Siegfried Wendt

Hasso-Plattner-Institut für Softwaresystemtechnik GmbH, Potsdam

Versuch einer ersten Grundlegung für die Ingenieurdisziplin „Softwaresystemtechnik“

Inhalt

1. Einführende Bemerkungen

1.1 Was bedeutet „Grundlegung für eine Ingenieurdisziplin“?

Die Entstehung einer Ingenieurdisziplin ist stets die Konsequenz des Entstehens eines Bedarfs an komplexen Systemen. Die Komplexität solcher Systeme kann nur arbeitsteilig bewältigt werden, und der gewünschte hohe Wirkungsgrad der Arbeitsteilung kann nur erreicht werden durch einen hohen Kommunikationswirkungsgrad. In den klassischen Ingenieurdisziplinen Bauwesen, Maschinenbau und Elektrotechnik wird dieser hohe Kommunikationswirkungsgrad dadurch erreicht, dass alle Fachleute in einer angemessenen Begriffswelt denken und genormte kommunikationsfreundliche Notationen für die Kommunikation verwenden.

Fachleute sind sich inzwischen weitgehend einig, dass auch die Softwaresystemtechnik eine Ingenieurdisziplin werden muss. Die Komplexität der zu beherrschenden Softwaresysteme ist gegenwärtig schon sehr hoch und wird in Zukunft noch weiter wachsen, jedoch wird diese Komplexität zur Zeit noch nicht befriedigend beherrscht. Eine Grundlegung für die Ingenieurdisziplin Softwaresystemtechnik muss die Erhöhung des Kommunikationswirkungsgrads zum Ziel haben. Das bedeutet, dass die Frage nach einer angemessenen Begriffswelt und nach kommunikationsfreundlichen Notationen im Zentrum dieser Grundlegung stehen muss.

1.2 Was ist das Charakteristikum der Softwaresystemtechnik?

Die Softwaresystemtechnik unterscheidet sich von den anderen Ingenieurdisziplinen grundlegend durch die Art, wie die bereitzustellenden Systeme geschaffen werden. In den klassischen Ingenieurdisziplinen werden die Systeme geplant und anschließend gefertigt. Das Ergebnis der Planung sind Systembeschreibungen, und nach Vorschrift dieser Beschreibungen werden in der Fertigung unterschiedliche Materialien geformt und als Komponenten zum System zusammengebaut. Im Unterschied hierzu wird bei der Fertigung eines Systems in der Softwaresystemtechnik kein Material geformt und kombiniert. Es genügt hier, die Systembeschreibung in den Speicher einer universellen Hardware einzubringen, damit diese Universalhardware zu dem gewünschten System wird. Diese Art und Weise, zu dem gewünschten System zu kommen, beruht auf dem Sachverhalt, dass dieses gewünschte System im Kern ein rein informationelles System ist. Nur bezüglich der Peripherie des Systems, welche zwischen dem Systemkern und seiner Umgebung sitzt, gibt es materiell-energetische Zwänge; solche Zwänge gibt es jedoch für den Kern eines informationellen Systems nicht.

Im Gegensatz hierzu wird bei den Systemen, die in den klassischen Ingenieurdisziplinen geschaffen werden, der Systemkern grundsätzlich durch materiell-energetische Zwänge bestimmt. Deshalb kann es für diese Systeme keine universelle Hardware geben, die durch bloßes Zusammenbringen mit der Beschreibung des gewünschten Systems zum gewünschten System wird. Man denke beispielsweise an den Unterschied zwischen einem Kühlschrank und einem Auto. Beide Systeme sind durch bestimmte materiell-energetische Vorgaben bestimmt, die nicht durch eine universelle Hardware erfüllt werden können. Nur im Bereich der rein informationellen Systeme gibt es die völlige Freiheit der Wahl physikalischer Effekte zur Systemrealisierung und damit unter anderem auch die Möglichkeit der extremen Miniaturisierung.

Grundsätzlich handelt es sich bei den Systemen, die in der Softwaresystemtechnik geschaffen werden, um simulierte Dienstleistungsunternehmen der Informationsverarbeitung.

Dem Vorteil, das jeweils gewünschte System unter Einsatz einer universellen Hardware realisieren zu können, steht der große Nachteil gegenüber, dass das Wissen über diese Systeme nur sehr schwer weiterzugeben ist. Bild 1 veranschaulicht diese Problematik.

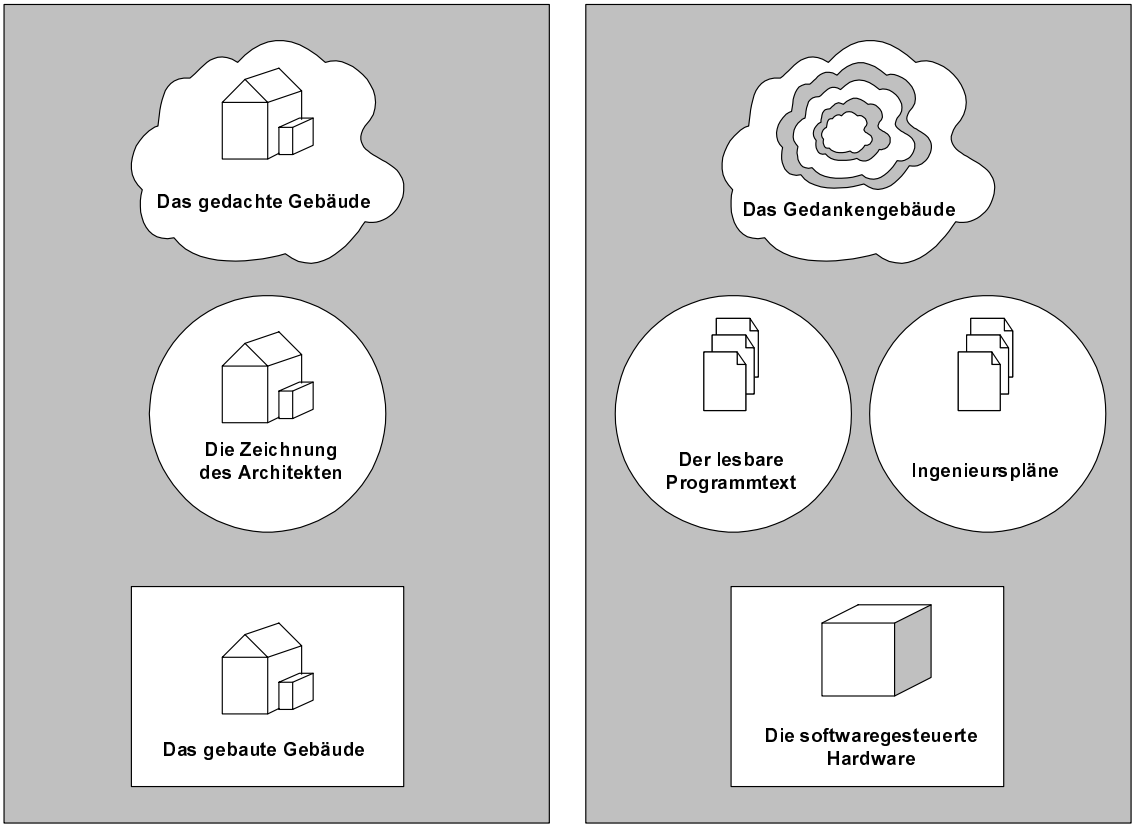

Bild 1

Das linke Teilbild veranschaulicht den Sachverhalt, dass es in den klassischen Ingenieurdisziplinen kein besonders schwer zu lösendes Darstellungsproblem gibt. Weil die Systeme gegenständlich sind, haben sie ein Aussehen, und deshalb kann bereits der Planer das System vor seinem geistigen Auge sehen. Wenn er es dann als Zeichnung zu Papier bringt, können alle Betrachter dieser Zeichnung genau das sehen, was der Planer zuvor vor seinem geistigen Auge sah. Und wenn schließlich das System realisiert ist, sieht es im Prinzip wieder so aus, wie man es schon von der Zeichnung her kannte.

Das rechte Teilbild veranschaulicht die Schwierigkeit der Systemdarstellung in der Softwaresystemtechnik. Informationelle Systeme haben kein funktionstypisches Aussehen, so dass der Planer vor seinem geistigen Auge kein systemcharakterisierendes Bild sehen kann. Eine grundsätzliche Sichtbarkeit gibt es erst bei den auf Grund der Planung geschaffenen Programmtexten. Aber aus diesen Programmtexten lässt sich ein Systemverständnis meistens nicht gewinnen, weil ihr Umfang für eine vollständige Erfassung durch einen Menschen viel zu groß ist – häufig viele Millionen Codezeilen.

Grundsätzliche Sichtbarkeit ist auch gegeben bezüglich der Hardware, die durch Einspeicherung der Software zu dem gewünschten System geworden ist. Da aber diese Hardware eine Universalhardware ist, kann ihr Aussehen wenig Aufschluss über die vom Planer gedachten Strukturen geben.

Wann die Softwaresystemtechnik zu einer Ingenieurdisziplin geworden ist, wird man leicht an Hand der Frage überprüfen können, ob das Problem der Darstellung komplexer Softwaresysteme befriedigend gelöst ist.

Die Tatsache, dass der sogenannten Dokumentation im Bereich der Softwaresystemtechnik im allgemeinen ein hoher Stellenwert eingeräumt wird, darf noch nicht als Hinweis darauf gesehen werden, dass das Kommunikationsproblem gelöst sei. Man braucht sich nur einmal zu fragen, weshalb in der Softwaresystemtechnik zwar viel von Dokumentation, aber so gut wie gar nicht von Plänen die Rede ist. Kein Architekt und kein Ingenieur des Bauwesens, des Maschinenbaus oder der Elektrotechnik wird seine Tätigkeit als Dokumentationstätigkeit bezeichnen, obwohl er doch Dokumente erzeugt. Er nennt diese Dokumente aber Pläne – Baupläne, Konstruktionspläne, Schaltpläne – und er grenzt diese planerische Tätigkeit deutlich von Dokumentationstätigkeiten ab. Dokumentation ist das Festhalten von Information für einen möglichen späteren Bedarf. Dokumentation hat im wesentlichen Protokollierungscharakter, das heißt es muss alles irgendwo aufgeschrieben oder aufgezeichnet werden. Pläne dagegen bilden die Kommunikationsbasis für die Organisation von Arbeitsteilung. An Hand von Plänen werden Lösungsalternativen diskutiert, und anhand von Plänen arbeiten sich neu zum Projekt hinzugekommene Mitarbeiter in den aktuellen Stand ein. Pläne werden also nicht für einen möglicherweise nie eintretenden Nachfragebedarf geschaffen, wie es bei der Dokumentation der Fall ist, sondern Pläne werden von Anfang an für einen selbstverständlichen Kommunikationsprozess geschaffen. Strukturdokumente, die nie zum Gegenstand einer Kommunikation werden, verdienen die Bezeichnung „Plan“ nicht.

1.3 Aus welchen Schichten besteht das vorgestellte Fundament?

Wenn man im Bild eines mehrstöckigen Gebäudes denkt, sieht man die Stockwerke übereinander liegen, und das Fundament befindet sich unter dem untersten Stockwerk. In der vorliegenden Betrachtung darf die Frage, ob die Softwaresystemtechnik als mehrstöckiges Gebäude angesehenen werden kann, unbeantwortet bleiben, denn es geht hier gar nicht um die Frage, welche unterschiedlichen „Räume“ es in der Softwaresystemtechnik gibt und wie sie zueinander angeordnet sind. Hier geht es lediglich um die Strukturierung des Fundaments. Es geht um die Frage, in welcher Reihenfolge die fundamentalen Begriffe eingeführt werden sollen, da diese selbstverständlich teilweise voneinander abhängen. Gewisse Begriffe sind sehr elementar und müssen zuerst vorgestellt werden. Unter Verwendung dieser Begriffe können dann weitere Begriffe vorgestellt werden.

Zuunterst im Fundament liegt der mathematische Strukturbegriff, der so allgemein ist, dass er ohne jeden Bezug zur Softwaresystemtechnik eingeführt werden kann. Unmittelbar darüber liegt aber bereits die Fundamentschicht, welche den Zusammenhang zwischen dem Strukturbegriff und der Softwaresystemtechnik herstellt. Hier geht es um die fundamentale Unterscheidung zwischen einer Beschreibung und dem Beschriebenen. Aus dieser Unterscheidung folgt zwangsläufig die weitere Schichtung des Fundaments. In der als nächstes folgenden Fundamentschicht geht es um das, was beschrieben werden soll, denn erst, wenn dies geklärt ist, kann man sich der Frage nach den Beschreibungsmöglichkeiten zuwenden.

Die Schicht, worin es nicht um die Beschreibung, sondern um das Beschreibbare geht, lässt sich selbst noch einmal in vier kleinere Schichten unterteilen. Es handelt sich um die Begriffswelt der dynamischen Systeme, die nach ihrer Einführung sukzessive auf bestimmte Systemtypen eingeschränkt wird. Über diesen vier Schichten, worin es um die Strukturen im Beschriebenen geht, liegt die letzte Fundamentschicht, worin die Strukturen der Beschreibung betrachtet werden.

1.4 Muss auf die erste Grundlegung eine zweite folgen?

Es wäre vermessen anzunehmen, die vorgestellte Grundlegung könnte bereits endgültig sein. Dafür ist die Softwaresystemtechnik noch eine viel zu junge Disziplin, worin noch – wie im jungen Wein – turbulente Gärungsprozesse ablaufen. Der Autor ist jedoch überzeugt, dass nicht alle hier vorgestellten Fundamentschichten in gleichem Maße unausgegoren sind. So wie man Kriterien kennt, anhand deren man feststellen kann, ob ein Gärungsprozess noch läuft oder schon sein Ende erreicht hat, gibt es auch ein Kriterium, anhand dessen man feststellen kann, ob eine begriffliche Grundlegung bereits eine akzeptable Endgültigkeit erreicht hat oder noch nicht.

Dass es so etwas wie einen endgültigen Reifegrad für konzeptionelle und notationelle Fundamente gibt, kann man sich leicht vor Augen führen, indem man das Konzept und die Notation für die Darstellung von Zahlen betrachtet. Vermutlich gab es zur Zeit der Römer viele Menschen, die mit der begrifflichen und notationellen Grundlegung der Zahlenwelt zufrieden waren, aber dies konnte nicht als Zeichen einer endgültigen Reife angesehen werden. Es genügte, dass mindestens ein Mensch unzufrieden war und sich nach anderen Möglichkeiten umsah.

Nach Einschätzung des Autors sind die hier vorgestellten Fundamentschichten umso reifer, je tiefer sie liegen. So hat mit Sicherheit die unterste Fundamentschicht, worin der mathematische Strukturbegriff liegt, ihre endgültige Reife erreicht. Im Gegensatz hierzu ist die oberste Fundamentschicht nur als erster Ansatz zu betrachten, der noch eine längere, durch entsprechende Forschungsbemühungen voranzutreibende Reifeperiode benötigt. Konkret heißt dies, dass die Einteilung der verschiedenen Trägersysteme in bestimmte Klassen noch mit Sicherheit verbessert werden kann.

2. Die beiden untersten Schichten des Fundaments

2.1 Der mathematische Strukturbegriff

In dieser Arbeit wird der Begriff „Struktur“ ausschließlich im mathematischen Sinne verwendet. In diesem Sinne versteht man unter einer Struktur ein Tupel aus Mengen und Relationen, welches mindestens eine nichtleere Menge und eine nichtleere Relation enthält. Jede Relation ist eine Teilmenge eines Kartesischen Produktes, dessen Faktoren Mengen oder Relationen des Strukturtupels sind. Ein Kartesisches Produkt lässt sich immer als n-dimensionaler Quader denken. Eine Relation lässt sich somit als eine Auswahl von Punkten aus einem solchen Quader denken. Recht häufig sind diese Quader nur zweidimensional; in diesen Fällen handelt es sich um Rechtecke, die man unmittelbar darstellen kann. Falls es sich um zweistellige Relationen handelt, deren Faktoren endliche Mengen sind, sind die Rechtecke als Matrizen darzustellen.

Neben der räumlichen Anschauung gibt es auch die logische Anschauung von Relationen. In der logischen Sicht ist eine Relation die Menge aller Individuentupel, die ein bestimmtes Prädikat erfüllen. Das jeweilige Prädikat wird als Aussageform formuliert, und diese Aussageform wird zu einer wahren Aussage, wenn man die Individuen eines Tupels, welches in der Relation enthalten ist, an die offenen Stellen der Aussageform setzt. Als Beispiel betrachten wir die Aussageform „Die Person *p* hat das Buch *b* gelesen." Die Betrachtung sei auf die endlichen Personenmenge {Anna, Otto, Karl, Emma, Luise} beschränkt. Die betrachtete Büchermenge sei auf die endliche Menge { Bibel, Koran, Goethe: Faust, Tolstoi: Krieg und Frieden, Hemingway: Wem die Stunde schlägt, Karl Marx: Das Kapital} beschränkt. Die Relationsmatrix könnte in diesem Falle beispielsweise folgendermaßen aussehen:

	Anna	Otto	Karl	Emma	Luise
Die Bibel	X	X			X
Der Koran		X		X	X
Goethe: Faust	X				X
Tolstoi: Krieg und Frieden			X		X
Hemingway: Wem die Stunde schlägt		X		X	X
Karl Marx: Das Kapital			X		

Bild 2

Zweistellige Relationen auf endlichen Mengen kann man nicht nur in Form von Matrizen darstellen. Man kann auch eine Graphendarstellung wählen. Dazu wählt man für jede der an der Relation beteiligte Menge ein eigenes Knotensymbol. Dann kann man alle Elemente der beteiligten Mengen als Knoten des Graphen in der Fläche platzieren. Jedes Relationselement, also jeder ausgewählte Kreuzungspunkt der Rechteckmatrix , wird im Graphen zu einer Verbindungslinie zwischen zwei Knoten. Die Struktur aus Lesern und Büchern, die in Form der Matrixdarstellung eingeführt wurde, lässt sich beispielsweise durch den Graphen im Bild 3 darstellen.

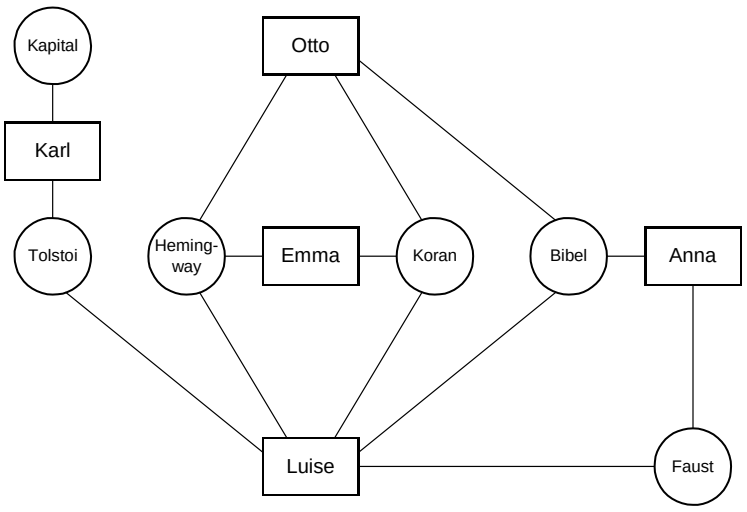

Bild 3

Selbstverständlich hat man eine große Freiheit bei der Gestaltung des Graphenlayouts. In Abhängigkeit davon, wo man die Knoten platziert, ergeben sich mehr oder weniger

übersichtliche Graphen. In Bild 3 waren noch alle Knoten einer Sorte gleich groß. Dies muss natürlich nicht so sein, denn es ist ja nur wichtig, dass man die Knotentypen leicht auseinander halten kann. In Bild 4 ist die Leser-Bücher-Struktur als Graph noch einmal dargestellt, wobei dieses Mal die Rechteckknoten nicht mehr alle die gleiche Größe haben. Durch die Variation der Knotengröße wurde es möglich, sämtliche Verbindungslinien senkrecht oder waagerecht zu ziehen.

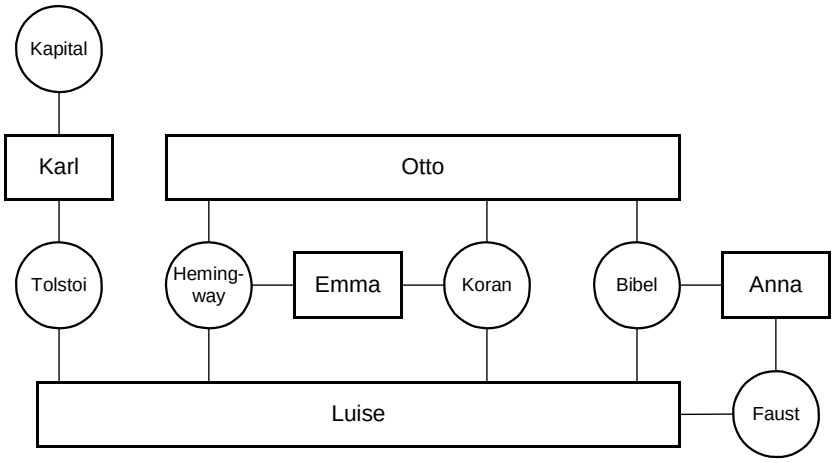

Bild 4

Selbstverständlich kann man nicht in jedem Falle erreichen, dass alle Verbindungslinien im Graphen kreuzungsfrei verlaufen. Ob dies möglich ist oder nicht, wird durch die gegebene Relation festgelegt.

Wenn das Kartesische Produkt, welches die Basis einer zweistelligen Relation bildet, zwei unterschiedliche Mengen als Faktoren hat, dann ergibt sich als Graphendarstellung ein sogenannter bipartiter Graph. Ein bipartiter Graph ist dadurch gekennzeichnet, dass er zwei Knotentypen hat und dass die Verbindungen jeweils nur zwischen zwei Knoten unterschiedlichen Typs liegen. Die Beispiele im Bild 3 und 4 sind bipartite Graphen. Eine zweistellige Relation kann aber auch eine sogenannte quadratische Relation sein, d.h. die Relation kann eine Teilmenge aus dem Kartesischen Quadrat einer gegebenen Menge sein. In diesem Falle enthält der darstellende Graph nur Knoten eines Typs. In diesem Falle müssen die Verbindungen gerichtet sein, weil man sonst die Reihenfolge der verbundenen Individuen im Relationstupel nicht erkennen kann. Die Relevanz der Reihenfolge sieht man leicht ein, indem man ein Beispiel betrachtet. Die Relation sei durch die Aussageform „ Die Person *p1* hat Person *p2* umgebracht.“ Offensichtlich ist es hier relevant, ob der Meyer den Müller umgebracht hat, oder ob es umgekehrt war.

Wenn quadratische Relationen als Graphen dargestellt werden, kann es auch vorkommen, dass ein Knoten mit sich selbst verbunden werden muss. In diesem Fall muss also eine Schleife über einen Knoten gezeichnet werden. Im Beispiel der Mordrelation würde eine solche Schleife den Sachverhalt zum Ausdruck bringen, dass jemand Selbstmord begangen hat.

Es gibt auch den Sonderfall, dass man eine quadratische Relation als ungerichteten Graphen darstellen kann. In diesem Falle muss es sich um ein symmetrische Relation handeln. Wenn man eine symmetrische Relation als gerichteten Graphen darstellt, enthält dieser Graph zu jedem Pfeil vom Knoten A zum Knoten B auch den in umgekehrter Richtung laufenden Pfeil. Deshalb kann man in diesem Fall den Graphen vereinfachen, indem man an Stelle der beiden

hin- und zurückführenden Pfeile eine einzige ungerichtete Verbindung einträgt. Man muss dann aber wissen, wie man dies zu interpretieren hat.

2.2 Die Unterscheidung zwischen Beschreibung und Beschriebenem

Die Überschrift erscheint trivial, denn es ist doch wohl selbstverständlich, dass niemand eine Beschreibung mit dem Beschriebenen verwechselt. Wie könnte man denn beispielsweise ein Buch über Afrika mit dem Kontinent Afrika selbst verwechseln. Es ist ganz offensichtlich, dass ein Buch über Afrika eine ganz andere Struktur hat als Afrika selbst. So hat dieses Buch einen Einband, es ist in Kapitel unterteilt, es hat ein Sachwortverzeichnis. All dies gilt für Afrika nicht. Das Buch ist etwas Statisches, d.h. es ist eine festliegende Form, die sich über der Zeit nicht verändert. Afrika dagegen ist ein dynamisches System, d.h. in Afrika und mit Afrika geschieht etwas im Laufe der Zeit. Eine Aussage, die heute auf Afrika korrekt zutrifft, könnte in einiger Zeit falsch geworden sein.

In dem Buch über Afrika gibt es verschiedene Darstellungen, die unterschiedlich weit von dem real in und mit Afrika Erlebbaren entfernt sind. Ein Text, worin etwas über Afrika berichtet wird, ist sehr weit von dem real Erlebbaren entfernt. Man darf hier nicht die Interpretation des Textes vor Augen haben, sondern muss an das Wahrnehmen des Textes selbst denken. Denken Sie sich eine Textseite aus dem Buch über Afrika und fragen Sie sich, wo man im realen Afrika ein Erlebnis haben kann, das dem Betrachten dieser Textseite nahe kommt. Das Erleben des Textes und das Erleben von Afrika sind sehr unterschiedliche Dinge.

Anders liegt der Fall, wenn in dem Buch über Afrika eine Landkarte abgedruckt ist. Das Erleben dieser Landkarte ist sehr viel näher am Erleben von Afrika. Heutzutage kann man sich vorstellen, man säße in einem Satelliten und schaue auf die Erde hinunter. Wenn der Satellit hoch genug fliegt, sieht man tatsächlich Afrika unter sich liegen in der Form, wie man es zuvor auf der Landkarte gesehen hat. Der Blick aus dem Satelliten auf das reale Afrika ist kein Erleben einer Beschreibung, sondern stellt ein Erleben des Beschriebenen dar.

Wenn man nun in dieser Situation, wo sowohl die Beschreibung als auch das Beschriebene erlebbar sind, nach Strukturen sucht, dann findet man drei unterschiedliche Strukturbereiche. Man findet einerseits Strukturen in der Beschreibung, also Relationen zwischen Elementen der Beschreibung. Da eine Beschreibung eine passive Struktur ist, kann es in dieser Struktur nur um Relationen zwischen Komponenten der Beschreibung gehen. Man denke beispielsweise an den Zusammenhang zwischen Einträgen in einem Sachwortverzeichnis und den Teilen der Beschreibung, auf die aus dem Sachwortverzeichnis heraus verwiesen wird. Wenn die Beschreibung ausschließlich aus Text bestünde, wäre die Struktur innerhalb der Beschreibung ausschließlich gegeben durch Beziehungen zwischen Textstücken.

Einen zweiten Strukturbereich findet man im Beschriebenen. Das Beschriebene selbst ist in unserem Fall ein dynamisches System. In diesem dynamischen System gibt es sehr viel erlebbare Individuen. Dies sind einerseits Komponenten, aus denen das System besteht, andererseits aber auch Vorgänge, die in diesem System stattfinden.

Es gibt aber noch einen dritten Strukturbereich, der bestimmt wird durch die Beziehungen zwischen der Beschreibung und dem Beschriebenen. In diesem Strukturbereich werden Individuen auf der Beschreibungsseite in Beziehung gesetzt zu Individuen auf der Seite des Beschriebenen. Am Beispiel des Buches über Afrika veranschaulicht heißt dies, dass es Beziehungen gibt zwischen einem bestimmten Textstück in diesem Buch, worin Aussagen über

den Fluss Kongo gemacht werden, und dem tatsächlichen Fluss Kongo in Afrika, der dort erlebbar ist.

In Bild 5 sind diese drei unterschiedlichen Strukturbereiche zueinander in Beziehung gesetzt. Dieses Bild sollte dem Leser immer vor Augen stehen, wenn er im konkreten Fall eine Strukturbeschreibung einordnen muss.

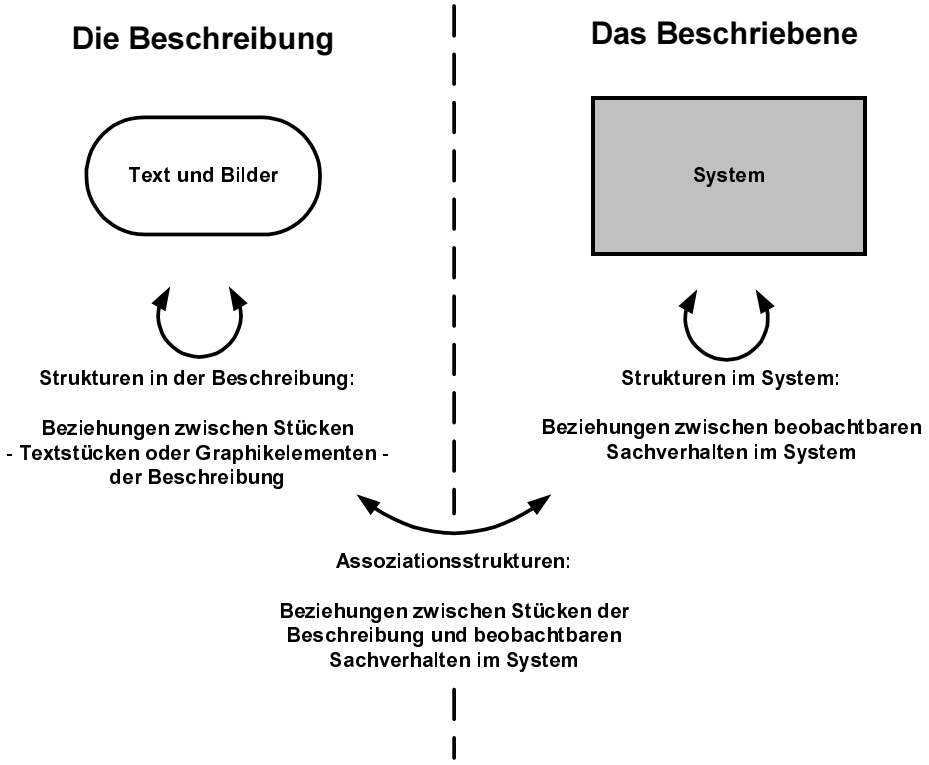

Bild 5

Im Unterschied zum Beispiel des Kontinents Afrika, den man vorgefunden hat und anschließend beschreiben konnte, handelt es sich in der Technik – und somit auch in der Softwaresystemtechnik – bei dem Beschriebenen nicht um etwas Vorgefundenes, sondern um etwas künstlich Geschaffenes oder erst noch zu Schaffendes. Bereits im Abschnitt 1.2. wurde betont, dass es sich bei den Systemen, die in der Softwaresystemtechnik geschaffen werden, um simulierte Dienstleistungsunternehmen der Informationsverarbeitung handelt. Deshalb werden nun als nächstes die Strukturen betrachtet, die man in derartigen Systemen finden kann.

3. Die systemtechnische Begriffswelt

3.1 Die vier fundamentalen Strukturen in dynamischen Systemen

3.1.1 Klassifikation dynamischer Systeme

Ein dynamisches System ist ein Gebilde, worin sich ein Geschehen abspielt. Das dynamische System umfasst genau alles das, was erforderlich ist, damit sich das Geschehen abspielen kann. Ein dynamisches System existiert nicht an sich, sondern ist immer eine gedankliche Konstruktion von Menschen. Die Festlegung auf bestimmte Typen von Geschehen, auf die sich das Interesse konzentrieren soll, führt zur Abgrenzung eines bestimmten Weltausschnitts. Innerhalb dieses Weltausschnitts liegt alles, was als Träger des interessierenden Geschehens gebraucht wird. Außerhalb dieses Weltausschnitts liegt der Rest der Welt, der bei der aktuellen Betrachtung nicht interessiert.

Das interessierende Geschehen kann kontinuierlich und diskret sein. Deshalb unterscheidet man zwischen kontinuierlichen und diskreten Systemen. Orthogonal zu dieser Einteilung der Systeme in zwei Klassen gibt es eine andere Zweierklassifikation, bei der zwischen materiell-energetischer Relevanz und informationeller Relevanz des Geschehens unterschieden wird. Im Falle der materiell-energetischen Relevanz des interessierenden Geschehens ist das Interesse am Geschehen durch materiell-energetische Ziele begründet. Beispiele für solche materiell-energetischen Ziele sind:

- Transport von Massen entgegen der Erdgravitation; Systeme für dieses Ziel sind beispielsweise Rolltreppen, Fahrstühle oder Raketen.

- Transport von Massen entgegen der Reibungskraft; Systeme für dieses Ziel sind beispielsweise Autos und Eisenbahnen.

- Verformung von Materie; Systeme für dieses Ziel sind Pflüge, die von Traktoren über den Acker gezogen werden, oder Pressen, die Karosserieteile formen.

- Veränderung der Temperatur von festen Körpern, Flüssigkeiten oder Gasen; Systeme für dieses Ziel sind beispielsweise Heizungen oder Kühlschränke.

Informationelle Relevanz liegt vor, wenn die materiell-energetischen Sachverhalte im Geschehen nicht interessieren, weil es nur auf die Interpretation der Sachverhalte ankommt. In diesem Fall müssen die Sachverhalte allerdings die Bedingung erfüllen, dass sie überhaupt interpretierbar sind, das heißt, dass es eine Vereinbarung gibt, die den wahrnehmbaren Sachverhalten jeweils eine hinzuzudenkende Bedeutung zuordnet. Als Beispiel für ein informationelles System denke man an ein System zur Durchführung einer numerischen Berechnung. Bei der Betrachtung dieses Systems genügt es zu wissen, dass im Laufe des Geschehens an verschiedenen Orten Zahlen beobachtbar sein werden, und es kann davon abstrahiert werden, durch welche konkreten wahrnehmbaren Formen diese Zahlen dargestellt werden. Informationelle Systeme können nicht solche sein, in denen es keine materiell-energetischen Sachverhalte gibt, denn alles wahrnehmbare muss eine materiell-energetische Basis haben. Informationelle Systeme zeichnen sich aber dadurch aus, dass die Wahl der materiell-energetischen Basis durch die Vorgabe des Ziels, dem das Geschehen im System dienen soll, nicht vorbestimmt ist.

Unabhängig von der Frage, ob das System kontinuierlich oder diskret sei oder ob es materiell-energetisch relevant oder informationell relevant sei, lassen sich allgemeingültige Strukturtypen feststellen, die es in allen dynamischen Systemen gibt. Bei der Suche nach diesen Strukturen fragt man zuerst nach den Individuen, auf die man beim Erleben dynamischer Systeme stößt, und anschließend fragt man nach den Beziehungen, die zwischen diesen Individuen bestehen können.

3.1.2 Wertebereichsstrukturen

Das Geschehen, um dessen Erleben es geht, wird nur erlebbar, wenn es Orte gibt, die der Beobachtung zugänglich sind. An diesen Orten gibt es Sachverhalte, die sich möglicherweise mit der Zeit ändern, und die man entweder unmittelbar mit den menschlichen Wahrnehmungsorganen oder mit Hilfe technischer Wahrnehmungsapparate registrieren kann. Jedem Beobachtungsort ist ein sogenannter Wertebereich zugeordnet, wobei unter dem Begriff Wertebereich die Menge aller an diesem Ort potentiell zu irgend einem Zeitpunkt beobachtbaren Sachverhalte zu verstehen ist. Je nach Systemtyp kann es sich um materiell-energetisch relevante oder um informationelle Sachverhalte handeln.

Die Beschreibung eines Wertebereichs besteht in der Spezifikation einer Menge. Die einfachsten Mengen sind endliche Mengen mit einer so kleinen Anzahl von Elementen, dass alle Elemente mit erträglichem Aufwand explizit aufgezählt werden können. Man denke beispielsweise an die Menge der Hauptstädte der europäischen Staaten. Diese Menge könnte man auf andere Weise als die explizite Aufzählung gar nicht spezifizieren. In vielen Fällen sind aber die Wertebereiche so umfangreich, dass eine explizite Aufzählung aller Werte völlig unmöglich ist. Diese Unmöglichkeit kommt nicht daher, dass die Wertebereiche tatsächlich unendlich wären, denn die technisch beherrschbaren Wertebereiche sind in jedem Falle endlich, weil die Speicherkapazitäten in den Systemen endlich sind, die Kanalkapazitäten endlich sind und die Auflösungen der Messgeräte endlich sind.

In der Systemtechnik ist es sehr wichtig, zwei Arten von Endlichkeit unterscheiden zu können, wogegen in der Mathematik eine solche Unterscheidung unnötig ist. Für einen Mathematiker ist eine Menge mit fünf Elementen in gleicher Weise endlich wie eine Menge mit fünf Billionen Elementen. Für den Systemtechniker aber ist eine Menge mit fünf Elementen praktikabel aufzählbar, wogegen eine Menge mit fünf Billionen Elementen als pseudounendlich betrachtet werden muß. Pseudounendliche Mengen werden im allgemeinen dadurch definiert, dass sie nach einem bestimmten Auswahlkriterium als endliche Teilmengen unendlicher Mengen festgelegt werden. Als Beispiel sei die Menge der mit 32 Bit im Zweierkomplement kodierten Integerzahlen betrachtet. Diese Menge enthält 2^{32} unterschiedliche Werte. Man spezifiziert diese Menge zweckmäßiger Weise durch die Intervallangabe

$$-2^{31} \leqq i \leqq +2^{31} - 1 \quad .$$

Durch diese Intervallangabe wird eine endliche Menge aus der unendlichen Menge der ganzen Zahlen ausgeschnitten. Die Intervallangabe kann aber nur von Jemanden verstanden werden, der die unendliche Menge der ganzen Zahlen schon verstanden hat.

In den meisten Fällen der Wertebereichsspezifikation geht es also darum, zuerst einmal eine unendliche Menge zu spezifizieren und anschließend in dieser unendlichen Menge eine endliche Menge abzugrenzen. Unendliche Mengen lassen sich grundsätzlich nur innerhalb von Strukturen definieren, was bedeutet, dass die zu spezifizierende Menge immer untrennbar mit

mindestens einer Relation auf dieser Menge eingeführt werden muss. So ist beispielsweise die Einführung der Menge der natürlichen Zahlen untrennbar mit der Linearordnung dieser Zahlen verbunden. Wenn man von Wertebereichen spricht, kann man also auch immer von Wertebereichsstrukturen sprechen, falls man bereit ist, auch den Sonderfall der praktikabel aufzählbaren Menge als Struktur gelten zu lassen.

3.1.3 Ablaufstrukturen

Die Beobachtungsergebnisse für die verschiedenen Orte kann man über der Zeit festhalten und auf diese Weise das Geschehen protokollieren. Jedes Protokoll dieser Art ist eine Struktur, denn jeder Verlauf von Werten über der Zeit stellt eine Beziehung zwischen dem Wertebereich und der Menge der Zeitpunkte dar.

Wenn jedes Protokoll möglich wäre, d.h. wenn jede Aufeinanderfolge von Werten im System möglich wäre, dann würden diese Protokolle keine Erkenntnis über das System vermitteln können. Man denke beispielsweise an ein Protokoll, worin die Aufeinanderfolge der an einem Roulettetisch in der Spielbank aufgetretenen Werte aus dem Zahlenbereich 0 bis 37 festgehalten ist. Wenn jede mögliche Folge gleichwahrscheinlich ist, was von der Spielbank selbstverständlich immer angestrebt wird, kann man aus dem Protokoll der in der Vergangenheit beobachteten Folge von Spielergebnissen keinerlei Schlüsse für die zukünftigen Spielergebnisse ziehen.

In der Softwaresystemtechnik sind die interessierenden Systeme normalerweise vollkommen deterministisch, was zur Folge hat, dass am jeweiligen Beobachtungsort meist nur eine Teilmenge aller kombinatorisch möglichen Werteverläufe vorkommen kann. Den Extremfall stellt der Ausgang eines Generators dar, wo garantiert nur eine ganz bestimmte Wertefolge auftreten kann, weil dies durch die Konstruktion des Generators festgelegt ist. Man denke an einen Sinusgenerator, den es als Gerät in einem Labor der Elektrotechnik geben kann. Bei einem solchen Generator stellt man die Amplitude und die Frequenz ein, und diese beiden Vorgaben legen das am Ausgang des Generators auftretende Signal vollständig fest.

In diesem Abschnitt 3.1, wo die Betrachtung noch nicht auf bestimmte Unterklassen dynamischer Systeme eingeschränkt ist, wird noch nicht auf die Möglichkeiten der Darstellung von Ablaufstrukturen eingegangen. Durch jeweils geeignete Darstellungen muss das Wissen vermittelt werden, welche Werteverläufe an welchen Beobachtungsorten unter welchen Bedingungen auftreten können.

3.1.4 Funktionale Strukturen

Das Festhalten von Ablaufstrukturen kann nur dann von Interesse sein, wenn es funktionale Abhängigkeiten zwischen bestimmten Beobachtungsergebnissen gibt. Bei der Formulierung von Ablaufstrukturen wird man also zwangsläufig auf Funktionen Bezug nehmen müssen.

Ganz allgemein muss man fragen, ob ein Beobachtungsergebnis am Ort v_2 zum Zeitpunkt t_2 funktional durch ein Beobachtungsergebnis am Ort v_1 zum Zeitpunkt t_1 festgelegt ist.
(Die Wahl der Ortsbezeichnungen v_1 und v_2 gründet sich auf die Bezeichnung „Variable“.)

Je nach dem, ob die beiden Beobachtungsorte v_1 und v_2 unterschiedliche Orte sind oder nicht und ob die beiden Zeitpunkte t_1 und t_2 unterschiedliche Zeitpunkte sind oder nicht, lassen sich drei Fälle unterscheiden:

1.) $v_1 \neq v_2$; $t_1 \neq t_2$

2.) $v_1 = v_2$; $t_1 \neq t_2$

3.) $v_1 \neq v_2$; $t_1 = t_2$

Der vierte Fall, bei dem $v_1 = v_2$ und $t_1 = t_2$ ist, ist ein trivialer Fall, denn hier steht ein beobachteter Sachverhalt in Beziehung zu sich selbst, und das kann nur die Identitätsbeziehung sein.

Funktionen sind mathematische Strukturen, die ohne jeden Bezug zur Systemtechnik betrachtet werden müssen. Man darf also nicht den Bereich der Ablaufstrukturen mit dem Bereich der Funktionsstrukturen verwechseln. Bei der Formulierung einer Ablaufstruktur muss man die Funktionen schon als bekannte Strukturen voraussetzen. Die Frage nach angemessenen Beschreibungen von Funktionen stellt also kein spezifisches Problem der Softwaresystemtechnik dar.

Zu jeder Funktion muss ein Argumentwertebereich und Ergebniswertebereich definiert sein. Die Funktion ordnet jedem Element des Argumentwertebereichs ein Element des Ergebniswertebereichs zu. Die Vielfalt der möglicherweise zu betrachtenden Funktionen ist so riesig, dass man keine allgemeingültigen Regeln für die Anfertigung zweckmäßiger Funktionsbeschreibungen angeben kann. Das Fehlen solcher Regeln stellt aber kein besonders schwieriges Problem dar. Die bisherigen Erfahrungen in der Mathematik haben gezeigt, dass man in jedem Einzelfall eine befriedigende Funktionsbeschreibung gefunden hat.

3.1.5 Aufbaustrukturen

Neben den Beobachtungsorten, die zwingend erforderlich sind, damit überhaupt ein Geschehen festgestellt werden kann, muss es im System noch etwas anderes geben, was man als den Träger des Geschehens bezeichnen kann. Wenn wir über den Träger des Geschehens nichts wissen außer dass er existieren muss, bezeichnen wir den Träger als Blackbox. Mit dieser Bezeichnung verbindet man die Anschauung einer verschlossenen Kiste, in die man nicht hineinschauen kann. Es gibt aber viele Systeme, bei denen man über den Träger des Geschehens mehr weiß als nur, dass er existiert. In diesen Fällen kennt man verschiedene unterschiedliche Komponenten, die miteinander in Wechselwirkung stehen und auf diese Weise das Geschehen tragen. Bei dieser Betrachtung sind die einzelnen Komponenten kleinere Blackboxes, in die man nicht hineinschauen kann.

Damit eine Wechselwirkung zwischen den Komponenten entstehen kann, müssen die Komponenten in irgendeiner Weise miteinander in Verbindung stehen. Durch die Verbindung von Komponenten ergeben sich zwangsläufig Orte zur Beobachtung des Systemgeschehens. Denn da man in Blackboxes nicht hineinschauen kann, können die Orte zur Beobachtung des Geschehens nicht im Inneren der Komponenten liegen; sie müssen vielmehr am Rand der Komponenten zu finden sein. Bei der Verbindung von Komponenten werden diese so nahe aneinander gerückt, dass sie sich an ihrem Rand berühren. Die gemeinsamen Berührungsstellen müssen zwangsläufig die Orte sein, über die eine Wechselwirkung zwischen den

Komponenten möglich wird. Und an diesen Orten wird das Geschehen beobachtbar. Der Beitrag einer Komponente zum gesamten Geschehen im System besteht im Beitrag der Komponente zu den Vorgängen an den Beobachtungsorten auf ihrem Rand.

Es wäre falsch anzunehmen, die Beobachtungsorte am Rand einer Komponente ließen sich in jedem Falle in Eingänge und Ausgänge der Komponente einteilen. Wenn der Ausgang einer Komponente mit dem Eingang einer anderen Komponente verbunden ist, liefern nicht beide Komponenten einen Beitrag zum Geschehen an diesem Berührungspunkt, sondern das Geschehen wird durch die Komponente bestimmt, für die der Berührungspunkt ein Ausgang ist. Es gibt aber auch die Möglichkeit, dass zwei oder mehr Komponenten, die sich an einer Berührungsstelle treffen, zum Geschehen an dieser Stelle beitragen. Systeme mit derartigem Komponentenverhalten sind beispielsweise elektrische Netzwerke. Die Komponenten sind elektrische Bauelemente wie Widerstände, Spulen, Kondensatoren, Spannungsquellen oder Halbleiterbauelemente. Das interessierende Geschehen ist in diesem Fall gegeben durch den zeitlichen Verlauf von Strömen und Spannungen, die an den Punkten gemessen werden, an denen Bauelementeanschlüsse miteinander verbunden sind. Die Anschlüsse von elektrischen Bauelementen sind nicht als Eingänge oder Ausgänge klassifizierbar. Der jeweilige Wert des Stromes oder der Spannung, der an einem Bauelementeanschluss gemessen wird, wird nicht ausschließlich von einem Bauelement bestimmt, sondern von der Gesamtheit aller in Wechselwirkung stehenden Bauelemente. Systeme dieser Art könnte man als Gleichgewichtssysteme bezeichnen.

Da es bei allen dynamischen Systemen einerseits die Orte gibt, an denen das Geschehen beobachtet wird, und andererseits den oder die Träger des Geschehens, kann man eine Struktur angeben, die man als Aufbaustruktur bezeichnen kann. Diese Struktur erfasst das Wissen darüber, welche Geschehensträger mit welchen Beobachtungsorten verbunden sind. Man kann diese Aufbaustruktur als bipartiten Graphen darstellen. Wenn ein Beobachtungsort ein Ausgang oder ein Eingang eines Geschehensträgers ist, kann man im Graphen den Trägerknoten mit dem Ortsknoten gerichtet verbinden, anderenfalls ist die Verbindung ungerichtet. Ein System mit nur einem einzigen Geschehensträger, also ein System, bei dem der Träger eine Blackbox ist, kann nur Beobachtungsorte haben, die als Ausgänge zu betrachten sind. Denn wenn nur ein Geschehensträger existiert, muss dieser alles beobachtbare Geschehen vollständig festlegen. Bild 6 zeigt die zugehörige Graphendarstellung.

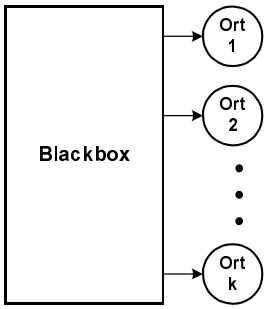

Bild 6

Der in Bild 6 gezeigte Fall des Systems mit nur einem einzigen Geschehensträger ist von sehr geringem Interesse. Sehr viel interessanter sind Systeme, bei denen zwei Komponenten in Wechselwirkung miteinander stehen. Bild 7 zeigt die grundsätzliche Struktur eines solchen Systems. In diesem aus zwei Komponenten bestehenden System ist die eine Komponente die eigentlich interessierende Komponente, wogegen die zweite Komponente die Umgebung dar-

stellt. Man stelle sich einen Menschen vor, der mit einem technischen Gerät experimentiert. Der Mensch und das Gerät stehen in gegenseitiger Wechselwirkung. Die Orte für die Beobachtung des Geschehens sind die Stellen, an denen sich die beiden Komponenten materiell oder energetisch berühren. Obwohl die eigentlich interessierende Komponente - im Beispiel also das technische Gerät - nur ein Teilsystem darstellt, bezeichnet man alles, was mit der Umgebung in Wechselwirkung steht, als das interessierende System.

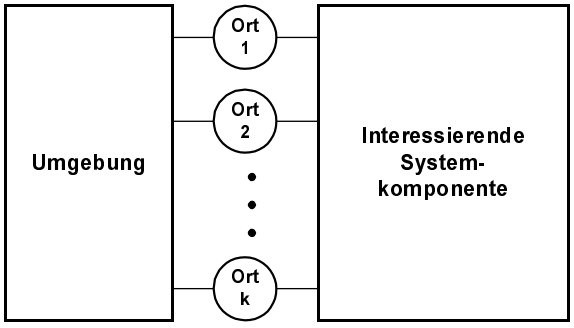

Bild 7

In Bild 8 ist ein Beispiel eines Systems dargestellt, welches schon ohne Umgebung abgeschlossen ist. Es handelt sich um ein elektrisches Netzwerk. Darunter ist der zugehörige größtenteils ungerichtete bipartite Graph dargestellt. Die Ungerichtetheit weist darauf hin, dass es sich hier um ein Gleichgewichtssystem handelt. Der Beitrag der einzelnen Komponenten zum Geschehen besteht darin, das Erfülltsein bestimmter komponentenspezifischer Gleichungen zu erzwingen, die jeweils in die entsprechenden Komponentenknoten eingetragen sind.

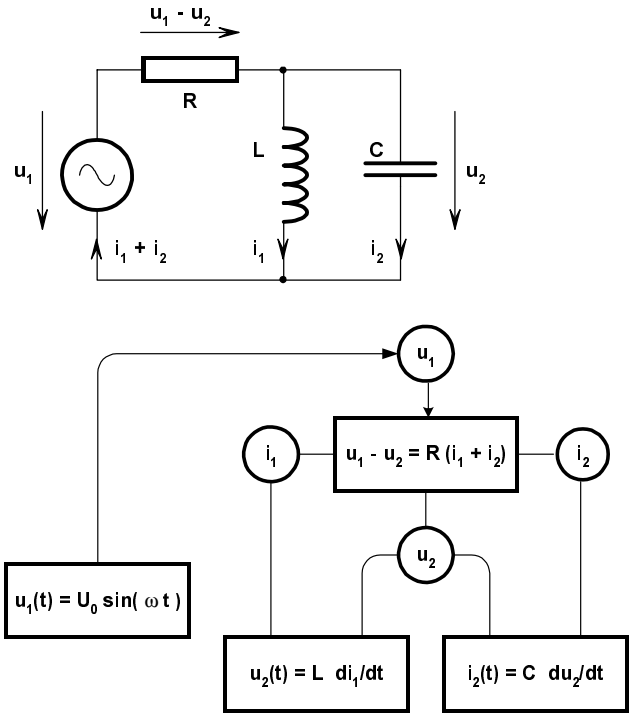

Bild 8

3.2 Einschränkung der Betrachtung auf diskrete Systeme

Ein diskretes System ist dadurch gekennzeichnet, dass die Wertebereiche für die an den Beobachtungsorten möglichen Beobachtungsergebnisse diskret sind. Da es sich bei den beobachteten Sachverhalten immer um materiell-energetische Sachverhalte handelt, die grundsätzlich als kontinuierlich veränderlich denkbar sind, kommt man zu diskreten Sachverhalten nur dadurch, dass man sich für bestimmte kontinuierliche Übergänge nicht interessiert. Ein sehr anschauliches Beispiel für die Beschränkung des Interesses auf diskrete Sachverhalte ist eine Bahnhofsuhr, bei der es für den großen Zeiger 60 diskrete Positionen gibt, die durch ein Zahnrad im Inneren der Uhr bedingt sind, welches 60 Zähne hat. Diese 60 Zähne bestimmen die 60 Winkelpositionen, in denen das Zahnrad und damit auch der große Zeiger verharren können. Selbstverständlich geschieht der Übergang von einer Winkelposition zur nächsten kontinuierlich, aber dieser Übergang kann für uninteressant erklärt werden. Nur dadurch erhält man die 60 diskreten Stellungen. Im Beispiel der Bahnhofsuhr liegt sogenannte Zustandsdiskretheit vor. Diese ist dadurch gekennzeichnet, dass für die Dauer eines bestimmten Zeitintervalls der beobachtbare Sachverhalt sich nicht ändert und aus dem diskreten Wertebereich stammt (s. Bild 9).

Neben der Zustandsdiskretheit gibt es auch noch die sogenannte Vorgangsdiskretheit. Die Vorgangsdiskretheit lässt sich leicht durch Betrachtung von Verkaufsautomaten veranschaulichen, bei denen Geldmünzen eingeworfen werden, damit anschließend Waren in ein Ausgabefach fallen. Das Einwerfen einer bestimmten Münze ist ein Vorgang von endlicher Dauer. Es handelt sich um ein an einem Beobachtungsort wahrnehmbares Geschehen. Die Diskretheit wird durch das endliche Repertoire unterschiedlicher einwerfbarer Münzen festgelegt. Während sich bei der Zustandsdiskretheit die Diskretheit dadurch ergibt, dass man nicht auf den Beobachtungsort schaut, wenn sich dort eine Veränderung abspielt, ist es im Falle der Vorgangsdiskretheit gerade umgekehrt: Man braucht auf den Beobachtungsort nicht zu schauen, solange sich dort kein Vorgang abspielt, denn in den Zeiten der Ruhe findet man dort immer die gleiche Situation vor (s. Bild 9). Das Repertoire der Ruhesachverhalte hat die Mächtigkeit 1. Jedesmal aber, wenn sich an diesem Ort ein Geschehen abspielt, kann es sich nur um ein Geschehen aus dem diskreten vorbestimmten Repertoire handeln.

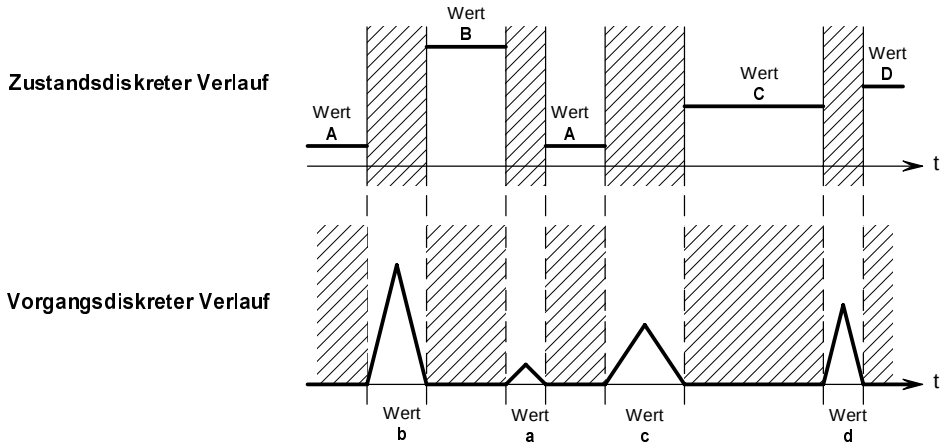

Bild 9

Den Unterschied zwischen Zustandsdiskretheit und Vorgangsdiskretheit kann man im Aufbaugraphen dadurch zum Ausdruck bringen, dass man die Knoten, welche die

Beobachtungsorte symbolisieren, in zwei unterschiedlichen Größen ausführt: ein großer Kreisknoten soll einen Ort mit zustandsdiskreten Erscheinungen symbolisieren, ein kleiner Kreis einen Ort mit vorgangsdiskreten Sachverhalten. Den großen Kreisknoten kann man sich als Behälter vorstellen, worin jeweils der diskrete Sachverhalt liegt. Den kleinen Kreis kann man sich als Wand eines Rohres vorstellen, durch welches die diskreten Elemente fließen.

Die Möglichkeit, zum System eine Aufbaustruktur in Form eines bipartiten Graphen anzugeben, wurde schon im Abschnitt 3.1.5 vorgestellt. Diese Darstellungsform für Aufbaustrukturen ergab sich also nicht erst durch die Einschränkung der Betrachtung auf diskrete Systeme. Die Einschränkung der Betrachtung auf diskrete Systeme bringt nun aber auch die Möglichkeit, Ablaufstrukturen in Form bipartiter Graphen darzustellen. Im Abschnitt 3.1 musste noch mit der Möglichkeit gerechnet werden, dass kontinuierliche Vorgänge an den Beobachtungsorten von Interesse sind. Zur Erfassung regelhafter Zusammenhänge zwischen kontinuierlichen Werteverläufen eignen sich in vielen Fällen die Differentialgleichungen. Nun aber, wo die Betrachtung auf diskrete Systeme eingeschränkt ist, gibt es kein Interesse mehr an kontinuierlichen Werteverläufen, so dass nun die Differentialgleichungen außer Betracht bleiben können. Es geht nun darum, regelhafte Zusammenhänge zwischen Folgen diskreter Vorgänge zu erfassen. Wenn man willkürlich zwei diskrete Vorgänge a_1 und a_2 aus dem gesamten Geschehen, welches sich im System abspielt, auswählt, kann es bezüglich ihrer zeitlichen Lage zueinander nur die folgenden Möglichkeiten geben (Die Wahl der Vorgangsbezeichnungen a_1 und a_2 gründet sich auf die Bezeichnung „Aktivität“.):

- Ihre tatsächliche Reihenfolge ist vollständig durch die Systemkonstruktion vorbestimmt, d.h. das Auftreten von a_1 und a_2 in der umgekehrten Reihenfolge oder die Gleichzeitigkeit sind durch die Systemkonstruktion ausgeschlossen.
  Beispiel: a_1 = Der große Zeiger der Bahnhofsuhr springt von 27 min auf 28 min.
  a_2 = Der große Zeiger der Bahnhofsuhr springt von 28 min auf 29 min.

- Die Systemkonstruktion schließt nur das gleichzeitige Auftreten der beiden Vorgänge aus, die tatsächliche Reihenfolge wird aber nicht durch die Systemkonstruktion bestimmt.
  Beispiel: a_1 = In den Münzeinwurfschlitz wird ein Markstück eingeworfen.
  a_2 = In den Münzeinwurfschlitz wird ein Fünfmarkstück eingeworfen.

- Die Systemkonstruktion bringt überhaupt keine Zwänge bezüglich der zeitlichen Lage dieser beiden Vorgänge mit sich, d.h. ob sie in einer bestimmten Reihenfolge oder gleichzeitig auftreten, ist keine Folge der Systemkonstruktion.
  Beispiel: a_1 = In der 3. Etage drückt jemand auf den Fahrstuhlanforderungsknopf.
  a_2 = In der 8. Etage drückt jemand auf den Fahrstuhlanforderungsknopf.

Im Abschnitt 3.1.3 über Ablaufstrukturen wurde gesagt, dass es darauf ankomme, in der unendlichen Menge aller kombinatorisch möglichen Werteverläufe die Teilmenge derjenigen Werteverläufe abzugrenzen, die an den jeweils betrachteten Orten auf Grund der Systemkonstruktion überhaupt möglich sind.

Die Werteverläufe haben keine wohldefinierte zeitliche Begrenzung, d.h. man muss mit potentiell unendlich langen Verläufen rechnen. Die Tatsache, dass das System zu irgendeinem Zeitpunkt außer Betrieb gesetzt wird, bringt keine wohldefinierte Beendigung der Verläufe, sondern lediglich einen willkürlichen Abbruch. Somit steht man vor dem Problem, in einer unendlichen Menge unendlich langer Werteverläufe eine Teilmenge abgrenzen zu müssen. Sowohl die Spezifikation der unendlichen Mengen als auch die Abgrenzung der Teilmenge müssen mit endlichem Aufwand darstellbar sein. Dieses Problem wurde bereits im Jahre 1961

von Carl Adam Petri theoretisch gelöst. Auf der Grundlage seiner Theorie wurde etwas später eine bestimmte Interpretation bipartiter Graphen entwickelt, wodurch diese Graphen zu den sogenannten Petrinetzen wurden. Es wird darauf verzichtet, die Theorie der Petrinetze im Rahmen des vorliegenden Aufsatzes einzuführen. Die vorliegende Betrachtung bleibt auf die sogenannten Stellen-Transitionsnetze beschränkt. Es handelt sich um gerichtete bipartite Graphen. Die durch Kreise symbolisierten Knoten heißen Stellen, und die durch Rechtecke symbolisierten Knoten heißen Transitionen. Diese gerichteten bipartiten Graphen werden dadurch zu Stellen-Transitionsnetzen, dass

1. jeder Stelle eine individuelle Kapazität in Form einer positiven ganzen Zahl zugeordnet wird,
2. eine sogenannte Anfangsmarkierung festgelegt wird, die zu jeder Stelle angibt, wieviel Marken jeweils dort liegen sollen. Die Anfangsmarkierung besteht also darin, dass jeder Stelle eine nichtnegative ganze Zahl zugeordnet wird, die die Kapazität der jeweiligen Stelle nicht überschreiten darf.
3. jeder gerichteten Verbindung eine sogenannte Paketgröße oder Flusszahl in Form einer positiven ganzen Zahl zugeordnet wird,
4. eine sogenannte Schaltregel angegeben wird.

Die Schaltregel dient dazu, die Anfangsmarkierung in andere Markierungen überführen zu können.

Die Schaltregel besteht aus zwei Teilen, der Definition des Schaltens und der Definition der Schaltbereitschaft.
Schalten: Beim Schalten werden von jeder Stelle, von der ein Pfeil zur Transition hinführt, so viele Marken entnommen, wie es die Paketgröße des Pfeils angibt, und anschließend werden auf allen Stellen, zu denen ein Pfeil von der Transition hinführt, so viele Marken hinzugefügt, wie die Paketgröße des jeweiligen Pfeils angibt. Dabei darf die Kapazität der Stellen nicht überschritten werden.
Schaltbereitschaft: Eine Transition ist schaltbereit, wenn alle an die Transition angrenzenden Stellen so markiert sind, dass ein Schalten möglich ist.

In Bild 10 sind vier Fälle dargestellt, wobei in allen vier Fällen die gleichen Kapazitätszahlen und Paketgrößen gelten:

- Im Fall a liegt keine Schaltbereitschaft vor, weil links oben nur eine Marke liegt, für das Schalten aber ein Paket der Größe 2 gebraucht wird.

- Im Fall b liegt keine Schaltbereitschaft vor, weil beim Herauslegen der Marken bei den beiden schattierten Stellen die Kapazität überschritten werden würde:
  Die Schleifenstelle links oben ist mit zwei Marken belegt; beim Schalten würde eine weggenommen und es müssten anschließend zwei hinzugelegt werden. Dadurch würden sich drei Marken auf der Stelle befinden, was die Kapazität 2 nicht zulässt.
  Unten rechts liegen bereits zwei Marken, beim Schalten kämen vier hinzu. Die Kapazität 5 erlaubt aber nicht die Aufnahme von insgesamt 6 Marken.

- In den Fällen c und d liegt jeweils Schaltbereitschaft vor. In diesen beiden Fällen ist zusätzlich zu der Situation, in der die Schaltbereitschaft geprüft wurde, auch noch die jeweilige Situation nach dem Schalten gezeigt.

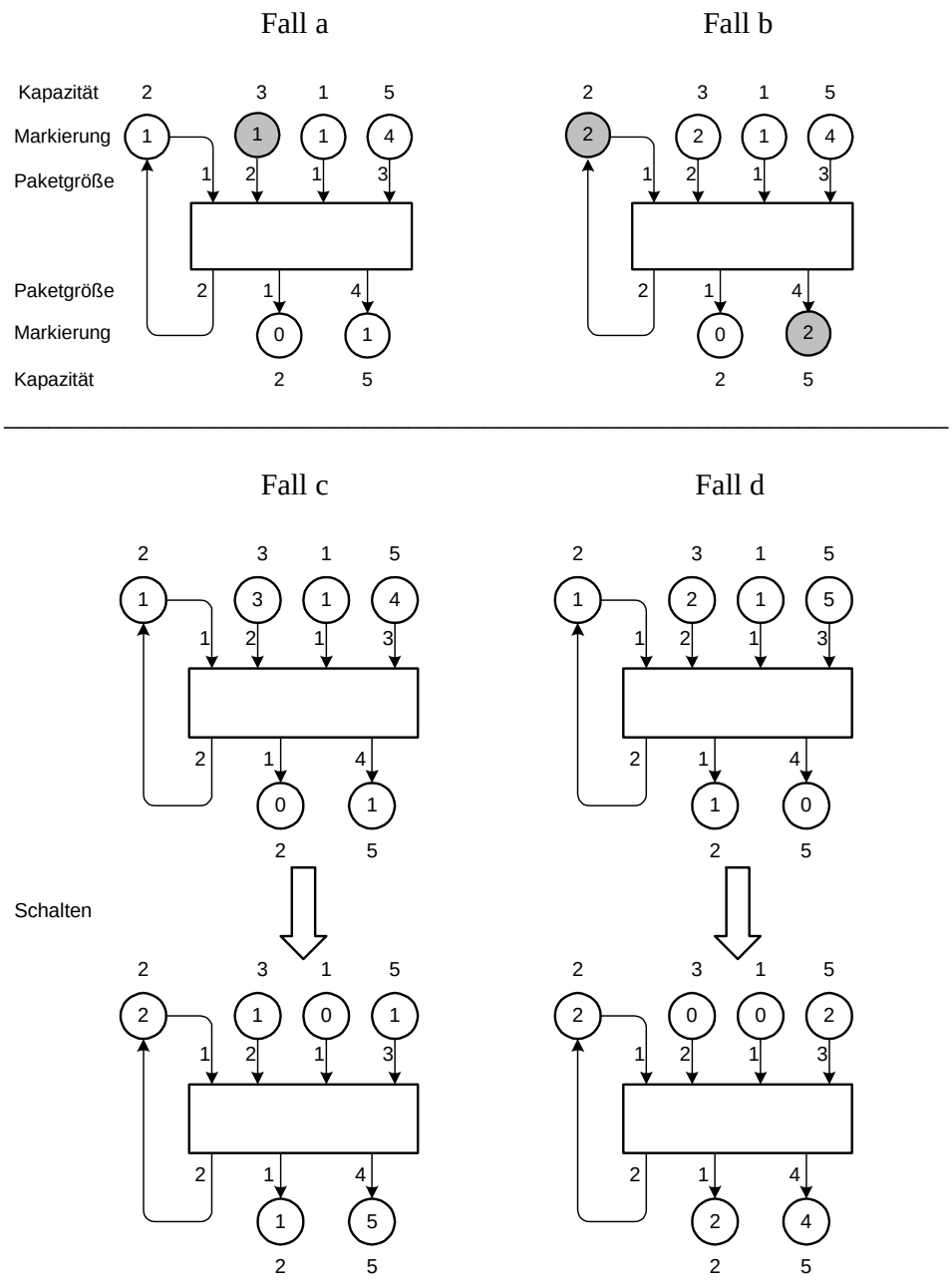

Bild 10

Weiter vorne wurde bereits festgestellt, dass zwei diskrete Vorgänge a_1 und a_2, deren Auftreten im System beobachtet wird, nur auf drei unterschiedliche Arten miteinander verbunden sein können. Diese drei Arten sind im Bild 11 jeweils durch Petrinetze veranschaulicht.

In diesen Petrinetzen sind weder Kapazitäten noch Paketgrößen eingetragen. Es gilt die Vereinbarung, dass die Kapazitäten und Paketgrößen alle den Wert 1 haben, wenn nichts angegeben ist. Dies ist der weitaus häufigste Fall. Die Markierung einer Stelle mit der Kapazität 1 kann nur 0 oder 1 sein. Aus Gründen der Anschaulichkeit wird in diesem Fall die Markierung nicht durch Eintragung einer Zahl angegeben, sondern durch einen schwarzen Punkt, der die Marke symbolisiert.

Im linken Netz in Bild 11 ist das Schalten nur nacheinander möglich, und zwar genau in der Reihenfolge a_1 vor a_2. Im mittleren Netz ist nur das gleichzeitige Schalten ausgeschlossen, denn da beide Transitionen für ihr Schalten die Marke von der mittleren Stelle benötigen, können sie nicht gleichzeitig schalten. Im rechten Netz ist das Schalten der beiden Transitionen unabhängig voneinander möglich.

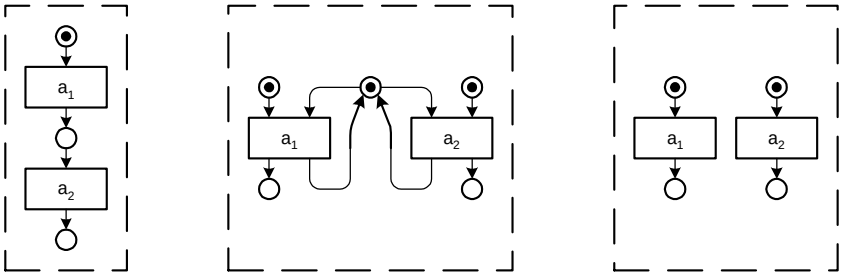

Bild 11

Ein Petrinetz ist keine Darstellung eines Protokolls über das Auftreten diskreter Prozessschritte. Es ist vielmehr die Formulierung einer Gesetzmäßigkeit für diskrete Abläufe. Das Petri-Netz ist ein generatives Schema, dem die damit generierbaren Strukturen zugeordnet sind.

Die Menge der Abläufe, die der als Petrinetz formulierten Gesetzmäßigkeit genügen, ist gleich der Menge aller Abläufe, die durch „Abwicklung“ des Petrinetzes nachvollziehbar sind. Dabei ist die anschauliche Vorstellung hilfreich, dass die Abwicklung von „einem Abwickler“ durchgeführt wird. Ausgehend von der Anfangsmarkierung sucht der Abwickler jeweils nach schaltbereiten Transitionen , von denen er anschließend alle oder nur eine Teilmenge schaltet. Es ist wichtig zu beachten, dass mit der Schaltbereitschaft einer Transition nicht der Zwang zu ihrem Schalten verbunden ist. Wegen der durch das Schalten bedingten Markierungsänderungen kann sich eine neue Menge schaltbereiter Transitionen ergeben, aus der wieder welche für das Schalten ausgewählt werden können.

Die durch die Abwicklung erzeugbaren Strukturen sind partialgeordnete Schaltereignisse. Ein Schaltereignis ist gekennzeichnet durch die Transition, die geschaltet wird, und durch die Ordnungsnummer, die angibt, wie oft diese Transition seit dem Beginn der Abwicklung geschaltet wurde. Beim ersten Schalten einer Transition T ergibt sich das Schaltereignis (T, 1). Ein Schaltereignis e_2 ist genau dann einem Schaltereignis e_1 unmittelbar nachgeordnet, wenn das gleichzeitige Schalten unmöglich war, aber unmittelbar nach e_1 die Schaltbereitschaft der zu e_2 gehörenden Transition entweder bestand oder durch das Schalten anderer Transitionen, die weder an e_1 noch an e_2 beteiligt waren, herbeigeführt werden konnte.

Die durch die Abwicklung von Petrinetzen generierbaren Strukturen lassen sich als gerichtete zyklenfreie Graphen darstellen, die man auch als Folgengeflechte bezeichnen kann. In Bild 12 ist ein Beispiel gezeigt. In diesem akademischen Beispiel können durch Abwicklung nur zwei endliche Folgengeflechte generiert werden, d.h. bei der Abwicklung dieses Netzes wird in jedem Falle eine sog. tote Markierung erreicht, bei der keine Transition mehr schaltbereit ist. Dagegen lassen sich mit Petrinetzen aus der Praxis meistens unendlich viele und unendlich lange Folgengeflechte generieren.

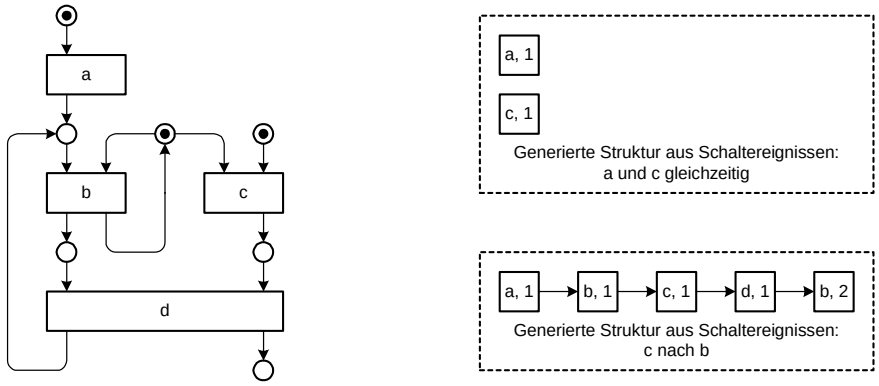

Bild 12

3.3 Einschränkung der Betrachtung auf diskrete und gerichtete Systeme.

Von nun an werden nur noch Systeme betrachtet, bei denen die Komponenten über Aus- und Eingänge miteinander verbunden sind. Die Vorstellung, dass zwei Systemkomponenten nur derart miteinander verbunden sein können, dass der Ausgang der einen Komponente mit dem Eingang der anderen Komponente verbunden ist, scheint auf den ersten Blick den Sachverhalt zu vereinfachen. Erst wenn man versucht, konkrete einfache Systeme auf dieses Modell abzubilden, stößt man auf Schwierigkeiten. Als Beispiel sei das System betrachtet, welches das Geschehen auf einem Schachbrett bei einem Schachspiel bestimmt. Als Systemkomponenten findet man hier sofort die beiden Schachspieler. Dann aber stellt sich die Frage, ob nicht auch das Schachbrett selbst eine Systemkomponente sei. Als Ort, auf dem das Geschehen beobachtbar wird, muss dieses Schachbrett sicher im Systemmodell vorkommen. Nun kann aber ein Ort nicht eine Komponente sein. Im Beispiel der elektrischen Schaltung im Bild 8 waren die Orte, an denen Ströme oder Spannungen gemessen wurden, selbstverständlich keine Komponenten des Systems. Die Frage, ob ein Ort auch eine Komponente sein dürfe, ergibt sich also erst als Konsequenz der Einschränkung der Betrachtung auf gerichtete diskrete Systeme.

Die Lösung dieses Problems ergibt sich dadurch, dass man zwischen aktiven und passiven Komponenten unterscheidet. Im Schachspiel gibt es als aktive Komponenten die beiden Schachspieler und als passive Komponente das Schachbrett. Dieses Schachbrett ist eine Komponente, mit der die beiden aktiven Komponenten in Wechselwirkung stehen. Sie können einerseits auf das Schachbrett einwirken, und sie können andererseits den Zustand des Schachbretts wahrnehmen. Die Aufbaustruktur dieses Systems ist in Bild 13 dargestellt. In diesem Bild gibt es allerdings keine symbolische Unterscheidung zwischen aktiven und passiven Systemkomponenten. Eine solche symbolische Unterscheidung wäre aber für das Systemverständnis sehr wünschenswert. Erfreulicherweise kann man eine solche symbolische Unterscheidung auf sehr einfache Weise erreichen.

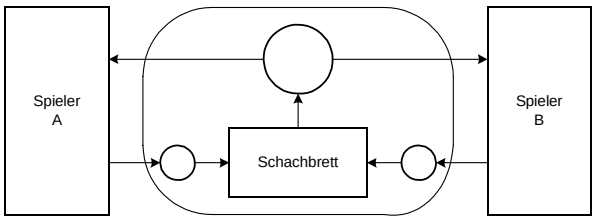

Bild 13

Man braucht nur die passive Komponente mit ihren Anschlüssen am Rand in ein Symbol der bisherigen Ortsklasse einzuschließen, denn allein auf der Grundlage der Pfeile, die zwischen diesem unechten Ort und den aktiven Komponenten liegen, kann man eindeutig darauf schließen, dass dies gar kein Ort sein kann. Man betrachte hierzu das Bild 14.

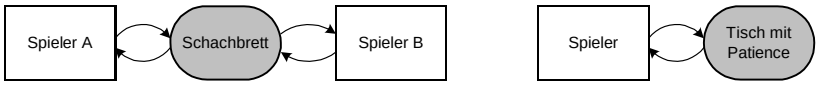

Bild 14

Wenn der Kreisknoten im Falle des Schachspiels ein Ort wäre, müsste er Ausgang von zwei Komponenten sein, denn von beiden Schachspielern geht ein Pfeil zu diesem Kreisknoten. Dies würde die Bedingung verletzen, dass ein Ort nur Ausgang einer Komponente sein darf. Also schließt man, dass dieser Kreisknoten eine passive Systemkomponente sein muss, und man kann unmittelbar die innere Struktur so angeben, wie sie in Bild 13 gezeigt ist. Auch im zweiten Beispiel, wo ein einzelner Akteur betrachtet wird, der sich durch Legen einer Patience mit Spielkarten die Zeit vertreibt, erkennt man, dass der Kreisknoten kein Ort sein kann, sondern eine passive Komponente sein muss. Diese passive Komponente ist der Tisch mit den darauf liegenden Spielkarten, auf denen der Spieler die Lage der Karten verändern kann. Wäre dieser Kreisknoten nur ein Ort, so wäre er gleichzeitig Ausgang und Eingang des Patiencespielers. Auch dies widerspricht der Festlegung, dass ein Ort in einem gerichteten System entweder nur Ausgang oder nur Eingang einer Komponente sein darf.

In Bild 14 wurden die beiden Kreisknoten, die passive Komponenten symbolisieren und keine Orte sind, schattiert dargestellt, um sie als Gegenstand der Problematik hervorzuheben. Im Folgenden aber wird auf eine Schattierung verzichtet, weil sich jeweils aus der Bepfeilung erkennen lässt, ob es sich um eine passive Komponente handeln muss.

Bei dieser Art der Symbolisierung passiver Komponenten bleibt allerdings doch noch ein Fall der Mehrdeutigkeit übrig. Man betrachte hierzu das Bild 15.

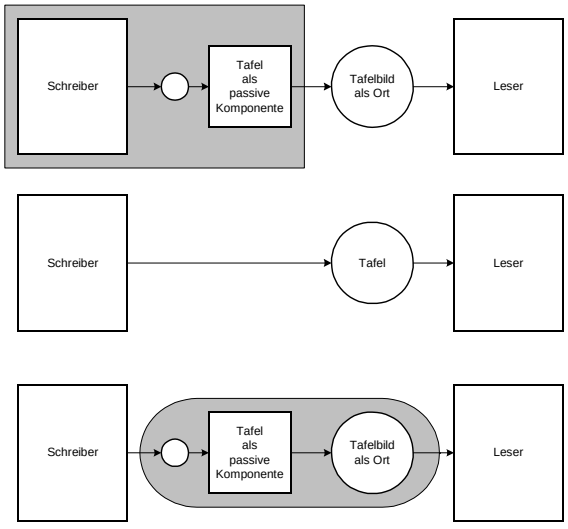

Bild 15

Bei der in der Mitte des Bildes liegenden Aufbaustruktur ist ein Schreiber mit einem Leser über eine Tafel gerichtet verbunden. Das Wort Tafel, welches im Kreisknoten steht, legt nicht eindeutig fest, ob es im Sinne von Tafelbild oder im Sinne von Tafel als Systemkomponente gedeutet werden soll. Oben im Bild ist eine Aufbaustruktur gezeigt, bei der die Tafel als

Komponente zusammen mit dem Schreiber einen umfassenderen Akteur bildet. Unten dagegen ist eine Aufbaustruktur gezeigt, bei der die Tafel als eigenständige passive Komponente im umfassenderen Kreisknoten liegt. Wenn man also ein Aufbaubild sieht, wie es in der Mitte des Bildes 15 gezeigt ist, weiß man, dass es einer späteren Verfeinerung vorbehalten bleiben muss zu entscheiden, ob der Kreisknoten in der Mitte tatsächlich als Ort oder als passive Komponente gemeint ist.

Mit der Beschränkung der Betrachtung auf gerichtete diskrete Systeme stößt man früher oder später auf die Möglichkeit, dass „sich ein System selbst verändern“ kann. Ein System bleibt unverändert, so lange sich seine Aufbaustruktur nicht ändert. Ein System, welches in der Lage ist, seine eigene Aufbaustruktur zu ändern, soll als strukturvariant bezeichnet werden.

Wenn für ein und dasselbe System zeitlich nacheinander unterschiedliche Aufbaustrukturen gelten, kann die Identität des Systems nicht durch seine Aufbaustruktur gegeben sein. Worauf sich die Identität eines strukturvarianten Systems gründet, erkennt man leicht, indem man Systeme aus Menschen betrachtet. Solche Systeme sind nämlich die anschaulichsten Beispiele für Strukturvarianz. In diesen Systemen bestehen die Vorgänge der Strukturvarianz darin, dass einzelne Menschen, die vorher nicht zum System gehörten, in das System aufgenommen werden, oder dass Menschen, die zum System gehören, aus dem System entfernt werden. Im Falle eines Unternehmens bedeutet die Hinzunahme das Einstellen neuer Mitarbeiter, und die Elimination bedeutet den Vorgang der Kündigung, bei der ein Mitarbeitet das Unternehmen verlässt. Man kann aber auch als Systembeispiel eine Familie betrachten. In diesem Fall geschieht die Hinzunahme durch den Vorgang der Geburt und die Elimination durch den Vorgang des Sterbens.

Die Einführung von Strukturvarianz ist zum einen an die Unterscheidung zwischen aktiven und passiven Komponenten und zum anderen an die Diskretheit der Vorgänge gebunden. Die Aktionen eines Akteurs lassen sich einteilen in strukturerhaltende Aktionen und in strukturverändernde Aktionen. Dies ist in Bild 16 veranschaulicht.

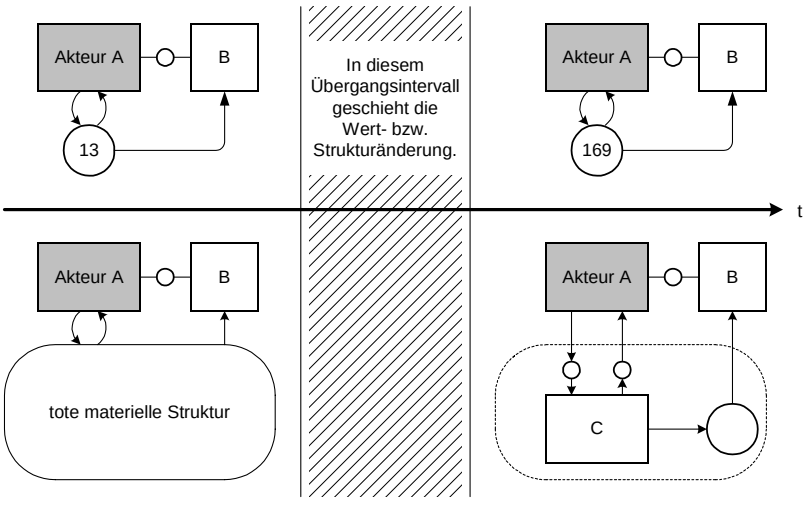

Bild 16

Oberhalb der Zeitachse ist eine strukturerhaltende Aktion des Akteurs A dargestellt. Der Akteur A hat Zugriff auf einen Speicher, worin vor der Aktion die Zahl 13 steht. Die Aktion besteht darin, dass der Akteur A die Zahl 13 aus dem Speicher löscht und das Quadrat von 13, also die Zahl 169, in den Speicher schreibt. Bei diesem Vorgang verändert sich die Aufbaustruktur des Systems nicht.

Unterhalb der Zeitachse ist eine strukturverändernde Aktion des Akteurs A dargestellt. Man stelle sich vor, der Akteur A sei ein begabter Mechaniker und Elektroniker, der einen Roboter gebaut hat. Der Roboter liege vorläufig noch als tote materielle Struktur auf der Werkbank, die als Behälter durch einen Rundknoten symbolisiert ist. Durch eine bestimmte Aktion des Akteurs A soll nun dieser Roboter „zum Leben erweckt“ werden. Man kann sich vorstellen, dass der Akteur einen Schalter am Roboter umlegt, der die Energieversorgung des Roboters einschaltet. Nach dieser Aktion ist der Roboter keine tote materielle Struktur mehr, sondern er ist nun zur Systemkomponente C geworden, die in Wechselwirkung mit den vorher schon vorhanden gewesenen Akteuren A und B tritt.

Die Elimination eines Akteurs aus dem System kann im betrachteten Beispiel nicht nur darin bestehen, dass der Akteur A den Roboter wieder ausschaltet und ihn dadurch wieder zu einer toten materiellen Struktur macht, die in einem Behälter liegt. Es könnte auch sein, dass der Roboter seinen Vater A erschlägt, wodurch dieser zu einer toten materiellen Struktur in einem Behälter wird.

Auf der Grundlage des Beispiels in Bild 16 kann man nun leicht definieren, was die Identität eines strukturvarianten Systems ausmacht. Ein strukturvariantes System behält seine Identität, so lange bei jedem strukturverändernden Vorgang mindestens einer seiner Akteure „am Leben bleibt“. Dies ist bei der Erzeugung neuer Komponenten immer der Fall, denn für eine solche Erzeugung wird immer einen Vater benötigt, und dieser verschwindet bei der Entstehung des Kindes nicht. Beim „Tod einer Komponente“ muss es dagegen nicht zwangsläufig einen Akteur geben, der die sterbende Komponente überlebt. Denn es könnte ja sein, dass der sterbende Akteur der letzte war, der im System noch lebte und der seinen eigenen Tod veranlasste. In diesem Fall verschwindet mit diesem Akteur das ganze System.

3.4 Einschränkung der Betrachtung auf diskrete informationelle Systeme

Im Abschnitt 3.3 war die Betrachtung auf diskrete und gerichtete Systeme eingeschränkt worden. Nun kommt als zusätzliche Einschränkung die Bedingung hinzu, dass die Systeme informationell sein sollen. Dass in der Überschrift zum vorliegenden Abschnitt das Attribut gerichtet nicht vorkommt, bedeutet nicht, dass nun eine frühere Einschränkung wieder aufgehoben worden sei. Das Attribut gerichtet konnte in der Überschrift entfallen, weil es durch die beiden Attribute diskret und informationell impliziert wird. Man kann sich kein ungerichtetes System vorstellen, welches diskret und informationell ist.

Man bezeichnet ein System als informationell, wenn es bei den Sachverhalten, die an den verschiedenen Orten im System beobachtet werden können, nicht auf die materiell-energetische Erscheinung ankommt, sondern nur auf die Interpretation. Die beobachtbaren Sachverhalte sind selbstverständlich auch in informationellen Systemen materiell-energetischer Natur, aber diese Natur ist für den Zweck des Systems irrelevant. Die Unterscheidung zwischen materiell-energetischer Relevanz und informationeller Relevanz war bereits Thema des Abschnitts 3.1.1.

In den bisherigen Betrachtungen gab es keinen Anlass, über die Wertebereiche, also über die Mengen aller an den verschiedenen Orten im System potentiell zu irgendeinem Zeitpunkt beobachtbaren Sachverhalte länger nachzudenken. Die im Abschnitt 3.1.2. angestellten Überlegungen erschienen als ausreichend. Solange man nur materiell-energetisch relevante Beobachtungssachverhalte betrachtet, kennt man die Wertebereiche schon aus der Physik,

denn es handelt sich um Messwerte physikalischer Größen. Durch die Einschränkung der Betrachtung auf informationelle Systeme ist nun aber das Thema Wertebereich zu einem zentralen Thema geworden. Es geht nun nicht mehr um messbare physikalische Größen, sondern um die Interpretation der gemessenen Sachverhalte. Interpretation aber beruht auf mehr oder weniger willkürlichen Festlegungen. Man denke beispielsweise an die Zeichen des lateinischen Alphabets oder an die arabischen Ziffern. Es sind zuerst einmal nur unterscheidbare grafische Formen. Ihre jeweilige Bedeutung lernt man in den ersten Jahren der Grundschule.

Über informationelle Systeme kann man nur angemessen kommunizieren, wenn man sorgfältig darauf achtet, die Betrachtung der Bedeutungen streng von der Betrachtung der Formen zu trennen. Für den Zweck eines informationellen Systems sind nur die Bedeutungen relevant, nicht aber die Formen, die der Konstrukteur als Träger der Bedeutungen gewählt hat. Wenn man also vom Wertebereich eines Ortes in einem informationellen Systems spricht, meint man immer die Menge der an diesem Ort potentiell zu irgendeinem Zeitpunkt findbaren Information, die man jeweils durch Interpretation einer aktuell nicht interessierenden Form gewinnt. Wenn man über einen solchen Wertebereich spricht, möchte man also nicht über den Interpretationsvorgang sprechen, sondern nur über das Ergebnis der Interpretation.

Ein informationeller Wertebereich kann elementar oder strukturiert sein. Ein Wertebereich für einen Ort ist elementar, wenn die an diesem Ort vorfindbare Information jeweils als atomar gedacht werden kann. Allgemein bekannte elementare Wertebereiche sind BOOLEAN, INTEGER, REAL und CHARACTER. Zusätzlich zu den in den Programmiersprachen eingeführten elementaren Wertebereichen kann man aber einfach durch Aufzählung endlicher Mengen neue elementare Wertebereiche definieren, beispielsweise den Wertebereich der Farben im Skatspiel {Karo, Herz, Pik, Kreuz}.

Es kennzeichnet die strukturierten Wertebereiche, dass sich die einzelnen Werte nur als Strukturen denken lassen. Dabei ist das Wort Struktur im mathematischen Sinne gemeint, wie er im Abschnitt 2.1. vorgestellt wurde. Ein Wert eines strukturierten Wertebereichs muss also mindestens eine Menge und eine Relation enthalten. Wenn jeder Wert eines strukturierten Wertebereichs eine Struktur ist, ist der Wertebereich also eine Menge von Strukturen. Wegen ihrer riesigen Mächtigkeit können diese Mengen von Strukturen nicht durch Aufzählung ihrer Elemente beschrieben werden. Wie man einen solchen Wertebereich beschreiben kann, erkennt man leicht durch die Betrachtung eines Beispiels.

In den Bildern 2 und 3 wurde ein Strukturbeispiel vorgestellt. Diese Struktur ist gekennzeichnet durch eine Menge von Personen, eine Menge von Büchern und eine Relation, die durch die Aussageform „Die Person p hat das Buch b gelesen.“ erfasst wird. Die durch Bild 2 oder Bild 3 definierte Struktur wird nun als Wert eines Wertebereichs betrachtet, bei dem alle Werte dadurch gekennzeichnet sind, dass es eine Personenmenge gibt, eine Büchermenge und eine Leserelation. Zur Beschreibung des Wertebereichs genügen also die einfachen Existenzaussagen, die angeben, welche Mengen und welche Relationen in den Werten des Bereichs vorkommen müssen. Die Beschreibung einer Menge von Strukturen durch eine Menge von Existenzaussagen über Mengen und Relationen bildet selbst wieder eine Struktur, denn die aufgezählten Mengen bilden eine Menge, die aufgezählten Relationen bilden eine zweite Menge und die Aussagen darüber, welche Mengen an welchen Relationen beteiligt sind, beschreiben eine Relation. Diese den Wertebereich charakterisierende Struktur kann man als bipartiten Graphen darstellen. Das betrachtete Beispiel eines Wertebereichs von Strukturen, wo jeweils eine Menge von Personen über die Leserelationen mit einer Menge von Büchern verbunden ist, wird durch den bipartiten Graphen in Bild 17 angemessen erfasst.

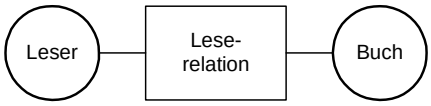

Bild 17

Die Entscheidung, in diesem Graphen die Mengen durch Rundknoten und die Relation durch einen Rechteckknoten zu symbolisieren, geschah im Hinblick darauf, dass man in der Mathematik Mengen schon seit sehr langer Zeit immer durch Kreise oder durch Ovale veranschaulicht hat.

In mathematischer Sicht haben Elemente keine Eigenschaften, sondern sie existieren und stehen in Relation zu anderen Elementen, die auch keine Eigenschaften haben. Die Unterscheidbarkeit der Elemente wird in der Mathematik dadurch garantiert, dass jedes Element auf eine individuelle Weise in die Struktur eingebunden ist. Grundsätzlich kann man auch jeden informationellen Wertebereich auf diese Weise sehen. Eine solche Sichtweise ist aber nicht verständnisfördernd. Es ist viel anschaulicher, Elemente mit Eigenschaften anzunehmen, die den Elementen ihre Individualität verleihen. Bei der Betrachtung des Bildes 17 stellt man sich vor, dass die Leser Eigenschaften haben, die sie unterscheidbar machen, und dass auch die Bücher durch ihre Eigenschaften unterscheidbar sind. Da die Leser und die Bücher gegenständliche Elemente sind, ist die Vorstellung, dass sie Eigenschaften haben, ganz natürlich. Man kann sich aber auch bei den nichtgegenständlichen Elementen der Leserelation vorstellen, dass sie mit irgendwelchen Informationen attributiert sind. Denn die Aussage, dass ein Leser ein bestimmtes Buch gelesen habe, bezieht sich ja auf einen konkreten Vorgang, und dieser Vorgang kann durch Attribute gekennzeichnet werden. Beispielsweise könnte es von Interesse sein, wie lange der Leser gebraucht hat, das Buch zu lesen, oder wie alt der Leser war, als er das Buch gelesen hat.

Dass die Frage der Attributierung keine im mathematischen Sinne entscheidbare Frage ist, sondern allein nach Zweckmäßigkeitsgesichtspunkten entschieden werden muss, soll anhand des Graphen im Bild 18 verdeutlicht werden.

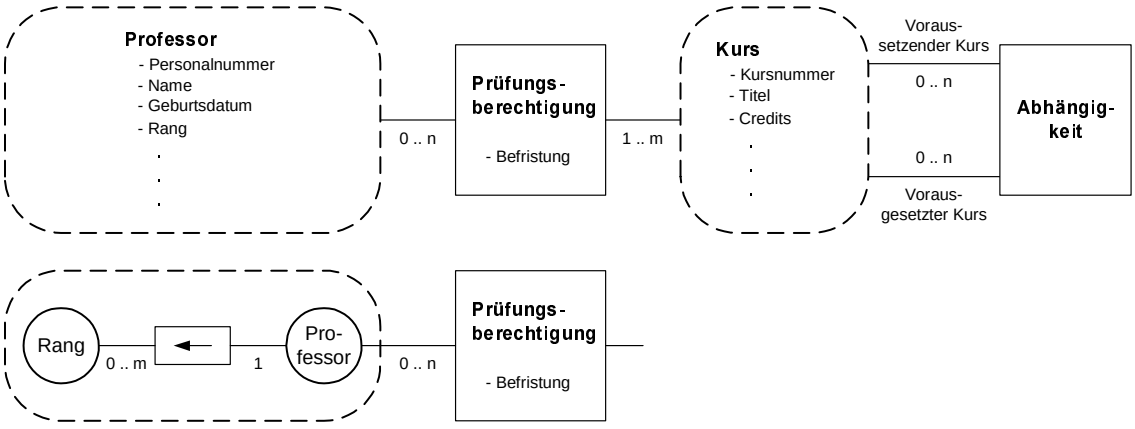

Bild 18

Der obere Graph besteht aus zwei Mengenknoten und zwei Relationsknoten. Es geht um Professoren und Kurse an einer Universität. Die eine Relation drückt aus, dass Professoren Prüfungsberechtigungen für Kurse erhalten können; die zweite Relation drückt aus, dass ein Kurs einen anderen Kurs zur Voraussetzung haben kann. In den Knoten sind beispielhaft unvollständige Attributlisten angegeben. Für die Voraussetzungsrelation gibt es keine

naheliegende Attributierung, dagegen konnte die Prüfungsberechtigung beispielhaft durch die Angabe einer Befristung attributiert werden.

Eines der Attribute eines Professors ist sein Rang, wobei angenommen werden soll, dass es die drei möglichen Ränge Assistant Professor, Associate Professor und Full Professor gibt. In der unteren Hälfte des Bildes 18 ist der Rang nicht mehr als Attribut eines Professors erfasst, sondern in Form einer eigenständigen Menge. In diesem Fall muss es eine Relation zwischen dem Professor und seinem Rang geben. In den Relationsknoten ist ein Pfeil eingetragen, der vom Professor zum Rang zeigt, was ausdrücken soll, dass bei gegebenem Professor eindeutig sein Rang festliegt.

Auf den ersten Blick scheint es so, als ob die beiden Graphen in Bild 18 jeweils den gleichen informationellen Wertebereich beschreiben. Tatsächlich aber enthält ein Wert aus dem zum unteren Graphen gehörenden Wertebereich mehr Information als der entsprechende Wert im Wertebereich, den der obere Graph beschreibt. Um dies einzusehen, braucht man sich nur vorzustellen, alle Professoren im aktuellen Wert hätten den gleichen Rang, beispielsweise den Rang Full Professor. Falls die Information aus dem Wertebereich stammt, der durch den oberen Graphen beschrieben wird, ist darin nicht das Wissen enthalten, welche anderen Ränge außer dem Full Professor es noch gibt, denn diese Ränge kommen ja nicht als Attributwerte der aktuellen Professoren vor. Wenn die Information dagegen aus dem Wertebereich stammt, der durch den unteren Graphen beschrieben wird, dann ist durchaus in dieser Information das Wissen über die gesamte Menge aller möglichen Ränge enthalten. Ob der untere Graph oder der obere Graph für den Wertebereich genommen wird, ist also keine Frage der Modellierung, sondern eine Frage der Systemkonstruktion, d.h. das Konstrukteur muss sich festlegen, ob er den oberen oder den unteren Wertebereich realisiert.

In Bild 18 sind die Kanten zwischen den Mengenknoten und den Relationsknoten beschriftet, und auf diese Beschriftung wurde bisher noch nicht eingegangen. Für das Verständnis der sogenannten Rollenbezeichnungen benötigt man keine Interpretationshilfe. Solche Rollenangaben stehen an den Kanten zwischen dem Mengenknoten der Kurse und der Voraussetzungs-
relation. Rollenangaben sind immer dann erforderlich, wenn es zwischen einem Relationsknoten und einem Mengenknoten mehr als eine Kante gibt, denn dann steht zwangsläufig jede dieser Kanten für eine andere Rolle, in der das Mengenelement an der Relation teilnimmt. Im vorliegenden Fall gibt es bei jeder Voraussetzungsabhängigkeit eine Vorlesung, die eine andere voraussetzt, und die Vorlesung, die vorausgesetzt wird.

Im Unterschied zu den Rollenangaben sind die sogenannten Kardinalitätsangaben ohne eine explizite Interpretationshilfe nicht verständlich. Diese Angaben beziehen sich auf den Sachverhalt, dass jede Relation eine Teilmenge eines kartesischen Produkts ist. Die Anzahl der Tupel im kartesischen Produkt, in denen ein bestimmtes Element einer Faktormenge vorkommt, ist gleich dem Produkt der Mächtigkeiten aller anderen Faktormengen. Die Anzahl der Tupel in der Relation, in denen dieses betrachtete Element vorkommt, kann höchstens so groß sein wie die entsprechende Anzahl im kartesischen Produkt. Da jede Kante zwischen einem Mengenknoten und einem Relationsknoten eine Faktormenge symbolisiert, kann man das Wissen über die möglichen Tupelvorkommen in Form einer Zahl oder eines Zahlenintervalls an die Kante schreiben. Dabei steht der Kleinbuchstabe n oder m jeweils für die durch die Mengenmächtigkeiten vorgegebene maximale Anzahl.

Diese allgemeinen Aussagen werden nun durch ein Beispiel veranschaulicht.

Professorenmenge = { p1, p2, p3, p4, p5 }

Kursmenge = { ka, kb, kc }

Prüfungsberechtigung = { (p1, ka), (p2, ka), (p4, ka), (p4, kb), (p5, kc) }

In dieser Relation gibt es die folgenden aktuellen Anzahlen der Elementvorkommen:

p1	1		ka	3
p2	1		kb	1
p3	0		kc	1
p4	2			
p5	1			

Die Kardinalitätsangaben in Bild 18 sind somit wie folgt zu interpretieren:

Die Angabe 0..n zwischen dem Professorenknoten und der Prüfungsberechtigungsrelation besagt, dass die Anzahl der Kurse, für die ein Professor die Prüfungsberechtigung haben kann, im Minimalfall 0 sein darf und im Maximalfall alle Kurse umfassen kann. Die Angabe 1..m auf der Seite der Kurse besagt, dass für einen Kurs mindestens ein Professor die Prüfungsberechtigung haben muss, dass es aber auch zulässig ist, dass alle Professoren für den betrachteten Kurs die Prüfungsberechtigung haben.

An den beiden Kanten, welche die Voraussetzungsrelation mit der Kursmenge verbinden, steht jeweils die Beschriftung 0..n. Dadurch wird zum Ausdruck gebracht, dass für beide Anzahlen die gleiche Obergrenze gilt; diese Obergrenze ist in diesem Fall gleich der um 1 verminderten Mächtigkeit der Kursmenge. Es kann nämlich sein, dass eine Vorlesung alle anderen Vorlesungen voraussetzt, und es kann umgekehrt sein, dass eine Vorlesung von allen anderen vorausgesetzt wird.

Bei der Relation, welche den Professoren ihren Rang zuordnet, steht auf der Seite der Professoren eine 1. Dies bedeutet, dass jedem Professor genau ein Rang zugeordnet sein muss. Auf der Seite der Ränge steht die Angabe 0..m, was bedeutet, dass es Ränge geben darf, die aktuell von keinem Professor besetzt sind, und dass es den Extremfall geben darf, dass alle Professoren den gleichen Rang haben. Der Kleinbuchstabe m steht also hier für die Mächtigkeit der Professorenmenge.

Wenn in einem Wertebereichsgraphen an mehreren Kanten der gleiche Kleinbuchstabe vorkommt, bedeutet dies nur dann eine relevante Gleichheitsaussage, wenn es sich um Beschriftungen der Kanten zu ein und derselben Relation handelt – in Bild 18 also im Falle des n an den Kanten der Voraussetzungsrelation. Wenn man nicht bei den verschiedenen Relationen immer wieder die gleichen Buchstaben verwenden dürfte, ohne damit eine inhaltliche Gleichheit zu implizieren, würde man bei umfangreicheren Graphen sehr bald das Alphabet erschöpfen.

Die hier vorgestellten Graphen sind allgemein unter der Bezeichnung Entity-Relationship-Diagramme, abgekürzt ERD bekannt. Allgemeine Verwendung haben sie zur Darstellung von Datenbankschemata gefunden. Es gibt aber nicht den geringsten Grund für diese eingeschränkte Verwendung, denn nicht nur in Datenbanken werden strukturierte Informationen vorgefunden, sondern auch an sehr vielen anderen Orten in informationellen Systemen.

Nachdem nun die Entity-Relationship-Diagramme eingeführt sind, muss noch gezeigt werden, dass diese Diagramme nicht ausreichen, sämtliche Sachverhalte bezüglich eines strukturierten informationellen Wertebereichs darzustellen. Hierzu wird das Beispiel in Bild 19 betrachtet. Es handelt sich um eine Erweiterung des Beispiels aus Bild 18.

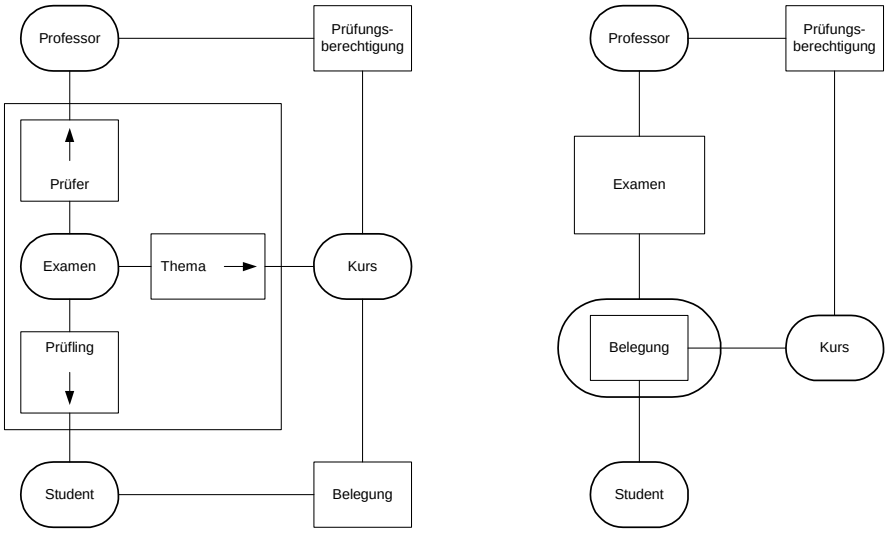

Bild 19

Zuerst soll das Diagramm in der linken Hälfte des Bildes 19 betrachtet werden. Zusätzlich zu den bereits in Bild 18 eingeführten Mengen und Relationen gibt es hier noch die Studentenmenge, die Examensmenge, drei Relationen zur Einbindung der Examensmenge in die Struktur und die Belegungsrelation, welche die Studenten mit den Kursen verbindet. Die Voraussetzungsrelation über den Kursen wurden gegenüber Bild 18 weggelassen. Mit den graphischen Mitteln des Diagramms kann nicht ausgedrückt werden, dass die Zuordnung eines Prüfers und eines Themas zu einer Prüfung im Einklang damit stehen muss, dass der gewählte Prüfer die Berechtigung für den gewählten Kurs haben muss. In gleicher Weise kann das Diagramm nicht zum Ausdruck bringen, dass ein Prüfling, der sich über ein bestimmtes Thema prüfen lassen will, den zugehörigen Kurs belegt haben muss. Beide Bedingungen hängen mit Maschen des Graphen zusammen. Die eine Masche wird gebildet durch die Knoten

Examen – Prüfer – Professor – Prüfungsberechtigung – Kurs – Thema.

Die andere Masche wird gebildet durch die Knoten

Examen – Thema – Kurse – Belegung – Student – Prüfling.

Die Existenz von Maschen in Entity-Relationship-Diagrammen ist immer ein Hinweis darauf, dass es möglicherweise Abhängigkeiten zwischen den in der jeweiligen Masche liegenden Relationen gibt, die graphisch nicht ausgedrückt werden können. Diese Abhängigkeiten kann man in der Sprache der Prädikatenlogik formulieren. Die entsprechende Formel, die als Ergänzung zum linken Graphen in Bild 19 hinzukommen muss, lautet:

$$\forall e, p, s, k: \quad [\,(e, p) \in \text{Prüfer}\,] \text{ UND } [\,(e, s) \in \text{Prüfling}\,] \text{ UND } [\,(e, k) \in \text{Thema}\,]$$
$$\rightarrow \quad [\,(p, k) \in \text{Prüfungsberechtigung}\,] \text{ UND } [\,(s, k) \in \text{Belegung}\,]$$

In der rechten Hälfte des Bildes 19 wird der gleiche Wertebereich beschrieben wie in der linken Bildhälfte. Es fällt auf, dass hier der Graph einfacher ist als in der linken Darstellung. Diese Vereinfachung wurde durch zwei Maßnahmen erreicht. Zum einen wurde die Examensmenge zusammen mit ihren drei Zuordnungsrelationen zu einer Relation zusammengefasst, und zum anderen wurde die Belegungsrelation als Menge aufgefasst, die an einer Relation beteiligt werden kann. Der rechte Graph enthält nur noch eine Masche, so dass hier nur noch die Abhängigkeit zwischen der Examensrelation und der Prüfungsberechtigungsrelation prädikatenlogisch formuliert werden muss:

$$\forall p, s, k: \quad [\,(p, (s, k)) \in \text{Examen}\,] \rightarrow [\,(p, k) \in \text{Prüfungsberechtigung}\,]$$

Eine weitere Vereinfachung des Graphen, in der auch noch die letzte Masche zum Verschwinden gebracht wäre, gibt es nicht. Das bedeutet, dass man hier in jedem Falle zusätzlich zum Graphen einen prädikatenlogischen Bedingungsausdruck benötigt.

Zum Thema Wertebereich gibt es nicht nur die nun abgeschlossenen Überlegungen über strukturierte Wertebereiche und ihre Darstellung in Form von Entity-Relationship-Diagrammen. Zum Thema Wertebereich gehört auch der Begriff der Polymorphie, der nun vorgestellt werden soll. Jeder Speicher ist ein Behälter für Werte aus einem ganz bestimmten Wertebereich. Bisher durfte der Leser die Vorstellung haben, dass einem Speicher ein bestimmter konkreter - elementarer oder strukturierter - Wertebereich, beispielsweise INTEGER oder REAL zugeordnet ist. Nun kann man aber auch einen neuen Wertebereich als Vereinigung bereits definierter Wertebereiche festlegen. So kann man sich beispielsweise einen Wertebereich vorstellen, der die Vereinigung der beiden Wertebereiche INTEGER und REAL darstellt. Ein Speicher mit diesem Wertebereich kann also jeweils eine Zahl enthalten, wobei man die Möglichkeit hat, wahlweise einmal eine INTEGER-Zahl und ein andermal eine REAL-Zahl in diesen Speicher einzubringen.

Wenn der Wertebereich eines Speichers als Vereinigung unterschiedlicher Wertebereiche festgelegt ist, spricht man von einem „polymorphen“ Speicher. Polymorph heißt vielgestaltig, was auf die Vielgestaltigkeit der möglichen Speicherinhalte hinweisen soll.

3.5 Semantische Probleme bei der Variation des Detaillierungsgrades

In den voranstehenden Abschnitten wurde als graphisches Darstellungsmittel für Aufbaustrukturen, Ablaufstrukturen und Wertebereichsstrukturen jeweils der bipartite Graph als geeignet erkannt. Bei der formalen Betrachtung bipartiter Graphen liegt der Begriff der bipartitheit-erhaltenden Vergröberung sehr nahe. Solche bipartitheit-erhaltenden Vergröberungen kamen in den bisherigen Abschnitten schon mehrfach vor, ohne dass dort explizit darauf hingewiesen wurde. Man betrachte die Graphen in den Bildern 13, 15, 16, 18 und 19. In den ersten drei der genannten Bilder sind Aufbaustrukturen dargestellt, in den letzten beiden Bildern Entity-Relationship-Diagramme.
Eine bipartitheit-erhaltende Vergröberung entsteht dadurch, dass man mehrere Knoten durch einen Oberknoten zusammenfasst, wobei alle Kanten, die den Rand des Oberknotens schneiden, im Inneren zu Knoten führen, die von der gleichen Art sind wie der Oberknoten. Wenn der Oberknoten ein Rechteckknoten ist, müssen also alle von außen in dieses Rechteck hineinführende Kanten im Inneren wieder auf Rechteckknoten führen. Wenn dagegen der

Oberknoten ein Rundknoten ist, dann müssen alle Kanten, die den Rand dieses Rundknotens schneiden, im Inneren wieder auf Rundknoten führen.

Während die Definition des Kriteriums, woran man eine bipartitheit-erhaltende Vergröberung erkennen kann, sehr einfach ist, ist die Frage, wie eine solche Vergröberung zu interpretieren sei, meist recht problematisch. Es gibt nur einen einzigen Fall, wo die Interpretation einer bipartitheit-erhaltenden Vergröberung unproblematisch ist, und dies ist der Fall der Vergröberung durch einen Rechteckknoten in einer Aufbaustruktur. Rechteckknoten in Aufbaustrukturen symbolisieren Akteure, und jede Zusammenfassung eines akteursberandeten Teilnetzes zu einem neuen Knoten stellt wieder einen Akteur dar. Die innerhalb dieses Oberakteurs liegenden Rundknoten stellen seine inneren Speicher und Kommunikationswege dar, und seine inneren Akteure tragen die Aktionen, die man dem Oberakteur zurechnet. Die Zusammenfassung mehrerer Akteure zu einem Oberakteur ist nur deshalb unproblematisch, weil von Akteuren nicht verlangt wird, dass sie sequenziell agieren. Sie müssen nicht unbedingt ihre einzelnen Aktionen nacheinander ausführen. Es wird zwar sicher in den Systemen sequenziell agierende Akteure geben, aber die Sequenzialität ist kein Bestimmungskriterium für den Akteursbegriff.

Neben diesem unproblematischen Vergröberungsfall gibt es noch fünf mehr oder weniger problematische Fälle:

(1) Vergröberung durch einen Rundknoten in einer Aufbaustruktur;
(2) Vergröberung durch einen Rundknoten in einer Ablaufstruktur;
(3) Vergröberung durch einen Rechteckknoten in einer Ablaufstruktur;
(4) Vergröberung durch einen Rundknoten in einem Entity-Relationship-Diagramm;
(5) Vergröberung durch einen Rechteckknoten in einem Entity-Relationship-Diagramm.

Der Fall (1) wurde bereits bei der Betrachtung von Aufbaustrukturen diskutiert. Diese Art der Vergröberung dient einerseits dazu, passive Komponenten von aktiven Komponenten zu unterscheiden, und andererseits dazu, Strukturvarianz zu symbolisieren.

Die Fälle (2) und (3) betreffen die Vergröberungen in Ablaufstrukturgraphen, also in Petrinetzen. Hier haben die meisten Vergröberungen keine sinnvolle Interpretation, und sie sind deshalb nutzlos. Bild 20 zeigt ein Beispiel für den seltenen Fall, dass Vergröberungen in Petrinetzen sinnvoll sein können. In diesem Bild ist sowohl eine Vergröberung zu einem Rechteckknoten als auch eine Vergröberung zu einem Rundknoten gezeigt, und man erkennt, dass das Kriterium dafür, dass diese Vergröberungen sinnvoll sind, für beide Fälle das gleiche ist: Die Vergröberungen müssen durch zwei prozessbegrenzende Transitionen eingeklammert sein. Diese beiden Transitionen sind in Bild 20 schattiert dargestellt. Bei der Rechteckvergröberung liegen diese beiden Transitionen im Inneren des umfassenden Knotens, bei der Rundknotenvergröberung liegen sie in unmittelbarer Nachbarschaft außen. Durch das Schalten der prozesseröffnenden Transition werden Schaltvorgänge im Inneren des umfassenden Rundknotens möglich. Durch das Schalten der prozessabschließenden Transition werden jegliche weiteren Schaltvorgänge im Inneren des umfassenden Rundknotens ausgeschlossen.

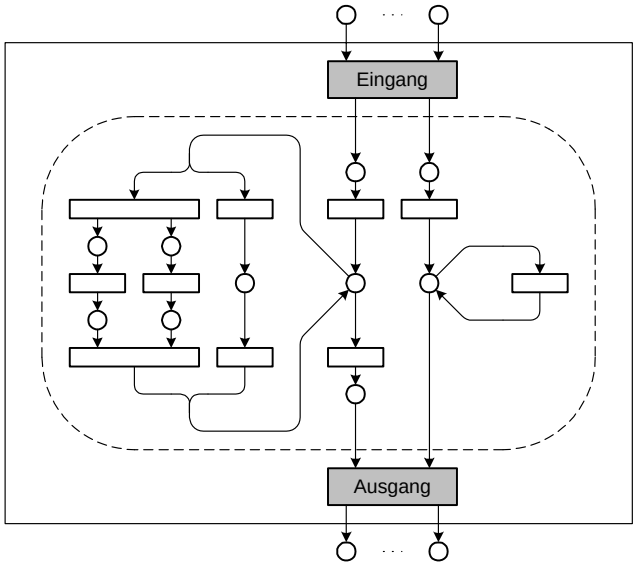

Bild 20

Der umfassende Rechteckknoten kann durchaus als Transition in einem Petrinetz betrachtet werden. Durch die Vergröberung wird zum Ausdruck gebracht, dass man sich für die Details beim Schalten dieser großen Transition nicht interessiert. Der umfassende Rundknoten kann durchaus als Stelle in einem Petrinetz angesehen werden. Solange diese Stelle markiert ist, können irgendwelche Vorgänge stattfinden, für die man sich nicht interessiert. Beide Vergröberungen verändern die generierbaren Partialordnungen für das Schalten der außerhalb der Vergröberung liegenden Transitionen nicht.

In gleicher Weise wie bei den Petrinetzen sind auch im Falle der Entity-Relationship-Diagramme die meisten bipartitheit-erhaltenden Vergröberungen nicht sinnvoll interpretierbar. Je ein Beispiel einer interpretierbaren Vergröberung für die Fälle (4) und (5) sind in den Bildern 18 und 19 gezeigt. Im einen Fall wird ein Teilnetz, welches aus einer Menge und lauter eindeutig zuordnenden Relationen besteht, durch eine Relation ersetzt. Im anderen Fall wird ein Teilnetz, dass nur über eine einzige Kante mit dem restlichen Graphen verbunden ist, durch einen Rundknoten ersetzt. Dieser umfassende Rundknoten wird als Symbol für eine Menge strukturierter Elemente angesehen, für deren Struktur man sich nicht interessiert.

4. Die softwaretechnische Begriffswelt

4.1 Rollensystem versus Trägersystem

Im Kapitel 3 wurde bei der Systembetrachtung nicht danach gefragt, ob die Systeme programmiert realisiert sind oder nicht. Von nun an aber konzentriert sich die Betrachtung auf programmiert realisierte Systeme. Diese Systeme zeichnen sich dadurch aus, dass eine geeignete Systembeschreibung in den Speicher einer universellen Hardware eingebracht wird, wodurch diese Universalhardware zu dem gewünschten System wird. Man kann sagen, dass die Universalhardware dazu gebracht wird, eine bestimmte Rolle zu spielen, so wie ein Schauspieler als Universalperson dazu gebracht werden kann, die Rolle einer bestimmten Person in einem Theaterstück zu spielen. Deshalb nennen wir das gewünschte System, zu dem die Hardware durch Einbringen der Beschreibung gemacht wird, das Rollensystem.

Es ist zweckmäßig, nicht nur den Fall zu betrachten, dass die Rollenbeschreibung in die nackte Hardware eingebracht wird, sondern auch den Fall, dass die Rollenbeschreibung in eine Hardware eingebracht wird, die durch eine zuvor schon eingebrachte Software zu einem System gemacht wurde, dessen Fähigkeiten über die der nackten Hardware hinausgehen. Deswegen soll das System, in das die Rollenbeschreibung eingebracht wird, nicht einfach als Hardware bezeichnet werden, sondern als das Trägersystem. In der Analogie zur Schauspielerei ist also der Schauspieler ein Trägersystem, welches in der Lage ist, Rollenbeschreibungen durch Lernen in seinen Speicher hereinzuholen.

Es ist wichtig zu erkennen, dass es sinnlos wäre, im Rollensystem das Trägersystem zu suchen. Das Trägersystem ist kein Teil des Rollensystems, sondern das Rollensystem ergibt sich nur durch eine besondere Sicht auf das Trägersystem. Wenn der Schauspieler Heinz Rühmann den Hauptmann von Köpenick spielt, dann ist beispielsweise die linke Hand des Schauspielers Rühmann identisch mit der linken Hand des Hauptmann von Köpenick. Es ist sinnlos, in der Gestalt des Hauptmann von Köpenick Teile des Schauspielers Rühmann zu suchen. Es gibt gar keine Komponenten im Hauptmann von Köpenick, die nicht auch Komponenten des Schauspielers Rühmann wären.

Der umgekehrte Fall ist jedoch möglich, das heißt der Schauspieler Rühmann kann Komponenten haben, die eine bestimmte von ihm gespielte Figur nicht hat. Man denke an die Möglichkeit, dass eine einarmige Theaterfigur von einem Schauspieler gespielt wird, der noch beide Arme hat. In diesem Fall muss der Schauspieler einen Arm bei seinem Rollenspiel verstecken, so dass er für die Zuschauer verborgen bleibt. Es darf also Komponenten im Trägersystem geben, die im Rollensystem nicht auftauchen. Man kann also ein Aufbaubild des Trägersystemes angeben, worin das Rollensystem eingebettet ist. Man kann jedoch kein Aufbaubild des Rollensystems angeben, worin das Trägersystem eingebettet wäre.

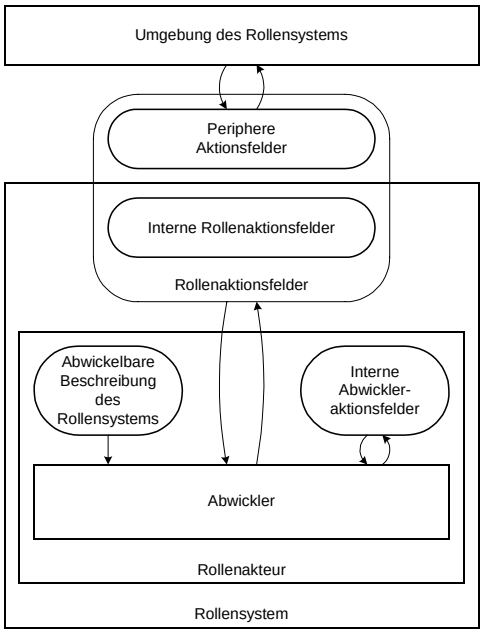

Bild 21

Im Bild 21 ist das allgemein gültige Aufbaumodell für Rollensysteme dargestellt. Das Rollensystem besteht aus einem Rollenakteur und den internen Rollenaktionsfeldern. Der Rollenakteur agiert auf den Rollenaktionsfeldern, das heißt dort werden die Aktionen des Rollenakteurs in Form von Zustandsänderungen sichtbar. Über die peripheren Rollenaktions-

felder steht der Rollenakteur in Wechselwirkung mit der Umgebung des Rollensystems. Wenn das Rollensystem durch Einbringen der Rollenbeschreibung in ein Trägersystem realisiert wird – und dieser Fall ist im Bild 21 angenommen, findet man im Inneren des Rollenakteurs drei Komponenten: Man findet den Speicher, worin die abwickelbare Beschreibung des Rollensystems abgelegt ist. Man findet den Abwickler, der die Beschreibung lesen kann und die Rolle abwickelt. Dazu agiert er nicht nur auf dem Rollenaktionsfeld, sondern auch auf einem internen Abwickleraktionsfeld, welches dem Beobachter des Rollensystemverhaltens verborgen bleibt. Man stelle sich beispielsweise vor, die Rollenbeschreibung läge als Ablaufstrukturplan in Form eines Petrinetzes vor. In diesem Fall wäre das interne Abwickleraktionsfeld erforderlich zur Speicherung der jeweils aktuellen Markierung, aus der sich die als nächste zu schaltenden Transitionen des Netzes ergeben.

Bild 22 zeigt ein Aufbaumodell des Trägersystems, worin das Rollensystem eingebettet ist. Man erkennt, dass das Trägersystem neben den für das Rollensystem benötigten Komponenten noch weitere Komponenten hat. Bevor das Trägersystem zum Rollensystem werden kann, muss die Rollenbeschreibung in den dafür vorgesehenen Speicher eingebracht werden. Dazu kommuniziert die Umgebung des Trägersystems mit dem Einbringassistenten, welcher schreibenden Zugriff auf den Rollenspeicher hat.

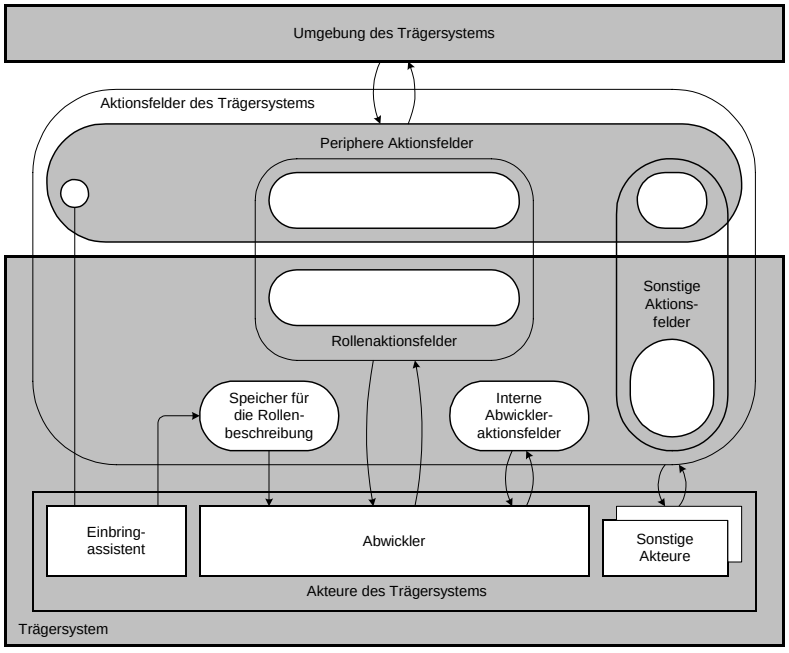

Bild 22

Das Bild sagt nichts darüber aus, in welcher Form die Umgebung des Trägersystems die Information über die zu spielende Rolle liefert. Der bislang häufigste Fall sieht so aus, dass die Rollenbeschreibung vollständig als Text formuliert vorliegt und übergeben wird. Es handelt sich um den vom Programmierer formulierten sogenannten Quelltext. Es gibt aber auch den derzeitig noch seltenen, aber in Zukunft möglicherweise häufiger vorkommenden Fall, dass die Übergabe der Information über die zu spielende Rolle in Form eines Interaktionsprozesses zwischen der Umgebung des Trägersystems und dem Einbringassistenten mitgeteilt wird. Man denke an Dialoge unter Verwendung graphischer Benutzeroberflächen.

Der Aufbaustrukturplan des Trägersystems in Bild 22 sagt nichts darüber aus, wie dieses Trägersystem realisiert wird. Es könnte sein, dass dieses Trägersystem ausschließlich durch Hardwarekonstruktion realisiert wird. Unmittelbar nach der Erfindung des Computers gab es

keine anderen Trägersysteme als die rein hardwarerealisierten. In der reinen Hardwarerealisierung muss der Einbringassistent eine vom Abwickler getrennte Hardwarekomponente sein. In den damaligen Computern gab es tatsächlich eine separate Hardware, die ausschließlich dem Zweck der Einspeicherung der Programme in den Programmspeicher diente. Während der Einbringaktion blieb der Abwickler in Wartestellung. Erst nach Abschluss des Einbringvorganges wurde der Abwickler beauftragt, mit der Abwicklung zu beginnen.

Bei reiner Hardwarerealisierung bleiben die Fähigkeiten des Einbringassistenten und des Abwicklers zwangsläufig recht begrenzt. Wenn man deren Fähigkeiten beträchtlich erweitern will, muss man dazu übergehen, den Einbringassistenten und den Abwickler programmiert zu realisieren. Das bedeutet, dass man das Trägersystem nun selbst zu einem Rollensystem erklärt, welches man in ein umfassenderes Trägersystem einbetten muss. Auf diese Weise kann man zu einer Schichtung von Trägersystemen kommen, die zusammen mit dem ursprünglichen Rollensystem wie die Schichten einer Zwiebel ineinander liegen. Die äußerste Schicht muss zwangsläufig ein ausschließlich durch Hardware realisiertes Trägersystem sein. Dann folgen nach innen nacheinander die jeweils durch Programmierung realisierten Trägersysteme, und ganz im Inneren eingebettet liegt dann das programmiert realisierte ursprüngliche Rollensystem.

4.2 Typen von Trägersystemen

4.2.1 Klassifikationskriterien

Die Universalität der Trägersysteme findet ihre Grenze dort, wo die Rollensysteme materiell-energetische Relevanz einbringen. Und die gilt immer für diejenigen Rollenaktionsfelder, über die das Rollensystem in Wechselwirkung mit seiner Umgebung steht. Wenn das Rollensystem verteilt ist, das heißt, wenn das Rollensystem an unterschiedlichen Orten mit seiner Umgebung in Wechselwirkung steht, dann muss selbstverständlich auch das Trägersystem entsprechend verteilt sein. Konkret bedeutet dies, dass ein Trägersystem mit den entsprechenden Ein- und Ausgabegeräten ausgestattet sein muss, wenn dies das Rollensystem verlangt. In diesem Fall muss auch die Beschreibung des Rollensystems, die in das Trägersystem eingebracht wird, auf diese Verteilung bezogen sein. Es würde den Rahmen der vorliegenden Schriften sprengen, wenn man versuchen wollte, ein reales Beispiel für ein verteiltes Trägersystem vorzustellen. Ohne Bezugnahme auf ein solches Beispiel lassen sich aber keine interessanten allgemeingültigen Erkenntnisse bezüglich der Verteilung von Trägersystemen vermitteln. Deshalb wird dieser Problembereich im folgenden nicht mehr weiter behandelt.

Damit ein Trägersystem zu dem gewollten Rollensystem werden kann, muss es die von seiner Umgebung gelieferte Beschreibung des Rollensystems verstehen können. Ein Trägersystem lässt sich deshalb anhand der Begriffswelt klassifizieren, die dem Trägersystem „per Konstruktion bekannt“ ist, so dass die Umgebung die Beschreibung des gewünschten Rollensystems unter Bezug auf diese Begriffswelt formulieren darf. Das bedeutet, dass eine Klassifikation von Programmiersprachen unmittelbar in eine Klassifikation von Trägersystemen überführt werden kann. Wenn man von Mischformen absieht, kann man die Programmiersprachen grob in zwei Klassen einteilen, die dann auch für die entsprechenden Trägersysteme gelten. Die eine Klasse umfasst die Sprachen bzw. die Trägersysteme für prozedurale Programmierung, die auch imperative Programmierung genannt wird, und die andere Klasse umfasst die Sprachen bzw. Trägersysteme für nichtprozedurale Programmierung.

Es muss an dieser Stelle darauf hingewiesen werden, dass sich die Klassifikation der Trägersysteme nicht automatisch auf die Klassifikation der Abwickler überträgt. Es kann sehr wohl sein, dass in einem Trägersystem für nichtprozedurale Programmierung ein Abwickler für prozedurale Programmierung sitzt. In einem solchen Trägersystem muss dann ein Einbringassistent sitzen, der das von der Umgebung gelieferte nichtprozedurale Programm in ein prozedurales Programm übersetzt. Das übersetzte Programm steht dem Abwickler im Speicher zur Verfügung.

4.2.2 Trägersysteme für prozedurale Programmierung

Wenn man von Programmierung spricht, meint man meistens implizit die prozedurale Programmierung. Die prozedurale Programmierung ist untrennbar mit der Erfindung des Computers verbunden, d.h. dass die Erfinder des Computers ein Trägersystem für prozedurale Programmierung erfunden haben.

Die anschaulichste Form eines prozeduralen Programms ist das Petrinetz, dessen Transitionen mit Operationsanweisungen beschriftet sind und an dessen Verzweigungsästen Prädikate stehen, die sich auf Belegungen der Operandenspeicher beziehen. In Bild 23 ist ein Beispiel eines in Form eines Petrinetzes dargestellten prozeduralen Programms angegeben. Wenn dieses Petrinetz im Programmspeicher stünde und der Abwickler ein Mensch wäre, könnte er dieses Programm abwickeln, falls er die Beschriftungen korrekt interpretieren kann.

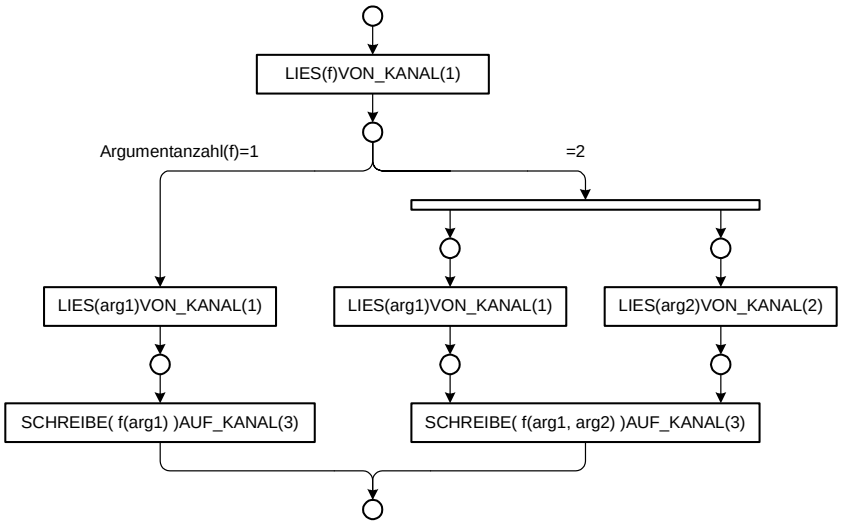

Bild 23

Bild 24 zeigt die Aufbaustruktur des Rollensystems, dessen Verhalten durch das Petrinetz in Bild 23 beschrieben wird. Das Rollensystem enthält hier nur einen einzigen Akteur, der lesen, berechnen und schreiben kann. Er berechnet das durch den Funktionsbezeichner f und das zugehörige Argumenttupel eindeutig bestimmte Ergebnis.

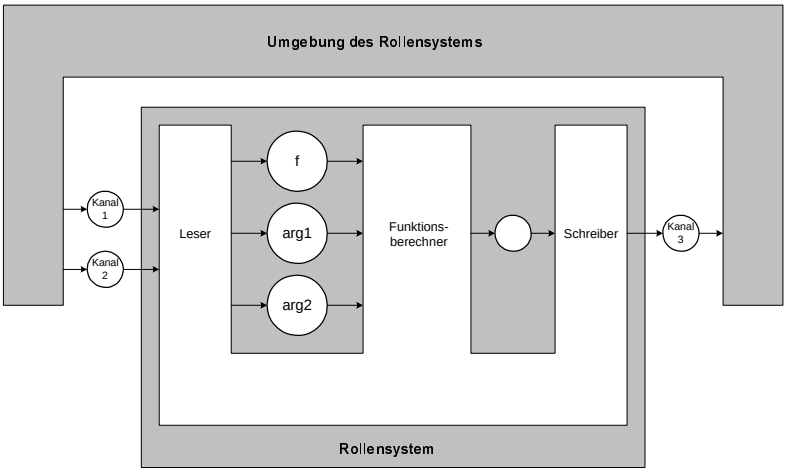

Bild 24

Wenn man einem Trägersystem oder einem Abwickler ein Programm übergibt, muss man immer davon ausgehen dürfen, dass per Konstruktion des Trägersystems bereits bestimmte Bezeichner verwendet werden dürfen, ohne dass der Programmierer dem System die diesen Bezeichnern zuzuordnenden Bedeutungen mitteilen muss. In dem Beispielsprogramm im Bild 23 kommen die folgenden Bezeichner vor:

Bezeichner für Speicherzellen:	f, arg1, arg2
Funktionsbezeichner:	Argumentanzahl ( )
Bezeichner für Operationen:	LIES ( ) VON_KANAL ( ), SCHREIBE( ) AUF_KANAL ( )
Bezeichner für Werte:	1, 2, 3

Es wird angenommen, dass das Trägersystem die Bezeichner für Operationen und die Bezeichner für Werte interpretieren kann. Außerdem darf angenommen werden, dass das Trägersystem erkennen kann, dass es sich bei den Bezeichnern arg1, arg2 und f um Speicherzellenbezeichner handelt und bei dem Bezeichner Argumentanzahl ( ) um einen Funktionsbezeichner. Was das System sicher nicht weiß und deshalb nur vom Programmierer erfahren kann, sind die Wertebereiche für die Belegung der Speicherzellen und die Definition der Funktion Argumentanzahl. Das gesamte abzuliefernde Programm kann also nicht nur aus dem Petrinetz bestehen, sondern muss noch zusätzliche Informationen enthalten, der hier als „Erklärungsblock“ bezeichnet werden soll. Der Erklärungsblock zum Petrinetz in Bild 23 könnte die folgende Form haben:

```
TYPE_DEFINITION
   funktionsmenge = {SIN, COS, SUM, PRODUCT};

FUNCTION_DEFINITION
   argumentanzahl:INTEGER(x:funktionsmenge);
   IF x ∈ {SIN, COS} THEN argumentanzahl = 1
                     ELSE argumentanzahl = 2;

STORAGE_ALLOCATION
   f:funktionsmenge;
   arg1, arg2:REAL;
```

In diesem Erklärungsblock kommen drei Arten von Bezeichnern vor:

- Bezeichner, deren Interpretation dem Trägersystem per Konstruktion bekannt ist; diese Bezeichner sind mit Großbuchstaben geschrieben;
- Bezeichner, die auch im Petrinetz vorkommen und deren Interpretation dem Trägersystem durch den Erklärungsblocksblock mitgeteilt werden soll;
- Der Bezeichner `Funktionsmenge`, der nur innerhalb des Erklärungsblockes als Bezeichner für einen Wertebereich verwendet wird.

Die am Beispiel vorgestellte Zweiteilung des Programms in den eigentlichen abwickelbaren Teil und den Erklärungsblock, der für die Interpretation des abwickelbaren Teils gebraucht wird, ist bei allen prozeduralen Programmen mehr oder weniger ausgeprägt zu finden. Den abwickelbaren Programmteil kann man den technischen Trägersystemen selbstverständlich nicht in Form eines Petrinetzgraphen übergeben; das Petrinetz lässt sich aber immer in einen Programmtext überführen, der bezüglich des Abwicklers die gleiche Information enthält wie der Graph.

In dem betrachteten Beispiel gibt es für die Grenzziehung zwischen dem abwickelbaren Teil des Programms und dem Erklärungsblock keine Alternative. Weder findet man im Petrinetz Teile, bei denen man sich fragen könnte, ob sie nicht vielleicht doch in den Erklärungsblock gehören, noch findet man im Erklärungsblock Teile, die man vielleicht eher ins Petrinetz hätte eintragen sollen. Es gibt jedoch viele prozedurale Programme, bei denen die Grenzziehung zwischen dem abwickelbaren Teil und dem Erklärungsblock keineswegs so klar auf der Hand liegt wie im betrachteten Beispiel. Diese Problematik soll anhand des im Bild 25 dargestellten Programmbeispiels verdeutlicht werden.

In dem Beispiel in Bild 25 geht es um ein Rollensystem, dessen Aktivität aus drei aufeinanderfolgenden Schritten besteht: Zuerst erfolgt ein Eingabevorgang, dann schließt sich ein Berechnungsvorgang an, und zuletzt kommt ein Ausgabevorgang. Es wird angenommen, dass der Programmierer drei Unterprogramme formuliert hat, jeweils eines für die Eingabe, die Berechnung und die Ausgabe. Das Hauptprogramm dient nur noch dazu, diese drei Unterprogramme in der richtigen Reihenfolge nacheinander aufzurufen.

Man könnte nun rein formal das gesamte Programm so aufteilen, wie es in der linken Hälfte des Bildes 25 gezeigt ist: Nur das Hauptprogramm wird als abwickelbarer Programmteil betrachtet, während die drei Unterprogramme dem Erklärungsblock zugeordnet werden. Denn definitionsgemäß soll der Erklärungsblock die Informationen enthalten, die der Abwickler benötigt, um den abwickelbaren Programmteil vollständig verstehen zu können. Bei dieser Aufteilung sieht man die drei Anweisungen des Hauptprogrammes nicht als reine Unterprogrammaufrufe, sondern man sieht sie als Anweisungen zur vollständigen Durchführung der in den Unterprogrammen näher beschriebenen Vorgänge.

Diese Sicht auf das Programm ist im betrachteten Falle sicher nicht zweckmäßig. Es ist viel angemessener, auch die Unterprogramme zum abwickelbaren Programmteil zu rechnen, wie dies in der rechten Hälfte des Bildes 25 dargestellt ist.

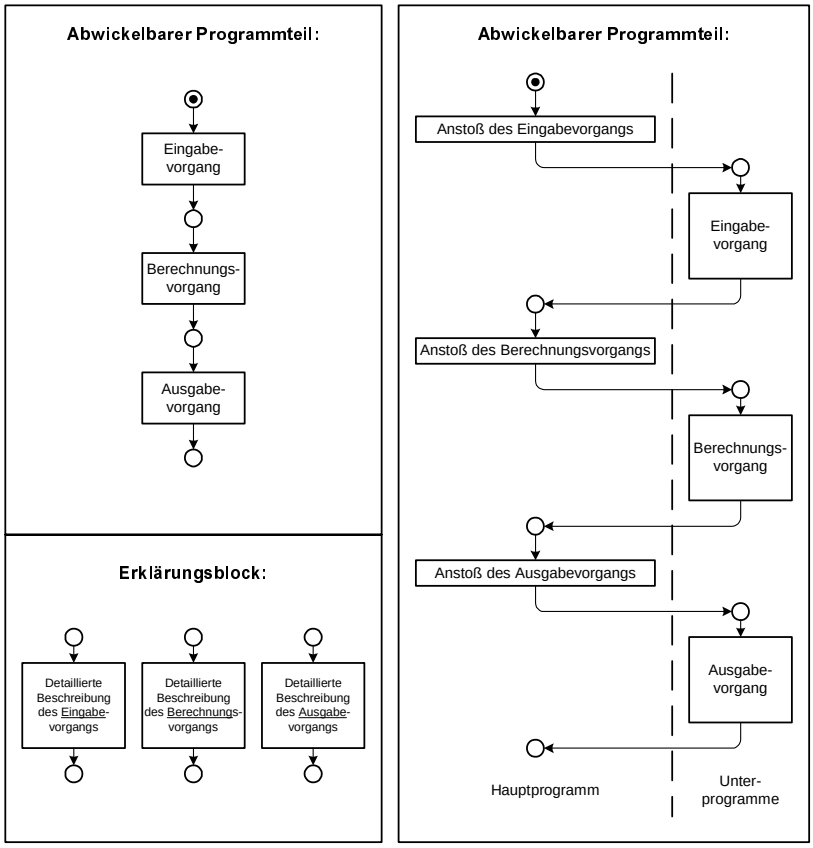

Bild 25 Modellierungsalternative 1 Modellierungsalternative 2

Man sollte keine Festlegung treffen derart, dass Unterprogramme grundsätzlich immer in den Erklärungsblock gehören, oder dass Unterprogramme grundsätzlich immer zum abwickelbaren Programmteil gehören. Es ist viel zweckmäßiger, in Abhängigkeit von der Art des Geschehens, welches im Unterprogramm beschrieben wird, das Unterprogramm entweder dem abwickelbaren Programmteil oder dem Erklärungsblock zuzuordnen. Die Zuordnung eines Unterprogramms zum Erklärungsblock ist immer dann angemessen, wenn im Unterprogramm ein Geschehen beschrieben wird, von dem man sich vorstellen kann, dass es der Abwickler auch schon hätte ausführen können, ohne dass es der Programmierer explizit formulieren musste. In diesen Fällen, wo die Unterprogramme zu Recht dem Erklärungsblock zugeordnet werden, stellen diese Unterprogramme also eine Erweiterung der Fähigkeiten des Abwicklers dar. Ganz typische Beispiele hierfür sind Unterprogramme zur Berechnung der Wurzel oder von trigonometrischen Funktionen wie Sinus und Cosinus. Aber auch ein Unterprogramm, welches einen Sortieralgorithmus beschreibt, gehört in diese Kategorie.

Man erkennt diese Art von Unterprogrammen daran, dass man sie ohne jeden Bezug zu den Hauptprogrammen einführen kann, aus denen heraus sie aufgerufen werden. Diese Bedingung war bei den drei Unterprogrammen des Bildes 25 nicht erfüllt. Die dortige Eingabeprozedur dient dazu, die Parameter zur Durchführung der Verarbeitung bereitzustellen, und deshalb ist die Eingabeprozedur ohne Betrachtung der Berechnungsprozedur nicht verständlich zu erklären. Auch die Berechnungsprozedur stellt keine Erweiterung der Fähigkeiten des Abwicklers dar, denn diese Prozedur stellt den Kern der Aufgabe des

Rollensystems dar und ist deshalb so speziell, dass eine Verwendung dieser Prozedur in einem anderen Kontext nicht vorstellbar ist.

In erster Näherung kann man sagen, dass der Erklärungsblock entfallen könnte, wenn der Abwickler ein Mensch wäre. Dann könnte dieser Mensch aufgrund der im abwickelbaren Programmteil vorkommenden Bezeichner für Werte, Speicherplätze, Funktionen und Operationen intuitiv auf deren Bedeutung schließen. So bräuchte man beispielsweise einem Menschen nicht zu erklären, was mit der Anweisung „Sortiere die gegebene Namensliste in alphabetischer Ordnung“ gemeint ist. Dem technischen Abwickler muss man aber die detaillierte Sortierprozedur mitteilen, weil für einen technischen Abwickler die Buchstabenfolge `sortiere` keine ausgezeichnete Stellung gegenüber anderen bezeichnenden Buchstabenfolgen wie beispielsweise `pp24f` oder `abcdxyz` hat.

Wenn man die Grenzziehung zwischen abwickelbarem Programmteil und Erklärungsblock in diesem Sinne wählt, hat man einen Beitrag zur Maximierung des Wirkungsgrades der Kommunikation über das betrachtete Programm geliefert. Man hat dann nämlich eine Grenze gezogen zwischen den Inhalten, die für das Verständnis unbedingt mitgeteilt werden müssen, und den restlichen Inhalten, die primär nicht für den menschlichen Kommunikationspartner, sondern nur für den technischen Abwickler zum Verständnis nötig sind.

Obwohl immer betont wird, dass die Objektorientierung ein völlig neues Programmierparadigma darstelle, sind auch die objektorientierten Programme grundsätzlich prozedural. In der folgenden Betrachtung soll herausgearbeitet werden, durch welche Merkmale sich die objektorientierten Programme von den konventionellen prozeduralen Programmen unterscheiden.

Alle prozeduralen Programme sind dadurch gekennzeichnet, dass man sich den Abwickler als Kern eines sogenannten Steuerkreises vorstellen kann, wie er im Bild 26 dargestellt ist.
Ein Steuerkreis besteht aus zwei Teilsystemen, die über zwei entgegengesetzt verlaufende Kanäle miteinander verbunden sind. Die Art der Verbindung ist der Grund für die Bezeichnung Kreis. Die üblicherweise unten liegend dargestellte Komponente wird als Steuerwerk, die oben liegende Komponente als Operationswerk bezeichnet. Die Verwendung des Begriffes „Werk“ stammt zwar aus der Schaltwerkstheorie, wird aber hier nicht in der engen schaltwerkstheoretischen Bedeutung verwendet.

Das Steuerwerk ist dadurch charakterisiert, dass es zwar immer weiß, welche Schritte auszuführen sind, dass es aber selbst diese Schritte nicht ausführen kann. Das Operationswerk ist dadurch gekennzeichnet, dass es einzelne Schritte auf Anforderung hin ausführen kann, aber selbst nichts über die Reihenfolge auszuführender Schritte weiß. Für das Steuerwerk äußern sich die auszuführenden Schritte als Anweisungen an das Operationswerk, und das Operationswerk muss so konstruiert sein, dass es jeden Schritt, den das Steuerwerk von ihm verlangt, ausführen kann. Dabei müssen nicht zwangsläufig alle Schritte nacheinander ausgeführt werden; es ist zulässig, dass das Steuerwerk die nebenläufige Ausführung mehrerer Schritte verlangt.

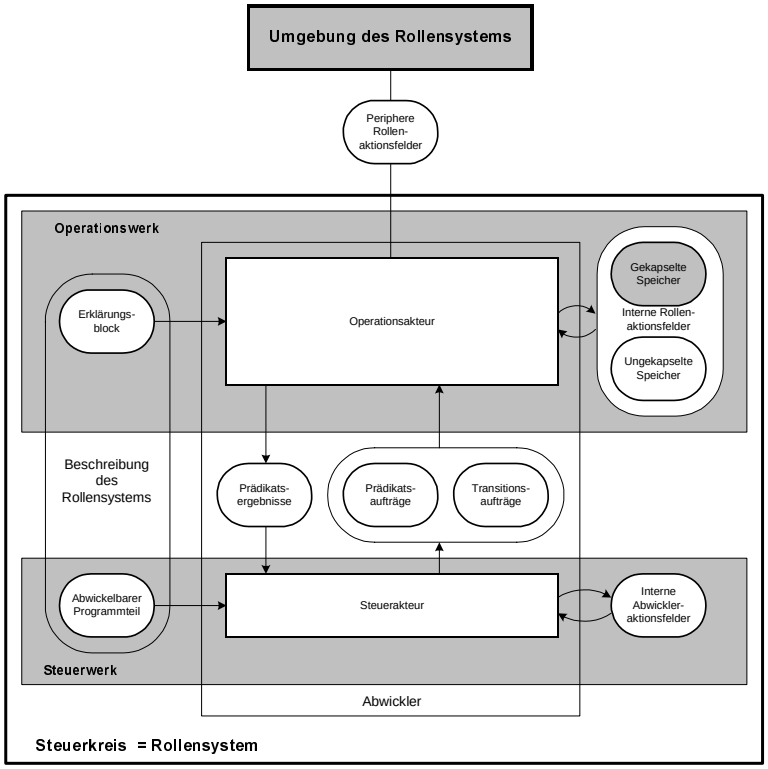

Bild 26

Da der Abwickler in Bild 26 aus zwei Teilen besteht, kann auch das Programm in zwei Teile zerlegt werden. Für den Steuerakteur ist nur der abwickelbare Teil des Programms interessant, während die Informationen im Erklärungsblock vom Operationsakteur gebraucht werden. Denn der Operationsakteur kann die vom Steuerakteur kommenden Aufträge nur verstehen, wenn er die im Erklärungsblock stehenden Informationen heranziehen kann.

Den abwickelbaren Programmteil kann man sich als beschriftetes Petrinetz vorstellen, wie es beispielsweise in Bild 23 vorgestellt wurde. Neben dem Programmspeicher findet man im Steuerwerk auch den Speicher für den zeitlich veränderlichen Steuerzustand. Dieser Speicher entspricht den internen Abwickleraktionsfeldern aus Bild 21. Wenn man sich das abwickelbare Programm als beschriftetes Petrinetz vorstellt, kann man sich den Steuerzustand als die zeitlich veränderliche Markierung des Petrinetzes vorstellen.

Das System in Bild 26 ist ein Sonderfall des Systems in Bild 21 ist. Im Falle des Bildes 21 ist die Art des Programms, also die Frage, ob das Programm prozedural sei oder nicht, noch offen geblieben. In Bild 26 aber ist nun das Programm prozedural, und dies hat dazu geführt, dass der Abwickler aus Bild 21 in einen Steuerakteur und in einen Operationsakteur aufgespalten werden konnte. Die Kanäle zur Verbindung des Steuerakteurs und des Operationsakteurs liegen innerhalb des Abwicklers. Anhand des Beispielprogramms in Bild 23 kann man sich leicht veranschaulichen, welche Informationen zwischen dem Steuerakteur und dem Operationsakteur ausgetauscht werden müssen. Der Steuerakteur muss dem Operationsakteur jeweils die Beschriftung der zu schaltenden Transitionen mitteilen. Hierzu dient der Kanal „Transitionsaufträge“. Außerdem muss der Steuerakteur dem Operationsakteur die Prädikate mitteilen, deren Wahrheitswert der Steuerakteur benötigt, um Schaltkonflikte zwischen Transitionen entscheiden zu können. Die Prädikate fließen über den Kanal „Prädikatsaufträge“ vom Steuerakteur zum Operationsakteur, und dieser liefert die entsprechenden Wahrheitswerte über den Kanal „Prädikatsergebnisse“ zurück.

Gegenüber der Darstellung in Bild 21 sind die internen Rollenaktionsfelder in Bild 26 zweigeteilt. Die internen Speicher wurden in gekapselte und ungekapselte Speicher unterteilt. Diese Unterteilung ist in der Unterschiedlichkeit dessen begründet, was in Transitions- und Prädikatsaufträgen bezüglich dieser Speicher verlangt werden kann. Die gekapselten Speicher sind dadurch gekennzeichnet, dass in den diesbezüglichen Aufträgen weder der Zuweisungsoperator noch die Identitätsfunktion zugelassen sind. Sowohl der Zuweisungsoperator als auch die Identitätsfunktion sind Elemente, die jeder prozedurale Abwickler per Konstruktion kennt. Bei der Verwendung des Zuweisungsoperators wird verlangt, dass ein auf der einen Seite des Operators identifizierter Wert an den auf der anderen Seite des Operators identifizierten Ort gebracht wird:

Ortsidentifikation := Wertidentifikation

Bei der Verwendung der Identitätsfunktion geht es darum, einen Wert durch die Identifkation des Ortes zu identifizieren, an dem dieser Wert gespeichert ist. So werden beispielsweise im Ausdruck i+1 drei Werte identifiziert:

- der Wert, der am Ort i gespeichert ist
- der durch das Symbol 1 identifizierte Wert eins
- die Summe der identifizierten beiden Summanden.

Ungekapselte Speicher kann man sich anschaulich als Tafeln vorstellen, die man voll im Blick hat, so dass man alles sieht, was darauf steht. Die Zulässigkeit des Zuweisungsoperators bedeutet, dass man unabhängig davon, was gerade auf der Tafel steht, jederzeit etwas beliebig Neues darauf schreiben kann. Die Zulässigkeit der Identitätsfunktion bedeutet, dass man durch die Identifikation der Tafel auch die auf der Tafel stehende Information identifiziert hat, denn man braucht ja nur auf die Tafel, die man nun kennt, hinzuschauen.

Einen gekapselten Speicher kann man sich anschaulich als das Gehirn eines Menschen vorstellen. In diesem Gehirn ist Information gespeichert, an die aber nur der Mensch durch „Hinschauen“ herankommt, dem das Gehirn gehört. Alle anderen Menschen können diesem Gehirn durch Hinschauen keine Information entnehmen. Der Verweis auf das Gehirn schafft ihnen also nicht automatisch den Zugang zu der dort gespeicherten Information. Die Identitätsfunktion ist also hier nicht anwendbar. Dass auch der Zuweisungsoperator nicht anwendbar ist, bedeutet in diesem anschaulichen Modell, dass man nicht auf einen Schlag einen völlig neuen vollständigen Gehirninhalt vorgeben kann, der dann eingespeichert werden soll. Vielmehr kann man immer nur Informationen anliefern, die der Empfänger auf irgendeine interne Weise zu einer Veränderung seines Gehirninhalts auswertet.

Das klassische Beispiel eines gekapselten Speichers, welches in den meisten Informatiklehrbüchern vorgestellt wird, ist der Stack. Der Wertebereich eines Stackspeichers ist strukturiert, d.h. man kann über ein einzelnes in diesem Speicher liegendes Exemplar vom Typ Stack nicht reden, ohne eine Struktur zu beschreiben. Das den Wertebereich darstellende Entity-Relationship-Diagramm ist in Bild 27 angegeben.

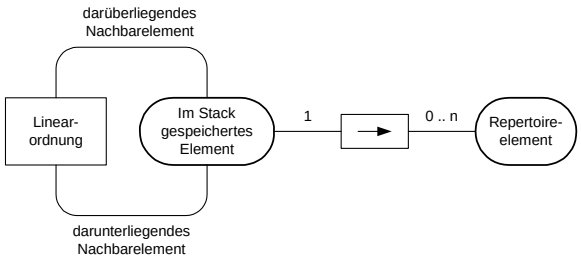

Bild 27

Üblicherweise werden für die Verwendung in Aufträgen bezüglich eines Stacks drei zustandsverändernde Operationen und zwei wertabfragende Funktionen zugelassen:

Operationen zur Zustandsveränderung:

CLEAR	verlangt, dass die Menge der im Stack gespeicherten Elemente gleich der leeren Menge sein soll;
PUSH(e)	verlangt, dass eine Kopie des Repertoire-Elements e als zusätzliches Element oben auf den bisherigen Stack gelegt wird;
POP	verlangt, dass das bisher oberste Element vom Stack entfernt wird;

Wertidentifizierende Funktionen:

TOP	identifiziert das aktuell zuoberst liegende Element;
HEIGHT	identifiziert die Anzahl der aktuell im Stack gespeicherten Elemente.

Wie sich die Unterscheidung zwischen gekapselten und ungekapselten Speicherzellen im Programmtext äußern könnte, zeigt das folgende Beispiel eines Programmauszugs:

Auszug aus dem Erklärungsblock:

```
OPEN_TYPE_DEFINITION
   complex = TUPEL(re:REAL, im:REAL);

CLOSED_TYPE_DEFINITION
   stack_of_'repertoire' = STRUCTURE …  ;
      OPERATIONS
      {  clear();
         push(e:repertoire);
         pop;
      }
      FUNCTIONS
      {  top:repertoire;
         height:INTEGER;
      };

FUNCTION_DEFINITION
   betrag:REAL(x:complex);
      betrag = SQRT(x.re * x.re  +  x.im * x.im)

STORAGE_ALLOCATION
     i, j:  INTEGER;
     x, y:  REAL;
   c1, c2:  complex;
        s:  stack_of_INTEGER;
```

Auszug aus dem abwickelbaren Programmteil:

```
        i := 5;
        j := 2*i+4;
        x := 6.75;
        y := 2.25*SQRT(x-1.2);
    c1.re := 4.0/x;
    c1.im := 4.0/y;
       c2 := c1*c1;
        y := betrag(c2)+c1.im;

    s.clear;
    s.push(j);
    s.push(6*i-1);

        j := s.top;
        i := s.height;
```

In diesem Programmbeispiel kommen zwei ungekapselte Speicher `c1` und `c2` für komplexe Zahlen vor und ein gekapselter Speicher `s` für einen Stack. Der Speichertyp für komplexe Zahlen wurde als Tupel definiert, wobei im Tupel zwei Speicher für reelle Zahlen zusammengefasst sind. Dass es sich hier um ungekapselte Speicher handelt, erkennt man daran, dass die Ortsbezeichnungen `c1.re`, `c1.im` und `c2` auf der linken Seite des Zuweisungsoperators vorkommen. Man erkennt es außerdem daran, dass die Ortsbezeichnungen `c1` und `c2` zum Zwecke der Wertidentifikation auf der rechten Seite von Zuweisungsanweisungen vorkommen.
Wenn man nur die Anweisung

```
y:= betrag(c2) + c1.im
```

sieht, kann man nicht entscheiden, ob es sich bei der Bezeichnung `c1.im` um die Identifikation eines ungekapselten Speichers innerhalb eines Tupels handelt oder um die Identifikation eines gekapselten Speichers `c1`, für den die wertidentifizierende Funktion `im` definiert ist. Man vergleiche hierzu die Anweisungen, wo auf der rechten Seite `s.top` bzw. `s.height` steht. Auch hier kann man nicht entscheiden, ob es sich hier um Ortsidentifikationen in einem ungekapselten Tupel `s` handelt oder um wertidentifizierende Funktionen, die für den gekapselten Ort `s` definiert sind. Dass es sich bei `s` um einen gekapselten Ort handelt, erkennt man an den Operationen `clear` und `push`, die auf diesem Speicher ausgeführt werden sollen.

Wenn man nur den Stack als Beispiel für einen gekapselten Speicher betrachtet, stößt man nicht auf zustandsverändernde Operationen, die auch wertidentifizierend sind. So etwas kann es aber durchaus geben. Als Beispiel betrachte man einen gekapselten Speicher für den Zustand eines sogenannten Zufallszahlengenerators. Bezüglich eines solchen gekapselten Speichers sind üblicherweise zwei Operationen definiert: Die Operation `reset` identifiziert keinen Wert, sondern dient nur dazu, den Zufallszahlengenerator in die Grundstellung zu bringen. Die Operation `next_random` identifiziert einen Wert, der über eine Funktion aus dem aktuellem Zustand abgeleitet wird, und sie führt außerdem zu einem Zustandsübergang. Dadurch wird erreicht, dass möglicherweise ein anderer Wert identifiziert wird, wenn die Operation `next_random` das nächste Mal ausgeführt wird.

Die Betrachtung gekapselter Speicher war die erste Etappe auf dem Weg zur objektorientierten Programmierung. Damit die Objektorientierung erreicht wird, müssen nun noch die beiden Konzepte Vererbung und Polymorphie hinzukommen. Vererbung und Polymorphie

äußern sich nur in der Formulierung des Erklärungsblockes, d.h. sie betreffen den abwickelbaren Programmteil nicht und sind deshalb für den Steuerakteur nicht sichtbar. Der Begriff „Polymorphie“ wurde bereits im Abschnitt 3.4 eingeführt. Bei der Objektorientierung wird die Polymorphie an die Vererbung gebunden.

In der objektorientierten Programmierung werden die Typen für gekapselte Speicher „Klassen“ genannt. Vererbung ist eine Beziehung zwischen Klassendefinitionen. Es kommt verhältnismäßig oft vor, dass eine neu einzuführende Klasse große Ähnlichkeiten zu anderen Klassen aufweist, die bereits innerhalb des Erklärungsblockes definiert wurden. Der Begriff „Vererbung“ für den Sachverhalt, dass man sich bei einer neuen Klassendefinition auf eine bereits vorhandene bezieht, ist in der Anschauung begründet, dass Kinder häufig bezüglich sehr vieler Eigenschaften ihren Eltern sehr ähnlich sind. Man sagt in solchen Fällen manchmal „ganz der Vater“ bzw. „ganz die Mutter“. Am einfachsten und am leichtesten verständlich sind die Fälle, bei denen in die neue Klasse alle Merkmale der Vaterklasse übernommen werden und sich die neue Klasse nur dadurch von ihrer Vaterklasse unterscheidet, dass noch zusätzliche Merkmale hinzugekommen sind. Die hinzugekommenen Merkmale können von zweierlei Art sein: Zum einen kann der Wertebereich des gekapselten Speichers erweitert worden sein, und zum anderen können zusätzliche Operationen oder Funktionen definiert worden sein.

Weniger einleuchtend und nicht so leicht zu kommunizieren sind die Fälle, wo gewisse Merkmale der Vaterklasse in der Sohnklasse nicht mehr existieren oder dort zwar noch unter der gleichen Bezeichnung vorkommen, aber nicht mehr so interpretiert werden dürfen wie beim Vater.

Es gibt Programmiersprachen, bei denen eine Klassendefinition von mehr als einer Vorgängerdefinition erben kann. Man spricht in diesem Fall von Mehrfachvererbung. Im Falle der Einfachvererbung lassen sich sogenannte Klassenbäume zeichnen. Die Knoten dieser Bäume sind die Klassendefinitionen, und die Kanten in diesen Bäumen drücken die Vererbungsbeziehung aus. Da immer nur von einer Klasse geerbt werden kann, die schon definiert wurde, muss es zwangsläufig eine als erste definierte Klasse geben. Diese ist die Wurzel des Baumes.

In Bild 28 ist als anschauliches Beispiel ein teilweise unvollständiger Klassenbaum dargestellt. In diesem Bild kommen nicht nur konkrete Klassen vor, sondern auch abstrakte. Eine Klasse ist abstrakt, wenn es keine konkreten Exemplare zu dieser Klasse geben kann. Man kann zwar diese Klasse als Wertebereich einem Speicher zuordnen, aber der Speicher wird trotzdem nie einen Inhalt haben, der ein konkretes Element aus genau diesem Wertebereich ist. Alle schattierten Klassen in Bild 28 sind abstrakt. Man betrachte beispielsweise die Wurzel des Baumes. In dieser abstrakten Klassendefinition wird lediglich festgelegt, dass der jeweilige Inhalt eines Speichers der betrachteten Klasse als Information über eine ebenflächige Figur angesehen werden kann, und dass auf dieser Figur zwei wertidentifizierende Funktionen existieren sollen, die den Umfang bzw. die Fläche der Figur identifizieren. Diese beiden Funktionen können an dieser Stelle nur bezüglich ihrer Existenz genannt werden, denn eine Berechnungsvorschrift kann noch nicht gegeben werden, solange die Art der flächigen Figur nicht näher bestimmt ist.

Solche näheren Bestimmungen kommen hinzu, wenn man in dem Baum absteigt. Dabei ist es durchaus möglich, dass abstrakte Klassen neben konkrete Klassen gestellt werden. Im betrachteten Beispiel steht beispielsweise die abstrakte Klasse 3, in der das Längenattribut b

eingeführt wird, neben den konkreten Klassen der gleichseitigen Dreiecke, der Quadrate und der Kreise.

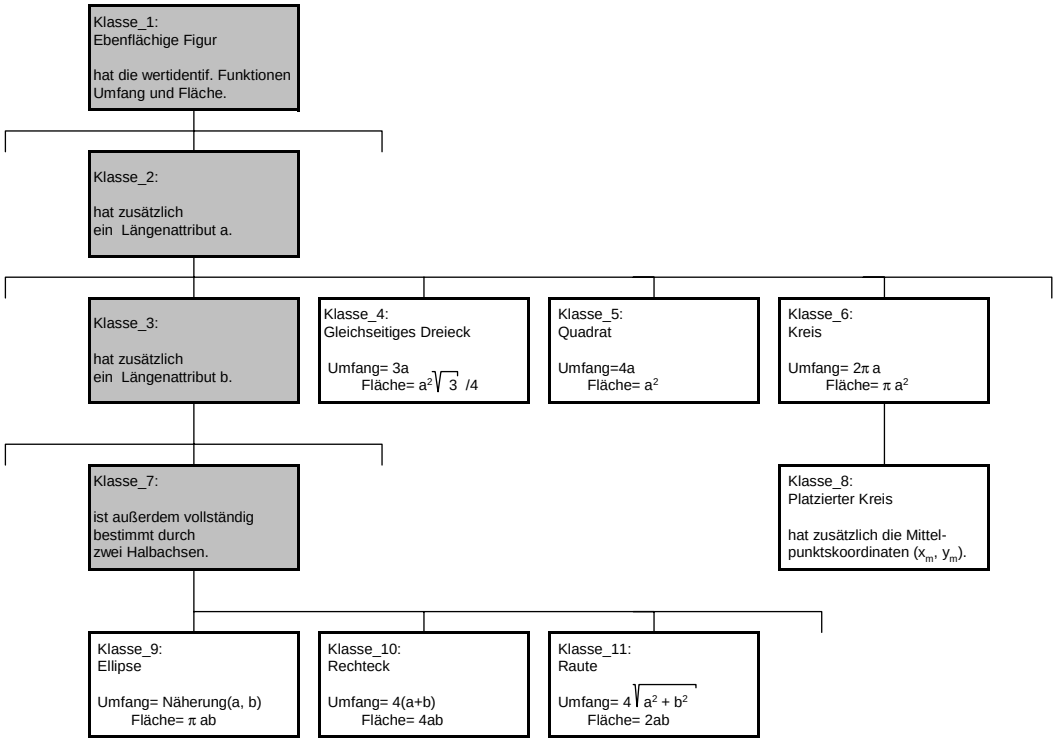

Bild 28

Der im Bild 28 dargestellte Baum zeichnet sich dadurch aus, dass auf dem Vererbungswege von oben nach unten an keiner Stelle irgendwelche weiter oben gemachten Aussagen revidiert werden. Alles, was für die darüberliegenden Klassen gesagt wurde, gilt auch noch weiter unten.

Nun muss noch erklärt werden, was die weiter oben gemachte Aussage bedeutet, dass bei der Objektorientierung das Polymorphieprinzip an die Vererbung gebunden worden sei.

Wenn beispielsweise die in Bild 28 eingeführten konkreten Klassen als Wertebereiche für gekapselte Speicher zur Auswahl stehen, so könnte beispielsweise ein Speicher `s1k` für den Wertebereich „Klasse_6: Kreis“ und ein zweiter Speicher `s2pk` für den Wertebereich „Klasse_8: Platzierter Kreis“ eingeführt werden. Wenn man in den Speicher `s1k` schaut, erwartet man dort einen konkreten Kreis, der selbstverständlich einen Radius hat, aber keine Mittelpunktskoordinaten. Wenn man dagegen in den Speicher `s2pk` schaut, erwartet man dort einen platzierten Kreis, der neben seinem Radius auch noch die beiden Mittelpunktskoordinaten hat. Man erwartet in diesem Speicher `s2pk` keinen Kreis, dem die Mittelpunktskoordinaten fehlen.

Bei objektorientiert programmierten Systemen werden diese Erwartungen zum Teil nicht erfüllt. Denn die Bindung der Polymorphie an die Vererbung besagt: Ein gekapselter Speicher, dem eine bestimmte Klasse als Wertebereich zugeordnet wurde, kann als Behälter nicht nur Exemplare dieser Klasse, sondern auch Exemplare aller im Vererbungsbaum darunterhängenden konkreten Klassen enthalten. Nach dieser Regel muss also der Speicher `s1k`, dem der Wertebereich „Klasse_6: Kreis“ zugeordnet wurde, einen Kreis enthalten können, der keine

Mittelpunktskoordinaten hat, aber alternativ auch einen Kreis mit Mittelpunktskoordinaten. Dagegen wird der Speicher `s2pk`, dem die „Klasse_8: Platzierter Kreis“ zugeordnet wurde, keine Kreise aufnehmen, denen die Mittelpunktskoordinaten fehlen.

Wenn man in einen gekapselten Speicher `s3a` hineinschaut, dem man die unmittelbar unter der Wurzel liegende Klasse_2 zugeordnet hat, wird man aktuell einen Speicherinhalt finden, der ein Exemplar aus einer der sieben in Bild 28 vorkommenden konkreten Klassen ist, denn alle diese konkreten Klassen liegen im Vererbungsbaum unter der abstrakten Klasse_2. Man könnte also beispielsweise in diesem Speicher ein bestimmtes gleichseitiges Dreieck finden. Wenn man später wieder nachschaut, könnte dort ein Rechteck liegen, und noch später möglicherweise ein platzierter Kreis. Damit nacheinander die verschiedenen Figuren in diesen Behälter `s3a` gelangen können, genügen selbstverständlich die in Bild 28 eingeführten Funktionen nicht. Es sind hierzu noch Operationen erforderlich, die in Bild 28 nicht angegeben sind.

Wenn der gekapselte Speicher `s3a` als Behälter für sehr unterschiedliche Figuren eingeführt wird, kann im abwickelbaren Programmteil der Umfang der jeweils aktuell im Behälter liegenden Figur einfach durch die Bezeichnung `s3a.umfang` identifiziert werden. Diese Bezeichnung kann also in einem Prädikatsauftrag oder einem Transitionsauftrag vorkommen. Da der Operationsakteur den Umfang der aktuellen Figur nach einer bestimmten Formel berechnen muss, muss er in der Lage sein, aus den sechs unterschiedlichen Umfangsformeln, die in Bild 28 vorkommen, die aktuell zutreffende auszuwählen. Alle diese Formeln müssen im Erklärungsblock stehen, auf den der Operationsakteur lesenden Zugriff hat. Der Operationsakteur muss in der Lage sein, festzustellen, zu welcher konkreten Klasse im Vererbungsbaum das Exemplar gehört, welches aktuell im gekapselten Speicher `s3a` liegt. Diese Information entscheidet die Auswahl der aktuell heranzuziehenden Berechnungsvorschrift für den Umfang.

Die Tatsache, dass die Fallunterscheidung zur Auswahl der aktuellen Berechnungsformel nicht im abwickelbaren Programmteil steht, so dass vom Steuerakteur keine Auswahl vorgenommen werden muss, wurde zum Anlass genommen, die objektorientierte Programmierung durch den Begriff CASELESS-Programming zu charakterisieren. Die Fallunterscheidung findet zwar immer noch statt, aber sie erfolgt im Operationsakteur und muss deshalb vom Programmierer nicht formuliert werden.

Zum Abschluss dieses Abschnitts wird in Bild 29 eine zusammenfassende Darstellung gezeigt, worin die verschiedenen Speichertypen übersichtlich gegeneinander abgegrenzt sind. In diesem Bild kommt ein Abgrenzungskriterium vor, welches bisher noch nicht behandelt wurde; es handelt sich um die Abgrenzung der „implementierungsverbergenden Speicher“. Ob ein Speicher implementierungsverbergend ist oder nicht, entscheidet sich an der Frage, ob der Abwickler dem Programmierer die Möglichkeit gibt, einen Speicherinhalt nicht nur als Interpretationsergebnis, sondern auch als interpretierbare Form zu behandeln. Die Speicher sollen grundsätzlich Behälter für informationelle Individuen sein, was aber zwangsläufig mit sich bringt, dass sie auch Behälter für interpretierbare Formen sein müssen. Denn das informationelle Individuum und die interpretierbare Form sind ja nur unterschiedliche Sichten auf ein und dieselbe Sache. Die Abbildung der Speicherung und Verknüpfung informationeller Individuen auf ein Formenspiel bezeichnen wir als Implementierung. Ein Speicher wird als implementierungsverbergender Speicher bezeichnet, wenn der Abwickler dem Programmierer keine Möglichkeit gibt, außerhalb der den Wertebereich definierenden Programmteile auf die Implementierung der Speicherinhalte Bezug zu nehmen, d.h. wenn die Sprache, die der Abwickler versteht, eine solche Bezugnahme nicht zulässt.

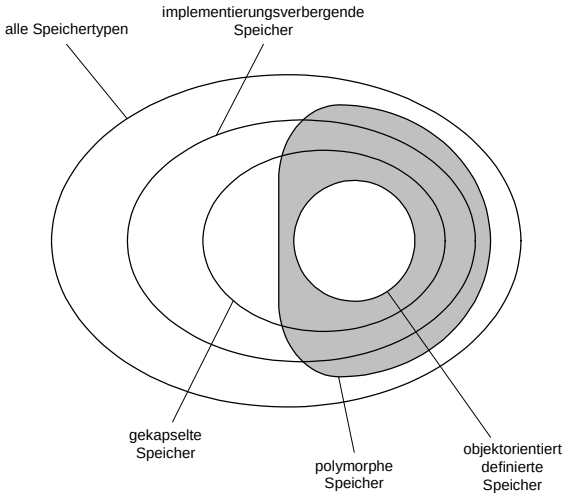

Bild 29

Als anschauliches Beispiel betrachte man einen Speicher für INTEGER-Zahlen. Der jeweilige Speicherinhalt ist ein informationelles Individuum, welches man sich atomar vorstellt. Bezüglich der Verwendung dieser Individuen im Programm benötigt man kein Wissen über die Implementierung. Man braucht nicht zu wissen, wie die ganzen Zahlen im Computersystem dargestellt werden. Man denke an zwei alternative Codierungen, nämlich einerseits die Darstellung im Zweierkomplement und andererseits die Darstellung durch Betrag und Vorzeichen. Es ist selbstverständlich, dass der Abwickler dem Programmierer die Möglichkeit bieten muss zu verlangen, dass die aktuell im Speicher i liegende ganze Zahl mit (–1) multipliziert wird, wodurch sich ihr Vorzeichen umkehrt. Es ist aber nicht selbstverständlich und nicht unbedingt zweckmäßig, dass der Programmierer die Möglichkeit hat, den Vorzeichenwechsel auf der Ebene der binärcodierten Zahlen zu formulieren. Im Falle der Implementierung durch Betrag und Vorzeichen bedeutet die Vorzeichenumkehr nur die Invertierung eines einzigen Bits, wogegen sie im Falle der Zweierkomplementcodierung die Invertierung vieler Binärstellen bedeuten kann.

Man betrachte hierzu die Beispiele in der folgenden Tabelle:

Zahl in Dezimaldarstellung mit Vorzeichen und Betrag	Zahl in Binärdarstellung		Zahl in Binärdarstellung
	Vorzeichen	Betrag $2^3\ 2^2\ 2^1\ 2^0$	Zweierkomplement $-2^4\quad 2^3\ 2^2\ 2^1\ 2^0$
5	0 (+)	0 1 0 1	0 0 1 0 1
-5	1	0 1 0 1	1 1 0 1 1
8	0	1 0 0 0	0 1 0 0 0
-8	1	1 0 0 0	1 1 0 0 0

Bild 30

Man beachte, dass ein implementierungsverbergender Speicher nicht notwendigerweise ein gekapselter Speicher sein muss. Andererseits aber gibt es keine gekapselten Speicher, die nicht auch implementierungsverbergend sind, denn wenn der Programmierer auf die Implementierung der gespeicherten Inhalte Bezug nehmen kann, ist dies unverträglich mit der Bedingung der Kapselung.

4.2.3 Trägersysteme für nichtprozedurale Programmierung

Die nichtprozedurale Programmierung im echten Sinne ist nicht in gleichem Maße universell wie die prozedurale Programmierung. Während ein nichtprozedurales Programm in jedem Falle eine Wertidentifikation zum Ziel hat, gilt diese Einschränkung für prozedurale Programme nicht. Zwar ist jedes prozedurale Programm, das als Formulierung eines Berechnungsalgorithmus entsteht, zusammen mit den vorgegebenen Argumenten eine wertidentifizierende Formel. Aber man kann ja die prozedurale Programmierung nicht nur dazu benutzen, Berechnungsalgorithmen zu formulieren, sondern man kann auch Prozesse spezifizieren, an denen man nicht wegen eines möglicherweise dabei entstehenden Ergebnisses interessiert ist. Der Unterschied zwischen einem ergebnisorientierten und einem prozessorientierten Programm wird durch den Interessenten am Programmablauf festgelegt, der sich entweder für ein am Ende vorliegendes Ergebnis oder aber für das Geschehen während des Ablaufs interessiert. Beispielsweise ist ein Betriebssystem ein Softwaresystem mit Prozessorientierung, denn es erzeugt kein Ergebnis, weil ja sein Ablauf gar kein reguläres Ende hat. Das anschaulichste Beispiel für einen Prozess, der zwar endet, aber durch den kein Ergebnis erzeugt wird, ist ein Konzert. Die Zuhörer sind ausschließlich am Verlauf des Konzertes interessiert, so dass es sinnlos wäre, während des Konzertes wegzugehen, wie man dies bei einem ergebnisorientierten Prozess durchaus tun kann.

Die Aussage, dass das Ziel nichtprozeduraler Programme ausschließlich eine Wertidentifikation ist, darf nicht dahingehend verstanden werden, dass die sogenannten nichtprozeduralen Programmiersprachen nur zur Formulierung wertidentifizierender Formeln geeignet seien. Denn die sogenannten nichtprozeduralen Programmiersprachen sind keineswegs vollkommen nichtprozedural. Vielmehr sind diese Sprachen immer mit bestimmten prozeduralen Anteilen versehen, ohne die sie keinerlei praktischen Nutzen hätten. So könnte beispielsweise die immer erforderliche Kommunikation zwischen dem Rollensystem und seiner Umgebung ohne die Möglichkeit, Operationen zu verlangen, gar nicht formuliert werden. Und alle diese Anweisungen, in denen die Ausführung von Operationen verlangt wird, sind ein Kennzeichen der prozeduralen Programmierung, auch wenn diese Anweisungen in den sogenannten nichtprozeduralen Sprachen syntaktisch in die Form von Funktionalausdrücken oder von Prädikaten gekleidet sind.

Im folgenden werden nur die nichtprozeduralen Aspekte betrachtet. Wenn man sich für die nichtprozedurale Programmierung interessiert, braucht man nur die Frage zu stellen, welche grundsätzlich unterschiedlichen Formen der Wertidentifikation es denn gibt. Wertidentifikation findet ja nicht nur in der nichtprozeduralen Programmierung statt, sondern taucht auch in prozeduralen Programmen an sehr vielen Stellen auf. Grundsätzlich steht auf der rechten Seite eines Zuweisungsoperators immer eine wertidentifizierende Formel. Allerdings darf man in diesen Formeln, die in prozeduralen Programmen rechts vom Zuweisungsoperatoren stehen, auf Speicherzellen Bezug nehmen, und dies muss in der reinen nichtprozeduralen Programmierung ausgeschlossen bleiben. Denn die Belegung solcher Speicherzellen mit Werten erfordert Operationen, und bei der nichtprozeduralen Programmierung dürfen keine Operationen verlangt werden.

Die Aufgaben der Rollensysteme im Bereich der nichtprozeduralen Programmierung lassen sich in zwei Klassen einteilen. Zum einen gibt es einen Bedarf an Rollensystemen, die in der Lage sind, eine vorgegebene funktionale Wertidentifikation in die als Norm festgelegte sogenannte kanonische Form zu überführen. Man spricht in diesem Fall von „funktionaler Programmierung“. Zum anderen gibt es einen Bedarf an Rollensystemen, die als Wissensträ-

ger eingesetzt werden können, denen man Fragen stellen darf, die sie auf der Grundlage ihres Wissens beantworten. Man spricht in diesem Fall von „deklarativer Programmierung“.

Worum es bei der funktionalen Programmierung geht, wird durch die Beispiele in Bild 31 veranschaulicht.

Funktionale Wertidentifikation	Kanonische Form der Wertidentifikation
3+4	7
SQRT(5∗6 - 2∗7)	4
EVALUATION(PRODUKT(f1, f2), (6, 9))	54
EVALUATION(d(„SIN(x)“)/d(„x“), (π/3))	0.5

Bild 31

In jeder Zeile der Tabelle wird ein Wert zweimal identifiziert. In der linken Spalte kann die wertidentifizierende Formel verhältnismäßig kompliziert sein, wogegen die Identifikationen in der rechten Spalte verhältnismäßig einfach sind. Die Identifikationen in der rechten Spalte haben die Form, die der Mensch als „eigentlich gewünschte Identifikationsform“ festgelegt hat.

Bild 32 zeigt den Aufbau des Systems, in dem sich der Kommunikationsprozess zwischen der Umgebung und dem Rollensystem abspielt. In diesem Fall ist das Rollensystem identisch mit dem Rollenakteur, weil es keine internen Rollenaktionsfelder gibt.

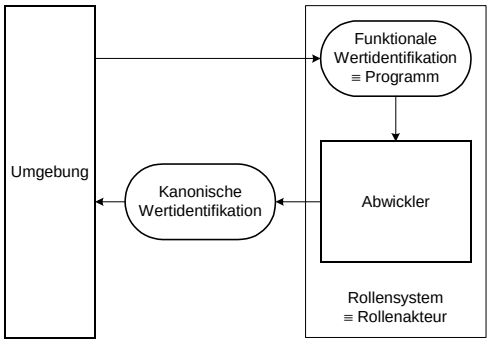

Bild 32

Die Umwandlung der funktionalen Wertidentifikation in die kanonische Identifikation wird durch den Abwickler erledigt. Da man die funktionale Wertumschreibung als Programm betrachtet, wird das Rollensystem zu einem bloßen Konstantenanzeiger, denn das Rollensystem hat ja nur einen einzigen Ausgang, auf dem ein ganz bestimmter, zeitlich unveränderlicher Wert angezeigt wird.

Als Beispiele zur Veranschaulichung wertidentifizierender Formeln wählt man gewohnheitsmäßig numerische Formeln zur Identifikation von Zahlenwerten – wie beispielsweise in Bild 31. Die Numerik ist aber nicht das bevorzugte Anwendungsgebiet der hier betrachteten funktionalen Programmierung. Die funktionale Programmierung erweist ihre Vorzüge insbesondere auf dem Gebiet der Symbolverarbeitung. Hierfür wurde auch die erste funktionale Sprache – LISP – entwickelt, die heute immer noch von großer Bedeutung ist. Als anschauliches Beispiel für anspruchsvolle Symbolverarbeitung stelle man sich die Aufgabe

vor, sämtliche Regeln des formalen Differenzierens in eine Funktion packen zu müssen. Als kanonische Wertidentifikation muss man sich in diesem Fall einen Funktionsausdruck vorstellen, der sich durch Differenziation eines anderen Funktionsausdruckes ergibt. Der zu differenzierende Funktionsausdruck gehört als Argument der Differenziationsfunktion zum Programm. Man beachte, dass in diesem Fall die Fähigkeit zu differenzieren nicht in den Abwickler hineinkonstruiert wurde, sondern dass das Rollensystem dadurch die Fähigkeit zu differenzieren erhielt, dass man die Differenziationsfunktion als Teil des Programms eingespeichert hat.

Nachdem nun die Umwandlung einer Wertidentifikation in die kanonische Form als die eine Art von Aufgaben für Rollenakteure im Bereich der nichtprozeduralen Programmierung betrachtet wurde, wird nun mit Bild 33 die andere Art von Aufgaben für Rollenakteure betrachtet .

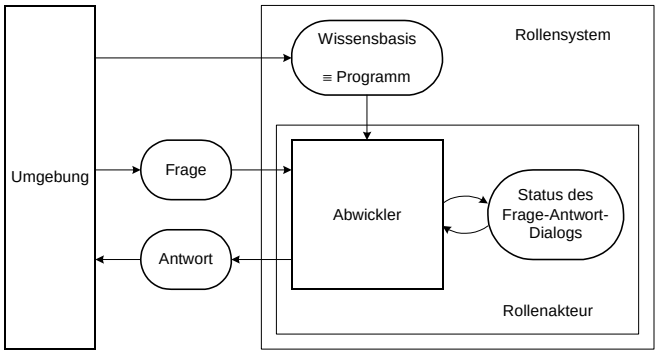

Bild 33

Das Gebiet der Programmierung, welches zum Bild 33 gehört, bezeichnet man als deklarativ und logisch. In diesem Falle bezeichnet man das Programm als Wissensbasis, die man sich als eine Menge von Aussagen vorstellen kann. Durch die Einspeicherung einer Wissensbasis bringt die Umgebung dem Rollensystem etwas bei. Man könnte sagen, dass die Umgebung in diesem Fall das Rollensystem belehrt, weil sie ihm Wissen beibringt. Aus einer Wissensbasis kann nur dann eine Ausgabeinformation gewonnen werden, wenn eine Anfrage gestellt wird. Das Rollensystem ist also in der Rolle eines Studenten, der nach der Belehrung geprüft wird. Die Belehrung besteht im Einspeichern des Wissens, die Prüfung besteht in dem Wechselspiel zwischen Prüfer und Prüfling. Der Prüfer stellt die Fragen, der Prüfling soll die Antworten liefern.

Dabei sollen die Antworten nicht trivial sein, d.h. es soll nicht einfach das, was vorher als Wissensbasis eingespeichert wurde, in Teilen wieder ausgegeben werden. Dies würde der Situation entsprechen, dass der Student die Lehrinhalte auswendig gelernt hat und nun die auswendig gelernten Teile als Reaktion auf die Fragen des Prüfers ausgibt. Man will vielmehr eine darüber hinausgehende Fähigkeit des Rollensystems: Man will, dass das Rollensystem logische Schlüsse ziehen kann, so dass auch solche Antworten gegeben werden können, die nicht original bereits als Teil der Wissensbasis formuliert sind.

Es wird nun ein Beispiel betrachtet, bei dem die Wissensbasis nur aus víer Aussagen besteht.

Wissensbasis:

(1) In der Relation V sind unter anderem die Paare (α, β), (β, γ), (γ, δ) und (δ, ε) enthalten.

(2) Die Relation V ist antireflexiv, d.h. es gilt

$$\forall x: (x, x) \notin V .$$

(3) Die Relation V ist antisymmetrisch, d.h. es gilt

$$\forall (x, y): [(x, y) \in V] \rightarrow [(y, x) \notin V] .$$

(4) Die Relation ist transitiv, d.h. es gilt

$$\forall (x, y, z): ([(x, y) \in V] \text{ UND } [(y, z) \in V]) \rightarrow [(x, z) \in V] .$$

Da diese Wissensbasis nur die Information enthält, wie die Relation V definiert ist, sind selbstverständlich auch nur Anfragen sinnvoll, die sich auf diese Relation V beziehen. Beispiele für Anfragen und die zugehörigen Antworten sind in der folgenden Tabelle zusammengestellt.

Frage	Antwort
$[(\beta, \varepsilon) \in V]$?	ja
$[(\gamma, \alpha) \in V]$?	nein
$[(\beta, x) \in V]$? d.h. welches x bildet hinter β stehend ein Paar in V?	γ
Gibt es noch ein weiteres x, und - falls ja – welches ist es?	δ
Gibt es noch ein weiteres x, und - falls ja – welches ist es?	ε
Gibt es noch ein weiteres x, und - falls ja – welches ist es?	nein

Auf die Frage, welches Element x hinter β stehend ein Paar bildet, das in V vorkommt, gibt es nicht nur eine Antwort. Deshalb benötigt der Rollenakteur einen Speicher, worin er sich den Status des Frage-Antwort-Dialogs merken kann. Dann kann er auf die jeweilige Frage, ob es noch ein weiteres x gibt, jeweils eine neue Antwort geben.

4.3 Implementierung als Beziehung zwischen Trägersystemen

Man halte sich noch einmal die drei unterschiedlichen Programmformen vor Augen, die nun vorgestellt worden sind:

Prozedurale Programmierung:	Die Sprachelemente im Programm sind Befehle, weshalb man auch anstelle von prozeduraler Programmierung von imperativer Programmierung spricht.
Funktionale Programmierung:	Die Elemente im Programm sind funktionale wertidentifizierende Ausdrücke.
Deklarative Programmierung:	Die Elemente im Programm sind Aussagen.

In großen Softwaresystemen kommen diese unterschiedlichen Programmierungsformen oft nebeneinander vor.

Man denke sich ein Softwaresystem, das 50 Millionen Zeilen Quellcode umfasst. Man wird es garantiert nicht schaffen, diese Software als Beschreibung eines einzigen Rollensystems aufzufassen. Ohne einen hierarchischen Zerlegungsansatz wird man nicht zu einer Beherrschung der durch den riesigen Umfang dieses Softwaresystems bedingten Komplexität kommen. Was man anstreben muss, soll durch ein stark vereinfachtes Rechenexempel plausibel gemacht werden. Es sei angenommen, dass jeweils 50 Zeilen Quellcode auf eine DIN A4 Seite passen. Damit benötigt man für das betrachtete Softwaresystem eine Million DIN A4 Seiten. Es sei außerdem angenommen, dass man durch das Studium einer solchen DIN A4 Seite ein Verständnis dieses Softwarestückes gewinnen könne, wenn man zuvor genügend Informationen über die Einbettung dieses Softwarestückes in das gesamte System bekommen hat. Als nächstes wird angenommen, dass jeweils 10 solche Programmstücke inhaltlich derart zusammengehören, dass man darüber ein zusammenfassendes Dokument setzen kann, dessen Studium ungefähr den gleichen Aufwand erfordert wie das Studium einer mit Quellcode beschrifteten DIN A4 Seite. Als letztes sei angenommen, dass jeweils über 10 solchen zusammenfassenden Dokumenten ein übergeordnetes zusammenfassendes Dokument existiert, dessen Studium wieder den Einheitsaufwand erfordert. Wenn man wie im gegebenen Beispiel von einer Million mit Quellcode beschrifteten DIN A 4 Seiten ausgeht und beim hierarchischen Aufstieg pro Schritt jeweils eine Verdichtung auf ein Zehntel erreicht, benötigt man insgesamt 6 Schritte zum Aufstieg bis zur Wurzel des Dokumentenbaumes. Dieser Dokumentenbaum enthält zusätzlich zu der Million Seiten mit Quellcode noch 111.111 übergeordnete Dokumente.

Damit man eine mit Quellcode beschriftete DIN A4 Seite in das Gesamtsystem einordnen kann, muss man mindestens alle Dokumente studiert und verstanden haben, die von der Wurzel des Dokumentenbaumes auf dem Weg bis zum Quellcodeblatt berührt werden. Das sind in dem betrachteten Zahlenbeispiel sechs übergeordnete Dokumente und ein Quellcodedokument. Es könnte jedoch sein, dass dieses Minimalstudium für eine befriedigende Einordnung des Quellcodeblattes in das Gesamtsystem nicht ausreicht. Der Aufwand für ein möglicherweise erforderliches Maximalstudium zum Verständnis einer quellcodebeschrifteten Seite ergibt sich aus der Annahme, dass man beim hierarchischen Aufstieg nicht nur die unmittelbar erreichten übergeordneten Dokumente studiert, sondern auch jeweils deren neun Nachbarn im Baum. Im betrachteten Zahlenbeispiel müsste man dann 51 übergeordnete Do-

kumente und eine Quellcodeseite studieren. Wenn man pro Dokument einen halben Tag Studieraufwand ansetzt, bräuchte man also rd. fünf Wochen, bis man sich das erforderliche Wissen erworben hat.

Diese fünf Wochen sind natürlich nur erforderlich, wenn man ganz frisch an das System herantritt und vorher noch nichts über das System wusste. Die Information, die man durch das Studium der zusammenfassenden Dokumente in Wurzelnähe gewonnen hat, dienen ja nicht nur dem Verständnis einer einzigen Quellcodeseite, sondern behalten ihren Wert auch in Hinblick auf andere Teile des betrachteten Softwaresystems. Grundsätzlich wäre es heutzutage sehr erfreulich, wenn der Einarbeitungsaufwand für einen Fachmann, der frisch zur Firma kommt und an dem System entwickelnd oder pflegend in Zukunft mitwirken soll, nur fünf Wochen betragen würde.

Die Betrachtung des Dokumentenbaumes wurde eingeleitet mit der Aussage, dass es sich um eine Wunschvorstellung handele. Da erhebt sich natürlich sofort die Frage, wie realistisch diese Wunschvorstellung ist. Die Frage nach der Realisierbarkeit dieser Wunschvorstellung ist gleichbedeutend mit der Frage, ob es gelingen kann, das Entstehen der übergeordneten Dokumente zu einer Selbstverständlichkeit werden zu lassen. In den traditionellen Ingenieurdisziplinen, also im Bauwesen, im Maschinenbau und in der Elektrotechnik ist dies nämlich der Fall. In diesen Disziplinen sind die übergeordneten Dokumente technische Pläne, bei denen der hierarchische Aufstieg in einer Verminderung der Auflösung besteht. Der hierarchische Abstieg bedeutet eine Erhöhung der Auflösung. Diese besteht darin, dass man immer näher an die Details herangeht und sich dabei in den Dokumenten immer nur auf Systemausschnitte beschränkt, so dass die technischen Pläne im Grunde auf jeder Ebene ungefähr den gleichen Erfassungsaufwand erfordern.

Es gehört zum Wesen der großen Softwaresysteme, dass die Vorstellung einer ausschließlichen Auflösungserhöhung beim hierarchischen Abstieg nicht angemessen ist. Das bedeutet, dass man nicht im Wurzeldokument eine erste Darstellung des Systems geben kann, aus der man dann durch sukzessive Verfeinerung immer näher an die Details des Systems herankommt, bis man schließlich auf der Quellcodeebene angekommen ist. Der Grund für die Unmöglichkeit des ausschließlich auflösungserhöhenden Abstiegs liegt im Verhältnis zwischen Rollensystem und Trägersystem, wie es im Abschnitt 4.2. behandelt wurde.

Während es in den traditionellen Ingenieurwissenschaften selbstverständlich ist, dass in den Dokumenten auf allen Ebenen des Dokumentenbaumes vom gleichen System die Rede ist, gilt diese Selbstverständlichkeit im Bereich der Softwaresystemtechnik nicht. Das bedeutet, dass zwischen den Dokumenten auf benachbarten Ebenen des Dokumentenbaumes eine Beziehung bestehen kann, die einem Systemwechsel entspricht. Es gibt zwei Möglichkeiten für einen solchen Wechsel des betrachteten Systems. Zum einen kann es sein, dass man auf der oberen Dokumentenebene das Rollensystem betrachtet, während man auf der darunter liegenden Ebene das Trägersystem der Modellierung zugrunde legt. Zum anderen kann es sein, dass man auf der oberen Dokumentenebene ein System betrachtet, dessen Implementierung man auf der darunter liegenden Ebene zeigt. Es ist wichtig zu erkennen, dass ein System und seine Implementierung nicht zwei Sichten auf dasselbe System sind, sondern dass tatsächlich das implementierte System nicht mehr mit dem ursprünglich gewollten System gleichgesetzt werden darf.

Die vorliegenden Überlegungen erfordern eine klare Definition des Begriffes der Implementierung. Das Wort „implementieren“ hat einerseits eine allgemeine umgangssprachliche Bedeutung im Englischen und andererseits eine spezielle Bedeutung in der Softwaresystem-

technik. In den traditionellen Ingenieurwissenschaften ist dieses Wort international nicht gebräuchlich.

Es wird hier nicht versucht, eine weltweit akzeptable Definition des Begriffes „implementieren“ zu geben. Es soll nur versucht werden, den Begriff für den vorliegenden Text verbindlich zu definieren. Implementierung wird hier definiert als der Übergang von einem gewünschten System hin zu einem System größerer Machbarkeit. Das bedeutet nicht, dass dieses System größerer Machbarkeit im endgültigen Sinne machbar sein muss, sondern nur, dass man eine Stufe hin zur endgültigen Machbarkeit hinter sich gebracht hat. Als erstes einfaches Beispiel für eine Implementierung denke man an eine Siedlung mit mehreren Einfamilienhäuschen. Es könnte sein, dass diese Siedlung, so sehr sie auch von den einzelnen Leuten gewünscht wird, nicht machbar ist, weil die Grundstückspreise zu hoch sind oder weil aus städtebaulichen Gründen eine Mindestgebäudehöhe von vier Stockwerken vorgeschrieben ist oder aus sonstigen anderen Gründen. Jedenfalls kann man von dieser gewünschten Einfamilienhaussiedlung zu einem Mietshaus übergehen, welches als machbarer zu betrachten ist. Es braucht weniger Grundstücksfläche, und es erfüllt möglicherweise schon die städtebaulichen Vorschriften. Ursprünglich gewollt war es aber nicht, d.h. es ist aus sachfremden Zwängen entstanden. Das Mietshaus hat zwar noch zu der Einfamilienhaussiedlung gewisse Beziehungen, aber es wäre verfehlt, von einer anderen Sicht auf das gleiche System zu sprechen.

Im Beispiel des Überganges von der Einfamilienhaussiedlung zum Mietshaus ist ein Implementierungsprinzip angewandt worden, welches man auch in der Softwaresystemtechnik immer wieder findet, nämlich das Multiplexprinzip. Man denke beispielsweise an das Prinzip des Timesharings, bei dem der Prozessor nacheinander in sehr kurzen Zeitintervallen semantisch völlig unabhängigen Aufträgen zugeordnet wird. Dieses Timesharing ist sicher nicht ein Thema der ursprünglich gewollten Systeme, sondern ist eine Konsequenz wirtschaftlicher Überlegungen.

Das Multiplexprinzip kennzeichnet eine bestimmte Implementierungsbeziehung zwischen zwei unterschiedlich aufgebauten Systemen. Was im ursprünglich gewollten System noch mehrfach vorhanden ist, wird durch Multiplex auf eine Sache reduziert. Beispielsweise hat in der Einfamilienhaussiedlung jeder Hausbesitzer seine eigene Haustüre, wogegen es im Mietshaus nur noch eine Haustüre gibt, durch die die Bewohner nacheinander gehen müssen. Dagegen ist die Gesamtheit der in der Einfamilienhaussiedlung vorhandenen Schlafzimmer auch im Mietshaus noch vorhanden, denn die Schlafzimmer liegen ja in den einzelnen Wohnungen auf den einzelnen Etagen. Man kann also immer beim Übergang vom ursprünglich gewünschten System zum implementierten System fragen, welche Systemkomponenten unverändert auch im neuen System vorkommen, und welche anderen durch Multiplex reduziert wurden.

Neben der einen Art von Implementierungsbeziehung, die sich durch Multiplex der Systemkomponenten äußert, gibt es noch eine völlig andere Art von Implementierungsbeziehung, die sich nicht in der Begriffswelt der Systeme, sondern in der Begriffswelt der Mathematik abspielt. Bei der Betrachtung informationeller Systeme muss ja nicht nur von Akteuren, Speichern, Kanälen, Prozessen und Zuständen geredet werden, sondern auch von Wertebereichen und Funktionen. Eine Implementierungsbeziehung, die zwei Systeme auf unterschiedlichen Ebenen verbindet, kann sich somit auch auf Wertebereiche und Funktionen, im mathematischen Sinne also auf Mengen und Relationen beziehen. Als anschauliches Beispiel denke man an die Menge ganzer Zahlen, die den Wertebereich INTEGER bilden. Die Elemente dieses Wertebereiches sind atomar, d.h. sie werden als nicht weiter strukturierte

Elemente gedacht. Wenn ein Mensch über ganze Zahlen nachdenkt, interessiert es ihn nicht, welche biochemischen Zustände in seinem Gehirn mit dem Nachdenken über eine bestimmte Zahl verbunden sind. Wenn der Neurophysiologe irgendwelche biochemischen Sachverhalte im Gehirn aufspürt, dann befindet er sich auf einer völlig anderen Systemebene. In seiner Welt spielen ganze Zahlen überhaupt keine Rolle, d.h. in seinen Systemdarstellungen wird über Elemente aus ganz anderen Kategorien geredet. Entsprechendes gibt es bei der Implementierung von INTEGER-Zahlen in technischen Systemen. Sie werden durch Binärwörter dargestellt, wobei der Konstrukteur die Wahl der Codierung nach Zweckmäßigkeiten entscheidet.

Im Unterschied zum Multiplex stellt die Codierung eine Art von Implementierungsbeziehung dar, die auch als Auflösungserhöhung betrachtet werden kann. Denn es wird ja nur etwas, was vorher atomar war, jetzt aus Einzelteilen bestehend erkannt. Dennoch passt auch hierfür die Definition des Begriffes Implementieren als Übergang zu einem System größerer Machbarkeit, denn man kann nun auf der Implementierungsebene über das System in einer Weise reden, bei der die mit den Strukturen verbundenen Interpretationsinhalte gar nicht mehr angesprochen werden. Das bedeutet, dass man auf dieser Ebene das System konstruieren kann, ohne zu wissen, was die einzelnen Binärwörter bedeuten. Denn das technische System, also der Computer, weiß ja auch nichts von Bedeutungen der Binärwörter. Der Übergang zu einem System größerer Machbarkeit bedeutet also in diesem Fall einen Schritt in Richtung auf ein System, für das nur noch die Regeln eines Formenspiels gelten, und erst dieses System ist tatsächlich machbar. Alles andere ist Interpretation, die nur durch den Menschen eingebracht wird.

Es gibt hier allerdings doch einen wesentlichen Unterschied zur Auflösungserhöhung in Systemen der traditionellen Ingenieurwissenschaften. Man betrachte beispielsweise ein Getriebe. Dieses Getriebe wird eingeführt als Komponente zur Wandlung des Produkts aus Drehzahl und Drehmoment. Wenn man die Auflösung der Betrachtung erhöht, d.h. wenn man das Getriebe aufschraubt und hineinschaut, findet man darin die Zahnräder, die Kugellager, die Wellen und alle anderen Komponenten, aus denen das Getriebe besteht. Aber auch diese Komponenten gehorchen den Gesetzen der Physik und wirken zusammen auch nur als Wandler von Drehzahl und Drehmoment. Es ist keinerlei zusätzliche Erklärung eines Menschen erforderlich, wenn man vom geschlossenen Getriebe zum geöffneten Getriebe übergeht. Anders liegt der Fall beim Übergang von atomaren ganzen Zahlen zu Binärwörtern. Denn die einzelnen Binärstellen eines Binärworts gehorchen keinen Regeln, die automatisch dafür sorgen, dass aus einem Binärwort eine ganze Zahl wird. Es ist nur der Mensch, der die Regeln festlegt, nach denen ein Binärwort zu interpretieren ist, so dass man zu der gemeinten ganzen Zahl kommt.

Die Auflösung atomarer Elemente in Strukturen bei der Implementierung bringt es mit sich, dass auch Funktionen, in denen diese ursprünglich atomaren Elemente als Argumente oder Ergebnisse vorkommen, nun nicht mehr als unstrukturiert definierte Zuordnungen gesehen werden können. Vielmehr müssen nun die entsprechenden Abbildungen auf die Strukturen übertragen werden.

Auch die Vorgabe von Algorithmen zur Berechnung von Funktionsergebnissen bei gegebenen Elementen stellt eine Implementierung dar. Denn es wird in diesem Fall einer mathematischen Funktion ein berechnendes System zugeordnet, so dass wieder ein Schritt zu einem System größerer Machbarkeit erfolgt. Das Wort „machbar“ in Bezug auf Funktionen hat die Bedeutung „berechenbar“, und Berechenbarkeit wird durch einen Algorithmus sicher-

gestellt. Deshalb ist die Bereitstellung eines Algorithmus ein Schritt in Richtung auf Machbarkeit.

Es soll nun noch einmal auf das Implementierungsprinzip „Multiplex“ eingegangen werden. Man unterscheidet hier Abwicklermultiplex und Kanalmultiplex. Im Falle des Abwicklermultiplex hat man im ursprünglich gewünschten System einen Bedarf an vielen Abwicklern. Das bedeutet, dass man sich das Rollensystem als Struktur aus vernetzten Akteuren vorstellt, worin jeder dieser Akteure selbst wieder als Rollensystem realisiert sein soll. Dann braucht man in jedem dieser Akteure einen Abwickler. Bei der Implementierung geht man dann zum Abwicklermultiplex über, d.h. man hat nur noch einen Abwickler, der nacheinander die Arbeit der ursprünglich vielen Abwickler erledigen muss.

Beim Kanalmultiplex hat man im ursprünglich gewünschten System mehrere unterschiedliche Kanäle, die man bei der Implementierung auf einen einzigen Kanal abbildet. Ein typisches anschauliches Beispiel hierfür ist die Fensterüberlagerung auf dem Monitorbildschirm. Jedes Kommunikationsfenster realisiert einen Kanal zu einem im Rechensystem gedachten Akteur. Teilweise haben diese Akteure überhaupt nichts miteinander zu tun. Grundsätzlich hätte man für jeden dieser Kanäle einen eigenen Bildschirm vorsehen können. Man hat dies aber aus Wirtschaftlichkeitsgründen nicht getan, sondern man hat alle diese Kanalfenster auf dieselbe Fläche gelegt, so dass sie sich teilweise überdecken.

Die verschiedenen Arten von Implementierungsbeziehungen bilden die Grundlage zu einer angemessenen und realisierbaren Form des Dokumentenbaumes. Es sei an dieser Stelle betont, dass hier nicht die Vorstellung vertreten wird, ein System entstehe dadurch, dass man zuerst das Wurzeldokument erstellt und dann nacheinander im Baum absteigend die verschiedenen Dokumente. Es bedeutet nur, dass man jede Information in Form fassbarer Dokumente festhalten muss und dass die Ablage dieser Dokumente der Baumstruktur genügen muss. Denn nur wenn die Dokumente in einer semantisch begründeten Ablagestruktur liegen, ist eine Navigation durch das Wissen über das System mit angemessenem Aufwand möglich. Es ist unumgänglich, dass bei riesigen Softwaresystemen auch riesige Mengen übergeordneter Dokumente entstehen. Die riesige Anzahl an Dokumenten braucht aber nicht abzuschrecken. Vielmehr ist gerade die semantisch nachvollziehbare Ablage dieser Dokumente die einzige Grundlage dafür, dass sich jeder an der Entwicklung und Pflege des Systems mitwirkende Fachmann mit minimalem Aufwand jeweils die Information beschaffen kann, die er zur optimalen Erledigung seiner Aufgabe benötigt.

5. Zur Wahl von Notationen

Über die Wahl von Notationen kann man sich erst Gedanken machen, nachdem man die Informationen, die man durch die Notationen kommunizierbar machen will, vollständig geklärt hat. Banal gesagt heißt das, dass man sich keine Gedanken über eine Zahlendarstellung zu machen braucht, so lange man den Begriff der natürlichen Zahl nicht denken kann. Gerade dieses Beispiel der natürlichen Zahlen hilft aber auch zu erkennen, dass es bei der Frage nach Notationen nicht einfach um die Auswahl irgendwelcher Symbole zur Erfassung informationeller Individuen geht, sondern dass durchaus die Frage nach strukturellen Darstellungskonzepten gelöst werden muss. Die Römer haben für die Darstellung natürlicher Zahlen ein recht unzweckmäßiges Konzept entwickelt, wogegen die Araber mit dem endlichen Ziffernrepertoire und der stellenabhängigen Potenzgewichtung ein optimales Notationsschema für

Zahlen gefunden haben. Das jeweils darzustellende informationelle Individuum war bei den Römern und den Arabern das gleiche, aber die einen haben eine optimale Notation gefunden, die anderen nicht. Es wird hier nicht versucht, ein strenges Optimalitätskriterium für Notationskonzepte zu finden. Es wird lediglich darauf hingewiesen, dass die Frage nach der Qualität von Notationskonzepten in jedem Falle bewusst behandelt werden muss.

Die Wahl von Symbolen und Notationskonzepten hat einen großen Einfluss auf den Kommunikationswirkungsgrad. Durch ungeschickte Wahl von Notationen kann man die Kommunikation über komplexe Sachverhalte drastisch verschlechtern. Da gerade eine Erhöhung des Kommunikationswirkungsgrades eine Grundvoraussetzung für die Komplexitätsbeherrschung im Bereich der Softwaresystemtechnik ist, müssen alle Anstrengungen unternommen werden, die Notationen zu optimieren. Dazu gehört zuallererst einmal das Bemühen, die aktuell benutzten Notationen auf ihre Qualität hin zu überprüfen. Nur wenn man mit einer Notationsform unzufrieden ist, wird man die Energie aufbringen, nach etwas besserem zu suchen.

Es gibt drei unterschiedliche Bereiche, in denen man Notationsentscheidungen zu fällen hat:

- Wahl von Symbolen zur Verwendung in Symbolfolgesprachen;
- Wahl von Knoten und Kantensymbolen für die Verwendung in zweidimensionalen Graphen;
- Festlegungen von Layoutregeln zur Erfassung von Relationen in zweidimensionalen Graphen.

Zur Beurteilung von Notationen muss man insbesondere die folgenden Kriterien heranziehen:

- Minimierung des Lernaufwandes;
- Minimierung der Missverständlichkeit oder Verwechselbarkeit;
- Maximierung der Interpretationssicherheit.

Es geht im allgemeinen nicht darum, einen bestimmten Notationsvorschlag absolut bezüglich dieser Kriterien zu beurteilen. Vielmehr geht es meistens darum, alternative Notationsvorschläge gegeneinander abzuwägen. Die relative Beurteilung ist sehr viel einfacher als die absolute.

Die angegebenen drei Qualitätskriterien sind nicht unabhängig von einander, dennoch ist es hilfreich, sie nebeneinander zu setzen. Der Lernaufwand wird mitbestimmt durch die Frage nach der bereits vorliegenden Alltagsvertrautheit, nach der Gestaltwahrnehmungsfähigkeit des Menschen und nach der möglicherweise gegebenen oder leicht zu erreichenden globalen Verbreitung. Als Grundregel zur Vermeidung von Missverständlichkeit oder Verwechselbarkeit gilt, dass die symbolische Distanz möglichst die semantische Distanz widerspiegeln sollte. So haben beispielsweise die beiden Wörter *tot* und *lebendig* sowohl eine große symbolische als auch eine große semantische Distanz. Dies garantiert, dass man nicht durch einen einfachen Tippfehler das eine Wort in das andere überführen kann. Bei der Gestaltung von Programmiersprachen wurde dieses Prinzip in der Vergangenheit oft missachtet. Man denke hier insbesondere an die Ähnlichkeit der drei Symbole `=` , `:=` , und `==` , die zwar eine sehr große semantische, aber nur eine sehr kleine symbolische Distanz haben.

Mit der Interpretationssicherheit ist meist auch die Interpretationsgeschwindigkeit verbunden. Wenn man lange braucht, bevor man einer Darstellung die enthaltene Information entnehmen kann, ist meistens auch die Interpretationssicherheit gering, weil man in diesem Fall das Gefühl haben kann, etwas übersehen zu haben. Man betrachte hierzu die zwei alternativen

Fallunterscheidungen in Bild 34. In der linken Darstellung stehen die drei unterschiedlichen Fälle gleichgewichtig nebeneinander, so dass man gar keine Mühe hat, dies als disjunkte Aufteilung in drei Fälle zu erkennen. In der rechten Darstellung, wo die Abfrage in eine Sequenz von zwei Binärabfragen überführt wurde, muss man länger hinsehen, bevor man die Äquivalenz mit der linken Dreierfallunterscheidung erkennt.

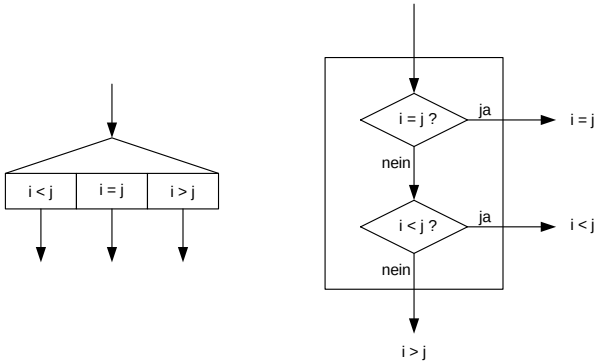

Bild 34

Bei der Wahl von Bezeichnern für Individuen muss man darauf achten, dass die Einordnungen in elementare Kategorien nicht verletzt werden. Damit ist gemeint, dass man Hinweise auf Aktivität oder Passivität nicht vermischen darf, oder dass man die Unterscheidung von Individuen und Eigenschaften nicht im Unklaren lassen darf. So sollte man beispielsweise weder das Wort „Buchung“ noch das Wort „Rechnung“ zur Benennung eines Akteurs verwenden, und man sollte auch nicht behaupten, im System gebe es einen Speicher für „Prozesse“.

Im ersten Ansatz hält man es leicht für irrelevant, welche Knotenformen man für die Darstellung von Individuen in Strukturen wählt. Dabei vergisst man, dass man gewohnheitsmäßig mit bestimmten Formen bestimmte Inhalte verbindet. So assoziiert man gewohnheitsmäßig zu einem Rundknoten nicht die Vorstellung einer aktiven Komponente. Rundknoten in Form von Kreisen oder Ellipsen wurden schon sehr früh in der Mathematik als Symbole für Behälter verwendet, und in diesen Behältern stellt man sich die darin befindlichen Elemente mathematischer Mengen vor. Der Techniker verbindet mit Funktionseinheiten gewohnheitsmäßig die Vorstellung von Rechtecken oder Quadern. So ist beispielsweise die Darstellung eines Vierpols in der Elektrotechnik rechteckförmig. Geräte der Elektronik befinden sich üblicherweise in quaderförmigen Gehäusen. Gedruckte Schaltungen befinden sich üblicherweise auf rechteckförmigen Platinen. Dagegen sind Behälter häufig rund, beispielsweise ein Eimer, ein Bottich oder eine Flasche. Im Grunde sind es Kleinigkeiten, die hier eine große Bedeutung bekommen. Rein formal gesehen könnte es ja gleichgültig sein, ob man einen Behälter durch einen Rundknoten oder durch einen Rechteckknoten symbolisiert. Wenn man aber feststellt, dass sich bipartite Graphen sehr gut zur Darstellung unterschiedlicher Strukturen eignen, wie dies im Abschnitt 3 gezeigt ist, dann bekommt die Frage nach der Wahl der Knotenform doch eine besondere Bedeutung.

Die Wahl des Konzeptes des bipartiten Graphen als übergreifendes Konzept für unterschiedliche Strukturinhalte stellt keine formale Einengung dar, sondern bringt durchaus bestimmte, den Lernaufwand minimierende Vorteile. Es sei hier an die Bedeutung der die Bipartitheit erhaltenden Vergröberungen erinnert.

Der Graph als technische Zeichnung kann durchaus auch gewisse ästhetische Bedürfnisse des Betrachters befriedigen. Man vergleiche hierzu die beiden alternativen Formen des Strukturgraphen in den Bildern 3 und 4. Dass die Planer von Editoren für graphische Darstellungen

häufig das Bedürfnis nach ästhetischer Gestaltung der Pläne nicht im Blick haben, erkennt man, wenn man die Funktionalität verschiedener graphischer Editoren, die als Produkte auf dem Markt sind, analysiert. Häufig hat der Benutzer keine Möglichkeit, die Knotengröße, den Linienstil, die Strichstärke oder die Schattierung von Flächen seinen ästhetischen Bedürfnissen anzupassen. Man hat den Eindruck, dass die Konstrukteure der Editoren im wesentlichen in den Begriffen der mathematischen Topologie denken. Dies bedeutet, dass sie nur die Möglichkeit schaffen, irgendwo auf der freien Fläche einen Knoten zu platzieren und diesen mit anderen bereits vorher platzierten Knoten zu verbinden. Dadurch entstehen Graphen in einer Darstellung, an die man sich nur noch als „Spaghettigraphen“ erinnert.

Wenn man schon graphische Darstellungen für abstrakte Strukturen erzeugt, dann sollte man die Fähigkeit des Menschen zur Gestaltwahrnehmung und zur Mustererkennung nutzen. Es gibt hier einen direkten Bezug zur Elektrotechnik oder zur Hydraulik, wo die elektrischen Schaltpläne bzw. die Hydraulikpläne auch nicht maßstäbliche Abbilder von Strukturen sind, die man in der Realität vorfindet. In diesen Disziplinen haben sich etliche Selbstverständlichkeiten herausgebildet, die jedem Fachmann geläufig sind und die dazu führen, dass man beim Betrachten solcher Pläne auf den ersten Blick typische Muster wiederfindet. Als klassisches Beispiel hierfür sei die Darstellung von Verstärkerschaltungen unter Verwendung eines Transistors erwähnt. Obwohl es nirgendwo offiziell genormt wurde, zeichnet hier jeder Entwickler elektronischer Schaltungen diese Verstärker in der gleichen Weise, wodurch sich der Kommunikationswirkungsgrad beträchtlich erhöht.

In einer Betrachtung über Notationen muss auch auf den Unterschied zwischen Primärnotationen und Sekundärnotationen hingewiesen werden. Primärnotationen zeichnen sich dadurch aus, dass sie besonders geeignet sind zur gleichzeitigen Einführung der zu notierenden Inhalte und der Form der Notationen. Am Beispiel veranschaulicht heißt dies, dass man gleichzeitig mit dem Begriff der Ablaufstruktur auch eine Notation für Ablaufstrukturen einführen muss, weil man sonst über den Begriff der Ablaufstruktur gar nicht kommunizieren kann. Die hierfür gewählte Notation sollte also den didaktischen Anforderungen für die Lehre genügen. Später, nachdem man die zu notierenden Inhalte, im Beispiel also den Begriff der Ablaufstruktur bereits verstanden hat, kann man sehr wohl zu Sekundärdarstellungen übergehen, wie sie sich möglicherweise in der Praxis herausgebildet haben. Am Beispiel konkretisiert heißt dies, dass man den Begriff der Ablaufstruktur nicht unter Verwendung der Nassi-Shneiderman-Notation einführen sollte, sondern unter Verwendung der Petrinetze, weil dort die Begriffe der Fallunterscheidung und der Wiederholungsschleife in Form sichtbarer Ablaufwege dargestellt sind (siehe Bild 35).

Außerdem stehen das Petrinetz und der Begriff der Ablaufstruktur in einer 1-zu-1-Beziehung zueinander, wogegen man mit Nassi-Shneiderman-Diagrammen nur sogenannte wohlstrukturierte Ablaufstrukturen darstellen kann. Erst nachdem man einmal den Begriff der Ablaufstruktur und seine kanonische Darstellungsform kennengelernt hat, kann man es sich erlauben, viele andere in der Praxis übliche Darstellungsformen für Ablaufstrukturen zu verwenden, weil man ja immer die Abbildung auf die Referenzdarstellung vor Augen haben kann.

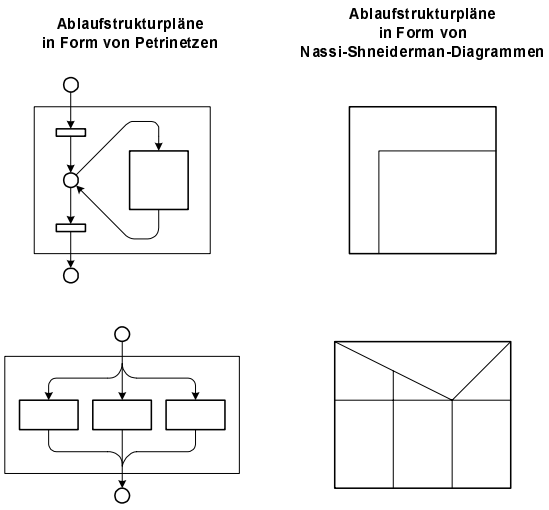

Bild 35

Zum Schluss sei noch auf ein Problem hingewiesen, welches sich durch die Tatsache ergibt, dass im Bereich der Softwaresystemtechnik die englische Sprache inzwischen zur internationalen Fachsprache geworden ist. Es müssen die in diesem Aufsatz vorstellten Begriffe mit englischen Bezeichnern ausgedrückt werden. Es geht also insbesondere um die Frage, wie man die Begriffe

- Aufbaustruktur
- Ablaufstruktur
- Wertebereichsstruktur
- Akteur
- Aktionsfeld
- Abwickler
- Trägersystem

ins Englische übersetzen soll. In Zusammenarbeit mit englischsprachigen Muttersprachlern, also mit sogenannten „native speakers“, soll in der nächsten Zeit versucht werden, die hier mit deutschen Wörtern vorgestellte Begriffswelt in englischer Sprache bereitzustellen.