösung anonymer Depots motivieren. Nach Auflösung von x und damit auch von k können keine Zeiger mehr existieren, die den Wert $\zeta(k)$ haben. Ist x ein Depot, sind x und die DA-Komponente y, zu der k unmittelbar π-*innen* eingeordnet ist, verschieden. Es ist jedoch gewährleistet, daß die Operation zur Abschlußsynchronisation von k in der Abschlußphase von $op(y)$ zeitlich vor der Auflösung von k ausgeführt wird. Mit der Auflösung von k wird k aus den Struktur-Relationen δ_t und ε_t ausgeordnet; die Menge X_t der DA-Komponenten wird um k verkleinert.

Die Regeln für die Auflösung von DA-Komponenten lassen sich zu folgender Regel zusammenfassen. Sie bezieht sich auf die Konfiguration $\mathcal{S}_t$ des Systems $\mathcal{S}$ zu einem Zeitpunkt t, in dem eine DA-Komponente $x \in X_t$ in den Zustand T überführt bzw. aufgelöst wird.

(3.32) Mit der Überführung der DA-Komponente $x \in X_t$ in den Zustand T werden alle M-Akteure $y \in X_t$ mit $(y, x) \in \tilde{\pi}_t$ aufgelöst.

Mit der Auflösung von x werden

- alle benannten K-Akteure und alle benannten Depots $b \in X_t$, für die $(b, x) \in \widetilde{\lambda}_t$ gilt, sowie
- alle anonymen K-Akteure und alle anonymen Depots $a \in X_t$, für die $x = \gamma_t(a)$ gilt,

aufgelöst.

Die Auflösungsregeln für DA-Komponenten haben zur Konsequenz, daß mit der Auflösung einer S-Order oder eines M-Akteur x, alle zu x λ-*inneren* benannten Depots und benannten K-Akteure sowie alle zu x ε-*inneren* anonymen Depots und anonymen K-Akteure aufgelöst werden.

Mit 33 wird der Zusammenhang zwischen der Definition (2.25) der Struktur-Relation ε-*innen*, die die Lebenszeitabhängigkeiten zwischen den DA-Komponenten eines Systems erfaßt, und den Regeln für die Auflösung von DA-Komponenten verdeutlicht.

(3.33) Sei $(x, y) \in \widetilde{\varepsilon}_t$. Dann wird x

- mit Auflösung von y aufgelöst, falls x Depot oder K-Akteur ist,
- mit Überführung von y in den Zustand T aufgelöst, falls x M-Akteur ist und
- mit der Überführung von x in den Zustand T aufgelöst, falls x eine S-Order ist.

Mit der Erklärung der Auflösung von DA-Komponenten und den vorangehenden Abschnitten sind die Möglichkeiten zur dynamischen Veränderung eines Systems beschrieben. Zusammen mit Kapitel 2 sind damit die Konzepte des verfolgten Ansatzes zur Konstruktion verteilter Systeme erklärt.

Kapitel 4

Spezielle Aspekte

4.1 Die Ausführungsumgebung

Die Akteure eines Systems $\mathcal{S}$ führen Berechnungen durch, indem sie Operationen auf Komponenten ausführen. Für jede Komponente $y \in X_t$ läßt sich die Menge der von y zum Zeitpunkt t benutzbaren Komponenten angeben. Diese Menge wird **Ausführungsumgebung** der DA-Komponente y zum Zeitpunkt t genannt und mit $U_t(y)$ bezeichnet. $U_t(y)$ ist die Teilmenge der Menge aller Komponenten des Systems $\mathcal{S}_t$, die bei Ausführung von $op(y)$ konzeptionell nutzbar sind.

Mit Definition (4.1) wird für eine DA-Komponente y festgelegt, welche lokalen Komponenten der DA-Komponente x, mit $(y, x) \in \tilde{\delta}_t$, für y **sichtbar** sind. Die Menge der sichtbaren Komponenten für y wird durch die sequentielle, textuelle Einordnung des Generators von y festgelegt.

Definition 4.1.:

Sei x DA-Komponente und G ein Generator mit $G \in L_0(x)$. Sei $\leq_{L_x}$ die lineare Ordnung auf den lokalen Komponenten von x, die der sequentiellen, textuellen Aufschreibung der Komponentendeklarationen des Generatorprogramms von x entspricht.
Sei y eine Inkarnation bezüglich des Generators G, dann gilt:
Der für y **sichtbare Teil** von x, $Visible(x, y, G)$, ist definiert durch

$$Visible(x, y, G) \triangleq \{z \mid z \in \tilde{L}(x) \ \wedge \ z \leq_{L_x} G\}.$$

□

Gemäß der Festlegungen von (4.1) gilt, daß für eine Komponente y diejenigen Komponenten von x mit $(y, x) \in \tilde{\delta}_t$ sichtbar sind, die der Deklaration des Generators G von y gemäß der linearen Ordnung $\leq_{L_x}$ textuell voranstehen, d.h. die im Programmtext vor G stehen.

Der aus der δ-Relation ableitbare Beitrag zur Ausführungsumgebung einer DA-Komponente y ist durch Definition (4.2) festgelegt.

Definition 4.2.:

Seien $x, y \in X_t$ DA-Komponenten und $G \in \tilde{L}(x)$ der Generator von y.

$$U^\delta(x,y,G) \triangleq \begin{cases} Visible(x,y,G) & \text{falls } x \text{ die Hauptkomponente ist,} \\ Visible(x,y,G) \cup U^\delta(z,x,W) & \text{sonst, mit } (x,z) \in \tilde{\delta}_t \text{ und} \\ & \quad W \text{ ist Generator von } x. \end{cases}$$

□

Gemäß Definition (4.2) werden in dem δ-Beitrag für die Ausführungsumgebung von y entlang der δ-Struktur alle DA- und DE-Komponenten 'eingesammelt', die auf der Basis von δ für y sichtbar sind.[1] Mit $U^\delta(x,y,G)$ ist die Menge aller unmittelbar sichtbaren Komponenten der Ausführungsumgebung von y erfaßt. In Abhängigkeit von der Komponentenart sind mittelbar über den direkten Zugriff auf eine Komponente weitere Komponenten, wie z.B. die mit dem Attribut E versehenen Komponenten eines Depots, nutzbar und gehören zur Ausführungsumgebung von y. Ausgehend von $U^\delta(x,y,G)$ lassen sich die mittelbar benutzbaren Komponenten bestimmen. Zunächst wird dazu die Abbildung $usable_t$ eingeführt, die für jede Komponente x des Systems die Menge der mittelbar über die Nutzung der Komponente x nutzbaren Komponenten erfaßt. Im gegensatz zu dem bisher gesagten, kann im folgenden die Komponente x entweder eine DA- oder eine DE-Komponente sein.

Definition 4.3.:

Sei K_t die Menge aller DA- und DE-Komponenten des Systems zum Zeitpunkt t.

$$usable_t : K_t \longrightarrow \mathcal{P}(K_t)$$

$$usable_t(x) \triangleq \begin{cases} \{x\} \cup \bigcup_{i=1}^{n} usable_t(k_i) & \text{falls } x \text{ ein Depot, eine Record- oder eine} \\ & \text{Feldkomponente ist, mit} \\ & k_1, \ldots, k_n \text{ sind die exportierten Depot-,} \\ & \text{die Record- oder die Feld- Komponenten;} \\ \{x\} \cup \{k_1, \ldots, k_n\} & \text{falls } x \text{ ein K-Akteur ist} \\ & \text{mit } k_1, \ldots, k_n \text{ sind die exportierten Komponenten;} \\ \{x\} \cup usable_t(z) & \text{falls } x \text{ ein Zeiger ist, mit} \\ & \zeta_t(x) \neq nul \ \wedge \ \zeta_t(x) = z \\ \{x\} & \text{sonst.} \end{cases}$$

□

Die Ausführungsumgebung $U_t(y)$ einer Komponente $y \in X_t$ ist dann durch (4.4) gegeben.

Definition 4.4.:

Seien x, y DA-Komponenten und $G \in \tilde{L}(x)$ der Generator von y.

$$U_t(y) \triangleq \bigcup_{w \in U^\delta(x,y,G)} usable_t(w) \ \cup \bigcup_{w \in L_0(y)} usable_t(w).$$

□

[1] Überdeckungen in Folge von Namensgleichheit werden hier nicht explizit behandelt. Es gilt, daß bei Namensgleichheit diejenige Komponente in der Ausführungsumgebung sichtbar ist, die gemäß der linearen Ordnung $\leq$ die größte der Komponenten gleichen Namens ist.

Die Ausführungsumgebung $U_t(y)$ einer Komponente y enthält alle bei der Ausführung von $op(y)$ benutzbaren DE- und DA-Komponenten. Dazu gehören insbesondere alle lokalen Komponenten, einschließlich der Parameter, von y.

Beispiel:

In Abbildung 4.1 ist ein Beispielsystem angegeben. Für die S-Order *so*1 wird beispielhaft die Ausführungsumgebung erarbeitet.

1. Es ist $U_t(so)$ zu bestimmen. Der Generator *S_Order_Gen_2* von *so* ist in *ma* deklariert. Es ergibt sich also gemäß Definition (4.2):

$$U_t(so) = \bigcup_{w \in U^\delta(ma, so, S\_Order\_Gen\_2)} usable(w) \cup \bigcup_{w \in L_0(so)} usable_t(w)$$

2. Die δ-Ausführungsumgebung $U^\delta(ma, so, S\_Order\_Gen\_2)$ ist gegeben durch:

$$\begin{aligned} & U^\delta(ma, so, S\_Order\_Gen\_2) = \\ = \;& Visible(ma, so, S\_Order\_Gen\_2) \cup U^\delta(hk, ma, M\_Akteur\_Gen) \\ = \;& Visible(ma, so, S\_Order\_Gen\_2) \cup Visible(hk, ma, M\_Akteur\_Gen) \\ = \;& \{S\_Order\_Gen\_2, Depot\_Ref, Depot\_Ref\_Gen, Depot\_Gen\_2, k\} \cup \\ & \{M\_Akteur\_Gen, K\_Akteur\_Ref, K\_Akteur\_Ref\_Gen, K\_Akteur\_Gen, \\ & \quad Depot\_1, Depot\_Gen\_1\} \end{aligned}$$

3. Für jede Komponente $w \in U^\delta(ma, so, S\_Order\_Gen\_2)$ ist die Menge $usable_t(w)$ der mittelbar benutzbaren Komponenten gemäß Definition (4.3) zu bestimmen.

$$\begin{aligned} usable_t(S\_Order\_Gen\_2) &= \{S\_Order\_Gen\_2\} \\ usable_t(Depot\_Ref) &= \{Depot\_Ref\} \cup usable_t(ad) \\ usable_t(ad) &= \{ad\} \cup usable_t(Depot\_3) \cup usable_t(S\_Order\_Gen\_3) \\ usable_t(Depot\_3) &= \{Depot\_3\} \cup usable_t(S\_Order\_Gen\_1(3)) \\ usable_t(S\_Order\_Gen\_1(3)) &= \{S\_Order\_Gen\_1(3)\} \\ usable_t(S\_Order\_Gen\_3) &= \{S\_Order\_Gen\_3\} \\ usable_t(Depot\_Ref\_Gen) &= \{Depot\_Ref\_Gen\} \\ usable_t(Depot\_Gen\_2) &= \{Depot\_Gen\_2\} \\ usable_t(k) &= \{k\} \cup \{K\_Order\_Gen(2)\} \\ usable_t(M\_Akteur\_Gen) &= \{M\_Akteur\_Gen\} \\ usable_t(K\_Akteur\_Ref) &= \{K\_Akteur\_Ref\} \cup usable_t(ak) \\ usable_t(ak) &= \{ak\} \cup \{K\_Order\_Gen(1)\} \\ usable_t(K\_Akteur\_Ref\_Gen) &= \{K\_Akteur\_Ref\_Gen\} \\ usable_t(K\_Akteur\_Gen) &= \{K\_Akteur\_Gen\} \\ usable_t(Depot\_1) &= \{Depot\_1\} \cup usable_t(S\_Order\_Gen\_1(1)) \\ usable_t(S\_Order\_Gen\_1(1)) &= \{S\_Order\_Gen\_1(1)\} \\ usable_t(Depot\_Gen\_1) &= \{Depot\_Gen\_1\} \end{aligned}$$

4. Für die Menge $\tilde{L}(so)$ der zu *so* lokalen Komponenten gilt:

$$\tilde{L}(so) = \{DE\_Gen, DE\}$$

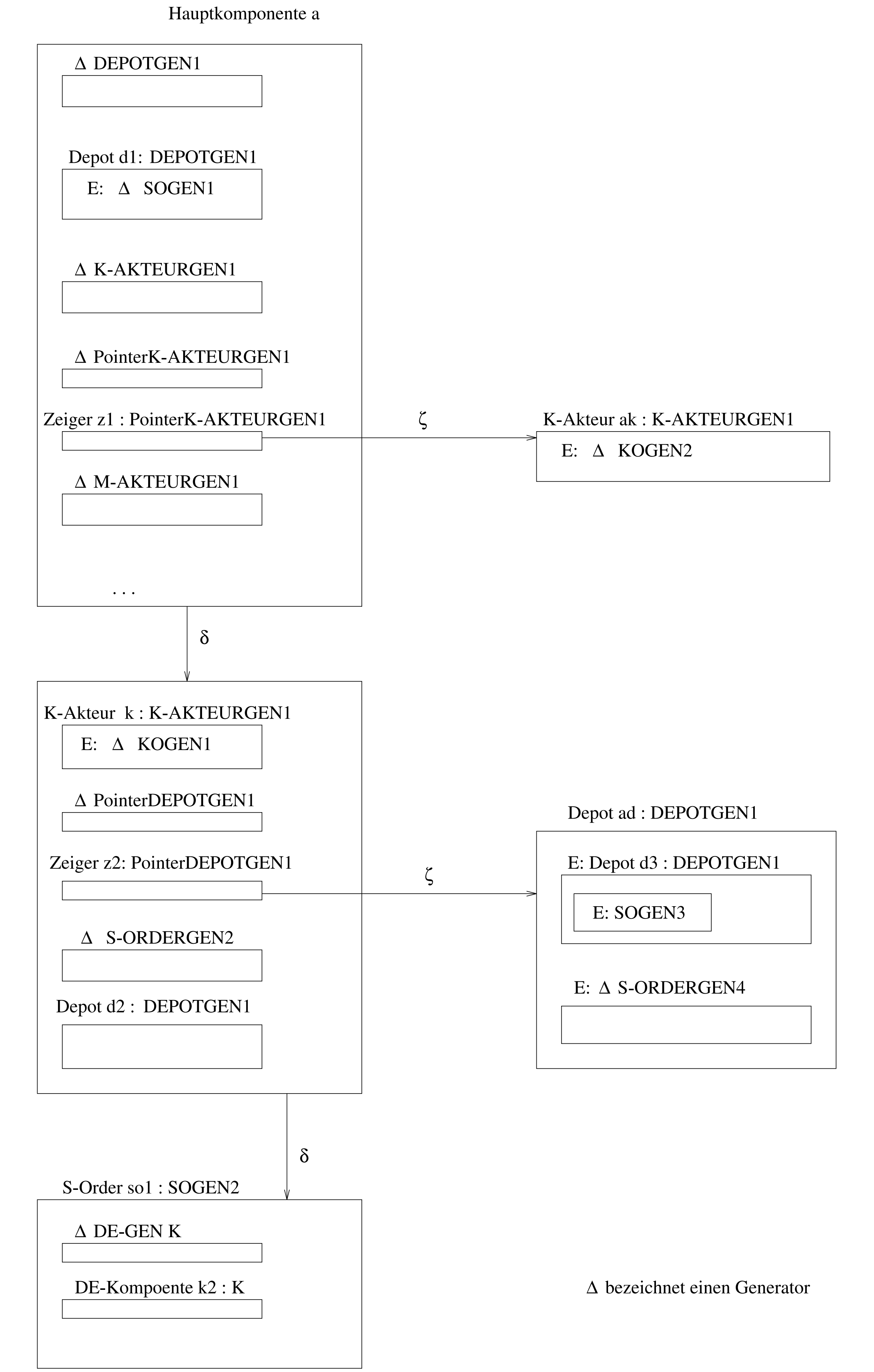


Abbildung 4.1: Beispiel zur Ausführungsumgebung

5. Die Ausführungsumgebung von *so* ergibt sich damit als:

$U_t(so) = \{$*S_Order_Gen_2, Depot_Ref, ad, Depot_3, S_Order_Gen_1(3), S_Order_Gen_3, Depot_Ref_Gen, Depot_Gen_2, k, K_Order_Gen(2), M_Akteur_Gen, K_Akteur_Ref, ak, K_Order_Gen(1), K_Akteur_Ref_Gen, K_Akteur_Gen, Depot_1, S_Order_Gen_1(1), Depot_Gen_1*$\} \cup \{$*DE_Gen, DE*$\}$

$\Diamond$

Die mit $U_t(y)$ für eine Komponente y definierte Menge von Komponenten beschreibt die Ausführungsumgebung von y vollständig und detailliert. Ist man nur an den für y benutzbaren DA-Komponenten des Systems interessiert, so ist es sinnvoll, die Menge $U_t(y)$ zur Menge $U_t^{DA}(y)$ zu vergröbern, wobei $U_t^{DA}(y)$ nur DA-Komponenten enthält.

Definition 4.5.:

Sei $y \in X_t$ DA-Komponente.

$$U_t^{DA}(y) \triangleq \{w \mid w \in U_t(y) \ \wedge \ w \in X_t\}$$

$\square$

4.2 Import-Export-Beschränkungen

In diesem Abschnitt werden Konzepte eingeführt, die es ermöglichen, die Ausführungsumgebung von Komponenten einzuschränken.

Die Definitions-Struktur eines Systems leistet wesentliche Beiträge zur Ausführungsumgebung einer DA-Komponente. Die Gründe hierfür liegen darin, daß eine DA-Komponente x die Eigenschaften (d.h. die Operationen) der lokalen N-Komponenten aller DA-Komponenten y, zu denen x δ-innen ist, benutzen kann. Wir sagen, daß x diese Komponenten **δ-importiert**.

Gemäß der in Kapitel 2 eingeführten Komponentenkonzepte gilt, daß eine DA-Komponente y alle ihre lokalen N-Komponenten den Komponenten x, die δ-innen zu y existieren können, zur Nutzung zur Verfügung stellt. Wir sagen y **δ-exportiert** ihre lokalen N-Komponenten.
In diesem Abschnitt werden Konzepte eingeführt, mit denen diese δ-Importe und -Exporte eingeschränkt werden können. Dabei sei (X_t, δ_t) die Definitions-Struktur eines Systems zum Zeitpunkt t.

(4.6) Sei $y \in X_t$ eine gemäß der Konzepte von Kapitel 2 konstruierte DA-Konponente.
Dann gilt für alle $k \in L_0(y)$: y *δ-exportiert* k. Die Komponente y heißt *δ-export-offen*.

Durch (4.6) ist eine Eigenschaft der DA-Komponente y definiert. Wenn y die lokale N-Komponente k δ-exportiert, dann ist damit zugelassen, daß eine DA-Komponente x die Komponente $k \in L_0(y)$ benutzt, d.h. eine der für k definierten äußeren Operationen aufruft. Dabei ist vorausgesetzt, daß x δ-innen zu y ist.

(4.7) Seien $x, y \in X_t$ gemäß der Konzepte von Kapitel 2 konstruierte DA-Konponenten, d.h. y ist δ-export-offen, und es gelte $(x, y) \in \delta_t$.
Dann gilt für alle $k \in L_0(y)$: x *δ-importiert* k.

Durch (4.7) ist eine Eigenschaft der DA-Komponente x definiert. Sie setzt voraus, daß die DA-Komponente y die lokale N-Komponente k δ-exportiert und x δ-innen zu y ist.

Im folgenden werden **Beschränkungen** für die δ-Importe und δ-Exporte von DA-Komponenten eingeführt. Die Beschränkungen sind Klasseneigenschaften. Ein DA-Generator definiert DA-Komponenten mit δ-Import- oder δ-Exportbeschränkungen, wenn in der Klassendefinition entsprechende **Attribute** auftreten. Bezüglich eines Generators, der DA-Komponenten mit δ-Import- oder δ-Exportbeschränkungen definiert, können Inkarnationen so erzeugt werden, wie es in Abschnitt 2.2.4 festgelegt wurde. Die so erzeugten DA-Komponenten werden nach den in Kapitel 3 angegebenen Regeln in die Definitions-Struktur eines Systems eingeordnet.

Sei (X_t, δ_t) weiterhin die Definitions-Struktur eines Systems zum Zeitpunkt t, wobei X_t DA-Komponenten mit δ-Import- und δ-Exportbeschränkungen enthalten kann.

Zunächst werden die Konzepte für **δ-Exportbeschränkungen** eingeführt. Mit δ-Exportbeschränkungen können für die lokalen N-Komponenten einer DA-Komponente die δ-Exporte auf *unmittelbar-δ-innere* DA-Komponenten eingeschränkt werden. δ-Exportbeschränkungen werden festgelegt, indem ein DA-Generator mit dem **Attribut** HC (Hidden and Closed) und lokale N-Komponenten mit dem *Attribut* HE (Hidden and partial Export opened) definiert werden. Wir nennen HC und HE *δ-export-beschränkende Attribute.*

(4.8) (a) Sei $y \in X_t$ und sei G der Generator von y. Wenn G mit dem Attribut HC attributiert ist, so ist y *δ-export-beschränkt.*

(b) Sei $y \in X_t$ *δ-export-beschränkt*, so gilt für jede Komponente $k \in L_0(y)$:

- falls k mit dem Attribut HE definiert ist, so δ-exportiert y die Komponente k zu $x \in X_t$ mit $(x, y) \in \delta_t$;
- falls k nicht mit dem Attribut HE definiert ist, so δ-exportiert y die Komponente k zu $x \in X_t$ mit $(x, y) \in \widetilde{\delta}_t$, falls k nicht mit dem Attribut HE definiert ist.

Wenn in einer Klassendefinition kein δ-export-beschränkendes Attribut auftritt, dann definiert der DA-Generator eine Klasse von δ-export-offenen DA-Komponenten.

Sei nun die DA-Komponente y δ-export-beschränkt. Dann können lokale N-Komponenten von y mit dem Attribut HE definiert sein. Wenn keine der lokalen N-Komponenten von y mit dem Attribut HE definiert ist, dann ist y *δ-export-abgeschlossen.* Ist $k \in L_0(y)$ und ist y δ-export-abgeschlossen, so δ-exportiert y k zu DA-Komponenten, die unmittelbar δ-innen zu y existieren können, und dies gilt für alle $k \in L_0(y)$. δ-Exporte zu DA-Komponenten x mit $(x, y) \in \delta$ und $(x, y) \notin \widetilde{\delta}$ sind unzulässig. Sei nun y δ-export-beschränkt und $k \in L_0(y)$ mit dem Attribut HE definiert. Dann δ-exportiert y k zu DA-Komponenten, die δ-innen zu y existieren können.

Anschließend werden Konzepte für **δ-Importbeschränkungen** eingeführt. Mit ihnen können potentielle δ-Importe für DA-Komponenten explizit festgelegt werden. δ-Importbeschränkungen werden festgelegt, indem ein DA-Generator mit dem **Attribut** IC (Import Closed) und δ-Importe mit dem **Attribut** I (Import) definiert werden.

Ein DA-Generator definiert eine Klasse von δ-import-beschränkten DA-Komponenten, wenn er mit IC definiert ist. Wenn ein Generator mit IC definiert ist, dann können δ-importierte Elemente dieser Klasse in der Definition des Generators mit dem Attribut I definiert werden. Wir nennen IC und I **δ-import-beschränkende Attribute**. Wenn in einer Klassendefinition kein δ-import-beschränkendes Attribut auftritt, dann definiert der DA-Generator eine Klasse von δ-import-offenen DA-Komponenten.

(4.9) Sei $x \in X_t$ und sei der Generator G für x mit dem Attribut IC attributiert, dann ist x *δ-import-beschränkt*. Mit dem Attribut I kann eine Komponente k importiert werden, falls gilt: $k \in L_0(y)$, $(x, y) \in \delta_t$ und k wird zu x von y δ-exportiert,

Wenn x ohne δ-Importe mit dem Attribut I definiert ist, dann ist x *δ-import-abgeschlossen*. Ist x δ-import-abgeschlossen, so sind für x δ-Importe unzulässig; x ist bzgl. δ-Importen von allen DA-Komponenten, zu denen x δ-innen ist, isoliert. Sei nun y eine DA-Komponente mit $(x, y) \in \delta_t$, die lokale N-Komponenten zu x δ-exportiert. Wenn x die benannte DA-Komponente y mit dem Attribut I definiert, dann sind die lokalen N-Komponenten, die y zu x δ-exportiert, für x als δ-Importe zulässig. Wenn y die lokale N-Komponente $k \in L_0(y)$ zu x δ-exportiert, und wenn x k mit dem Attribut I definiert, dann ist k als δ-Import für x zulässig.

Eine DA-Komponente kann δ-export- und δ-import-offen sein; sie kann δ-export-offen und δ-import-beschränkt, δ-export-beschränkt und δ-import-offen oder δ-import- und δ-export-beschränkt sein.

Wie bereits festgelegt wurde, werden δ-export- oder δ-import-beschränkte DA-Komponenten wie δ-export- und δ-import-offene DA-Komponenten in die Definitions-Struktur einer System-Konfiguration eingeordnet. Die Definitions-Struktur bleibt zwar Basis für die Ausführungsumgebungen der DA-Komponenten, ihre Beiträge zu den Ausführungsumgebungen sind jedoch durch die δ-Export- und δ-Importbeschränkungen festgelegt.

Die von einer DA-Komponente x importierbaren Komponenten ergeben sich entsprechend dem bisher Gesagten aus den δ-Importbeschränkungen von x sowie den δ-Export- und δ-Importbeschränkungen der Komponenten, zu denen x δ-innen ist. Das bedeutet, daß jede Export- bzw. Importbeschränkung einer Komponente y mit $(x, y) \in \delta_t$ die Menge der von x benutzbaren Komponenten entsprechend einschränkt. Analog zu der mit dem Objekt-Prinzip durchgesetzten Daten-Kapselung nach außen ist folglich mit den δ-Export- und δ-Importbeschränkungen eine entsprechende Möglichkeit zur Einschränkung von Nutzungsmöglichkeiten nach innen gegeben.[2]

[2] In weiterführenden Arbeiten soll das hier vorgestellte Konzepterepertoire erweitert werden um ein Konzept, das es erlaubt, Generatoren unvollständig zu definieren und inkrementell zu einem späteren Zeitpunkt zu vervollständigen. Die durch die Export- und Importbeschränkungen zur Verfügung stehenden Möglichkeiten sind insbesondere bei der Anwendung solcher unvollständigen Definition sinnvoll, um die Möglichkeiten zur dynamischen Entwicklung eines Systems zu beschränken, bzw. den Programmierer, der die unvollständige Definition vervollständigt, zu beschränken.

4.3 Die Lebenszeit-orientierte Akteur-Struktur

Mit den in Kapitel 2 definierten System-Konfigurationen lassen sich die Eigenschaften und damit der Entwicklungsstand eines Systems $\mathcal{S}$ in jedem Zeitpunkt seiner Existenz beschreiben. Ist die Anzahl der Komponenten, aus denen das System $\mathcal{S}$ besteht, groß und sind die Eigenschaften und Abhängigkeiten der Komponenten komplex, ist es sinnvoll, sich einerseits auf bestimmte Eigenschaften und Abhängigkeiten zu beschränken, und andererseits Komponenten so zu Komponenten-Mengen zusammenzufassen, daß die betrachteten Abhängigkeiten zu Abhängigkeiten zwischen diesen Komponenten-Mengen vergröbert werden. Mit diesem Vorgehen ergeben sich Vergröberungen der System-Strukturen. Die Gesichtspunkte, nach denen vergröbert wird, können je nach interessierenden Eigenschaften eines Systems unterschiedlich sein. Zum Beispiel sind zur Lösung der bei der Realisierung der mit den vorgestellten Konzepten konstruierten Systeme anfallenden Verwaltungsprobleme unterschiedliche an die jeweilige Verwaltungsaufgabe angepaßte Vergröberungskriterien von Interesse.

Da die Akteure eines Systems $\mathcal{S}$ zum Zeitpunkt t mit der Konfiguration $\mathcal{S}_t$ die Komponenten sind, die Berechnungen durchführen und damit eine zentrale Rolle spielen, besteht eine naheliegende Vorgehensweise darin, sich auf Abhängigkeiten zwischen Akteuren zu beschränken und entsprechend jede passive DA-Komponente der Konfiguration $\mathcal{S}_t$ eindeutig einem der Akteure von $\mathcal{S}_t$ zuzuordnen. Ein Kriterium für diese Zuordnung, das insbesondere im Rahmen von Verwaltungsaufgaben von Interesse ist, sind die Lebenszeitabhängigkeiten zwischen den DA-Komponenten so wie sie in der Lebenszeit-Struktur erfaßt werden. Im folgenden wird die Lebenszeit-orientierte Akteur-Struktur erklärt, die sich aus der Beschränkung auf die Lebenszeit-Abhängigkeiten zwischen DA-Komponenten ergibt und in der jedem Akteur die von ihm lebenszeitmäßig abhängigen passiven DA-Komponenten zugeordnet sind.

Dazu sei $\mathcal{S}_t = (X_t, \delta_t, \alpha_t, \lambda_t, \zeta_t, \varepsilon_t)$ die Konfiguration des Systems $\mathcal{S}$ zum Zeitpunkt t und A_t die Menge der Akteure von X_t. Jede DA-Komponente der Konfiguration $\mathcal{S}_t$ wird nun genau einem der Akteure aus A_t zugeordnet. Diese Zuordnung orientiert sich hier an der in Abschnitt 2.3.5 definierten Lebenszeit-Struktur (X_t, ε_t). Dementsprechend wird eine passive DA-Komponente $p \in X_t$ dem Akteur $a \in A_t$ zugeordnet, von dem sie lebenszeitmäßig abhängig ist. Um diese Zuordnung formal zu erfassen, wird zunächst folgende Menge definiert:

(4.10) Sei $a \in A_t$. Dann ist $A_t^{\varepsilon}(a) \triangleq \{b \in A_t : (b, a) \in \varepsilon_t\}$ die Menge der Akteure von A_t, die *ε-innen* zu dem Akteur a sind.

Damit ergibt sich für die eindeutige Zuordnung der passiven DA-Komponenten zu den Akteuren folgende Regel:

(4.11) Sei $p \in X_t$ Order oder Depot. p wird dem Akteur a zugeordnet, für den gilt:

$$(p, a) \in \varepsilon_t \text{ und es gibt kein } b \in A_t^{\varepsilon}(a) \text{ mit } (p, b) \in \varepsilon_t.$$

Da jede DA-Komponente für ihre gesamte Lebenszeit in die Lebenszeit-Struktur eingeordnet ist und es für jede DA-Komponente $x \in X_t$ genau eine DA-Komponente $y \in X_t$ mit $(x, y) \in \tilde{\varepsilon}_t$ gibt, läßt sich gemäß Regel (4.11) jede DA-Komponente eindeutig einem Akteur aus A_t zuordnen. Für die Menge $Q_t(a)$ der einem Akteur $a \in A_t$ gemäß der Lebenszeit-orientierten Akteur-Struktur zugeordneten DA-Komponenten ergibt sich:

(4.12) $$Q_t(a) \triangleq \{a\} \cup \{p \in X_t - A_t : (p,a) \in \varepsilon_t \text{ und es gibt kein } b \in A_t^\varepsilon(a) \text{ mit } (p,b) \in \varepsilon_t\}.$$

Die Menge $Q_t(a)$ enthält neben dem Akteur a genau die passiven DA-Komponenten von X_t, die a gemäß Regel (4.11) zugeordnet sind. Da diese Zuordnung eindeutig ist, gilt:

(4.13) $$Q_t \triangleq \{Q_t(a) : a \in A_t\} \text{ ist eine Zerlegung von } X_t.$$

Um nun die Lebenszeit-Abhängigkeiten vergröbert auf die Mengen $Q_t(a)$ mit $a \in A_t$ zu beschreiben, wird auf Q_t die Relation *ε^v-innen*, kurz ε^v, definiert. ε^v ist die transitive Hülle der Relation *unmittelbar-ε^v-innen*, die wie folgt definiert ist:

Definition 4.14.:

Seien $Q_t(a), Q_t(b) \in Q_t$.
$Q_t(a)$ ist *unmittelbar-ε^v-innen* zu $Q_t(b)$ – $(Q_t(a), Q_t(b)) \in \tilde{\varepsilon}_t^v$ – $\stackrel{\triangle}{\Longleftrightarrow}$ es ein $x \in Q_t(b)$ mit $(a,x) \in \tilde{\varepsilon}_t$ gibt, wobei $\tilde{\varepsilon}_t$ gemäß (2.25) definiert ist.

□

Das Paar (Q_t, ε_t^v) ist die **Lebenszeit-orientierte Akteur-Struktur** des Systems $\mathcal{S}$ zum Zeitpunkt t. Da es für jeden Akteur $a \in A_t$, mit Ausnahme der Hauptkomponente $\underline{a}$, genau eine Komponente $x \in X_t$ mit $(a,x) \in \tilde{\varepsilon}_t$ gibt und Q_t eine Zerlegung von X_t ist, ergibt sich, daß $(Q_t, \tilde{\varepsilon}_t^v)$ ein Baum mit der Wurzel $Q_t(\underline{a})$ ist. Die Lebenszeit-orientierte Akteur-Struktur ist eine Vergröberung der Lebenszeit-Struktur auf die Akteure des Systems $\mathcal{S}$ zum Zeitpunkt t. Sie beschreibt damit die Schachtelung der Lebenszeiten der Komponenten vergröbert auf die Akteure.

Beispiel:
In Abbildung 4.2 ist zunächst die Zerlegung Q_t für die System-Konfiguration aus Abbildung 2.7 dargestellt. Die Abbildung 4.3 zeigt die sich ergebende Lebenszeit-orientierte Akteur-Struktur (Q_t, ε_t^v). ◊

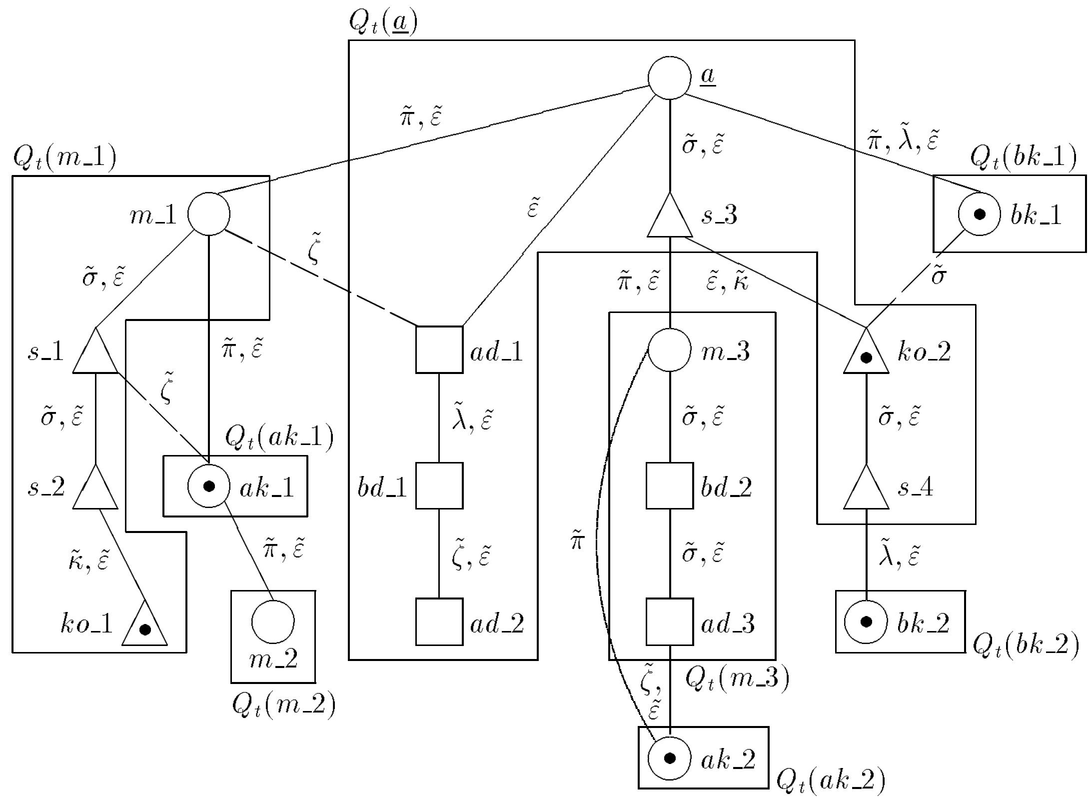


Abbildung 4.2: Die Zerlegung Q_t für die System-Konfiguration aus Abbildung 2.5

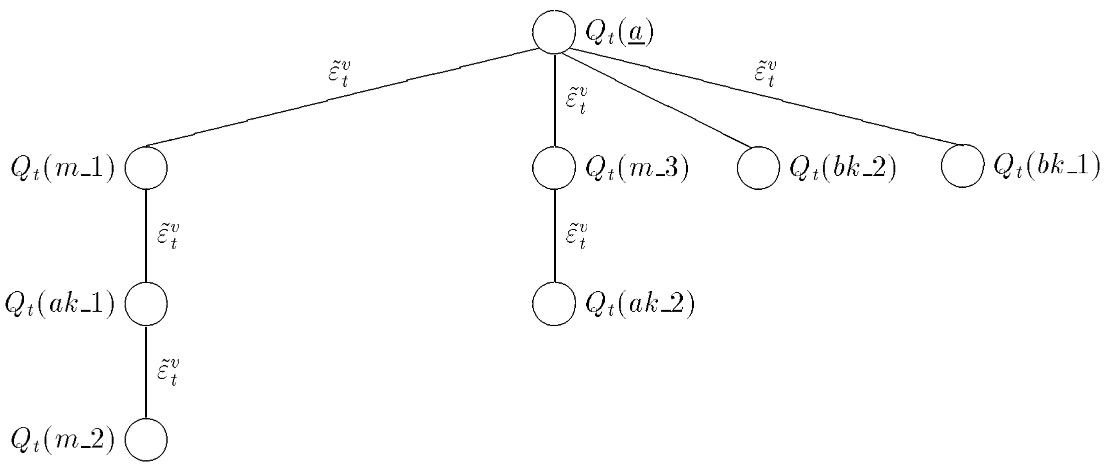


Abbildung 4.3: Der Baum der Lebenszeit-orientierten Akteur-Struktur zu Abbildung 4.1

Kapitel 5

Zusammenfassung und Ausblick

In dem vorliegenden Bericht wurde ein Vorrat von Konzepten eingeführt und erklärt, mit dem verteilte Systeme als konzeptionell homogene Systeme konstruiert werden können. Den Konzepten liegen das Objektprinzip, das Schachtelungsprinzip und das Prinzip der Klassenbildung zugrunde. Systeme werden konstruiert, indem System-Komponenten konstruiert werden.
Ein System besteht aus **DA-Komponenten**, die dadurch ausgezeichnet sind, daß sie mit konzeptionell wohlunterschiedenen äußeren und inneren Eigenschaften definiert werden können. Es gibt drei Arten von DA-Komponenten, nämlich Order, Depots und Akteure. Order sind operationsdefinierende DA-Komponenten. Depots sind speicherdefinierende DA-Komponenten. Akteure sind aktive DA-Komponenten; sie sind fähig, Operationen auszuführen. Alle Operationen eines Systems werden von seinen Akteuren ausgeführt. Zu jeder der drei Arten von DA-Komponenten gibt es Unterarten. Jede DA-Komponente definiert ihre kanonische Operation; sie kann insbesondere weitere DA-Komponenten enthalten.

Ein auf der Basis der zur Verfügung gestellten Konzepte konstruierte System entwickelt sich **dynamisch** aus einer Hauptkomponente, indem Komponenten erzeugt und aufgelöst werden. Die Gesetzmäßigkeiten für diese dynamischen Entwicklungen sind durch die Konzepte und durch die Ausprägungen, welche diese durch die einzelnen Komponentenarten und Komponenten erhalten können, festgelegt.
Der Zustand eines Systems zu einem Zeitpunkt läßt sich vollständig durch die jeweilige **System-Konfiguration** beschreiben. Eine System-Konfiguration beschreibt die Menge der DA-Komponenten, aus denen das System besteht, und die Abhängigkeiten zwischen diesen DA-Komponenten. Die Abhängigkeiten zwischen den DA-Komponenten definieren die Struktur des Systems; sie wird durch Struktur-Relationen beschrieben. Die Definitions-Struktur einer Konfiguration beschreibt die definitorischen Abhängigkeiten zwischen den DA-Komponenten. Die Ausführungs-Struktur einer Konfiguration beschreibt primär die sequentiellen und parallelen Einordnungen der kanonischen Operationen der DA-Komponenten; zusammen mit der Relation ε-innen beschreibt sie zudem die Lebenszeitabhängigkeiten zwischen den DA-Komponenten. Die Lokalitäts- und die Zeiger-Struktur einer Konfiguration beschreiben Abhängigkeiten zwischen DA-Komponenten, aus denen sich zusammen mit der Definitions-Struktur die Ausführungsumgebungen der DA-Komponenten ergeben.

Die den angegebenen Konzepten entsprechende Programmiersprache unterscheidet sich von den für die Konstruktion von Verteilten Systemen entwickelten insbesondere dadurch, daß sie kein Konzept für Plazierungseinheiten, also für Einheiten, die immer vollständig mit einem Stellenrechner zu realisieren sind, kennt. Dem entspricht, daß die hier angegebenen Konzepte für DA-Komponenten-übergreifende Operationen nicht zwischen eng oder lose gekoppelten Komponenten unterscheiden. Das Fehlen von Konzepten für Plazierungseinheiten und das Ignorieren von engen oder losen Kopplungen zwischen Komponenten entspricht der Tatsache, daß derartige Konzepte nicht Problemlösungseigenschaften, sondern Implementierungseigenschaften entsprechen. Für die Realisierung eines Systems, das mit den hier eingeführten Konzepten konstruiert wird, sind natürlich Plazierungseinheiten und die Unterscheidung zwischen eng und lose gekoppelten Komponenten notwendig; damit werden zugleich Eigenschaften des Nachrichten-Transportsystems einer Hardware-Konfiguration relevant. Eine der anstehenden Aufgaben besteht demnach darin, aus DA-Komponenten Plazierungseinheiten zu konstruieren und die Randbedingungen für deren Realisierung festzulegen.

Die erklärten Konzepte zusammen mit der sie konkretisierenden Sprache INSEL sind das Instrumentarium mit dem VP–Systeme mit den für algorithmische Anwendungsproblemlösungen relevanten Eigenschaften konstruiert und beschrieben werden sollen. Ein entsprechendes System ist abstrakt verteilt; es hat, beschrieben durch die jeweilige System–Konfiguration, auf dem gewählten Abstraktiosniveau die Eigenschaften, die zum Verständnis von Problemlösungen und der dafür nötigen parallelen und kooperativen Berechnungen wesentlich sind.
Die Ergebnisse der Arbeiten, über die hier berichtet ist, sind Ausgangsbasis für weitere Arbeiten, die zwei Schwerpunkte haben: Realisierungen von VP–Systemen sowie Ergänzungen und Erweiterungen des jetzt zur Verfügung stehenden Konzepterepertoires.
Dem sprachbasierten Gesamtsystem–Ansatz entsprechend sollen VP–Systeme konstruiert werden, indem mit den festgelegten Konzepten Programme konstruiert und dann ausgeführt werden. Mit den **Arbeiten zur Realisierung von VP–Systemen** werden Konzepte und Verfahren für die verteilten Betriebssysteme entwickelt, die dazu erforderlich sind, die erklärten Programme effizient auf Hardwarekonfigurationen mit vernetzten Stellen auszuführen. Die benötigten Betriebssysteme sollen an die abstrakten Systeme angepaßt und dynamisch anpaßbar sein; ihre Eigenschaften sind dem entspechend aus denen der abstrakten Systeme abzuleiten und mit den jeweils benutzten Hardwarekonfigurationen zu realisieren. Mit den laufenden Arbeiten werden einerseits eine Architektur für anpassungsfähige, verteilte Betriebssysteme und andererseits Experimentalsysteme, die INSEL–Programme ausführen können, entwickelt.
Die **Architektur für ein anpassungsfähiges, verteiltes System** ist aus einer Vergröberung des zu realisierenden abstrakten Systems abgeleitet, nämlich aus der in Kapitel 4.3 erklärten Lebenszeit–orientierten Akteur–Struktur. Diese Vergröberung liefert, wie dort erklärt, für ein abstraktes System jeweils einen Baum, dessen Knoten Zusammenfassungen je eines Akteurs und der ihm nach der Lebenszeit–Struktur zugeordneten DA–Komponenten, Akteur–Sphären genannt, sind. Jeder dieser Akteur–Sphären wird als Betriebssystemkomponente ein Manager zugeordnet; ein Manager hat die Aufgabe, dem Akteur seiner Sphäre die Ausführung seiner Berechnungen zu ermöglichen und die dafür benötigten Komponenten des abstrakten Systems zu realisieren. Das Betriebssystem, das sich ergibt, besteht aus einem Mikrokern für jede Stelle der benutzten Hardwarekonfiguration, der die Basiskonzepte und –dienste zur Nutzung der Stelle zur Verfügung stellt, und den oben erklärten Managern. Jeder Manager wird zusammen mit dem Akteur, dem er zugeordnet ist, auf einer Stelle realisiert. Die Manager werden mit den Akteuren, denen sie zugeordnet sind, erzeugt und aufgelöst; damit wird die Dynamik des abstrakten Systems vergröbert auf die des realisierenden Betriebssystems übertragen. Ebenso wird die Lebenszeit–orientierte Akteur–Struktur auf die Mengen der Manager übertragen. Die Manager realisieren die Komponenten, welche

die Akteure für ihre Berechnungen benötigen; sie verwalten die Ressourcen, die dafür mit der Hardwarekonfiguration zur Verfügung stehen, gemeinsam und verteilt, indem sie der Baumstruktur entsprechend kooperieren.
Die **Experimentalsysteme**, die entwickelt werden, dienen dazu, den verfolgten Ansatz frühzeitig an Anwendungen zu erproben; sie tragen zudem dem beträchtlichen Aufwand, der für die Entwicklung eines verteilten Betriebssystems, das der oben erklärten Architektur entspricht, zu leisten ist, dadurch Rechnung, daß sie Konzepte und Dienste bereits vorhandener Betriebssysteme verwenden. Ein Experimentalsystem stellt eine Ausführungsumgebung für INSEL–Programme auf **vernetzten UNIX–Workstations** zur Verfügung; es benutzt UNIX-Prozesse auf den Stellen und systemweite Threads zur Realisierung der Akteure, der übrigen DA–Komponentenarten und der Operations– und Kooperationskonzepte von INSEL. Ein weiteres Experimentalsystem stellt eine Ausführungsumgebung für INSEL–Programme auf **vernetzten PCs mit Mach–Mikrokernen** zur Verfügung; es benutzt Mach–Tasks und –Threads sowie weitere Mach–Konzepte und –Dienste zur Realisierung von INSEL–Systemen, wobei insbesondere Verfahren zur Speicherverwaltung entwickelt werden, die den oben erklärten Manager–Ansatz mit DSM (Distributed Shared Memory) kombinieren. Die Arbeiten an beiden Experimentalsystemen sind so weit fortgeschritten, daß die bisher erzielten Ergebnisse die Erfolgsaussichten des verfolgten, sprachbasierten Gesamtsystem–Ansatzes klar bestätigen. Insbesondere wird deutlich, was für Realisierungen von leistungsfähigen VP–Systemen auf Hardwarekonfigurationen mit vernetzten Stellen nötig ist: Eine Sprache, die Konzepte zur Verfügung stellt, mit denen Anwendungsproblemlösungen unter Nutzung von Parallelität und Kooperation auf hohem Abstraktionsniveau frei von realisierungsbedingten Beschränkungen formuliert werden können, und ein an diese Konzepte und an die sich mit ihnen ergebenden Anforderungen angepaßtes und anpassungsfähiges, verteiltes Betriebssystem, welches das Potential, das die Hardwarekonfiguration bietet, ausschöpft.
Die weiterführenden **Arbeiten zur Ergänzung** des jetzt zur Verfügung stehenden **Konzepterepertoires** betreffen die Behebung seit längerem erkannter Mängel. Dazu gehört zunächst die Erweiterung von Depots um Synchronisationsausdrücke zur Verbesserung der Einsatzmöglichkeiten der Depots als gemeinsam benutzbare, abstrakte Speicher. Dazu gehört weiter die Einführung unvollständig definierter Generatoren und deren Vervollständigung als konzeptionelle Maßnahmen, mit denen die Handhabung des verfolgten Gesamtsystem–Ansatzes vereinfacht wird.
Die **Arbeiten zur Erweiterung des** jetzt zur Verfügung stehenden **Konzepterepertoires** zielen zum einen darauf ab, Regeln festzulegen, mit denen VP–Systeme konstruiert werden können, die aus Subsystemen mit anwendungsbezogenen Charakteristika bestehen. Zum anderen zielen sie darauf, mit den Subsystemen eine Basis zur Festlegung von Qualitätseigenschaften und für qualitätssteigernde Maßnahmen zu schaffen, und schließlich die erforderlichen Konstruktionen durch geeignetere Konzepte zu unterstützen. Dafür sind ergänzend zu den jetzt benutzten System–Konfigurationen formalisierte Beschreibungen des Verhaltens eines Systems erforderlich, mit denen die Wirkungen ausgeführter und auszuführender Berechnungen so erfaßt und klassifiziert werden können, daß auf dieser Grundlage DA–Komponentenmengen berechnungs– und wirkungsbezogen zusammenfaßbar und gegeneinander abgrenzbar werden. Für ein System, das aus anwendungsbezogenen Subsystemen besteht, sind die Subsyteme die Teile des Systems, an die von Anwendungen Qualitätsanforderungen gestellt werden; sie sind festzulegen und durchzusetzen. Die qualitativen Eigenschaften, die vorrangig interessieren, sind Zuverlässigkeit und (Rechts-)Sicherheit. Für beide ist bekannt und wichtig, daß mit entsprechenden Festlegungen die angestrebten Ziele nur dann erreicht werden, wenn sie für ein System vollständig sind, also alle (beteiligten) Komponenten des Systems lückenlos erfassen. Der verfolgte Gesamtsystem–Ansatz liefert die Voraussetzungen

hierfür, wenn die Subsysteme so definiert werden, daß sie anwendungs– und berechnungsbezogene Vergröberungen des Systems sind.

Literaturverzeichnis

[Ada83] Ada. *The Programming Language Ada Reference Manual*, volume 155 of *Lecture Notes in Computer Science*. Springer–Verlag, Berlin, 1983.

[RW94] R. Radermacher and F. Weimer. INSEL-Sprachbeschreibung. Technischer Bericht, Technische Universität München, 1994.

[Str91] Bjarne Stroustrup. *The C++ Programming Language*. Addison–Wesley, Reading, MA, 2nd edition, 1991.

[Weg87] Peter Wegner. Dimensions of Object–Based Language Design. In Norman Meyrowit, editor, *OOPSLA'87 Conference Proceedings*, pages 168–182, Orlando, Florida, October 4–8 1987.

Index