Andreas Winter

Referenz-Metaschema für visuelle Modellierungssprachen

Vom Promotionsausschuß des Fachbreichs 4: Informatik der Universität Koblenz-Landau zur Verleihung des akademischen Grades Doktor der Naturwissenschaften (Dr. rer. nat.) genehmigte Dissertation.

Vorsitzender des Promotionsausschusses: Prof. Dr. Jürgen Ebert

Promotionskommission

Vorsitzender:	Prof. Dr. Ulrich Frank
Berichterstatter:	Prof. Dr. Jürgen Ebert
	Prof. Dr. Franz Lehner

Tag der wissenschaftlichen Aussprache: 14. Februar 2000

Geleitwort

Sowohl bei der Beschreibung von Unternehmen, Verwaltungen und Organisationen als auch bei der Beschreibung von Softwaresystemen werden zahlreiche – häufig visuelle – Sprachen eingesetzt. Die Sprachenvielfalt ist dabei sowohl in der Organisationslehre als auch in der Softwaretechnik sehr hoch.

Die vorliegende Dissertation bringt in die Menge der Sprachen zur Beschreibung von Organisationen und von Softwaresystemen ein umfassendes und leistungsfähiges konzeptuelles Ordnungssystem, das geeignet ist, diese Sprachen zusammen zu erfassen und miteinander (syntaktisch) in Beziehung zu setzen. Damit wird insbesondere die Grundlage für den Bau generischer sprachspezifischer und sprachübergreifender Werkzeuge weiter abgesichert.

Herr Winter vereint die visuellen Sprachen beider Bereiche in einer ganzheitlichen Sicht und zeigt, daß sie alle zusammen in systematischer Form beschrieben und erfaßt werden können:

Aufbauend auf einer stark literatur-basierten Diskussion der Grundbegriffe „Organisation“ und „Softwaresystem“ und den jeweiligen methodischen Ansätzen für deren Analyse und Gestaltung zeigt er, daß Organisationstechnik und Softwaretechnik gegenseitig eng miteinander verflochten sind, so daß im Kontext der vorliegenden Arbeit nur eine gemeinsame, integrierte Betrachtung in Frage kommt.

Um die Beschreibungsparadigmen umfassend und integriert darstellen zu können, wird dann im Verlauf der Arbeit ein gemeinsames *Referenz-Metaschema* erstellt, das alle behandelten Sprachen im wesentlichen als konkret-syntaktisch notierte Instanzen hierzu umfaßt.

Der Autor validiert dieses Referenz-Metaschema dadurch, daß er existierende multiparadigmatische Beschreibungsansätze als Spezialfälle des integrierten Referenz-Metaschemas beschreibt. Wie durch den Begriff „Referenz“ nahegelegt, sind hierzu nur kleinere Anpassungen erforderlich.

Die vorliegende Dissertation ist ein wichtiger Beitrag zu einem wesentlichen Gebiet der Softwaretechnik, nämlich der Definition und Formalisierung von visuellen Beschreibungssprachen. Mit dieser Arbeit sind diejenigen Sprachen, die sich im praktischen Einsatz befinden, inklusive der notwendigen Kontextbedingungen erfaßt.

Durch die Arbeit wird der Stand der Technik in der Organisationstheorie *und* in der Softwaretechnik vollständig berücksichtigt. Dabei ist zu beachten, daß nicht die Konzepte in der wissenschaftlichen Behandlung von Organisationen und Softwaresystemen (also in der Anwendungsdomäne der Sprachen) erfaßt werden, sondern die Konzepte, die in den *Sprachen* zu deren Beschreibung verwendet werden. Da softwaretechnische Methoden stets um Beschreibungssprachen herum

entwickelt werden, ist somit für das Method Engineering eine Basis geschaffen, die auch den semantischen und den transformativen Teil von Methoden formalisieren läßt.

Das eingeführte Referenzschema ist insgesamt als ein konsistenter und hinreichend vollständiger Vorschlag zu sehen, der sich in der Diskussion in der Wissenschaft, vor allem aber bei der Implementierung in CASE-Werkzeugen noch weiter bewähren muß. Die Definition von UML durch die OMG über ein schwach untermauertes Metamodell ist dagegen ein eher blasser Ansatz. Der Autor hat gezeigt, daß sich sein Referenz-Metaschema mit den eingeführten Mitteln vollständig beschreiben läßt und daß Variantenbildung – wie die zahllosen Beispiele zeigen – ausgesprochen einfach ist.

Durch die Veröffentlichung dieser Dissertation durch den Deutschen Universitäts-Verlag kann die Arbeit die Verbreitung erlangen, die sie verdient und die sie benötigt, um die Diskussion über visuelle Sprachen in der Softwaretechnik und deren Präzisierung voranzubringen.

Jürgen Ebert

Vorwort

Diese Dissertation entstand während meiner Anstellung als wissenschaftlicher Mitarbeiter in der Forschungsgruppe Softwaretechnik der Universität Koblenz-Landau bei Herrn Prof. Dr. Jürgen Ebert. Die Arbeit wurde teilweise durch ein Graduiertenstipendium des Landes Rheinland-Pfalz unterstützt. Ich möchte mich an dieser Stelle bei allen bedanken, die zur Erstellung dieser Arbeit beigetragen haben.

Meinem Mentor, Herrn Prof. Dr. Jürgen Ebert, danke ich für das mir entgegengebrachte Vertrauen, das diese Arbeit erst ermöglichte. Er hat mir das weite Feld der Softwaretechnik eröffnet und mich in die wissenschaftliche Beschäftigung mit der Softwaretechnik eingeführt. In seiner Arbeitsgruppe fand ich die Freiräume und die konstruktive und diskussionsfreudige Atmosphäre, ohne die praxisnahe, wissenschaftliche Arbeit nicht möglich ist.

Herrn Prof. Dr. Franz Lehner danke ich für sein Interesse an dieser Arbeit und die Bereitschaft, die Berichterstattung zu übernehmen. Intensive Diskussionen mit ihm erlaubten es mir, die Softwaretechnik auch aus Sicht der Wirtschaftsinformatik zu sehen. Für seine Diskussionsbereitschaft und seine wertvollen Kommentare und Bemerkungen, die diese Arbeit deutlich beeinflußt haben, bedanke ich mich sehr herzlich.

Herrn Prof. Dr. Heino Kaack, der die Fertigstellung dieser Dissertation leider nicht mehr erleben konnte, danke ich für seine Unterstützung bei den ersten Schritten zu dieser Arbeit. Er brachte mich den Fragestellungen der Verwaltungsinformatik und der Organisationsmodellierung näher. In seiner Forschungsstelle für Verwaltungsinformatik ermöglichte er mir meine ersten wissenschaftlichen Gehversuche. Gemeinsam mit Herrn Prof. Dr. Jürgen Ebert und Herrn Dr. Andreas Engel brachte er mich auf die Idee zu dieser Arbeit und ermutigte mich zur Realisierung dieser Dissertation.

Meinen Kollegen Bernt Kullbach und Roger Süttenbach schulde ich Dank für das Lesen und Kommentieren früherer Versionen dieser Arbeit. Trotz ihrer eigenen zeitlichen Belastung, nahmen sie sich stets die Zeit für intensive und wertvolle Diskussionen zu meiner Arbeit. Diese Diskussionen mit ihnen möchte ich nicht missen. Bei Bernt Kullbach bedanke ich mich darüber hinaus auch sehr herzlich, daß er mich in den letzten Monaten in *GUPRO* deutlich entlastete.

Der in dieser Arbeit verwendete *EER/GRAL*-Ansatz zur Konzeptmodellierung basiert auf gemeinsamen Arbeiten mit Martin Carstensen, Peter Dahm, Prof. Dr. Jürgen Ebert, Angelika Franzke und Manfred Kamp. Ihnen danke ich für viele Gespräche zur Metamodellierung und zur Formalisierung von *EER/GRAL*.

Intensive Diskussionen zu Teilbereichen meiner Arbeit konnte ich mit Dr. Andreas Engel, Prof. Dr. Ulrich Frank, Christel Heil, Rudolf Kruse, Prof. Dr. Albrecht Meißner, Dr. Stephan Philip-

pi, Michael Prasse, Ulrich Remus, Martin Schulze, Thomas Schumm, Carlo Simon, Christopher Thomann, Ingar Uhe und Prof. Dr. Alfred Winter und der GI/GMDS Arbeitsgruppe „Methoden und Werkzeuge für das Management von Krankenhausinformationssystemen“ führen. Ihnen danke ich für viele hilfreiche Kommentare, die die vorliegende Dissertation deutlich geprägt haben.

Für die Hilfe beim Umgang mit LaTeX, Linux, Tgif und diversen Druckern danke ich Dr. Marcel Bresink, Detlev Droege, Rainer Krienke, Christoph Litauer und Friedbert Widmann.

Schließlich danke ich meinen Eltern für ihre vielfältige Unterstützung und dafür, daß sie bereits in frühen Jahren für meine Ausbildung gesorgt haben. Ohne sie wäre vieles nicht möglich gewesen.

Andreas Winter

Kurzfassung

Die Modellierung von Organisationen und Softwaresystemen erfolgt heute multiperspektivisch aus unterschiedlichen Darstellungssichten. Hierzu wird sowohl in der Organisationstechnik als auch in der Softwaretechnik eine große Vielfalt unterschiedlicher visueller Sprachen verwendet. Diese Arbeit integriert diese Beschreibungsmittel auf konzeptioneller Ebene und identifiziert die Querbezüge der Darstellungen unterschiedlicher Sichten.

Zur Strukturierung des weiten Spektrums unterschiedlicher Modellierungssprachen wird ausgehend von den zentralen Modellierungssichten ein *Klassifikationsschema* entwickelt, durch das die verschiedenen Beschreibungsmittel auf zehn grundlegende *Beschreibungsparadigmen* zurückgeführt werden können.

Die durch Sprachen dieser Paradigmen modellierten Konzepte und deren Beziehungen werden durch *Referenz-Metaschemata* formalisiert. Diese Referenz-Metaschemata sind einerseits so allgemein gehalten, daß die Ableitung spezialisierter Metaschemata konkreter Modellierungsmittel leicht möglich ist. Andererseits sind sie auch so konkret, daß diese Spezialisierungen nur geringfügige Anpassungen der Referenz-Metaschemata erfordern. Das *Referenz-Metaschema für visuelle Modellierungssprachen* faßt diese Referenz-Metaschemata der Beschreibungsmittel für Organisationen und Softwaresysteme in einem integrierten Referenz-Metaschema zusammen.

Das Referenz-Metaschema für visuelle Modellierungssprachen bietet einen umfassenden Überblick über die heute zur Modellierung von Organisationen und Softwaresystemen verwendeten Beschreibungsmittel. Unabhängig von konkreten Notationen werden diese Sprachen entlang ihrer Modellierungskonzepte dargestellt und Querbezüge zwischen den verschiedenen Darstellungsmitteln der unterschiedlichen Sichten und Paradigmen herausgestellt.

Anwendung findet dieses Referenz-Metaschema neben der Festlegung der Modellierungskonzepte und deren Beziehungen u. a. auch als *Modellierungsmittel* zur Entwicklung spezialisierter Metaschemata konkreter, multiperspektivischer Modellierungssprachen, und es kann als *Vergleichsmaßstab* zur Einordnung dieser Sprachen herangezogen werden. Für die *Entwicklung von Modellierungswerkzeugen* definiert das Referenz-Metaschema bzw. die hieraus abgeleiteten Spezialisierungen Repositorystrukturen zur internen Verwaltung von Modelldaten und spezifiziert die Konsistenz der Teilmodelle unterschiedlicher Beschreibungsmittel zueinander.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung und Zielsetzung

Die Gestaltung von *Organisationen* und die Entwicklung von *Softwaresystemen* sind komplexe Aufgaben, die kaum ohne methodische Unterstützung durchgeführt werden können. Sowohl in der Organisationstechnik als auch in der Softwaretechnik existieren Verfahren und Hilfsmittel, die die Modellierung und Analyse von organisatorischen bzw. softwaretechnischen Zusammenhängen unterstützen. Diese Hilfsmittel weisen, sowohl in bezug auf die darzustellenden Beschreibungsinhalte als auch auf die zur Modellierung verwendeten Beschreibungsmittel, Techniken und Methoden, große Ähnlichkeiten auf.

Die *Organisationstechnik* beschäftigt sich mit der Gestaltung und Realisierung von Organisationssystemen. Die Beschreibung von Organisationen bezieht sich sowohl auf die Einbindung der Organisation in das sie umgebende soziale System als auch auf die Modellierung der eigentlichen Organisationsstruktur, der Organisationsabläufe und der durch die Organisation bearbeiteten Daten und Objekte. Hierbei umfaßt die Strukturierung der Organisationen sowohl die Zerlegung der Organisationsaufgaben in Teilaufgaben als auch die Gliederung der Organisationseinheiten, die mit der Erledigung der Aufgaben betraut sind. Neben diesen statischen Organisationsaspekten sind auch die Organisationsprozesse, durch die die zeitlich/logische Reihenfolge der Aufgabenerledigung beschrieben wird, zu erfassen bzw. festzulegen. Diese Prozesse sind ebenfalls mit den von der Organisation benötigten bzw. erzeugten Objekten und Daten in Beziehung zu setzen.

Beschreibungsinhalt der *Softwaretechnik* ist einerseits die Einbettung zu erstellender bzw. weiterzuentwickelnder Softwaresysteme in die sie umgebenden Systeme und andererseits die Modellierung des Softwaresystems selbst. Insbesondere bei der Entwicklung und Gestaltung von Informationssystemen ist hierbei auch die Einbindung des Softwaresystems in die umgebende Organisation zu beachten. Hierzu sind die durch das Softwaresystem zu unterstützenden Aufgaben und Prozesse sowie deren Zerlegung darzustellen und die Querbezüge zu den mit der Bearbeitung befaßten Organisationseinheiten zu beschreiben. Die Modellierung des eigentlichen Softwaresystems erfordert neben der Darstellung der Objekt- und Datenstrukturen auch die Beschreibung der Prozesse und des dynamischen Verhaltens der Software.

Sowohl die Modellierung von Organisationen als auch von Softwaresystemen erfordert somit die Betrachtung von *Aufgaben*, *Organisationseinheiten*, *Prozessen* und *Objektstrukturen* sowie deren Querbezüge.

Zur Erfassung und Darstellung dieser Aspekte von Organisationen und Softwaresystemen wurden in der Organisationstechnik und in der Softwaretechnik vielfältige Methoden und Techniken entwickelt. Unter *Techniken* werden Vorgehensweisen zur Erstellung solcher Modellierungen aus einzelnen Perspektiven verstanden. Die Techniken zur Organisations- und Softwaremodellierung betonen i. allg. Perspektiven, die sich auf die zuvor skizzierten Aspekte Aufgaben, Organisati-

onseinheiten, Organisations- bzw. Softwareprozesse und Objekt- und Datenstrukturen beziehen. Neben dem Vorgehen zur Modellierung werden in Techniken auch die zu verwendenden konkreten *Beschreibungsmittel* festgelegt, die durch graphische oder textuelle Symbole und Regeln zur korrekten Verwendung näher beschrieben sind. In *Methoden*, die grundsätzliche Denk- und Vorgehensweisen zur Modellierung widerspiegeln, werden diese Techniken geeignet zusammengefaßt. Durch Methoden wird festgelegt, in welcher Reihenfolge die Techniken zur Erreichung des Modellierungsziels angewandt werden und wie die Teilmodelle der einzelnen Techniken zueinander in Beziehung stehen. Die Anwendung von Beschreibungsmitteln, Techniken und Methoden zur Modellierung von Organisationen und Softwaresystemen wird durch softwarebasierte *Werkzeuge* unterstützt, die sowohl die Modellerstellung als auch die Dokumentation und Analyse der Modellierungsergebnisse ermöglichen.

Methoden und Techniken

Moderne Ansätze zur Organisations- und Softwaremodellierung betrachten Organisationen und Softwaresysteme aus unterschiedlichen *Sichten*. Solche *multiperspektivischen Betrachtungen* heben jeweils einige Aspekte des zu modellierenden Systems hervor und blenden andere, aus der jeweiligen Sicht als weniger wichtig eingestufte Aspekte, aus. Die sichtenorientierte Modellierung ermöglicht die Strukturierung und Zerlegung komplexer Modelle in kleinere, überschaubarere Teilmodelle.

Ausgehend von den Aspekten, die zur Modellierung von Organisationen und Softwaresystemen zu beschreiben sind, kann eine Aufteilung in Sichten auf Aufgaben (*Aufgabensicht*), auf Organisationseinheiten (*Stellensicht*), auf Prozesse (*Prozeßsicht*) und auf Objektstrukturen (*Objektsicht*) abgeleitet werden. Ähnliche Sichteneinteilungen (vgl. auch die Diskussion in Kapitel 3.1), bei denen in Abhängigkeit vom Modellierungsziel einzelne Sichten ausgeklammert werden oder andere Sichten durch weitere Aufgliederung stärker betont werden, finden sich z. B. auch in [Olle et al., 1991], [Gutzwiller, 1994], [Ebert / Engels, 1994], [Scheer, 1992], [Jablonski et al., 1997, S. 98ff] und [Partsch, 1998, S. 44]. Die Modellierung aus jeder dieser Sichten wird durch unterschiedliche Modellierungstechniken und Beschreibungsmittel unterstützt. Einen umfassenden Überblick über den zu modellierenden Sachverhalt bietet aber nur eine *integrierte Beschreibung* solcher sichtenspezifischer Teilmodelle.

Durch die verschiedenen Methoden zur Modellierung von Organisationen und Softwaresystemen wird i. allg. eine dieser Sichten als zentraler Ausgangspunkt der Modellierung festgelegt. Die Teilmodelle der anderen Sichten ergänzen dieses zentrale Modell.

Die *Methoden der Organisationstechnik* können grob in funktionsorientierte und prozeßorientierte Denkweisen zur Organisationsmodellierung unterschieden werden. Während in der (klassischen) *funktionsorientierten Organisationsmodellierung*, die den Prinzipien der Arbeitsteilung folgt (vgl. z. B. [Kosiol, 1976], [Nordsieck, 1972], [Smith, 1990]), die hierarchische Zerlegung der Aufgaben den Ausgangspunkt zur Festlegung der Organisationsstruktur und der Unternehmensprozesse bildet, geht die *prozeßorientierte Organisationsmodellierung* (vgl. z. B. [Gaitanides, 1983], [Scheer, 1992], [Gaitanides et al., 1994a], [Eversheim, 1996], [Hammer / Champy, 1996, 52ff], [Ferstl / Sinz, 1996]) von der Darstellung der Unternehmensprozesse aus, die die Grundlage zur Schaffung funktional zusammenhängender Organisationsstrukturen bildet.

Die grundlegenden *Modellierungsmethoden der Softwaretechnik* können in strukturierte und objektorientierte Denkweisen unterschieden werden. Bei der Modellierung nach der *strukturierten Analyse* (vgl. z. B. [DeMarco, 1978], [Gane / Sarson, 1979], [Ward / Mellor, 1985], [Yourdon, 1989]) dominiert die prozeßorientierte Untersuchung des Softwaresystems durch die Modellierung der Aufgaben und Datenflußabhängigkeiten. Diese Modellierungen werden um Darstellungen der Informationsstrukturen und der Systemdynamik ergänzt. *Objektorientierte Methoden* (vgl. z. B. [Rumbaugh et al., 1991], [Jacobson et al., 1993], [Booch, 1994]) gehen von einer Systembetrachtung aus Sicht der Objekte aus, die sowohl durch ihre Datenstrukturen (Zustände) als auch durch ihr Verhalten (Dynamik) charakterisiert sind.

Neuere Ansätze zur Organisationsmodellierung (vgl. z. B. [Jacobson et al., 1994], [Balzert, 1998a, S. 721ff], [Rumbaugh et al., 1999]) kombinieren objektorientierte Methoden mit prozeßorientierten Methoden und führen Organisations- und Softwaremodellierung zusammen. Ausgehend von der Einbettung der Organisation in das sie umgebende Umfeld erfolgt eine anwendungsfallbezogene Beschreibung der Organisationsprozesse. Hier dominiert ebenfalls die Modellierung aus Prozeßsicht, die mit einem objektorientierten Fokus um die Darstellung der restlichen Sichten ergänzt wird.

In diesen grundlegenden Modellierungsmethoden (vgl. hierzu auch Kapitel 2) werden Organisationen und Softwaresysteme i. allg. aus der Aufgabensicht, aus der Stellensicht, aus der Prozeßsicht und aus der Objektsicht betrachtet. Die Methoden unterscheiden sich in der Art und Weise, wie die Betrachtungen der einzelnen Sichten miteinander kombiniert werden. Bezogen auf die funktionsorientierten und prozeßorientierten Methoden zur Organisationsgestaltung bzw. auf die strukturierten oder objektorientierten Methoden zur Softwareentwicklung unterscheiden sich die Modellierungstechniken nur in der Bedeutung, die die einzelnen Sprachmittel innerhalb des Modellierungsprozesses einnehmen.

Visuelle Modellierungssprachen

Die Methoden und Techniken der Organisations- und Softwaremodellierung bedienen sich zur Beschreibung der Modellierungsinhalte visueller Modellierungssprachen. Neben der Unterstützung bei der Erfassung und Dokumentation sind diese graphischen und textuellen Beschreibungsmittel auch wesentliche Hilfsmittel zur Kommunikation bei der (Weiter-) Entwicklung von Organisationen und Softwaresystemen. Zentraler Betrachtungsgegenstand dieser Arbeit sind diese zur Organisations- und Softwaremodellierung verwendeten *visuellen Modellierungsprachen*.

Zur Darstellung von Organisationen und Softwaresystemen aus verschiedenen Sichten wurde sowohl in der Organisationstechnik als auch in der Softwaretechnik eine große Anzahl unterschiedlicher visueller Sprachen entwickelt. Zur Modellierung aus *Aufgabensicht* werden u. a. Funktionsbäume, Aufgabenstrukturbäume, Funktionsstrukturdiagramme und Aufgabengliederungspläne eingesetzt. Die Beschreibung von leitungs- und interaktionsbezogenen Zusammenhängen zwischen einzelnen Organisationseinheiten aus *Stellensicht* erfolgt z. B. durch Abteilungsgliederungspläne, Kommunigramme, Kooperationsbilder, Organigramme, Organisationspläne oder Stellenpläne. Zur Beschreibung aus der *Prozeßsicht* werden neben vielen anderen Beschreibungsmittel auch Aktivitätsdiagramme, Datenflußdiagramme, Prozeßfolgen, Netzpläne, Petri-Netze oder Statecharts verwendet. Objektzusammenhänge werden aus *Objektsicht* z. B.

durch Objektdiagramme, Interaktionsdiagramme, Datenlexika, Entity-Relationshipdiagramme und Klassendiagramme beschrieben. Einen umfassenden Überblick der wichtigsten zur Modellierung eingesetzten Sprachen gibt auch Abbildung 3.3 auf Seite 39.

Diese Beschreibungsmittel teilen sich weiter in verschiedene *Dialekte* und *Varianten* auf. Zur Notation ähnlicher Sachverhalte werden *unterschiedliche graphische oder textuelle Primitive* verwendet. So werden beispielsweise Prozesse in Datenflußdiagrammen oder Aufgaben in Aufgabengliederungsplänen je nach Dialekt durch Ovale, Kreise, Rechtecke u. ä. beschrieben. Verschiedene Dialekte der Klassen- bzw. Entity-Relationshipdiagramme verwenden zur Darstellung der Objektklassen z. B. unterschiedlich untergliederte Rechtecke oder Wolkensymbole. Beziehungsklassen werden durch Linien oder Rautensymbole beschrieben. Einige Modellierungssprachen schreiben auch die *Anordnung der darzustellenden Konzepte* vor. Organisationseinheiten werden z. B. in unterschiedlichen Organigrammdialekten in Blockdarstellung, in Kreisform, in Pyramidenform, in Satellitenform oder in Säulenform notiert. Varianten der Datenflußdiagramme wie z. B. SADT oder IDEF0 fordern eine stufenartig angeordnete Folge von Prozessen, während in klassischen Datenflußdiagrammen keine spezielle Anordnung gefordert wird. Zum Teil verfolgen ähnliche Beschreibungsmittel auch *unterschiedliche Schwerpunktsetzungen* in der Abbildung der dargestellten Konzepte. Prozeßfolgen werden beispielsweise in Ereignisgesteuerten Prozeßketten durch Prozesse und die durch sie bewirkten bzw. ihre Bearbeitung auslösenden Ereignisse beschrieben, während in der Darstellung durch Programmablaufpläne oder Aktivitätsdiagramme solche Ereignisse kaum beachtet werden. Konzeptionelle Unterschiede zwischen Varianten einzelner Beschreibungsmittel wirken sich auch auf die *Ausdrucksfähigkeit* der einzelnen Sprachen aus. So werden z. B. in Statecharts und erweiterten Entity-Relationshipdiagrammen Konzepte zur Strukturierung der Modelle angeboten, während diese in den ursprünglichen Formen der Automaten bzw. (klassischen) Entity-Relationshipdiagrammen fehlen.

In den konkreten Methoden zur Organisations- und Softwaremodellierung werden zur Darstellung aus den verschiedenen Sichten einige Varianten der Beschreibungsmittel fest vereinbart. Als Vertreter der Methoden zur *prozeßorientierten Organisationsmodellierung* können beispielsweise die Methode der Architektur integrierter Informationssysteme (ARIS) [Scheer, 1992] und die Bonapart-Methode [Krallmann / Wood, 1998] genannt werden. Während ARIS die Prozeßdarstellung mittels Ereignisgesteuerter Prozeßketten in den Vordergrund stellt, basiert die Modellierung mit Bonapart eher auf einer datenflußorientierten Prozeßbeschreibung durch Bonapart-Prozeßmodelle. In beiden Methoden werden weiter Organigramme in unterschiedlichen Notationen zur Beschreibung der Organisationsstrukturen und Funktionsbäume bzw. Aufgabenstrukturbäume zur Darstellung von Aufgabenstrukturen verwendet. Die Objektmodellierung erfolgt in ARIS durch klassische Entity-Relationshipdiagramme und Attribut-Zuordnungsdiagramme. Bonapart verwendet eingeschränkte Formen der Klassendiagramme u. a. zur schematischen Beschreibung von Informations- und Organisationsstrukturen.

In den Methoden der *strukturierten Analyse* sind ebenfalls datenflußorientierte Prozeßbeschreibungen zentral. In der ursprünglichen Form der strukturierten Analyse nach [DeMarco, 1978] wurden einfache Datenflußdiagramme und Datenlexika verwendet. Spätere Erweiterungen [Ward / Mellor, 1985], [Yourdon, 1989] ergänzten die Datenflußdiagramme um Mittel zur Beschreibung von Kontrollstrukturen und fügen weitere Modellierungsprachen z. B. Nassi/Shneiderman-Diagramme, Programmablaufpläne und einfache Zustandsübergangsdiagramme zur Beschreibung der Systemdynamik hinzu. Die Mittel zur Datenmodellierung wurden um Entity-Relationshipdiagramme ergänzt.

Wesentliches Beschreibungsmittel der *objektorientierten Methoden*, wie z. B. der Object Modeling Technique (OMT) [Rumbaugh et al., 1991] oder der Booch-Methode [Booch, 1994] sind jeweils verschieden notierte Klassendiagramme. Zur Modellierung funktionaler Prozeßzusammenhänge verwendet OMT auch die aus der strukturierten Analyse entnommenen Datenflußdiagramme. Die Beschreibung der Systemdynamik erfolgt durch Ereignisflußdiagramme und Ereignispfaddiagramme bzw. Interaktions- und Kollaborationsdiagramme. Darüber hinaus werden zur Modellierung von Systemzuständen und deren Änderungen noch jeweils eigene Statechart-Varianten eingesetzt. Mit der Unified Modeling Language (UML) [Booch et al., 1999], [Rumbaugh et al., 1999] werden die verschiedenen Sprachmittel diverser objektorientierter Methoden zusammengeführt und quasi standardisiert. Diese Sammlung, der sowohl in Methoden zur Organisationsmodellierung als auch in Methoden zur Softwaremodellierung einsetzbaren visuellen Sprachen, legt Notationen für Aktivitätsdiagramme, Anwendungsfalldiagramme, Kollaborationsdiagramme, Klassendiagramme, Objektdiagramme, Sequenzdiagramme und Zustandsübergangsdiagramme fest.

Zusammenfassend kann festgestellt werden, daß zur Modellierung von Organisationen und Softwaresystemen sehr viele, unterschiedliche Beschreibungsmittel verwendet werden, die in den diversen Modellierungsmethoden häufig in eigenen Darstellungsdialekten und -varianten eingesetzt werden. Der Vergleich und die Einordnung der zur Modellierung verwendeten Beschreibungsmittel erfordert eine geeignete *Strukturierung des Spektrums der visuellen Modellierungssprachen*. Entlang einer Klassifikation der Beschreibungsmittel können die Gemeinsamkeiten und Unterschiede der einzelnen Sprachen, ihrer Dialekte und Varianten aufgezeigt und vor dem Hintergrund konkreter Modellierungsaufgaben bewertet werden. Ebenso wird hierdurch die Einschätzung der Modellierungsmächtigkeit der verschiedenen Modellierungsmethoden entlang der verwendeten Beschreibungsmittel ermöglicht.

> Ein Ziel dieser Arbeit ist es, eine ***Klassifikation der visuellen Modellierungssprachen***, die zur Darstellung organisatorischer und softwaretechnischer Zusammenhänge verwendet werden, zu schaffen.

Betrachtet man hierzu die verschiedenen Beschreibungsmittel unabhängig von ihrer *konkreten Notation* entlang der jeweils dargestellten Modellierungskonzepte, läßt sich die Vielfalt der Beschreibungsmittel auf *wenige Beschreibungsparadigmen*[1] reduzieren. Den Ausgangspunkt zur Festlegung dieser Paradigmen bilden die *Konzepte* und die hierzwischen vorliegenden *Beziehungen*, die durch die einzelnen Beschreibungsmittel herausgestellt werden.

[1] Der Paradigmenbegriff wird in dieser Arbeit in seiner ursprünglichen Form (Paradigma, griech. παράδειγμα: Beispiel, Vorbild, Muster) und nicht in seiner durch die Wissenschaftstheorie oder durch die Linguistik geprägten Verwendung genutzt. In der Wissenschaftstheorie werden in Paradigmen allgemein anerkannte Methoden, Annahmen und Aussagen einer Wissenschaft [Lehner et al., 1995, S. 23] zusammengefaßt. Paradigmen dienen hier zur Unterscheidung und Abgrenzung wissenschaftlicher Gemeinschaften oder Schulen, denen grundlegende Annahmen, Vorstellungen und Weltbilder gemeinsam sind (vgl. u. a. [Stegmüller, 1973], [Kuhn, 1979]). Paradigmen der Linguistik unterscheiden Wortklassen, die gleichen Deklinations- oder Konjugationsformen folgen [Bußmann, 1990]. Der Paradigmenbegriff, der dieser Arbeit zugrunde liegt, dient zur Klassifikation textueller und graphischer Sprachen. Die hier verwendeten *Beschreibungsparadigmen* dienen zur Gruppierung solcher visueller Sprachen, die einem gemeinsamen *Beschreibungsmuster* folgen.

In Kapitel 3.1 wird ein solches Klassifikationsschema entwickelt, das eine sprachunabhängige Untersuchung und Einführung der Beschreibungsmittel entlang weniger Grundformen ermöglicht und als Rahmen für die weiteren Betrachtungen der visuellen Modellierungssprachen in dieser Arbeit dient.

Werkzeuge

Zur Modellierung und Analyse von Organisationen und der Entwicklung von Softwaresystemen werden viele Softwarewerkzeuge angeboten[2]. Je nach unterstützter Modellierungsmethode stellen diese Werkzeuge u. a. mehr oder weniger integrierte Editoren zur Erfassung und Analyse von Darstellungen gemäß der in der Methode verwendeten Beschreibungstechniken bereit.

Zur *Modellierung von Organisationen und Geschäftsprozessen* werden z. B. das ARIS Toolset 3.0, (IDS Scheer AG, Saarbrücken) [IDS, 1995], [IDS, 1998], [Scheer, 1998] zur Unterstützung der ARIS-Methode und Professional Bonapart 2.3 (PRO UBIS GmbH, Berlin) [Krallmann / Wood, 1998], [Pro Ubis, 1999a], [Pro Ubis, 1999b] zur Unterstützung der Bonapart-Methode angeboten. Beide Werkzeuge erlauben neben der prozeßorientierten Organisationsmodellierung auch die funktionsorientierte Modellierung. Neben den wesentlichen Beschreibungsmitteln der ARIS- und Bonapart-Methoden unterstützen diese Werkzeuge auch weitere Notationsformen. Unter anderem unterstützt das ARIS-Toolset die Verwendung einiger Beschreibungsmittel der Unified Modeling Language [Booch et al., 1999]. Zur Prozeßmodellierung können bei der Modellierung mit Professional Bonapart neben den datenflußartigen Prozeßmodellen auch Ereignisgesteuerte Prozeßketten verwendet werden.

Die Methoden der *strukturierten Analyse* werden z. B. durch die Werkzeuge CASE/4/0 (microTOOL GmbH, Berlin) [Micro Tool, 1997], [Micro Tool, 1998] und Easy CASE Professional 4.22 (Visible Systems Corporation, Boston) unterstützt. Diese Werkzeuge unterstützen diverse Dialekte der Datenflußdiagramme (u. a. [DeMarco, 1978], [Gane / Sarson, 1979], [Ward / Mellor, 1985], [Longworth / Nicholls, 1986] und [Yourdon, 1989]) und der Entity-Relationshipdiagramme (z. B. nach [Chen, 1976], [Martin / McClure, 1985b] und IDEF-IX [Menzel / Mayer, 1998]).

Die Werkzeuge der *objektorientierten Modellierung* stellen heute in unterschiedlicher Vollständigkeit die Beschreibungsmittel der Unified Modeling Language (UML) [Booch et al., 1999] bereit. So unterstützt beispielsweise ObjectiF (microTOOL GmbH, Berlin) die Modellierung mit Anwendungsfalldiagrammen, Klassendiagrammen, Sequenzdiagrammen und Zustandsdiagram-

[2] Einen umfangreichen, aber leider nicht mehr aktuellen Überblick zu Modelliergungswerkzeugen bietet [Balzert, 1993]. Aktuelle (Juni 1999) Überblickslisten zu Modellierungswerkzeugen finden sich z. B. auf `http://www.vtt.fi/tte/staff/ojp/process.htm` (Tools for Process Workflow Modeling and Simulation), auf `http://pwp.starnetinc.com/larryg/process.html` (Process Modeling Tools), auf `http://www.methods-tools.com/tools/modeling.html` (Graphical Modeling Tools), auf `http://www.qucis.queensu.ca/Software-Engineering/case.html` (CASE Tool Information) und auf `http://www.rhein-neckar.de/~cetus/oo_ooa_ood_tools.html` (Architecture and Design: Object-Oriented Analysis & Design Tools).
Einen groben Überblick über objektorientierte Modellierungswerkzeuge bzw. UML-Werkzeuge bieten auch [Balzert / Balzert, 1994], [Versteegen / Versteegen, 1998], [Hunt, 1999] und `http://www.jeckle.de/umltools.htm` (3.1.2000). Weitere Werkzeuge zur Organisations- und Softwaremodellierung werden auch auf den CD-Roms [Balzert, 1996b] und [Balzert, 1998b] vorgestellt.

men. Die Object Engineering Workbench, OEW 3.0.3 (Innovative Software GmbH, Frankfurt) [ISG, 1999] und Rational Rose 98i (Rational Software Corporation, Santa Clara) bieten darüber hinaus auch die Modellierung durch Aktivitätsdiagramme und Kollaborationsdiagramme.

Neben diesen Werkzeugen, die einzelne Modellierungsmethoden in evtl. unterschiedlichen Notationsformen unterstützen, erlaubt das Werkzeug System Architect 2001 (Popkin Software, New York) sowohl die Organisationsmodellierung mit funktions- und prozeßorientierten Methoden als auch die Softwaremodellierung mit strukturierten und objektorientierten Methoden. Aufgrund der Anlage als *Multi-Methoden Werkzeug* wird durch den System Architect 2001 ein Großteil der wesentlichen Beschreibungsmittel und -dialekte unterstützt.

Generische Ansätze zur Erstellung von Modellierungwerkzeugen werden in den *Meta-CASE-Werkzeugen KOGGE* (Institut für Softwaretechnik, Koblenz) [Ebert et al., 1997b], [Kölzer / Uhe, 1997], [Ebert et al., 1999b] und MetaEdit+ (MetaCase Consulting, Jyväskylä) [Kelly et al., 1996] verfolgt. Die Modellierung von Organisationen und Softwaresystemen wird z. B. durch *KOGGE*-Werkzeuge u. a. zur Erstellung von Aufgabengliederungsplänen, Organigrammen, Datenflußdiagrammen, Statecharts und Klassendiagrammen in verschiedenen Notationen unterstützt. Instanzen von MetaEdit+ liegen für die Modellierung sowohl entlang der wesentlichen Methoden der strukturierten und objektorientierten Softwareentwicklung (u. a. durch Datenflußdiagramme, Anwendungsfalldiagramme, Statecharts, Aktivitätsdiagramme und Klassendiagramme) als auch zur Geschäftsprozeßmodellierung (u. a. durch Prozeßmodelle und Prozeßmatrizen) vor.

Weitere Werkzeuge wie beispielsweise Neptun/NetCase (Institut für Softwaretechnik, Koblenz) [Simon et al., 1997], [Marx, 1998] kombinieren Ansätze der objektorientierten Modellierung mit Petri-Netzen zur Modellierung von Workflows. Ebenfalls basieren Werkzeuge zur Projektplanung wie z. B. MS Project (Microsoft Corporation, Redmond) [Staff, 1996], [Microsoft, 1998] auf Beschreibungstechniken zur Prozeßmodellierung (Balkendiagramme und Vorgangsknotennetzpläne), die um geeignete Analysetechniken ergänzt wurden.

Graphische Werkzeuge wie z. B. TGif (William Chia-Wei Cheng) oder Visio (Visio Corporation, Seattle) bieten in der Regel keine Methodenunterstützung. Hier werden lediglich graphische Primitive (Linien, Rechtecke, Kreise, Polygone etc.) oder Symbole aus (erweiterbaren) Symbolbibliotheken angeboten, die „geeignet“ durch den Benutzer methoden- und technikkonform zu kombinieren sind.

Integrierte Referenz-Metaschemata

Die Methodenunterstützung der Modellierungswerkzeuge ermöglicht die Erstellung syntaktisch korrekter Modelle im Sinn der gewählten Methoden und Techniken. Die Erkennung syntaktisch korrekter bzw. fehlerhafter Modelle und die Unterstützung eines konkreten Modellierungsvorgehens setzt eine formale Beschreibung der Modellierungsmethode voraus. Hierzu werden *Metamodelle* verwendet, die die Eigenschaften der zur Modellierung verwendeten Modellierungskonzepte, deren Repräsentationsformen und deren Verwendung spezifizieren. Metamodelle umfassen *Metaaktivtitätsmodelle* zur Beschreibung des Modellierungvorgehens und *Metaschemata* zur Beschreibung der abstrakten Syntax der hierzu verwendeten Beschreibungsmittel (vgl. auch die Diskussion des Metamodellbegriffs in Kapitel 4.3).

Metaschemata zu Modellierungsmethoden und -techniken spezifizieren die syntaktische Korrektheit der Modelle und definieren die Repository-Strukturen der Modellierungswerkzeuge. Sie legen die Modellierungskonzepte der verwendeten visuellen Sprachen und die hierzwischen erlaubten Beziehungen fest.

Sowohl Modelle der Organisationstechnik als auch Modelle der Softwaretechnik bestehen aufgrund der multiperspektivischen Modellierungsansätze aus Teilmodellen mehrerer Sichten, die durch verschiedene Beschreibungsmittel notiert sind. Diese Teilmodelle müssen zueinander konsistent sein. So spiegeln beispielsweise Aufgabengliederungspläne die Verfeinerungsstruktur der Prozesse in Datenflußdiagrammen wider und Entity-Relationshipdiagramme oder Klassendiagramme müssen mit den Daten- und Objektbezügen in Prozeßdarstellung z. B. durch Ereignisgesteuerte Prozeßketten oder durch Datenflußdiagramme verträglich sein.

Die Metaschemata der Modellierungsmethoden beschreiben daher neben der abstrakten Syntax der jeweils verwendeten Beschreibungstechniken auch die Querbezüge zwischen den Konzepten der verschiedenen Modellierungssprachen. Sie definieren ein *integrieres Metaschema* der jeweils in der Methode zusammengefaßten visuellen Modellierungssprachen.

Für die meisten Modellierungsmethoden wurden in den letzten Jahren Metaschemata[3] entwickelt. Metaschemata der strukturierten Analyse wurden z. B. in [Verhoef et al., 1991], [Färberböck et al., 1991] und beschränkt auf Datenflußdiagramme in [Drüke, 1996] vorgestellt. [Süttenbach/Ebert, 1997] beschreiben ein Metamodell der Booch-Methode [Booch, 1994] und [Ebert / Süttenbach, 1997a] eines der OMT-Methode nach [Rumbaugh et al., 1991]. Weitere Metaschemata objektorientierter Methoden wurden auch in [Goor et al., 1992] und [Strahringer, 1996] vorgestellt. Während diese Metaschemata i. allg. nicht von den Autoren der Methoden, sondern deutlich später, nach Vorstellung der jeweiligen Methoden erstellt wurden, werden neuere Modellierungssprachen wie z. B. die Unified Modeling Language (UML) [OMG, 1999] direkt entlang ihrer Metaschemata eingeführt.

Diese Metaschemata beziehen sich jedoch nur auf die konkreten, in den einzelnen Modellierungsmethoden bzw. in konkreten Werkzeugen verwendeten Beschreibungsmittel. Ein weiter gefaßtes, möglichst viele visuelle Modellierungssprachen umfassendes, *integriertes Metaschema* existiert nicht. Umfassende Metaschemata wie beispielsweise das UML-Metaschema [OMG, 1999] betrachteten ausschließlich die Beschreibungsmittel verschiedener Sichten auf Organisationen und Softwaresysteme, integrieren diese Teil-Metaschemta aber nicht. Im Metaschema des ARIS-Ansatzes [Scheer, 1992] findet eine leichte Integration der Metaschemata einzelner Beschreibungsmittel auf rein syntaktischer Ebene statt. Konsistenzbedingungen, die Teilmodelle unterschiedlicher Beschreibungsmittel miteinander synchronisieren, werden jedoch nicht formalisiert.

Ein weitgehend integriertes Metaschema der visuellen Modellierungssprachen bildet eine unabhängige Basis für den *Vergleich* und die *Einordnung* unterschiedlicher Modellierungsmethoden und -werkzeuge. Da es von konkreten Modellierungsmethoden abstrahiert und diverse Beschreibungsmittel konzeptionell integriert, legt es eine *methodenunabhängige Terminologie* der Modellierungsmittel fest und eignet sich auch als *methodenunabhängiges Schulungsmittel*. Ebenfalls beschreibt es eine generelle Repository-Struktur für Modellierungswerkzeuge und könnte daher

[3] Einige Autoren verwenden hierfür auch den allgemeineren Begriff *Metamodell*, betrachten jedoch ausschließlich syntaktische Aspekte der Methoden ohne Berücksichtigung des Modellierungsvorgehens.

sowohl zur *Entwicklung weiterer Modellierungswerkzeuge* als auch zur Definition eines *generellen Austauschformats* zwischen verschiedenen Modellierungswerkzeugen genutzt werden.

Arbeiten im Bereich der Metamodellierung für CASE- und Reengineering-Werkzeuge haben jedoch gezeigt (vgl. z. B. [Ernst, 1997], [Ebert et al., 1999a]), daß es aufgrund der Heterogenität der hier abzubildenden Domänen nicht möglich ist, ein solches, allgemeines Metaschema, das alle bekannten visuellen Modellierungssprachen umfaßt, zu entwickeln. Das auf den ersten Blick recht umfangreiche Metaschema der Unified Modeling Language (UML) [OMG, 1999] bildet trotz Einschränkung auf wenige Beschreibungsmittel auch über die UML-Sprachen hinaus weitere Modellierungsmittel ab. So enthält der Teil des Metaschemas für Aktivitätsdiagramme auch Komponenten zur Abbildung von Datenflußdiagrammen und der Teil zur Darstellung von Klassendiagrammen ermöglicht auch die Repräsentation von Entity-Relationshipdiagrammen und Datenlexika. Die UML, die sowohl zur Modellierung von Organisationen als auch zur Modellierung von Softwaresystemen eingesetzt werden soll, umfaßt aber z. B. keine Notationen der Stellensicht. Mittel zur Beschreibung von Organigrammen oder Stellenplänen sind dementsprechend im UML-Metamodell nicht enthalten. Ebenso sieht die UML keine Beschreibungsmittel zur Darstellung harter Echtzeit-Anforderungen zur Modellierung von Echtzeit-Systemen bereit. Auch zu einem solchen, umfassenden Metaschema sind immer Beschreibungskonzepte denkbar, die *nicht* in ihm enthalten sind.

Anstelle eines allumfassenden Metaschemas kann jedoch ein Referenzmodell entwickelt werden. *Referenzmodelle* (vgl. zur Diskussion des Referenzmodellbegriffs auch Kapitel 4.2) sind ausgewiesene Modelle, die die charakteristischen Eigenschaften einer Modelldomäne allgemeingültig beschreiben. *Spezielle Modelle* für konkrete Modellierungsaufgaben erhält man durch Übernahme und ggfs. Anpassung des Referenzmodells.

Das integrierte Referenz-Metaschema für visuelle Modellierungssprachen umfaßt die wesentlichen Konzepte der zur Modellierung von Organisationen und Softwaresystemen verwendeten Beschreibungsmittel und deren Zusammenhänge. Dieses Referenz-Metaschema ist so anzulegen, daß es mit möglichst geringem Aufwand an die Anforderungen spezieller Metaschemata angepaßt werden kann. Ein solches Referenz-Metaschema wird in Kapitel 6 definiert und in Teil III durch Anwendung zur Methodenbeschreibung und zum Werkzeugbau validiert.

Ziel dieser Arbeit ist die Entwicklung

- eines ***integrierten Metaschemas*** der visuellen Modellierungssprachen für Organisationen und Softwaresysteme, das die konzeptionellen Grundlagen und Zusammenhänge dieser Beschreibungsmittel formalisiert, und
- als ***Referenz*** u. a. zur Auswahl und Einordnung von visuellen Modellierungsprachen in Modellierungsmethoden und zur Entwicklung von Modellierungswerkzeugen dient.

Übersicht über die nachfolgenden Kapitel

Die Entwicklung des Referenzmetaschemas für visuelle Modellierungssprachen ist in drei größere Teile gegliedert.

Teil I schafft die Grundlagen zur Herleitung des Referenz-Metaschemas. In Kapitel 2 werden hierzu die Gemeinsamkeiten von Organisations- und Softwaremodellierungen in bezug auf Vorgehensweise und Beschreibungsmittel herausgestellt. Das *Klassifikationsschema* zur Einordnung der visuellen Modellierungssprachen wird in Kapitel 3 hergeleitet. Hierzu werden grundlegende Beschreibungsparadigmen identifiziert. Entlang dieser Paradigmen werden die verschiedenen Beschreibungsmittel anhand ihrer konkreten Notation eingeführt und Anforderungen an das Referenz-Metaschema formuliert. Als durchgängige Beispiele dienen hierzu Teilmodelle zur Beschreibung der Organisation Krankenhaus. In Kapitel 4 wird das in Teil II entwickelte Referenz-Metaschema als Modell, als Metamodell und als Referenzmodell eingeordnet. Dieses Kapitel enthält auch die Abgrenzung zu vergleichbaren Ansätzen.

Die Herleitung des Referenz-Metaschemas erfolgt in Teil II. Kapitel 5 führt in die graphbasierte Konzeptmodellierung mit dem hierzu verwendeten *EER/GRAL*-Ansatz ein. Entlang des in Kapitel 3 entwickelten Klassifikationsschemas wird in Kapitel 6 das Referenz-Metaschema hergeleitet. Hierzu werden zunächst einzelne Referenz-Metaschemata der grundlegenden visuellen Modellierungssprachen erstellt und zur Metamodellierung verschiedener konkreter Beschreibungsformen dieser Paradigmen angewandt. Anschließend erfolgt die Zusammenfassung dieser Teilschemata zum integrierten Referenz-Metaschema für visuelle Modellierungssprachen der Organisations- und Softwaretechnik.

Teil III umfaßt die *Validierung* des integrierten Referenz-Metaschemas. Hierzu wird exemplarisch aus dem integrierten Referenz-Metaschema das Metaschema der Beschreibungsmittel des ARIS-Ansatzes (vgl. Kapitel 7) und das Metaschema der Beschreibungsmittel der Unified Modeling Language (vgl. Kapitel 8) abgeleitet. Die Anwendung des Referenz-Metaschemas zur Entwicklung von Modellierungswerkzeugen nach dem *KOGGE*-Ansatz wird in Kapitel 9 fallstudienartig für ein Werkzeug zur Unterstützung der Evaluation von Softwaresystemen nachgewiesen.

Kapitel 10 umfaßt eine kurze Zusammenfassung der Arbeit und schließt mit einem Ausblick ab.

Der Anhang enthält eine Zusammenfassung der Formalisierung des Referenz-Metaschemas und einen graphisches Glossar der grundlegenden Begriffe des Referenz-Metaschemas.

Teil I

Grundlagen und Einordnung

Ein Ziel dieser Arbeit ist die Klassifikation und Metamodellierung der zur Beschreibung von Organisationen und Softwaresystemen eingesetzten visuellen Modellierungssprachen. Kapitel 2 skizziert die Gemeinsamkeiten und Überschneidungen der zur Organisations- und zur Softwaremodellierung eingesetzten Beschreibungstechniken entlang der hierzu verwendeten Vorgehensmodelle und motiviert somit die integrierte Betrachtung der Beschreibungsmittel der Organisations- und Softwaretechnik.

Ein Klassifikationsschema zur Einordnung der visuellen Modellierungssprachen wird in Kapitel 3 entwickelt. Die Beschreibungsmittel zur Modellierung von Organisationen und Softwaresystemen werden auf zehn Paradigmen zurückgeführt, entlang derer die diversen Sprachen in ihren konkreten Notationen eingeführt werden.

Die Darstellung und Klassifikation der einzelen Beschreibungsmittel erfolgt in Teil II durch Modelle, die die wesentlichen Eigenschaften der visuellen Sprachen der einzelnen Paradigmen herausstellen. Diesen Modellen wird einerseits Referenzcharakter zugesprochen, da sie leicht zu speziellen Modellen konkreter Beschreibungsformen erweitert oder eingeschränkt werden können. Da diese Modelle andererseits auch Modelle von Modellierungsmitteln darstellen, sind sie auch als Metamodelle aufzufassen. In Kapitel 4 erfolgt daher die Einordung des im folgenden entwickelten Referenz-Metaschemas der visuellen Modellierungssprachen der Organisations- und Softwaretechnik als Modell, als Referenzmodell und als Metamodell.

2 Organisationen und Softwaresysteme

Die Modellierung von Organisationen und von Softwaresystemen weist vielfältige Gemeinsamkeiten auf. In beiden Bereichen werden strukturelle und dynamische Zusammenhänge näher beschrieben. Hierbei werden ähnliche Vorgehensmodelle verfolgt und gleichartige graphische und textuelle Beschreibungsmittel eingesetzt.

Im folgenden Kapitel werden diese Gemeinsamkeiten herausgestellt. Ausgehend von einer Eingrenzung der Begriffe „Organisationstechnik“ und „Softwaretechnik“ werden hierzu die in beiden Bereichen verwendeten Vorgehensmodelle, Methoden und Beschreibungsmittel isoliert skizziert. Anschließend werden die Gemeinsamkeiten und Querbezüge der Organisationsmodellierung und Softwaremodellierung herausgestellt.

2.1 Modellierung von Organisationen

Mit ***Organisation*** werden umgangssprachlich Institutionen wie z. B. Behörden, Industrie- und Handwerksbetriebe, Universitäten, Krankenhäuser und Vereine bezeichnet. In der Organisationslehre sind im wesentlichen zwei unterschiedliche Begriffsbildungen zu finden, die den Organisationsbegriff auf das organisierte System als Ganzes (*institutionaler Organisationsbegriff*) oder auf das Regelwerk zur Organisation des Systems (*strukturaler oder instrumentaler Organisationsbegriff*) beziehen.

Institutionaler Organisationsbegriff

Der umgangssprachlichen Verwendung folgt der ***institutionale Organisationsbegriff***. Der Begriff Organisation wird hierbei als allgemeiner Oberbegriff für komplexe, soziale Systeme aufgefaßt, die durch die Interaktion mehrerer Menschen geprägt sind. Diese von der Soziologie geprägte Begriffsbildung ist vor allem in der *angelsächsischen Organisationslehre* verbreitet.

[Leavitt, 1965] konkretisiert diesen Organisationsbegriff durch vier Systemvariablen. Organisationen werden von ihm als komplexe Systeme aufgefaßt, die mindestens durch die Variablen Aufgabe, Person/Akteur, Technik und Struktur charakterisiert sind. Die Systemvariable *Aufgabe* umfaßt hierbei alle Tätigkeiten der Organisation einschließlich deren Zerlegung in Teilaufgaben. Mitglieder der Organisation wie auch die von den Aktivitäten der Organisation betroffenen Kunden etc. werden durch die Systemvariable *Person/Akteur* beschrieben. Insbesondere umfaßt diese Variable auch die Qualifikation der Mitarbeiter. Die Systemvariable *Technik* faßt alle Hilfsmittel wie z. B. Maschinen und Computersysteme zusammen, die zur Aufgabenerledigung eingesetzt

werden. Durch die Systemvariable *Struktur* werden die Aktivitäten der Organisationsmitglieder gesteuert. Die Struktur einer Organisation kann als eine Sammlung von Regeln zur Koordination des Verhaltens der Organisationsmitglieder zur Aufgabenerfüllung aufgefaßt werden. Hierunter fallen z. B. Regeln zur Kommunikation zwischen den Mitgliedern oder Festlegungen konkreter Arbeitsabläufe [Kieser / Kubicek, 1992, S. 16ff]. In [Leavitt / Bahrami, 1988, S. 246ff] wird als zusätzliche Systemvariable noch die *Umwelt* ergänzt, von der die betrachtete Organisation abzugrenzen ist.

Aus soziologischer Sicht versteht [Mayntz, 1985] unter Organisation soziale Gebilde, die auf die zielgerichtete Erfüllung umrissener Aufgaben ausgerichtet sind. Nach dieser Begriffsbildung sind Organisationen durch Angabe bzw. Festlegung ihrer Mitglieder deutlich von ihrer Systemumgebung abgegrenzt. Merkmale dieser Begriffsbildungen werden von [Kieser / Kubicek, 1992, S. 4ff] zusammengefaßt: Unter Organisation werden solche „*soziale[n] Gebilde*“ verstanden, „*die dauerhaft ein Ziel verfolgen und eine formale Struktur aufweisen, mit deren Hilfe Aktivitäten der Mitglieder auf das verfolgte Ziel ausgerichtet werden sollen*“.

Einen allgemeineren institutionalen Organisationsbegriff führen [Falkenberg et al., 1998, S. 84] ein. Beliebige Zusammenfassungen von (nicht notwendig menschlichen) Akteuren und Gegenständen werden hier als Organisationen aufgefaßt. Wesentlich für den Organisationsbegriff nach [Falkenberg et al., 1998] ist die Interaktion zwischen den Akteuren, die zum Funktionieren der Organisation allgemein akzeptierten Regeln genügt. Der Zusammenhang zwischen den Akteuren einer Organisation kann durch ein gemeinsames Ziel festgelegt sein. Im Gegensatz zu [Mayntz, 1985] und [Kieser / Kubicek, 1992] kann aber dieser auch unabhängig von einem Organisationsziel, beliebig formuliert sein. Diese Begriffsbildung erscheint aber zu weit gefaßt, da die Aktivitäten der Organisationsmitglieder auf ein gemeinsam akzeptiertes Ziel (vgl. zu Organisationszielen auch [Kieser / Kubicek, 1992, S. 5ff]) ausgerichtet sind. Die Zielerreichung determiniert somit auch die Interaktion der Organisationsmitglieder, so daß die Zielorientierung auch als wesentliches Charakteristikum einer Organisation betrachtet werden muß.

Strukturaler Organisationsbegriff

Eine Begriffsauffassung, die die Strukturkomponente sozialer Systeme in den Mittelpunkt der Betrachtung rückt, wird durch den ***strukturalen*** oder ***instrumentalen Organisationsbegriff*** vertreten. Hiernach wird die Organisation als ein System formaler Regeln aufgefaßt, das als Instrument (griech.: organon) zur Steuerung des sozialen Systems dient (vgl. [Hoffmann, 1980]). Dieser Organisationsbegriff wird vorallem durch die *deutschsprachige Organisationslehre* vertreten.

So versteht [Nordsieck, 1962] in Analogie zur Verfassung eines Staates die Organisation einer Unternehmung als ein auf Dauer angelegtes „*Gefüge einer Vielzahl von Regelungen*“. Diese Regeln stellen das Zusammenwirken des organisierten Systems sicher, das als eine „*mit Hilfsmitteln ausgestattete Gemeinschaft aller Arbeitskräfte, die zur Erfüllung der Betriebsaufgabe zusammenwirken*“ [Nordsieck, 1972] charakterisiert ist. Die Notwendigkeit der Abgrenzung zwischen dem organisierten System (hier Betrieb) und seiner Organisation begründet [Nordsieck, 1972] dadurch, daß er den Betrieb als materiell, gegenständlich auffaßt, während das ihn steuernde Regelsystem abstrakt, nicht gegenständlich ist. Insbesondere gehören hiernach die im Unternehmen tätigen Personen *nicht* zur Organisation wohl aber zum organisierten System.

Die Bestimmung des Organisationsbegriffs bei [Kosiol, 1976] folgt ebenfalls der instrumentalen Auffassung und stellt so die Struktur des sozialen Systems als wesentlich heraus. Kosiol vertritt darüber hinaus einen ***funktionalen Organisationsbegriff***. Organisation wird hier als das (technische) Handeln der Menschen zur *„dauerhafte[n], integrative[n] Strukturierung von Gefügesystemen“* [Kosiol, 1976, S. 5] aufgefaßt. Dieser funktionalen Begriffsauffassung folgt auch die Gesellschaft für Organisation e. V., die Organisation als *„ganzheitliches Gestalten der Beziehungen zwischen Aufgaben, Menschen, Sachmitteln und Informationen in sozialen Systemen“* [Büchli / Chrobok, 1997, S. 103] auffaßt. Im Gegensatz zu Nordsiecks Begriffsbildung, die eher das in einem Regelsystem vorliegende Ergebnis eines Strukturierungsprozesses betont, stellen diese Begriffsbildungen auch die zielgerichtete Tätigkeit zur Bildung dieser Strukturen heraus.

Wie [Nordsieck, 1962] und [Kosiol, 1976] bezieht auch [Grochla, 1982] den Organisationsbegriff auf Regelsysteme zur Steuerung und Ordnung von Institutionen. Ausgangspunkt seiner Überlegungen bilden *sozio-technische Systeme* [Grochla, 1978, S. 8ff]. Während bei der Betrachtung sozialer Systeme in erster Linie die Interaktion mehrerer Individuen betont wird, werden bei der Untersuchung sozio-technischer Systeme auch die (technischen) Hilfsmittel einbezogen, die zur Leistungserbringung genutzt werden. Werden zusätzlich die Hilfsmittel der Informationstechnik in den Vordergrund gestellt, spricht man auch von sozio-informationstechnischen Systemen [Euler, 1989], [Ebert et al., 1992, S. 53]. Sozio- (informations-) technische Systeme sind definiert als *„eine Menge von in Beziehung stehenden Menschen und Maschinen, die unter bestimmten Bedingungen nach festgelegten Regeln bestimmte Aufgaben erfüllen“* [Grochla, 1978, S. 9]. Aus systemtheoretischer Sicht stellt Grochla hier die Verhaltensweisen und Interaktionen der Elemente des Systems als das systembildende Kriterium heraus. Erst durch die systembezogene Rolle der Menschen wird das betrachtete System von seiner Systemumwelt abgegrenzt (vgl. hierzu auch [Willke, 1982, S. 36ff]). Das Zusammenwirken der Menschen und Maschinen zur Aufgabenerfüllung wird durch festgelegte Regeln gesteuert, die sich sowohl auf das Verhalten der Menschen als auch auf die Funktion der Maschinen beziehen. Die Gesamtheit dieser festgelegten (organisatorischen) Regeln bildet die Organisation des sozio-technische Systems [Grochla, 1978, S. 12ff], [Grochla, 1982].

Auch [Hill et al., 1994] vertreten den instrumentalen Organisationsbegriff. Sie fassen unter dem Begriff Organisation die Gesamtheit aller auf die Erfüllung eines Ziels gerichteten Maßnahmen auf, die ein soziales System strukturieren und die Aktivitäten der Menschen in diesem System, den Einsatz der hierzu benötigten Mittel und die Verarbeitung von Informationen ordnet.

Objektbereich der Organisationslehre

Der Objektbereich der Organisationslehre umfaßt alle durch formale Regeln gesteuerten sozio- (informations-) technischen Systeme. Solche formalen Regeln werden in einem bewußten Gestaltungsprozeß (funktionaler Organisationsbegriff) festgelegt und als gültig vereinbart. Der Gegenstandsbereich der Organisationslehre umfaßt somit einerseits sozio- (informations-) technische Systeme und andererseits auch die in Regeln festgelegte (instrumentale) Organisationen. Somit behandelt die Organisationslehre solche Systeme, die durch den institutionalen Organisationsbegriff zusammengefaßt sind.

> Unter dem Begriff ***Organisation*** wird sowohl die organisatorische Struktur als auch die Einbettung der Organisationsstruktur in das umgebende soziotechnische System verstanden.

Zusammenfassende Gegenüberstellungen unterschiedlicher Organisationsbegriffe finden sich z. B. auch in [Kosiol, 1976, S. 15ff], [Hoffmann, 1980], [Steinebach, 1983, S. 79ff], [Voßbein, 1987, S. 8ff] und [Engel, 1993].

Organisationstechnik

Wesentlicher Inhalt der Organisationslehre ist die Gestaltung von Organisationen. In Anlehnung an den Begriff „software engineering“ führt [Kaucky, 1988] den Begriff *Organisations-Engineering* ein. Die ***Organisationstechnik***[1] bezeichnet nach [Kaucky, 1988, S. 100] die „*Anwendung von Prinzipien, Methoden und Werkzeugen [...] zur Durchführung gezielter organisatorischer Änderungen unter Berücksichtigung der Informationstechnik*“. Ähnlich definieren [Jacobson et al., 1994, S. 2] den Begriff „Business Engineering“ als die Menge der Techniken, die ein Unternehmen zur Entwicklung der Strukturen zur Erreichung des Unternehmensziels nutzt. Durch den Begriff „Engineering“, der i. allg. eher auf die Entwicklung technischer Systeme bezogen ist, soll hier insbesondere die methodische Strukturierung der Organisationsgestaltung hervorgehoben werden. Die ingenieursmäßige Gestaltung von Organisationen betont auch [Balzert, 1998a, S. 691], der jedoch anstelle des Begriffs „Organisationstechnik“[2] bewußt den „*weniger befremdlich klingenden*“ Begriff “Organisationsmodellierung“ verwendet.

Neben der strukturierten Durchführung von Organisationsgestaltungen ist auch die Entwicklung und Einordung der hierzu benötigten Prinzipien, Methoden und Werkzeuge als Teil der Organisationstechnik aufzufassen. Auf die in der Begriffsbildung von [Kaucky, 1988] besonders herausgestellte Berücksichtigung der Informationstechnik wird dagegen in den weiteren Betrachtungen verzichtet, da die Informationstechnik heute immanenter Teil der Organisationsgestaltung ist (vgl. auch Kapitel 2.3) und daher nicht mehr speziell erwähnt werden muß.

> Die ***Organisationstechnik*** beschäftigt sich sowohl mit der Entwicklung von Prinzipien, Methoden, Techniken und Werkzeugen zur Organisationsgestaltung als auch mit der Anwendung dieser Mittel zur Durchführung gezielter organisatorischer Änderungen.

2.1.1 Vorgehen zur Organisationsgestaltung

Ein allgemeines Vorgehensmodell zur Gestaltung von Organisationen wird Abbildung 2.1 vorgestellt (vgl. hierzu auch [Krüger, 1981, Bild 2], [Kaucky, 1988, S. 110ff], [Büchli / Chrobok,

[1] Analog zur „Softwaretechnik“ (vgl. Kapitel 2.2) wird im folgenden anstelle des Begriffs „Organisations-Engineering“ der Begriff Organisationstechnik verwendet.

[2] Konkret beziehen [Jacobson et al., 1994], [Balzert, 1998a] ihre Betrachtungen auf *Unternehmen*. Da der Unternehmensbegriff jedoch für betriebliche Unternehmen und kaum für Behörden, Verwaltungen, Krankenhäuser etc. verwendet wird, wird in dieser Arbeit der allgemeinere Begriff „Organisation“ genutzt.

1997, S. 75ff], [Frank, 1997a, Abb. 8]). Nach Erteilung eines Auftrags zur Organisationsgestaltung und einer Machbarkeitsüberprüfung ist zunächst in einer *Analysephase* (Organisation analysieren) das zu bearbeitende Organisationsproblem zu strukturieren sowie die Ziele der Organisationsgestaltung festzulegen. Die *Entwurfsphase* (Organisation entwerfen) hat die Planung, Entwicklung und Bewertung von Lösungsalternativen und die Konzeption der Problemlösung zum Ziel. Die Erarbeitung der Problemlösung setzt aus der Analysephase eine detailierte Ermittlung der Anforderungen an die zu entwickelnden Lösungen voraus. In der *Realisierungsphase* (Organisation realisieren) schließlich werden die zuvor entwickelten Konzepte umgesetzt. Es schließt sich eine kurze *Einführungsphase* (Organisation einführen) an, auf die die *Wirkungsphase* (Organisation verwenden und erhalten) folgt. Mit der Systemverwendung einher geht die Erhaltung des Systems durch Beseitigung von Störungen sowie kleinere Anpassungen an geänderte Bedingungen.

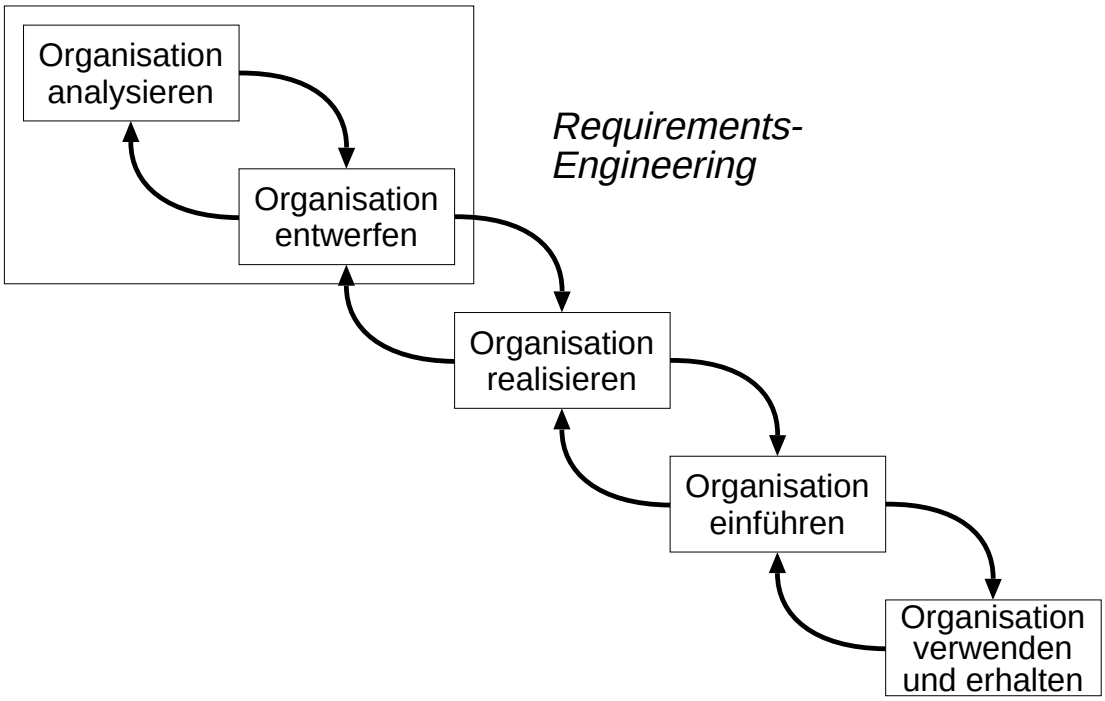


Abbildung 2.1: Ein Wasserfallmodell zur Organisationsgestaltung

Ein verallgemeinertes, *zyklisches Vorgehensmodell* zur Projektdurchführung wird in [Schmidt, 1980], [Schmidt, 1989, S. 39ff] vor dem Hintergrund der Organisationsgestaltung diskutiert. Der Organisationsprozeß erfolgt in mehreren Stufen, wobei die der Analyse- und Entwurfsphase zuzurechnenden Stufen eine ähnliche Feinstruktur aufweisen. Nach dem Anstoß Organisationsgestaltung erfolgt die Erstellung diverser Studien (Vorstudie, Hauptstudie, Teilstudien) zur Konzeption der Problemlösung. In diesen Teilphasen werden zyklisch Schritte zur Festlegung des Vorgehensplanes, zur Erhebung und Analyse der relevanten Informationen des Ist-Zustandes, zur Bewertung des Ist-Zustandes, zur Gestaltung einer Solllösung und zur Bewertung der Auswahlentscheidung durchlaufen. Nach Fertigstellen der Hauptstudie erfolgt dann die Realisierung der erarbeiteten Problemlösung, deren Einführung und Verwendung.

2.1.2 Methoden des Requirements-Engineering der Organisationstechnik

Die Analyse- und die Entwurfssphase der Organisationsgestaltung, in denen die konzeptionelle Entwicklung der zu gestaltenden Organisation erfolgt, werden im folgenden unter dem Begriff ***Requirements-Engineering***[3] zusammengefaßt.

Das Requirements-Engineering der Organisationstechnik kann ausgehend von der Aufbauorganisation oder der Ablauforganisation durchgeführt werden (vgl. [Nordsieck, 1962, s. 9f],[Kosiol, 1976, S. 32ff]). Die ***Aufbauorganisation*** behandelt die Untersuchung der statischen Organisationsaspekte. Hier stehen die Aufgaben der Organisation und die mit der Aufgabenbewältigung beauftragten Organisationseinheiten sowie die hierzwischen vorliegenden Querbezüge im Mittelpunkt der Betrachtung. Dynamische Organisationszusammenhänge werden in der ***Ablauforganisation*** untersucht. Beschreibungsschwerpunkt ist hier die räumliche und zeitliche Strukturierung der Aufgabenerledigung.

Methoden des Requirements-Engineering der Organisationstechnik können in *funktionsorientierte* und *prozeßorientierte* Ansätze unterschieden werden. Während in der Organisationslehre *Funktion* Zusammenfassungen von Aufgaben entlang der Aufbauorganisation bezeichnen, versteht man unter *Prozessen* solche Aufgabenzusammenfassungen, die bezogen auf die Ablauforganisation erfolgen [Eversheim, 1996, S. 14f] [Rolf, 1998b, S. 69]. Funktions- bzw. prozeßorientierte Methoden der Organisationstechnik gehen jeweils von der Betrachtung der Aufbau- bzw. Ablauforganisation aus.

Funktionsorientiertes Requirements-Engineering:

Die klassische, ***funktionsorientierte*** Gestaltung von Organistationen wird von der Zerlegung der zu erledigenden Aufgaben in einfache Unteraufgaben geprägt (vgl. z. B. [Kosiol, 1976], [Nordsieck, 1972], [Smith, 1990]). Der eigentlichen Organisationsmodellierung (in der Entwurfsphase) ist die Analyse der Organisationsaufgabe (*Aufgabenanalyse*) vorgeschaltet, bei der diese Aufgabengliederung, unabhängig von zukünftig beabsichtigten Aufgabengruppierungen, erfolgt. Die Aufgabenanalyse bildet die Grundlage der *Aufgabensynthese*, bei der im Rahmen der Stellenbildung zunächst die ermittelten Teilaufgaben im Hinblick auf die späteren Aufgabenträger zu Aufgabenkomplexen gruppiert werden. Diese Stellen werden anschließend in Organisationseinheiten zusammengefaßt. Neben diesen aufbauorganisatorischen Aspekten ist weiter der Arbeitsprozeß zur Erledigung der Organisationsaufgabe zu regeln. In der *Arbeitsanalyse* werden die in der Aufgabenanalyse ermittelten, nicht weiter zerlegten Teilaufgaben vor dem Hintergrund ihrer Ausführung näher untersucht. Diese tiefere Zerlegung der einzelnen Aufgaben bildet in der *Arbeitssynthese* die Grundlage zur Festlegung von Arbeitsgängen, in der u. a. die Ausführung durch die Aufgabenträger und die Abfolge zur Aufgabenerledigung bestimmt wird.

Zentraler Schritt der funktionsorientierten Organisationsgestaltung ist die statische Zerlegung der Aufgaben in Teilaufgaben. Dieses Vorgehen führt i. allg. zu einer Betrachtung der Organisation aus Sicht der wesentlichen Unternehmensfunktionen (Beschaffung, Produktion, Absatz, Finanz-, Rechnungs- und Personalwesen). Funktionale Abhängigkeiten einzelner Arbeitsschritte fließen jedoch in die Organisationsgestaltung nicht ein. Die ablauf-

[3] Der Begriff Requirements-Engineering wird hier in Analogie zum Begriff Requirements-Engineering der Softwaretechnik (vgl. Kapitel 2.2) eingeführt.

organisatiorische Gestaltung bezieht sich dann, bei gegebener Stellenbildung, nur noch auf die zeitliche Anordnung der Arbeitsschritte der beteiligten Stellen. Die Ablaufplanung des funktionsorientierten Requirements-Engineerings optimiert eher in bezug auf Verweilzeit pro Stelle und nicht in bezug auf die gesamte Bearbeitungsdauer pro Prozeß [Gaitanides, 1983, 61ff]. In funktionsorientiert entwickelten Organisationen führen spezialisierte Stellen häufig wenige Funktionen an wenigen Objekten aus. Zur Weiterbearbeitung werden die Ergebnisse einzelner Arbeitsschritte an andere Organisationseinheiten weitergegeben. Hieraus resultierte ein häufiges Wechseln der Zuständigkeiten innerhalb der Bearbeitung eines Prozesses.

Prozeßorientiertes Requirements-Engineering:

Im ***prozeßorientierten Requirements-Engineering*** (vgl. z. B. [Gaitanides, 1983], [Scheer, 1992], [Gaitanides et al., 1994a], [Eversheim, 1996], [Hammer / Champy, 1996, 52ff], [Ferstl / Sinz, 1996]) erfolgt die Organisationsgestaltung in Umkehrung des funktionsorientierten Requirements-Engineerings nicht entlang der hierarchisch orientierten Aufgabenteilung sondern ausgehend vom Ablauf der Unternehmensprozesse. Dem prozeßorientierten Entwurf einer Organisation geht hierbei eine ausführliche Prozeßanalyse voraus. Nach Festlegung und Eingrenzung der für die Organisation wesentlichen Prozeße (Geschäftsprozesse) durch Angabe des Prozeßbeginns, Prozeßendes und der Randbedingungen erfolgt zunächst eine hierarchische Zerlegung in Teilprozesse und (atomare) Prozeßelemente. Die hierbei ermittelten Prozeßelemente werden anschließend gemäß ihrer zeitlichen und logischen Bearbereitungsabhängigkeiten geordnet und um Bearbeitungszeiten ergänzt. Diese Prozeßbeschreibungen bilden dann (erst) die Grundlage zur Stellenbildung, indem funktional sinnvoll zusammengehörige Prozeßelemente zusammengefaßt werden. Im letzten Schritt der Organisationsgestaltung erfolgt dann die Koordination der Prozeßelemente innerhalb der Prozesse sowie die Abstimmung der einzelnen Prozesse untereinander. [Gaitanides, 1983, S. 64ff]

Die funktionsorientierte Methode des Requirements-Engineering zielt auf eine zweckmäßige Zuordnung der einzelnen Aufgaben bzw. Aufgabenausführungen auf Stellen und Organisationseinheiten. Die Bildung der Aufbauorganisation dominiert dieses Vorgehen. Erst nach der Stellenbildung wird die Ablauforganisation betrachtet. Beim prozeßorientierten Requirements-Engineering bestimmen dagegen zentrale Prozesse (z. B. der Prozeß der Produktentwicklung oder der Auftragsabwicklung) die Gestaltung der Organisationsstruktur. Ziel ist es hierbei, die Prozesse optimal zu gestalten. Die Geschäftsprozeß-Modellierung determiniert die Aufbauorganisation (vgl. auch [Jablonski et al., 1997, S. 8] und [Rolf, 1998a, S. 68f]).

2.1.3 Visuelle Modellierungssprachen der Organisationstechnik

Für diese Methoden des Requirements-Enginnering der Organisationstechnik werden Darstellungsmittel benötigt, die sowohl aufbauorganisatorische als auch ablauforganisatorische bzw. prozeßbezogene Zusammenhänge herausstellen. Neben der Organisationsstruktur sollten diese Modellierungssprachen auch die Einbettung der Organisation in das sie umgebende System unterstützen.

Visuelle Modellierungssprachen des Requirements-Engineering der Organisationstechnik dienen als Hilfsmittel zur Erhebung und Beschreibung organisatorischer Zusammenhänge bezogen auf die statische und dynamische Organisationsstruktur und das umgebende sozio-technische System.

Eingesetzt werden diese Sprachen sowohl zur Erhebung und Dokumentation dieser Zusammenhänge als auch zur Kommunikation bei der Entwicklung, Weiterentwicklung und Vermittlung von Organisationen. Die in den folgenden Kapiteln betrachteten Modellierungssprachen dienen sowohl zur Beschreibung der Ausgangssituation in der Analysephase bzw. zu Beginn der Entwurfsphase als auch zur Entwicklung, Beschreibung und Umsetzung der Problemlösung in der Entwurfs- und Realisierungsphase.

Die visuellen Modellierungssprachen des Requirements-Engineering der Organisationstechnik werden in Kapitel 3 ausführlich dargestellt. Überblicksdarstellungen zu graphischen und textuellen Sprachen der Organisationstechnik finden sich z. B. in [Nordsieck, 1962], [Acker, 1973], [Joschke, 1980], [Schmidt, 1980], [Grochla, 1982, S. 295ff], [Blum, 1991] und [Lehner et al., 1991, S. 264ff].

2.2 Modellierung von Softwaresystemen

Die ***Softwaretechnik*** (engl. software engineering) befaßt sich mit dem Erstellen umfangreicher Softwaresysteme. Sie folgt ingenieurmäßigen Prinzipien, die nach [Balzert, 1996a, S. 36] einerseits „*Künstlertum*“ ausschließt und andererseits die ziel- und marktorientierte Entwicklung optimaler Problemlösungskompromisse unterstützt.

Der Begriff “software engineering“ wurde zum Ende der 60er Jahre als Reaktion auf die allgemeine Unzufriedenheit mit produzierter Software (Software Krise) geprägt. Bauer faßte diese Problematik im Umfeld der ersten Konferenz mit dem Titel „software engineering“ (vgl. [Naur / Randell, 1969]) passend zusammen: „*The whole trouble comes from the fact that there is so much tinkering with software. It is not made in a clean fabrication process, which it should be. [. . .] What we need, is software engineering.*“ (nach [Bauer, 1993]). Folglich wurde auch für die Entwicklung und Anwendung von Software nach (ingenieur-) wissenschaftlich fundierten Grundlagen gesucht.

Die Verwendung dieser wissenschaftlichen und mathematischen Grundlagen zur Nutzbarmachung von Computersystemen für den Menschen durch Programme, Verfahren und Dokumente betrachten [Boehm, 1981] und [Denert, 1991] als wesentlichen Schwerpunkt der Softwaretechnik. Ähnlich beschreibt auch [ANSI/IEEE Std. 729, 1983] die Softwaretechnik als den systematischen Ansatz zur Entwicklung, zum Betrieb und zur Wartung von Software. Boehm stellt auch die Dokumentation, die zur Erstellung, zur Verwendung und zur Wartung der Software nötig ist, deutlich heraus. So definiert er in [Boehm, 1976] die Softwaretechnik als die praktische Anwendung wissenschaftlicher Kenntnisse zur Entwicklung und Realisierung von Computerprogrammen einschließlich der zur Entwicklung, zum Betrieb und zur Wartung benötigten Dokumentation.

Auch die Entwicklung der bei der Softwareerstellung benötigten wissenschaftlichen Grundlagen ist als Aufgabe der Softwaretechnik zu sehen. Thema der Softwaretechnik ist nach [Hesse et

al., 1984] die Bereitstellung und systematische Verwendung von Methoden und Werkzeugen für die Herstellung und Anwendung von Software. Ähnlich argumentiert auch [Sommerville, 1996, S. 4]. Softwaretechnik beschäftigt sich mit Theorien, Methoden und Werkzeugen, die zur Entwicklung von Computersoftware benötigt werden. Diese Begriffsbildungen zusammenfassend versteht [Balzert, 1996a, S. 36f] unter Softwaretechnik die „*zielorientierte Bereitstellung und systematische Verwendung von Prinzipien, Methoden, Konzepten, Notationen und Werkzeugen für die arbeitsteilige, ingenieurmäßige Entwicklung von umfangreichen Softwaresystemen.*“

> Die Softwaretechnik befaßt sich sowohl mit der wissenschaftlichen Entwicklung von Prinzipien, Methoden, Techniken und Werkzeugen zur Softwareentwicklung als auch mit der Anwendung dieser Mittel zur Erstellung, zum Betrieb und zur Wartung von umfangreichen Softwareprodukten.

Diskussionen verschiedener Softwaretechnik-Begriffe finden sich auch bei [Balzert, 1996a, S. 35] und [Pressman, 1997, S. 22].

2.2.1 Vorgehen zur Softwareerstellung

Wie auch in der Organisationstechnik wird der Softwareerstellungsprozeß durch Vorgehensmodelle strukturiert. Ein einfaches und nicht empfehlenwertes Modell aus den ersten Tagen der Programmierung ist im *code-and-fix-Modell* [Boehm, 1988] zu sehen. Programmcode wurde erstellt, anschließend wurden die Fehler beseitigt. Ein auf einer Analyse- und Synthesephase beruhendes Zwei-Phasenmodell wird als Ausgangspunkt zur Entwicklung des *Phasenmodells* in [Royce, 1970] vorgestellt. Bei diesem Vorgehensmodell folgen die grundlegenden Entwicklungsschritte der Softwareentwicklung nach Abnahme der Teilergebnisse aufeinander. Royce entwickelte dieses Modell anschließend zu dem von [Boehm, 1981, S. 35] als *Wasserfallmodell* bekannt gemachten Vorgangsmodell weiter. Die einzelnen Entwicklungsschritte können hier auch iterativ bearbeitet werden. Ebenso wurde das Wasserfallmodell um prototypische Studien ergänzt. Die in der Literatur beschriebenen Phasen der Wasserfallmodelle zur Softwareentwicklung variieren sowohl in den Bezeichnungen wie in den jeweils zugeordneten Teiltätigkeiten. Die Grundstruktur ist jedoch für alle Modelle ähnlich und bietet ein gutes Strukturierungsmittel der einzelnen Schritte zur Softwareentwicklung. Eine ausführliche Beschreibung des kanonischen Lebenszyklus-Modells, der in Abbildung 2.2, angelehnt an Abbildung 2.1, komprimiert dargestellt ist, findet sich z. B. in [Boehm, 1981, S. 37] oder [McDermid / Rook, 1991].

Die Softwareentwicklung beginnt in der *Analysephase* oder *Definitionsphase* (Software analysieren) mit der Festlegung der qualitativen und quantitativen Anforderungen an das zu realisierende System. Hierbei sind insbesondere die funktionalen Eigenschaften des Softwaresystems aus Anwendersicht zu erheben sowie die Realisierungsmöglichkeiten des Vorhabens zu überprüfen. In der *Entwurfsphase* (Software entwerfen) erfolgt die Entwicklung eines Realisierungskonzepts, das die zuvor definierten Anforderungen erfüllt. Die Umsetzung dieser Entwürfe in ausführbare Programme erfolgt in der *Implementierungsphase* oder *Realisierungsphase* (Software implementieren). Neben der Programmierung der einzelnen Komponenten erfolgt hier auch die Integration dieser Module in das Gesamtsystem. Dieses System ist anschließend in der *Einführungsphase* (Software einführen) beim Anwender einzuführen. Es schließt sich die *Wirkungsphase* (Software

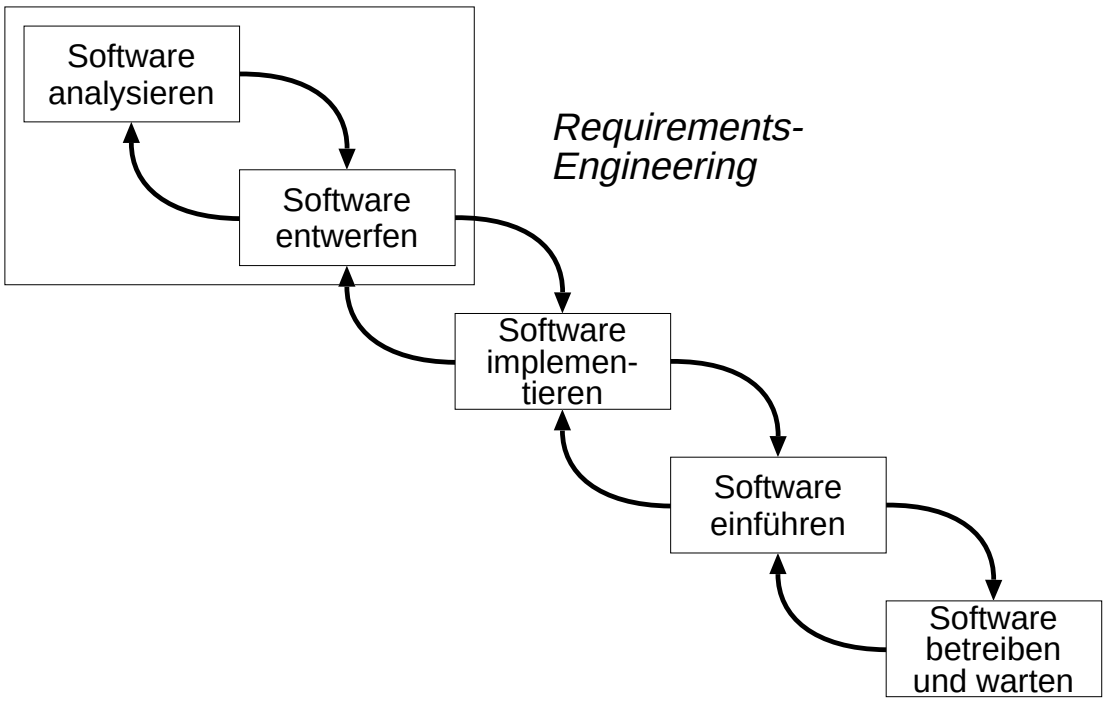


Abbildung 2.2: Ein Wasserfallmodell des Software-Lebenszyklus

betreiben und warten) mit dem Betrieb und der Pflege und Wartung des Systems an. Die Softwareentwicklung nach dem Wasserfallmodell läuft jedoch nicht generell sequentiell ab. Rückgriffe auf vorhergehende Schritte sowie Teilentwicklungen in unterschiedlichen Phasen sind durchaus möglich.

Das Wasserfallmodell bildet die Grundlage zur Entwicklung weiterer Vorgangsmodelle. Die bekannteste Erweiterung ist im *Spiralmodell* [Boehm, 1988] (vgl. auch das zyklische Vorgehensmodell zur Organisationsgestaltung nach [Schmidt, 1980] auf Seite 17) zu sehen, in dem sowohl die schon bei [Royce, 1970] angesprochene Verwendung von Prototypen als auch die Risiko-Abschätzung der einzelnen Entwicklungsstufen weiter ausgeprägt wurde. Jede Phase der Softwareentwicklung (z. B. Erstellung der Machbarkeitsanalyse, Erheben der Anforderungen) wird in vier Teilschritten, der Festlegung des Phasenziels, der Risikoabschätzung, der Entwicklung und Validierung und der Planung der nächsten Phase bearbeitet.

Das Spiralmodell kann als Referenzmodell für Vorgehensmodelle zur Softwareentwicklung aufgefaßt werden, da aus ihm ein Großteil der anderen Vorgehensmodelle abgeleitet werden kann. [McDermid / Rook, 1991] gibt einen umfassenden Überblick zu diesen Vorgangsmodellen zur Softwareentwicklung. Weitere Diskussionen dieser Vorgehensmodelle zur Softwareerstellung findet sich z. B. auch in [Boehm, 1988] und [Lehner et al., 1995, S. 87ff].

2.2.2 Methoden des Requirements-Engineering der Softwaretechnik

Die frühen Phasen der Softwareentwicklung werden ebenfalls unter dem Begriff ***Requirements-Engineering*** zusammengefaßt. Ziel des Requirements-Engineering der Softwaretechnik ist es, eine vollständige, konsistente, realisierbare und eindeutige Spezifikation bereitzustellen, in der

beschrieben ist, *was* ein Softwaresystem tun soll [Boehm, 1979], [ANSI/IEEE Std. 729, 1983]. Die Beschreibungsmittel des Requirements-Engineering beziehen sich nur bedingt auf die Darstellung der Funktionsweise eines Softwaresystems. Programmiersprachen werden daher auch in den folgenden Betrachtungen ausgeklammert.

Bei den Methoden des Requirements-Engineering der Softwaretechnik werden die eher prozeßorientierten Ansätze der *(modernen) strukturierten Analyse* und die *objektorientierten* Ansätze unterschieden. In beiden Ansätzen sind Daten bzw. Datenstrukturen, Kontrollstrukturen und funktionale Zusammenhänge einzelner Softwarebausteine zu erheben und zu beschreiben.

(Moderne) Strukturierte Analyse

Die Methoden der *strukturierten Analyse* (z. B. [DeMarco, 1978], [Gane / Sarson, 1979]) basieren auf der prozeßorientierten Beschreibung des Systems durch den Datenaustausch zwischen Systemprozessen. Die Modellierung nach der strukturierten Analyse beginnt mit der Abgrenzung des Softwaresystems von seiner Umwelt. Hierzu sind die Datenflußbeziehungen, über die das Softwaresystem mit seiner Umwelt kommuniziert, zu erheben. Ausgehend von diesen Datenflüssen wird das Softwaresystem in Teilprozesse zerlegt, die ebenfalls durch ihren Datenaustausch charakterisiert sind. Nicht weiter zerlegte Prozesse werden durch Angabe ihrer Kontrollflüsse konkretisiert. Die *moderne strukturiere Analyse* [Yourdon, 1989] erweitert die strukturierte Analyse um die Beschreibung von der Systemdynamik und der Informationsstrukturen. Die Steuerung der Abarbeitung der einzelnen Prozesse wird durch zusätzliche Kontrollprozesse (vgl. auch die Echt-Zeit-Variante der strukturierten Analyse [Ward / Mellor, 1985]) modelliert. Parallel und (nahezu) unabhängig von der Prozeßmodellierung erfolgt die Erstellung des Informationsstrukturmodells, das anschließend mit dem Datenflußmodell abgeglichen wird. Die Modellierung nach der modernen stukturierten Analyse erfolgt aus unterschiedlichen Blickwinkeln, die zusammengefaßt das System vollständig abbilden. Die unabhängige Entwicklung der einzelnen Teilmodelle führt jedoch leicht zu Inkonsistenzen in der Modellierung.

Objektorientierte Analyse

Während bei der (modernen) strukturierten Analyse die prozeßorientierte Untersuchung deutlich dominiert, stellen die *objektorientierten Methoden* (z. B. [Rumbaugh et al., 1991], [Booch, 1994], [Booch et al., 1999]) die Untersuchung der Daten einschließlich der hierauf auszuführenden Operationen in den Vordergrund. Zentraler Betrachtungsgegenstand sind Objekte, die sowohl durch ihre Datenstrukturen als auch durch ihr Verhalten charakterisiert sind. In den objektorientierten Ansätzen werden statische und dynamische Strukturen des Systems als Sichten auf das gleiche System aufgefaßt. So besteht z. B. eine Modellierung nach der Object Modeling Technique (OMT) [Rumbaugh et al., 1991, S. 6ff] aus einem *Objektmodell*, zur Beschreibung der statischen Klassen-Struktur, einem *dynamischen Modell*, zur Beschreibung der Veränderungen der (internen) Systemzustände und einem *funktionalen Modell* zur datenflußorientierten Beschreibung des nach außen sichtbaren Verhaltens. [Booch, 1994] bzw. [Booch et al., 1999] unterscheiden analog die *statische* bzw. die *strukturelle Sicht* zur Darstellung der Systemstrukturen, und die *dynamische* bzw. die *Verhaltenssicht* zur Betrachtung der Veränderungen des Systemzustands. Idealtypische objektorientierte Modellierungen beginnen mit der Erstellung des Objektmodells.

Zunächst werden die relevanten Klassen, deren Attribute und Methoden sowie die Beziehungen der Klassen untereinander erhoben und beschrieben. Diese statische Modellierung bildet gemeinsam mit Szenarien, die typisches und untypisches Systemverhalten abbilden, den Ausgangspunkt zur Beschreibung des möglichen dynamischen Verhaltens der Objekte. Erst nach Erstellung des Objektmodells und des dynamischen Modells wird nach der OMT [Rumbaugh et al., 1991, S. 179] das funktionale Modell erstellt. Hierbei werden ausgehend von den im dynamischen Modell notierten Aktivitäten und den im Objektmodell modellierten Objekten und Attributen die Daten- und Kontrollflüsse beschrieben. Vgl. hierzu auch die Vorgehensmodelle zur objektorientierten Analyse in [Balzert, 1996a, S. 359] oder in [Rumbaugh et al., 1991, S. 148ff].

Sowohl die strukturierte Analyse als auch die objektorientierten Methoden benötigen zum Requirements-Engineering Beschreibungsmittel, die statische und dynamische Systemaspekte hervorheben. Beide Ansätze verwenden letztendlich die gleichen oder ähnliche visuelle Sprachen (vgl. auch [Ebert / Engels, 1994]), die sich häufig nur in ihrer konkreten Notation unterscheiden und bei unterschiedlichen Methoden zu unterschiedlichen Zeitpunkten im Modellierungsprozeß eingesetzt werden. Eine gute Übersicht über die konkreten Beschreibungsmittel, die in den verschiedenen Methoden der strukturierten Analyse oder der objektorientierten Modellierung verwendet werden, bietet auch [Wieringa, 1998].

2.2.3 Visuelle Modellierungssprachen der Softwaretechnik

So wie sich die visuellen Sprachen zur Organisationsbeschreibung sowohl auf organisatorische Regeln als auch auf die Einbettung in ihr organisatorisches Umfeld beziehen, unterstützen die Sprachen des Requirements-Engineering der Softwaretechnik sowohl die Beschreibung der Strukturen und Abläufe innerhalb eines Softwaresystems als auch die Einbettung des Softwaresystems in seinen Anwendungskontext. Insbesondere unterstützen die Sprachen, die in der Analysephase eingesetzt werden, die Abgrenzung des zu beschreibenden Softwaresystems von seiner Systemumwelt, während die Beschreibungsmittel der Entwurfsphase eher auf die internen Zusammenhänge ausgerichtet sind.

> Visuelle Modellierungssprachen des Requirements-Engineering der Softwaretechnik dienen als Hilfsmittel zur Erhebung und Beschreibung von Softwaresystemen bezogen auf ihre statische und dynamische Struktur sowie auf das sie umgebende Anwendungssystem.

Diese Sprachen dienen zur Unterstützung der Erhebung und Dokumentation von Entwurfsentscheidungen während der Softwareerstellung. Sie unterstützen darüber hinaus auch die Kommunikation zwischen den bei der Softwareentwicklung betroffenen Personen. Die Beschreibungsmittel, die in der Analysephase eingesetzt werden, dienen hier als Kommunikationsmittel mit den Anwendern bzw. Auftraggebern. In der Entwurfs- und Implementierungsphase werden diese Sprachen eher als Kommunikationsmittel zwischen Systementwicklern eingesetzt. Die Beschreibungsmittel, die insbesondere in der Entwurfsphase eingesetzt werden, unterstützen den Softwareentwickler bei der Umsetzung der eher informell beschriebenen Anforderungen aus der Analysephase in formale Beschreibungen und ausführbare Progamme.

In Kapitel 3 werden auch die visuellen Modellierunssprachen des Requirements-Engineerings der Softwaretechnik ausführlicher dargestellt. Überblicksdarstellungen zu diesen graphischen und textuellen Beschreibungsmitteln bieten auch [Martin / McClure, 1985a], [Ebert / Engels, 1994], [Wieringa, 1998] und [Partsch, 1998].

2.3 Modellierung von Organisationen und Softwaresystemen

Sowohl die Modellierung von Organisationen als auch die Modellierung von Softwaresystemen beschäftigt sich mit der Gestaltung komplexer Systeme. Wie auch die groben Phasenmodelle in Abbildung 2.1 und 2.2 zeigen, werden zur Organisationsgestaltung und zur Softwareentwicklung ähnlich strukturierte Vorgehensmodelle eingesetzt, die gemeinsame Beschreibungsmittel zur Darstellung statischer und dynamischer Systemzusammenhänge im Requirements-Engineering verwenden.

Die Ursprünge der Techniken zur Beschreibung statischer Aspekte der Aufgabenanalyse und zur Aufbauorganisation können eher in der Organisationslehre (insbesondere bei [Nordsieck, 1962] und [Kosiol, 1976]) gefunden werden. Die Entwicklung der Beschreibungsmittel zur Darstellung ablauforganisatorischer Zusammenhänge ist aber auch eng mit der Entwicklung in der Informatik verbunden. Zur Beschreibung von Ablaufstrukturen in Software wurden hier z. B. Programmablaufpläne [DIN 66001, 1983], Struktogramme [Nassi / Shneiderman, 1973] oder Datenflußdiagramme [Ross, 1977], [DeMarco, 1978] entwickelt, die in gleicher Form auch zur Beschreibung organisatorischer Ablaufstrukturen eingesetzt werden. Eine Vorreiterfunktion der prozeßorientierten Organisationsgestaltung weisen [Gaitanides et al., 1994a] sowohl der Fertigungstechnik als auch der Informatik zu. Die Beschreibung von Daten- bzw. Objektstrukturen durch Objekt-Beziehungsdiagramme durch [Chen, 1976] oder Klassendiagramme z. B. durch [Rumbaugh et al., 1991], [Booch, 1994] und [Booch et al., 1999] wurde ebenfalls eher durch die Informatik geprägt. Zusammenfassend kann festgestellt werden, daß die heute im Requirements-Engineering verwendeten visuellen Modellierungssprachen ihren Ursprung sowohl in der Organisationslehre als auch in der Informatik finden (vgl. auch [Keller et al., 1992]).

Darüber hinaus bedingen sich Organisationsgestaltung und Softwareentwicklung gegenseitig in der Form, daß weder die Organisationsgestaltung ohne Bezug zur Softwareentwicklung noch die Softwareentwicklung ohne Bezug zur Organisationsgestaltung gesehen werden kann.

- Aus Sicht der ***Organisationstechnik*** kommen Organisationen heute kaum noch ohne softwaretechnische Hilfsmittel aus. Die Umsetzung und Koordination von Geschäftsprozessen erfordert die Bereitstellung entsprechender softwaretechnischer Unterstützung, die wiederum ihre Anforderung an Organisationsstrukturen und -abläufe stellt. Diese Hilfsmittel unterstützen sowohl die Erhebung, Berechnung, Weitergabe und Bereitstellung von Daten als auch die Steuerung konkreter Abläufe. Um diese Aufgaben angemessen zu unterstützen, muß in dieser Software Wissen über die Organisationsstruktur, über die Abläufe zur Aufgabenerledigung und über die einzusetzenden Hilfsmittel abgebildet sein.

- Aus Sicht der ***Softwaretechnik*** bezieht sich die Entwicklung von Problemlösungen nicht ausschließlich auf die Programmierung von performanten Softwaresystemen. Die Problemlösungen müssen auch die Einbettung der Softwaresysteme in die ihre Anwendungs-

umgebung berücksichtigen. Da Informationssysteme mit dem Ziel erstellt werden (sollten), Hilfsmittel zur Unterstützung und Vereinfachung von Arbeitsabläufen zu sein, ist auch eine umfangreiche Erhebung der Anforderungen der Anwendungsbereiche erforderlich. Hierbei sind ebenso strukturelle Organisationszusammenhänge wie z. B. Aufgabenzerlegungen und Aufgabenverteilungen als auch dynamische Organisationsaspekte wie z. B. die zu unterstützenden Geschäftsprozesse zu beachten.

Stärkere Querbezüge zwischen Informatik und Organisationstechnik wurden bereits in den 70er Jahren gefordert. Ausgehend von der Feststellung, daß Informatiker bei der Lösung von Anwendungsproblemen auch für die Einbindung der Softwarelösung in die Betriebsorganisation zuständig sind, forderte [Zemanek, 1971] eine organisationstheoretische Fundierung der Informatik.

Eine stärkere Berücksichtigung sozialer und organisatorischer Zusammenhänge in der Informatik fordert auch [Coy, 1989]. Die Analyse von Arbeitsprozessen und ihre maschinelle Unterstützung, bei der nicht die Maschine, sondern die Organisation und Gestaltung der Arbeitsplätze im Vordergrund steht, sieht er als wesentliche Aufgabe der Informatik. Aus der Auffassung der Informatik als Disziplin, die sich mit der Organisation und Steuerung formal beschreibbarer und durch Prozessoren ausführbarer Prozesse beschäftigt, leitet [Kornwachs, 1997] die „*Theorie der Organisation als Mutterwissenschaft*“ der Theorie der Informatik ab.

Kling geht noch einen Schritt weiter und führt „Organizational Informatics“ als eigenständige Disziplin ein, die sich mit der Entwicklung und Verwendung von computerbasierten Informations- und Kommunikationssystemen in Organisationen befaßt [Kling, 1993]. Gegenüber der technisch ausgerichteten „Computer Science“ soll die „Organizational Informatics“ vor allem die Bedingungen untersuchen, unter denen der Einsatz von Computersystemen die Aufgabenunterstützung der Organisationen verbessert.

Die Verbindungen zwischen Softwaretechnik und Organisationstechnik stellt auch [Rolf, 1998a, 7ff], [Rolf, 1998b] in seinen Betrachtungen zur „*Organisations-Informatik*“ heraus. Er leitet aus diesen Querbezügen die Entwicklung von Methoden, Modellen und Werkzeugen, die sowohl organisatorische als auch softwaretechnische Perspektiven berücksichtigen, als eine Herausforderung für die (Wirtschafts-) Informatik ab.

Auch bewegen sich akutelle Ansätze zur Modellierung von Organisationen und Softwaresystemen deutlich aufeinander zu. [Flatscher, 1998, S. 13] faßt beispielsweise Softwarewerkzeuge zur Aufbau- und Ablaufmodellierung unter den Begriff der CASE-Werkzeuge. Auch durch die Standardisierung der Modellierungssprachen in der Unified Modeling Language [Booch et al., 1999] wird ein Satz an Bescheibungsmitteln vorgeschlagen, der sowohl zur Organisations- als auch zur Softwaremodellierung eingesetzt werden kann. Beispielsweise beschäftigen sich [Jacobson et al., 1994], [Balzert, 1998a, s. 721ff], [Oestereich, 1998], [Versteegen, 1998] und [Rumbaugh et al., 1999] mit der Anwendung objektorientierter Techniken zur Organisationsmodellierung. Organisationsmodelle werden durch ihre Einbettung in die sie umgebende Systemumwelt und durch Geschäftsprozesse aus externer und aus interner Sicht einschließlich der benötigten Objekte dargestellt [Balzert, 1998a, S. 722]. Diese in erster Linie prozeßorientierte Organisationsmodellierung erfolgt durch Beschreibungsmittel (u. a. Anwendungsfalldiagramme und Aktivitätendiagramme), die aus objektorientierten Methoden der Softwareentwicklung entnommen wurden. Daher wird hierfür auch häufig der Begriff „objektorientierte Unternehmensmodellie-

rung“ verwendet. Auf rein notationeller Ebene ermöglichen diese Ansätze die Integration von Organisations- und Softwaremodellierungen.

Eine gemeinsame Betrachtung organisatorischer und softwaretechnischer Zusammenhänge ist heute in nahezu allen Gestaltungsbereichen der Organisations- und Softwaretechnik anzutreffen. Beispiele für die gemeinsame Betrachtung organisatorischer und softwaretechnischer Aspekte finden sich u. a. bei der Gestaltung von Informationssystemen, der Auswahl und Einführung von Standardsoftware, dem Business Reengineering und der Modellierung von Workflows.

Informationssysteme

Informationssysteme werden als sozio-(informations)-technische Teilsysteme von Unternehmen aufgefaßt. Diese Teilsysteme umfassen die informationsverarbeitenden Aktivitäten, die hieran beteiligten Mitarbeiter in ihrer informationsverarbeitenden Rolle und die hierzu eingesetzten Hilfsmittel [Haux et al., 1998]. Informationssysteme dienen der Beschaffung, Herstellung, Bevorratung und Verwendung von Informationen [Heinrich, 1997] und beschreiben somit einen Ausschnitt der Unternehmensmodellierung [Hoppen, 1992, S. 61], [Lehner et al., 1995, S. 104]. Neben der Betrachtung des Informationssystems, sind zur Unternehmensmodellierung auch die Unternehmensstrategie und die Organisationsstrukturen, die sowohl die Aufbau- wie die Ablauforganisation umfassen, zu beachten [Frank, 1994] [Frank, 1997a].

Die Modellierung von Informationssystemen erfordert sowohl die Beschreibung der informationsverarbeitenden Aktivitäten als auch die organisatorische Einbettung dieser Systeme. Ansätze zur Modellierung von Informationssystemen (vgl. z. B. [Olle et al., 1991], [Scheer, 1992], [Frank, 1994]) fordern daher auch die Modellierung von *Aufgaben* und *Prozessen*, die durch das Informationssystem ausgeführt werden, von *Objektstrukturen* bzw. *Daten*, die durch das Informationssystem bearbeitet werden und von *Organisationsstrukturen*, die die Aufgabenerledigung des Informationssystems steuern.

Auswahl und Einführung von Standardsoftware

Das Angebot von Standardsoftware geht von der Annahme aus, daß die verschiedenen Anforderungen unterschiedlicher Organisationen noch so viele Gemeinsamkeiten aufweisen, daß es möglich ist, ein standardisiertes Softwaresystem zu erstellen, daß in vielen Organisationen eingesetzt werden kann. Im Gegensatz zu allgemeiner Standardsoftware, wie beispielsweise (weit verbreitete) Textverabeitungen, Tabellenkalkulationen, Datenbankmanagementsysteme oder Modellierungswerkzeuge, durch die in der Regel einzelne Funktion unterstützt werden, ist *betriebswirtschaftliche Standardsoftware* dadurch charakterisiert, daß sie die wesentlichen Geschäftsprozesse vieler Organisationen unterstützt. Die Anpassbarkeit an die individuellen Anforderung konkreter Organisationen wird dadurch erreicht, daß diese Produkte auf einem spezialisierbaren Geschäftsprozeßmodell aufsetzen (vgl. auch [Staud, 1999, S. 21ff]).

Die Auswahl und Einführung von (betriebswirtschaftlicher) Standardsoftware erfordert den Vergleich der durch die Organisation geforderte und der durch die Standardsoftware angebotene Unterstützung der Geschäftsprozesse. Standardsoftware wird hierzu häufig durch *Software-Referenzmodelle* beschrieben, die funktionale Eigenschaften, die verwendeten Objekt- und Datenstrukturen sowie die organisatorischen Voraussetzungen zur Ver-

wendung dieser Software abbilden. Wie auch für die Entwicklung von Informationssystemen sind bei der Auswahl und Einführung von Standardsoftware unterstützte *Aufgaben*, *Geschäftsprozesse*, benötigte *Daten* und *Organisationsstrukturen* zu modellieren und mit den Software-Referenzmodellen abzugleichen [Dumslaff et al., 1994], [Franzke / Winter, 1996].

Business Reengineering

Im Gegensatz zum *Business Improvement*, durch das die Pflege und Weiterentwicklung bestehender Unternehmen beschrieben wird, wird das *Business Reengineering* (alias Business Process Reengineering [Johansson et al., 1993], Process Innovation [Davenport, 1993]) als „*fundamentales Überdenken und radikales Redesign von Unternehmen oder wesentlichen Unternehmensprozessen*“ [Hammer / Champy, 1996, S. 48] eingeführt. Ziel des Business Reengineerings[4] sind deutliche und meßbare Verbesserungen der Unternehmensparameter Kosten, Qualität, Service und Aufwand.

Ergebnisse solcher Business-Reengineering-Vorhaben sind zu realisierende Organisationsmodelle sowie hiermit verträgliche Informationssysteme zur informationstechnischen Unterstützung der Unternehmensprozesse [Jacobson et al., 1993, S. 21]. Wie auch zur Modellierung von Informationssystemen (s.o.) werden im Business Reengineering Organisationen durch *Aufgaben* (Funktionen), durch *Organisationsstrukturen*, durch *Daten*- bzw. *Objektstrukturen* und im besonderen durch *Unternehmensprozesse* beschrieben.

Workflows

Workflow-Management-Systeme zielen auf eine möglichst vollständige computerbasierte Koordination der Vorgangsbearbeitung innerhalb einer Organisation. Ausgangspunkt der Betrachtungen sind hierbei die Vorgänge, die das Geschehen innerhalb der Organisation abbilden. Vorgänge, deren Ablauf beschrieben ist und dessen Bearbeitung durch ein Workflow-Management-System koordiniert wird, werden *Workflow* genannt [Jablonski / Bussler, 1996], [Jablonski et al., 1997, S. 23ff].

Die Vorgangsunterstützung durch Workflow-Management-Systeme setzt eine schematische Beschreibung der *Vorgänge* (*Workflow-Schema*) voraus, die geeignet ist, die Bearbeitung konkreter Vorgänge zu steuern. Zentraler Beschreibungsgegenstand ist hierbei die Abarbeitung dieser Workflows. Neben diesen Ablaufaspekten sind ferner auch die zur Abarbeitung benötigten bzw. erzeugten *Daten* sowie die *Akteure*, durch die Vorgänge umgesetzt werden, zu berücksichtigen [Jablonski et al., 1997, S. 98ff].

Zusammenfassend kann festgestellt werden, daß in der Organisations- und der Softwaregestaltung in vielen Anwendungbereichen organisatorische und softwaretechnische Zusammenhänge gemeinsam zu modellieren sind. Die Darstellungen aus eher organisatorischer und eher softwaretechnischer Sicht bedingen sich hierbei gegenseitig.

[4] Der Reengineering-Begriff des Business Reengineering wird deutlich radikaler aufgefaßt als der Reengineering-Begriff der Softwaretechnik. Während Software Reengineering sowohl das Verstehen (understanding) als auch das Verändern und das eher evolutionäre Weiterenwickeln (renovation) bestehender Softwaresysteme umfaßt (vgl. z. B. [Arnold, 1993a], [Kullbach / Winter, 1999]) bezieht sich das Business Reengineering eher auf die grundlegende, revolutionäre Überarbeitung und Änderung von Organisationen.

Die Beschreibung von Organisationen und Softwaresystemen umfaßt *Aufgabenstrukturen*, *Aufbaustrukturen*, *Daten- bzw. Objektstrukturen* und *Ablaufstrukturen*. Dieses ist auch von der jeweils gewählten Modellierungsmethode unabhängig. Die Methoden der funktionsorientierten oder prozeßorientierten Organisationsgestaltung bzw. der strukturierten oder der objektorientierten Softwareentwicklung unterscheiden sich nur in der unterschiedlichen Betonung der einzelnen Strukturen. Es ist daher notwendig und sinnvoll, diese Beschreibungsmittel auch zueinander in Beziehung zu setzen, um eine durchgängige Verwendung der Ergebnisse einzelner Modellierungssichten zu gewährleisten.

2.4 Einordnung in die Zielsetzung dieser Arbeit

Organisationstechnik und Softwaretechnik weisen eine Reihe von Gemeinsamkeiten in bezug auf Vorgehensweisen und Beschreibungsmittel auf. Auch bedingen sich Organisationstechnik und Softwaretechnik gegenseitig, so daß heute weder die Organisationsgestaltung ohne Berücksichtigung der Softwaretechnik noch die Softwareentwicklung ohne Berücksichtigung der Organsiationsgestaltung erfolgen kann. Eine integrierte Betrachtung von Organisationstechnik und Softwaretechnik ist folglich sinnvoll.

Ziel dieser Arbeit ist die Integration der in beiden Bereichen verwendeten visuellen Modellierungssprachen, um so auf notationeller Ebene Beschreibungstechniken der Organisations- und Softwaretechnik kombinieren zu können. Das in Kapitel 6 entwickelte *Referenz-Metaschema für visuelle Modellierungssprachen* stellt eine integrierte Darstellung der fundamentalen Konzepte der Beschreibungsmittel aus Organisationstechnik und Softwaretechnik bereit.

3 Visuelle Modellierungssprachen der Organisations- und Softwaretechnik

Visuelle Modellierungssprachen für organisatorische und softwaretechnische Zusammenhänge legen unterschiedliche Schwerpunkte bei der Betrachtung von Organisationen und Softwaresystemen. Die Beschreibungsmittel der einzelnen Betrachtungssichten oder -perspektiven verfolgen verschiedene Paradigmen zur Systembeschreibung. Diese Sichten und Paradigmen bilden die Grundlage eines Klassifikationsschemas, anhand dessen die Beschreibungsmittel steckbriefartig eingeführt werden.

Hierzu werden die einzelnen Sprachen entlang der Paradigmen in ihrer konkreten Syntax dargestellt. Die durch die Beschreibungsmittel der einzelnen Paradigmen abgebildeten Konzepte legen die Anforderungen an das Referenz-Metaschema für visuelle Modellierungssprachen der Organisations und Softwaretechnik in Kapitel 6 fest.

3.1 Klassifikationsschema für visuelle Modellierungssprachen

Zur Darstellung der visuellen Modellierungssprachen des Requirements-Engineering der Organisationstechnik und der Softwaretechnik wird im folgenden ein Klassifikationsschema eingeführt, daß eine möglichst umfassende Überblicksdarstellung dieser Beschreibungsmittel erlaubt, aber auch die Entwicklung eines Referenz-Metaschemas dieser Sprachen ermöglicht. [Partsch, 1998, S. 53ff] schlägt als mögliche Ordnungskriterien die Darstellungsform, den Grad der Präzision und die jeweils betrachtete Sicht (Systemaspekt) auf das zu beschreibende System vor.

Nach der *Darstellungform* unterscheidet man textuelle und graphische Beschreibungen. Textuelle Beschreibungen werden in freie Texte und in durch Hervorhebungen, Einrückungen oder ähnliche Strukturierungsmittel systematisierte Texte unterschieden. Zu den systematisierten Texten gehören u. a. Pseudo-Code, Listen und Tabellen. Graphische Darstellungen verwenden einfache geometrische Objekte wie z. B. Punkte, Linien, Kreise und Rechtecke, die in ihrem Zusammenhang dargestellt werden. Einige Werkzeuge (z. B. das ARIS Toolset [IDS, 1998], Bonapart Professional [Pro Ubis, 1999b], System Architect 2001 [Popkin, 1999] oder Neptun/NetCase [Marx, 1998]) unterstützen verschiedene Darstellungsformen, die neben diesen abstrakten Formen auch Piktogramme zur Beschreibung verwenden. Textuelle und graphische Mittel zur Darstellung organisatorischer und softwaretechnischer Zusammenhänge beschreiben häufig gleiche Inhalte und sind ineinander überführbar. So werden Aufgabengliederungen z. B. textuell durch Aufgabenli-

sten und graphisch durch Funktionsbäume äquivalent dargestellt. Textuelle und graphische Darstellungen unterscheiden sich lediglich in ihrer konkreten Notation. Das Referenz-Metaschema zielt jedoch auf eine Darstellung der konzeptionellen Zusammenhänge dieser Sprachen, so daß eine Klassifikation nach Darstellungsform in diesem Kontext nicht sinnvoll erscheint.

Nach dem *Grad der Präzision* unterscheidet man formale, semiformale und informale Beschreibungsmittel. Unter formale Sprachen werden z. B. algebraische Spezifikationen [Ehrig / Mahr, 1985], VDM [Jones, 1990] oder Z [Spivey, 1992] gefaßt, die vollständig auf algebraischen, mengentheoretischen und logischen Fundamenten aufbauen. In semi-formalen und informalen Beschreibungsmitteln tritt die mathematische Fundierung in den Hintergrund. Diese Sprachen sind eher auf Kommunikation und weniger auf exakte, mathematische Formalisierungen ausgerichtet. Die weiteren Betrachtungen beziehen sich auf diese semi-formalen und informalen Sprachen. Diese Sprachen enthalten aber auch Bezüge zu formalen Beschreibungsformen wie z. B. prädikatenlogische Annotation an Objekt-Beziehungsdiagrammen oder durch Prädikate notierte Vor-/Nachbedingungen zur Methodenbeschreibung. Die Unterscheidung nach Grad der Präzision wird aufgrund des Betrachtungsschwerpunkts auf semiformalen und informalen Beschreibungsmitteln im weiteren nicht zur Klassifikation der Sprachen verwendet.

Bei der Klassifikation nach der *Sicht* auf das betrachtete System stehen die von einzelnen Modellierungssprachen besonders betonten Systemaspekte im Vordergrund. Ausgehend von verschiedenen Ansätzen zur Beschreibung von Soft- und Hardwareystem in EPOS stellt [Göhner, 1984] hier die Aspekte *Funktion, Ablauf, Ereignis, Datenfluß, Datenstruktur* und *Gerät* heraus. In [Partsch, 1998, S. 44] werden dieses Aspekte weiter abstrahiert. Er gliedert die visuellen Modellierungssprachen des Requirements-Engineering nach *System und Komponentenstrukturen*, nach *Kontrollaspekten und Steuerung* und nach *Abläufen und funktionalem Verhalten*. Im Klassifikationssystem von [Scheer, 1994] zur Strukturierung der Sprachen zur Modellierung von Informationssystemen wird neben der *Datensicht* und der *Funktionssicht* noch die *Organisationssicht* ergänzt, in der strukturelle Aspekte der Organisationsmodellierung betrachtet werden. Daten- und Funktionssicht entsprechen in etwa den Sichten aus [Partsch, 1998], wobei hier die Kontrollaspekte mit dem funktionalen Verhalten zusammengefaßt wurden. Eine ähnliche Sichteneinteilung verfolgen auch [Jablonski et al., 1997, S. 98ff]. Neben der *Organisationssicht* und der *Informationssicht* (Datensicht) analog zu Scheer werden hier die eher ablauforientierten Aspekte auf eine *Funktionssicht*, eine *Verhaltenssicht* und eine gegenüber Partsch zusätzliche *Operationssicht* aufgeteilt.

3.1.1 Beschreibungssichten

Auch wenn sich diese Sichteneinteilungen in ihren Bezeichnungen und den genauen Einteilungen der Sprachen unterscheiden, bieten sie ein gutes Strukturierungsmittel für die verschiedenen Beschreibungsmittel, so daß auch im folgenden eine Klassifikation der visuellen Modellierungssprachen nach Sichten verwendet wird. Unterschiedliche Darstellungsmittel heben jeweils einige Aspekte des zu modellierenden Systems hervor und blenden andere aus. Eine vollständige Darstellung aller zu untersuchenden Zusammenhänge erlaubt nur die gemeinsame Beschreibung aller Sichten.

Beispiel 3.1 (Beschreibung einer Krankenhaus-Organisation aus verschiedenen Sichten)

Abbildung 3.1 zeigt einen Ausschnitt einer Krankenhausinformationssystem-Modellierung aus vier verschiedenen Beschreibungssichten (vgl. [Winter / Ebert, 1997]).

Der Ausriß des *Aufgabengliederungsplans* beschreibt die Aufgaben der Pflegedienstleitung (PDL) und deren Untergliederung. Das *Organigramm* skizziert die Abteilungsstruktur der Aufbauorganisation des Krankenhauses. Im *Objekt-Beziehungsdiagramm* ist ein Teil der Datenstrukturen für Patientendaten dargestellt. Einen Einblick in die Prozesse des Labors erlaubt das *Datenflußdiagramm*. Es beschreibt die Teilprozesse, die zur Befunderstellung nötig sind, einschließlich deren Datenaustauschbeziehungen und Einbettung in das Gesamtsystem Krankenhaus. □

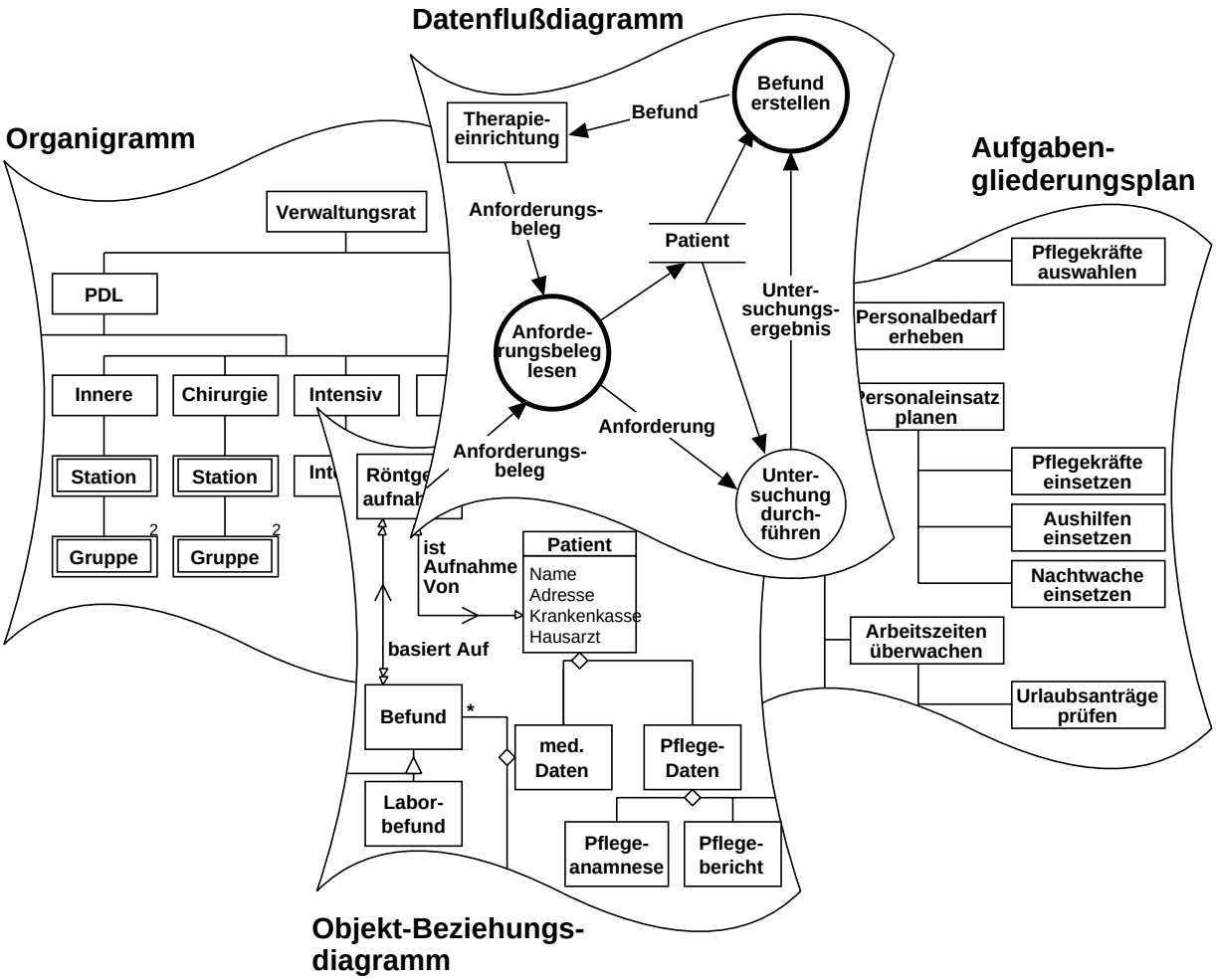


Abbildung 3.1: Beschreibung einer Krankenhaus-Organisation aus verschiedenen Sichten

Mit Hilfe der vier Diagramme in Abbildung 3.1 wird das Krankenhaus aus Blickwinkeln zur Darstellung der Aufgabenstruktur, der Aufbaustruktur, der Prozeßstrukturen und der Objektstrukturen beschrieben. Diese vier Blickwinkel liefern auch die Strukturierung zur Betrachtung der verschiedenen Beschreibungsmittel in dieser Arbeit.

Zur Klassifikation der visuellen Modellierungssprachen für Organisationen und Softwaresysteme werden

- die ***Aufgabensicht***,
- die ***Aufbausicht***,
- die ***Prozeßsicht*** und
- die ***Objektsicht***

verwendet. In jeder dieser Sichten wird ein Aspekt der Organisation bzw. des Softwaresystems besonders hervorgehoben. In der Aufgabensicht sind dieses die *Aufgaben*, die durch die Organisation bzw. das Softwaresystem zu bearbeiten bzw. zu unterstützen sind. In den zuvor skizzierten Sichtenkonzepten wird diese Sicht als Teil der Funktionssicht zur Beschreibung des funktionalen Verhaltens oder als Teil statischer Organisations- oder Struktursichten betrachtet. Die *Organisationseinheiten*, die diese Aufgaben erledigen bzw. deren Arbeit durch Softwaresysteme unterstützt wird, werden in der Aufbaussicht[1] untersucht. Die Aufbausicht entspricht den Organisationssichten für den Entwurf von Informationssystemen nach [Scheer, 1994] oder von Workflowsystemen nach [Jablonski et al., 1997]. Die Prozeßsicht faßt die Funktionssicht, die Verhaltenssicht, die Operationssicht nach [Scheer, 1994] und [Jablonski et al., 1997] sowie die Kontroll- und Ablaufaspekte aus [Partsch, 1998] zusammen. Diese Sicht betont *zeitliche und logische Prozeßabläufe* in Organisationen und Softwaresystemen. Die Objektsicht hebt die *Objekte* der Organisation und der Softwaresysteme hervor. Diese Sicht umfaßt die Daten- oder Informationssicht aus [Scheer, 1994] oder [Jablonski et al., 1997].

3.1.2 Beschreibungsparadigmen

Zur Einordnung der visuellen Modellierungssprachen reicht die Sichteneinteilung noch nicht aus, da zur Darstellung aus einer Sicht auch konzeptionell verschiedene Beschreibungsmuster verwendet werden. Zur Modellierung der Organisationseinheiten aus Aufbausicht werden z. B. Organigramme verwendet, die hierarchische Strukturbeziehungen darstellen. Ebenso finden hier auch Kommunikationsdiagramme zur Modellierung von Kommunikationsbeziehungen Verwendung. Modellierungen aus Prozeßsicht verwenden datenflußorientierte Beschreibungsmittel (z. B. SADT-Diagramme), zustandsübergangsorientierte Beschreibungsmittel (z. B. Statecharts) netzartige Beschreibungsmittel (z. B. Netzpläne) oder kontrollflußorientierte Beschreibungsmittel (z. B. Nassi-Shneiderman-Diagramme). Aus Objektsicht werden sowohl schematische Beschreibungen (z. B. Klassendiagramme) als auch Instanzbeschreibungen (z. B. Objektdiagramme oder Interaktionsdiagramme) zur Darstellung von Objektzusammenhängen eingesetzt.

Die Beschreibungsmittel einer Sicht folgen unterschiedlichen ***Paradigmen***. Ähnlich wie Programmiersprachen nach dem imperativen, dem funktionalen oder dem logischen Sprachparadgima klassifiziert werden können, kann dieses auch für die hier betrachteten Modellierungssprachen erfolgen. Während durch die Sichten festgelegt wird, welche *Konzepte* hervorgehoben werden, wird durch die *Paradigmen* die Art der Beziehungen zwischen diesen Konzepten unterschieden.

[1] Der Begriff *Organisationssicht*, der i. allg. auf aufbauorganisatorische Zusammenhänge eingeschränkt verwendet wird, wird hier nicht genutzt, da der Organisationsbegriff dieser Arbeit weiter gefaßt ist.

Sprachen des Softwareentwurfs werden in [Ebert / Engels, 1994] entlang des *Datenflußparadigmas*, des *Zustandsübergangsparadigmas*, des *Kontrollflußparadigmas*, des *Modulparadigmas* und des *Objekt-Beziehungsparadigmas* eingeordnet. Hierbei beziehen sich die ersten drei Paradigmen auf die Beschreibungsmittel der *Prozeßsicht*. Beziehungen zwischen einzelnen Aktivitäten oder Prozessen[2] werden hierbei entweder durch Datenflüsse, durch Zustandsübergänge oder durch Kontrollflüsse dargestellt. Das Modulparadigma und das Objekt-Beziehungsparadigma stellen grundlegende Beschreibungsparadigmen der *Objektsicht* dar. Objekte werden hier einschließlich ihrer Beziehungen *schematisch* beschrieben. Gegenüber dem Objekt-Beziehungsparadigma kann das Modulparadigma als Spezialfall aufgefaßt werden, in dem spezielle Objekt- und Beziehungsklassen zur Darstellung von Modulen und hierzwischen vorliegende Benutzungsbeziehungen betrachtet werden.

3.1.3 Beschreibungssichten und -paradigmen

Diese Paradigmen werden auch für die feinere Einordnung der hier behandelten Sprachen innerhalb der vier Sichten übernommen. Für die Betrachtung der Sprachen zur Organisationsmodellierung und Softwareentwicklung werden Paradigmen für die Prozeß- und Objektsicht ergänzt. Ebenso werden für die in [Ebert / Engels, 1994] nicht betrachtete Aufgaben- und Aufbausicht jeweils eigene Paradigmen eingeführt:

- Aus *Aufgabensicht* werden die Aufgaben einer Organisation oder eines Softwaresystems betont. Die Darstellung hierarchischer Gliederungen von Aufgaben in Teilaufgaben erfolgt entlang des *Aufgabengliederungsparadigmas*.

- Bei der Beschreibung aus *Aufbausicht* werden Organisationseinheiten, d. h. Stellen und Abteilungen, betrachtet. Nach der Art der Beziehungen zwischen den organisatorischen Einheiten werden das *Stellengliederungsparadigma* und das *Kommunikationsparadigma* unterschieden. Untergliederungen organisatorischer Einheiten sowie Leitungs- und Weisungsbeziehungen werden mit den Beschreibungsmitteln des Stellengliederungsparadigmas beschrieben. Kommunikations- und Interaktionsbeziehungen zwischen Stellen oder Abteilungen werden durch die Sprachen des Kommunikationsparadigmas notiert.

- Neben dem *Datenfluß-*, dem *Zustandsübergangs-* und dem *Kontrollflußparadigma* ergänzt das *Netzparadigma* die Paradigmen der Prozeßsicht zur Beschreibung zeitlicher oder logischer Abläufe. Die Sprachen des Netzparadigmas beschreiben Prozeßzusammenhänge durch ein Netz von durch Flußbeziehungen verbundenen aktiven und/oder passiven Komponenten (Prozesse und Ereignisse).

[2] Im Sprachgebrauch der Softwaretechnik werden diese Aktivitäten auch als *Prozesse* bezeichnet. Dieser Prozeßbegriff unterscheidet sich aber deutlich vom Prozeß- oder Geschäftsprozeß-Begriff der Organisationstechnik, der ablauforientierte Zusammenfassungen von Aufgaben bezeichnet (vgl. Seite 18). Ähnliches gilt für den Begriff *Funktion*. Während dieser in der Informatik als mathematische Funktion verstanden wird, durch die Werte eines Eingabewertebereichs in Werte eines Ausgabewertebereichs transformiert werden, faßt die Organisationstechnik Funktionen eher als aufbauorientierte Zusammenfassungen von Aufgaben auf. Soweit die Verwendung dieser Begriff aus dem Kontext erschließbar ist, werden sie sowohl für softwaretechnische als auch für organisationstechnische Zusammenhänge verwendet.

- Die Objekte, die eine Organisation oder ein Softwaresystem bilden, und deren Zusammenhänge werden aus der *Objektsicht* beschrieben. Zur Modellierung von Objekten gibt es Darstellungsmittel sowohl auf Schema- als auch auf Instanzebene. Das *Objekt-Beziehungsparadigma*[3] zur Beschreibung schematischer Objektzusammenhänge wird noch um das *Objekt-Instanzparadigma* und das *Objekt-Interaktionsparadigma* ergänzt. Die Sprachen des Objekt-Instanzparadigmas werden zur exemplarischen Darstellung von konkreten Objekten und deren statischen Beziehungen verwendet. Mit den Beschreibungsmittel des Objekt-Interaktionsparadigmas wird der Nachrichtenaustausch zwischen Objekten dargestellt.

Abbildung 3.2 faßt das im folgenden verwendete Schema zur Klassifikation der visuellen Modellierungssprachen für Organisationen und Softwaresysteme zusammen. Zu den vier Beschreibungssichten werden jeweils die zentralen Komponenten und ihre Beschreibungsparadigmen angegeben. Überblicksdarstellungen zu Beschreibungssichten und -paradigmen bieten auch [Ebert / Engels, 1994], [Winter / Ebert, 1996] und [Winter, 1996].

Dieses Klassifikationsschema ist nicht als starres, universell gültiges Raster zur Einordnung von Beschreibungsmitteln zu betrachten. Auch andere Schemata sind begründbar. So wird in [Ebert / Engels, 1994] für netzartige Beschreibungsmittel der Prozeßsicht kein eigenes Paradigma eingeführt. Notationen wie z. B. Netzgraphen der Petri-Netze werden hier unter das Datenflußparadigma gefaßt. Diese Einordnung erfolgte aus der Anschauung heraus, daß in Netzgraphen der Stellen-Transitions-Netze eine datenflußartige Kommunikation zwischen Petri-Netz-Stellen vorliegt. Petri-Netz-Stellen können aber auch als Zustände aufgefaßt werden, wonach eine Einordnung unter das Zustandsübergangsparadigma begründet werden kann. Ähnlich könnte auch aus den in Netzgraphen modellierten Abfolgen von Schaltvorgängen eine Zuordnung in das Kontrollflußparadigma abgeleitet werden. Da auf Prozeßbeschreibungen durch bipartite Graphen, in denen Abläufe durch die Interaktion passiver Komponenten (Zustände, Stellen etc.) und aktiver Komponenten (Prozesse, Transitionen etc.) modelliert werden, auch andere Beschreibungsmittel basieren, kann ebenfalls — wie in der hier verwendeten Klassifikation — eine eigene Sicht gerechtfertigt werden.

Das Kontrollflußparadigma nach [Ebert / Engels, 1994] umfaßt auch Ablaufdarstellungen, die nahezu beliebige Prozeßfolgen erlauben. Das im folgenden verwendete Kontrollflußparadigma ist dagegen ausschließlich auf regulär strukturierte Folgedarstellungen durch Sequenzen, Verzweigungen und Iterationen beschränkt. Darstellungen beliebiger Prozeßfolgen wie beispielsweise durch Aufgabenfolgepläne oder (generelle) Programmablaufpläne werden hier unter das Netzparadigma gefaßt. Diese Beschreibungsmittel verwenden ähnliche Mittel zur Beschreibung der Folgestrukturen wie auch die Notationsformen, die aktive und passive Komponenten deutlich unterscheiden, reduzieren die Darstellung jedoch auf die aktiven Komponenten (vgl. z. B. die Grundmuster zur Darstellung von Folgestrukturen in [Schmidt, 1989, S. 301ff] und [Keller et al., 1992, Abb. 4]).

Diese Entscheidungsspielräume sind nicht auf die Paradigmen einer Sicht begrenzt. Die visuellen Sprachen beispielsweise des Objekt-Interaktionsparadigmas wurden hier der Objektsicht zugeordnet. Diese Einteilung stellt die Objekte, zwischen denen Nachrichten ausgetauscht werden, in

[3] Ein korrekterer Name für dieses Paradigma wäre sicherlich *Objektklassen-Beziehungsklassenparadigma*. Im allgemeinen Sprachgebrauch bezogen auf die Beschreibungsmittel hat sich der Hinweis auf den schematischen Beschreibungsinhalt außer bei NIAM-Informationsstruktur-Diagrammen nicht durchgesetzt.

Aufgabensicht: (Aufgaben)

- ***Aufgabengliederungsparadigma:***
  Untergliederung von Aufgaben in Teilaufgaben

Aufbausicht: (Organisationseinheiten)

- ***Stellengliederungsparadigma:***
  Untergliederung von Organisationseinheiten
- ***Kommunikationsparadigma:***
  Interaktionsstrukturen zwischen Organisationseinheiten

Prozeßsicht: (Prozesse)

- ***Datenflußparadigma***
  Prozeßdarstellungen durch Datenflüsse
- ***Zustandsübergangspardigma***
  Prozeßdarstellungen durch Zustandsveränderungen
- ***Netzparadigma:***
  Prozeßdarstellungen durch netzartige Folgestrukturen
- ***Kontrollflußparadigma***
  Prozeßdarstellungen durch reguläre Folgestrukturen

Objektsicht: (Objekte)

- ***Objekt-Instanzparadigma***
  Zusammenhänge zwischen Objektinstanzen
- ***Objekt-Interaktionsparadigma***
  Interaktionsbeziehungen zwischen Objektinstanzen
- ***Objekt-Beziehungsparadigma***
  Zusammenhänge zwischen Objekt- und Beziehungsstrukturen

Abbildung 3.2: Beschreibungssichten und Paradigmen

den Vordergrund. In der Unified Modeling Language [Booch et al., 1999] werden die Sprachen des Objekt-Interaktionsparadigmas als Mittel zur Beschreibung der Systemdynamik eingeführt. Dieser Einordung liegt die Anschauung zugrunde, daß durch diese Beschreibungsmittel in erster Linie der Kontrollfluß des Nachrichtenaustauschs zwischen Objekten beschrieben wird.

Die Festlegung der Paradigmen ist, wie auch die Definition der verwendeten Sichten, einer gewissen „Willkür“ unterworfen, die jedoch aus dem individuellen Standpunkt des Modellierers heraus begründbar ist.

3.1.4 Visuelle Modellierungssprachen

Anstatt alle unterschiedlichen visuellen Modellierungssprachen der Organisationsbeschreibung und des Softwareentwurfs einzeln zu beschreiben (vgl. z. B. [Grupp, 1990]) erlaubt das Klassifikationsschema aus Abbildung 3.2 eine kompaktere Betrachtung entlang der jeweils abgebildeten Konzepte. Abbildung 3.3 ordnet diese Beschreibungsmittel den einzelnen Darstellungsparadigmen zu.

Gleiche Beschreibungsmittel werden häufig unterschiedlich bezeichnet. Aufgabengliederungspläne werden auch als Funktionsbäume bezeichnet, und Struktogramme sind auch als Nassi-Shneiderman-Diagramme bekannt. Manche Beschreibungsmittel können (notationell) als *Erweiterungen* anderer Sprachen aufgefaßt werden. So sind Statecharts allgemeinere Zustandsübergangsdiagramme, die lediglich kompakter notiert werden. Ähnlich verhält es sich mit Objekt-Beziehungsdiagrammen, die zu erweiterten Objekt-Beziehungsdiagrammen und Klassendiagrammen weiter entwickelt wurden. Andere Sprachen können als (notationelle) *Einschränkungen* einer Grundform betrachtet werden. Datenlexika entsprechen konzeptionell den Beschreibungen durch erweiterte Objekt-Beziehungsdiagramme, die auf die Beschreibung durch Attributierungen, Generalisierungen und Aggregationen eingeschränkt sind. Darstellungen durch Ablaufkarten oder Programmablaufpläne entsprechen in ähnlicher Weise Stimulus-Response-Darstellungen verkürzt um die Stimuli. Auch haben unterschiedliche Modellierungssprachen gelegentlich denselben Namen. Unter den Begriff „Objektdiagramm“ werden bei [Rumbaugh et al., 1991, S. 23] sowohl Darstellungen auf der Instanzebene (Instanzdiagramme) als auch auf Schemaebene (Klassendiagramme) gefaßt. [Booch, 1994, S. 208ff] verwendet „Objektdiagramme“ zur Beschreibung der Interaktion zwischen Objekten. In [Booch et al., 1999] beziehen sich Objektdiagramme, wie auch in dieser Arbeit, ausschließlich auf die Darstellung des statischen Zusammenhangs von Instanzen.

Nicht alle visuellen Modellierungssprachen lassen sich eindeutig direkt einer Beschreibungssicht oder einem Paradigma zuordnen. So gibt es auch *multiparadigmatische Sprachen*. Vorgangskettendiagramme und erweiterte Ereignisgesteuerte Prozeßketten beschreiben Vorgangsfolgen sowohl aus Prozeßsicht, aus Objektsicht und aus Aufbausicht. Mit diesen Diagrammen werden Organisationen aus diesen Sichten in „*zusammenhängender Form*“ [Scheer, 1992, S. 57] abgebildet. Zentrales Element sind jedoch alternierende Folgen von Ereignissen und Vorgängen, die jeweils um Elemente aus der Objekt- und der Aufbausicht ergänzt werden. Vorgangskettendiagramme werden daher als Beschreibungsmittel des Netzparadigmas der Prozeßsicht eingestuft. Ähnlich verhält es sich mit Funktionendiagrammen. Diese stellen die Querbezüge zwischen Aufgaben und Organisationseinheiten einschließlich der jeweiligen Gliederungen dar. Funktionendiagramme erlauben somit eine übergreifende Darstellung aus Aufgaben- und Aufbausicht. Für die weiteren Betrachtungen wird diese Notationsform dem Stellengliederungsparadigma zugeordnet.

Aufgabensicht	
Paradigma	Beschreibungsmittel
Aufgabengliederungsparadigma	Aufgabengliederungspläne, Aufgabengliederungsschaubilder, Aufgabenlisten, Aufgabenstrukturbilder, Aufgabenstrukturbäume, Funktionsbäume, Funktionshierarchiediagramme, Funktionsstrukturdiagramme
Aufbausicht	
Paradigma	Beschreibungsmittel
Stellengliederungsparadigma	Abteilungsgliederungspläne, Funktionendiagramme, Organigramme, Organisationspläne, Organisationsschaubilder, Stellenbeschreibungen, Stellengliederungspläne, Stellenpläne, Strukturpläne
Kommunikationsparadigma	Kommunigramme, Kommunikationsdiagramme, Kommunikationsgraphen, Kommunikationstabellen, Kooperationsbilder
Prozeßsicht	
Paradigma	Beschreibungsmittel
Datenflußparadigma	Anwendungsfalldiagramme, Bonapart Prozeßmodelle, Bubble-Charts, Datenflußdiagramme, Datenflußpläne, Funktionszuordnungsdiagramme, IDEF0-Diagramme, Kontextdiagramme, SADT-Aktivitätsdiagramme, SADT-Datendiagramme, Verkehrsschaubilder
Zustandsübergangsparadigma	(hierarchische) Automaten, IDEF3-Zustandsübergangsdiagramme, Statecharts, Zustandsautomaten, Zustands-(übergangs)-Diagramme, Zustands-(übergangs)-Matrizen, Zustandsübergangstabellen
Netzparadigma	Ablaufkarten, Aktivitätsdiagramme, Aufgabenfolgepläne, Balkendiagramme, (erweiterte) Ereignisgesteuerte Prozeßketten, Folgestrukturen, Folgepläne, IDEF3-Prozeßflußdiagramme, Netzgraphen, Netzpläne, Petri-Netze, Programmablaufpläne, Stimulus-Response-Folgen, Vorgangskettendiagramme
Kontrollflußparadigma	Entscheidungstabellen, Jackson-Diagramme, Nassi-Shneiderman-Diagramme, Pseudo-Code, Struktogramme, Warnier-Orr-Diagramme
Objektsicht	
Paradigma	Beschreibungsmittel
Objekt-Instanzparadigma	Instanzdiagramme, Objektdiagramme
Objekt-Interaktionsparadigma	Ereignisflußdiagramme, Ereignispfaddiagramme, Interaktionsdiagramme, Kollaborationsdiagramme, Message-Sequence-Charts, Objektdiagramme, Sequenzdiagramme
Objekt-Beziehungsparadigma	Datenlexika, (erweiterte) Entity-Relationship-Diagramme, (erweiterte) Objekt-Beziehungs-Diagramme, IDEF1X-Datenmodelle, Jackson-Diagramme, Klassendiagramme, NIAM-Informationsstruktur-Diagramme

Abbildung 3.3: Sichten, Paradigmen und Beschreibungsmittel

In den Kapiteln 3.2 bis 3.5 werden die wesentlichen visuellen Modellierungssprachen der Aufgabensicht, der Aufbausicht, der Prozeßsicht und der Objektsicht entlang der Paradigmen vorgestellt. Ausgehend von einer heute üblichen Notation werden verschiedene notationelle Varianten der einzelnen Darstellungsparadigmen diskutiert und hieraus Forderungen an das Referenz-Metaschema abgeleitet.

3.2 Visuelle Modellierungsprachen der Aufgabensicht

Untersuchungsgegenstand der Aufgabensicht sind die *Aufgaben* (oder *Funktionen*), die durch Organisationen zu bearbeiten bzw. durch Softwaresysteme zu unterstützen sind. Aufgaben bezeichnen klar umrissene, zielorientierte Handlungsweisen innerhalb eines größeren Zusammenhangs [Balzert, 1996a, S. 116], die durch eine *Verrichtung* an einem *Objekt* charakterisiert sind [Nordsieck, 1962, S. 13], [Kosiol, 1976, S. 43]. Hierbei beschreibt die Verrichtung den Vorgang oder Prozeß, der zur Aufgabenerledigung durchzuführen ist. Das Objekt bezeichnet die Klasse der Gegenstände, an denen die Verrichtung durchgeführt wird.

Zur Beschreibung von Aufgaben sind folglich deren Verrichtungs- und Objektkomponenten darzustellen. Eine Aufgabenbeschreibung kann optional um weitere Aufgabenaspekte ergänzt werden (vgl. [Kosiol, 1976, S. 43] und [Hub / Fischer, 1977, S. 15]). Der Aspekt *Raum* gibt an, *wo* die Aufgabenerledigung stattfindt. Zeitliche Aspekte werden unterschieden in den *Zeitpunkt*, zu dem die Aufgabenerfüllung stattfindet, und die *Dauer*, die zur Umsetzung benötigt wird. Der *Umfang* einer Aufgabe wird durch die Anzahl der zu bearbeitenden Objekte bestimmt. Sach- und Arbeitsmittel, die zur Aufgabenerledigung benötigt oder eingesetzt werden, werden unter dem Aspekt *Hilfsmittel* zusammengefaßt.

Zu beschreibende Aufgaben können beliebig komplex werden. Zur Bewältigung dieser Komplexität ist es üblich, diese Aufgaben in Teilaufgaben mit geringerer Komplexität zu zerlegen. Die visuellen Modellierungssprachen des *Aufgabengliederungsparadigmas* dokumentieren diese Zerlegungen in Teilaufgaben.

3.2.1 Visuelle Modellierungsprachen des Aufgabengliederungsparadigmas

Eine Aufgabengliederung beschreibt die Zerlegung einer Aufgabe in Teilaufgaben. Diese Zerlegungen werden so lange wiederholt, bis Elementaraufgaben ermittelt sind, die nicht weiter untergliedert sind. Wird die Aufgabengliederung im Rahmen einer Aufgabenanalyse zur Vorbereitung der Stellenbildung bzw. der Prozeßgestaltung durchgeführt, sind diese Elementaraufgaben dann erreicht, wenn sie einem einzigen Aufgabenträger zugeordnet werden können [Hub / Fischer, 1977, S. 18] bzw. elementare Handlungen beschreiben. Dient die Aufgabengliederung zur Zerlegung eines softwaretechnischen Problems in unabhängige Module, bricht die Gliederung bei solchen Teilaufgaben ab, die durch einzelne Methoden oder Funktionen implementiert werden können.

Notiert werden diese Aufgabenzerlegungen durch Aufgabengliederungspläne, Aufgabengliederungsschaubilder, Aufgabenlisten, Aufgabenstrukturbilder und durch Funktionsbäume.

Notationelle Grundform: Aufgabengliederungsplan

Zur Darstellung von Aufgabengliederungen werden baumartige Diagrammformen verwendet, in denen die einzelnen Aufgaben als Knoten dargestellt sind. Die Verbindungen zwischen diesen Knoten beschreiben die Zerlegungsbeziehungen, die i. allg. von oben nach unten oder von links nach rechts gelesen werden. Um die Objekt- und Verrichtungskomponenten einer Aufgabe zu verdeutlichen, sollten die Aufgaben durch Verbalphrasen bezeichnet werden. Beispiel 3.4 skizziert eine solche Zerlegung der Aufgaben der Radiologie eines Krankenhauses in einem Aufgabengliederungsplan (vgl. [Schumm et al., 1995]).

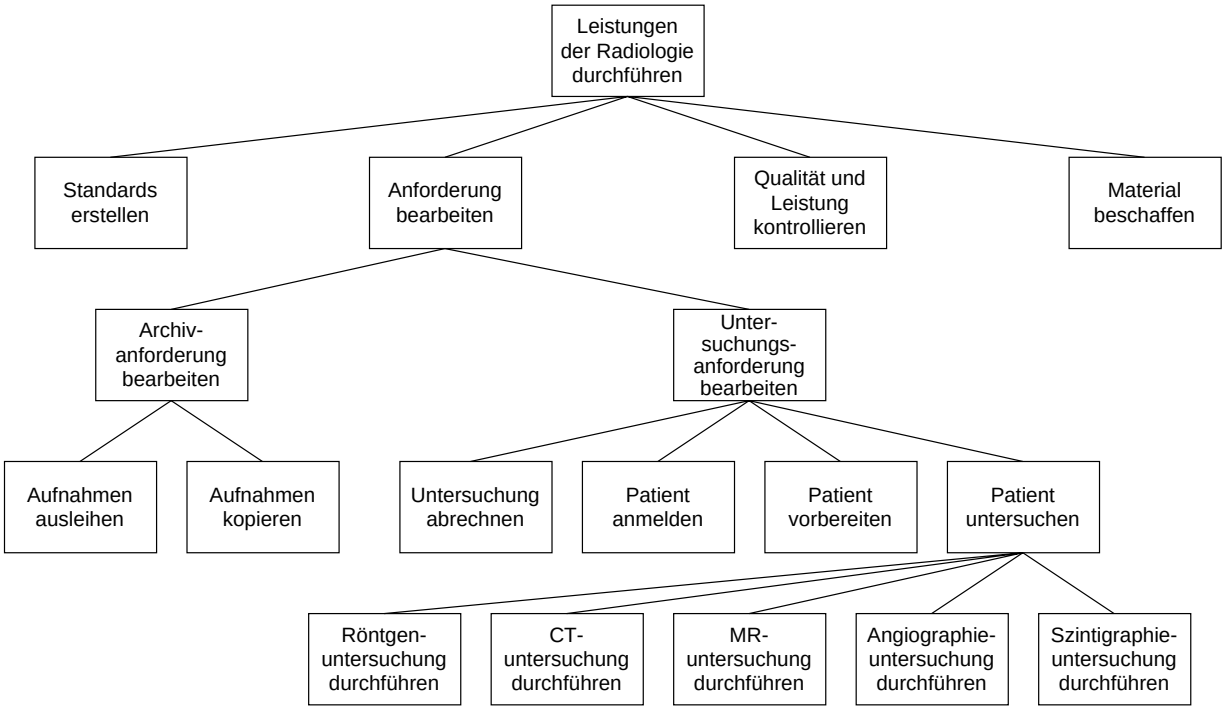


Abbildung 3.4: Aufgabengliederungsplan

Die Gliederung von Aufgaben in Teilaufgaben kann entlang mehrerer Grundmuster (vgl. [Nordsieck, 1962, S. 14], [Kosiol, 1976, S. 49ff], [Hub / Fischer, 1977, S. 19ff]) erfolgen. Ein wesentliches Gliederungskriterium bieten die Aufgabenkomponenten Verrichtung und Objekt. Bei der *Gliederung nach Verrichtungen* werden einzelne Tätigkeiten unterschieden, die zur Erledigung der Gesamtaufgabe nötig sind. Analog erfolgt die *Gliederung nach Objekten* durch Konkretisieren der (Teil-) Objekte, die zur Aufgabenerfüllung bearbeitet werden müssen. Unabhängig von der betrachteten Komponente kann eine Aufgabe in solche Teilaufgaben zerlegt werden, die alle zur Aufgabenerledigung zu erfüllen sind (*Und-Gliederung*) oder die Alternativen in der Aufgabenerledigung darstellen (*Oder-Gliederungen*).

[Kosiol, 1976, S. 52ff] schlägt als weitere Gliederungsmerkmale noch den Rang, die Phase und den Zweck vor. Hieraus resultierende Aufgabengliederungen können als Spezialfälle der Und-Verrichtungsgliederung aufgefaßt werden, bei denen die Verrichtungen Leiten, Planen, Kontrollieren, Verwalten und Ausführen unterschieden werden. Gliederungen nach *Rang* zerlegen Aufgaben in Teilaufgaben, die die Koordination der Gesamtaufgabe (Leitungsaufgabe) von der eigentlichen Ausführung (Ausführungsaufgabe) unterscheiden. Das Kriterium *Phase* gliedert Auf-

gaben in Planungs-, Ausführungs- und Kontrollaufgaben. Planungsaufgaben umfassen hierbei die Aufgabenerfüllung vorbereitende und Kontrollaufgaben nachbereitende Tätigkeiten. Nach dem Kriterium des *Zwecks* werden Aufgaben in primäre Aufgaben (Ausführungsaufgaben) und in sekundäre Aufgaben (Verwaltungsaufgaben) zur Unterstützung der Ausführungsaufgaben unterschieden (vgl. auch [Nordsieck, 1962, S. 14f], Ausgliederung der Verwaltungsaufgaben).

In der Praxis treten Aufgabengliederungen auch in Mischformen auf. Die zuvor skizzierten Gliederungskriterien liefern aber ein geeignetes Raster zur Durchführung von Aufgabengliederungen. Zur Erstellung einer Aufgabengliederung ist für jede Aufgabe zu entscheiden, nach welchem Kriterium sie geeignet zerlegt wird. Sind die zur Erfüllung einer Aufgabe nötigen Verrichtungen stärker voneinander abhängig als die zu bearbeitenden Objekte, sollte die Gliederung entlang der Verrichtung erfolgen, andernfalls entlang des Objekts [Nordsieck, 1962, S. 14]. Als grundsätzliches Vorgehen zur Aufgabengliederung schlägt [Krüger, 1981] vor, zunächst zu überprüfen, ob durch die Aufgabe unterschiedliche, selbständige Objekte alternativ zueinander bearbeitet werden. In diesem Fall bietet sich eine Oder-Objekt-Gliederung an. Besteht das Objekt aus mehreren unterschiedlichen Teilobjekten, die getrennt bearbeitet werden können, sollte die Aufgabe durch eine Und-Objekt-Gliederung strukturiert werden. Gibt es verschiedene Verfahren zur Bearbeitung des Objekts, bietet sich die Oder-Verrichtungs-Gliederung an. Andernfalls sollte eine Und-Verrichtungsgliederung vorgenommen werden.

Beispiel 3.2 (Gliederung der Aufgaben der Radiologie)

Die Gliederung der Aufgaben der Radiologie aus Abbildung 3.4 erfolgte entlang der zuvor skizzierten Kriterien. Die Aufgabe der Radiologie (*Leistungen der Radiologie durchführen*) wird zunächst entlang der Phase in die Planungsaufgabe *Standards erstellen*, die Ausführungsaufgabe *Anforderung bearbeiten* und die Kontrollaufgabe *Qualität und Leistung kontrollieren* unterteilt. Ergänzt wird diese Gliederung um die Verwaltungsaufgabe *Material beschaffen*. Diese Teilaufgaben stellen eine Und-Verrichtungs-Gliederung der Hauptaufgabe dar.

Anforderungen an die Radiologie können sich sowohl auf die Aufnahmen aus dem Archiv als auch auf die durchzuführende Untersuchungen beziehen. Die Bearbeitung der Anforderungen wird daher entlang der Objekte Archivanforderung und Untersuchungsanforderung in die Teilaufgaben *Archivanforderung bearbeiten* und *Untersuchungsanforderung bearbeiten* oder-gegliedert. Archivanforderungen können unterschieden werden in die Verrichtungen Ausleihen oder Kopieren. Hiernach ergeben sich die oder-gegliederten Teilaufgaben *Aufnahmen ausleihen* und *Aufnahmen kopieren*.

Nach dem Zweck wird die Bearbeitung einer Untersuchungsanforderung zunächst in die Verwaltungsaufgabe *Untersuchung abrechnen* und die Ausführungsaufgaben *Patient anmelden, Patient vorbereiten* und *Patient untersuchen* zerlegt. Die Gliederung der Ausführungsaufgaben erfolgt hierbei nach den Verrichtungen Anmelden, Vorbereiten und Untersuchen. Die weitere Verfeinerung der Untersuchung erfolgt entlang der benötigten Hilfsmittel. In Radiologischen Abteilungen werden hierzu u. a. Röntgenaufnahmen, Computer-Tomographen-Bilder, Magnet-Resonanz-Bilder, Angiografie-Bilder oder Szintigramme verwendet. Eine Oder-Verrichtungs-Gliederung nach diesen Hilfsmitteln ergibt die Teilaufgaben *Röntgenuntersuchung durchführen, CT-Untersuchung durchführen, MR-Untersuchung durchführen, Angiographieuntersuchung durchführen* und *Szintigraphieuntersuchung durchführen*. □

Notationelle Varianten

Zur Darstellung solcher Aufgabenzerlegungen gibt es weitere Notationsformen, die in ihrer Grundform jedoch alle der baumartigen Darstellung aus Abbildung 3.4 entsprechen. Abbildung 3.5 skizzierte einige dieser notationellen Varianten anhand der Aufgabe *Untersuchungsanforderung bearbeiten.*

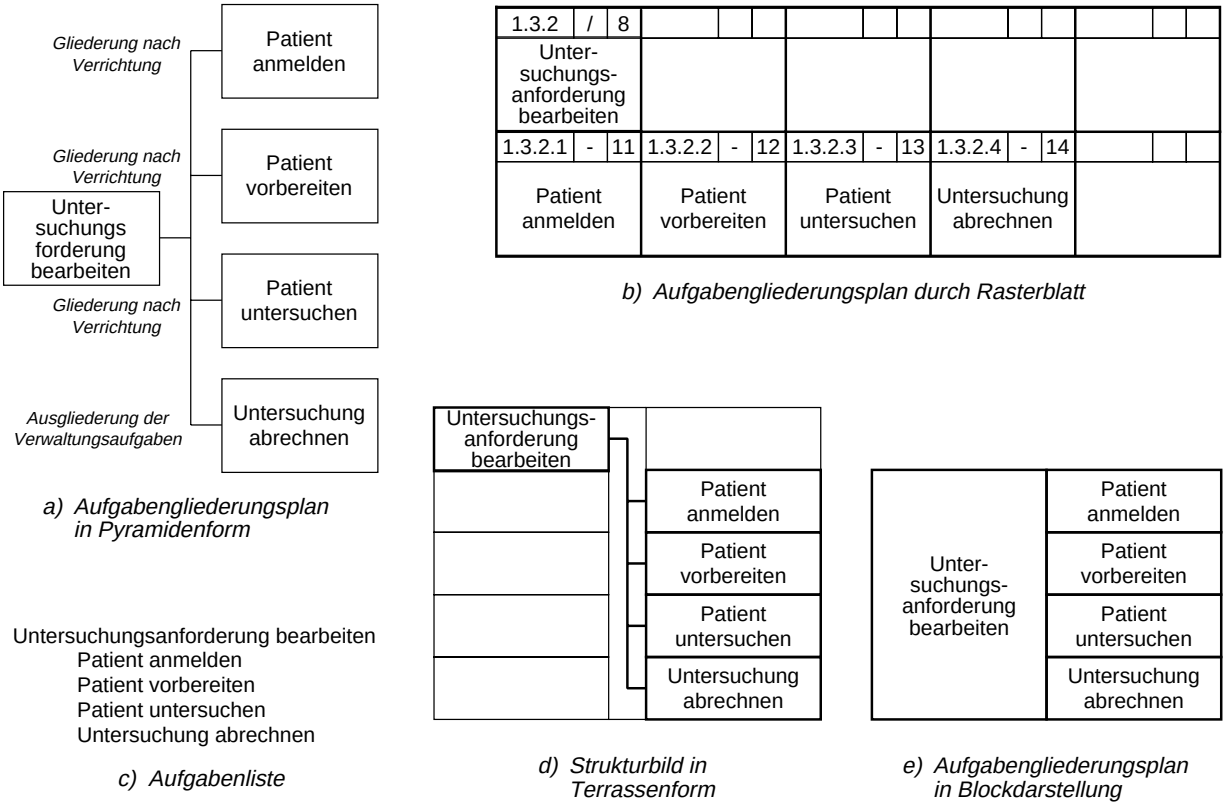


Abbildung 3.5: Aufgabengliederungspläne, notationelle Varianten

Die baumartige Gliederung der Aufgaben erfolgt häufig auch horizontal von links nach rechts. Hierbei werden *Pyramidenformen*, bei der die gegliederte Aufgabe zentriert zu den Teilaufgaben dargestellt wird (vgl. Abbildung 3.5a) und *Terrassenformen*, bei denen die Teilaufgaben unterhalb der Aufgabe notiert sind (vgl. Abbildung 3.5d), unterschieden. Eine Aufgabengliederung mit Hilfe eines *Rasterblatts* zeigt Abbildung 3.5b. Die Teilaufgaben zu einer Aufgabe, die sowohl durch die Gliederung widerspiegelnde Nummer (links) als auch durch eine laufende Aufgabennummer (rechts) nummeriert sind, werden hierbei in einer Zeile des Formulars aufgeführt. Rasterblätter werden in erster Linie als Hilfsmittel bei der Erhebung von Aufgabengliederungen eingesetzt [REFA, 1992, S. 88f]. Aufgaben, die während der Erstellung bereits untergliedert wurden, werden durch „/“ und nicht weiter zerlegte Aufgaben durch „-“ markiert. Nach der Erhebung werden die Rasterblätter in *Strukturbilder* (Abbildung 3.5d) übertragen. Eher listenartige Darstellungen bieten *Aufgabenlisten* (Abbildung 3.5c) und *Blockdarstellungen* (Abbildung 3.5e).

In einigen Notationen wird bei der Bezeichnung der Aufgaben auf die Angabe der Verrichtungs- oder Objektkomponente verzichtet. Bei Verrichtungsgliederungen werden häufig nur die Verrichtungen und bei Objektgliederungen nur die Objekte genannt (vgl. z. B. [Kosiol, 1976, S. 50ff],

[Schmidt, 1989, S.168ff]). Das jeweils verwendete Gliederungskriterium wird in einigen Dialekten der Aufgabengliederungsplänen (z. B. [Nordsieck, 1962, S. 16], [Kosiol, 1976, S. 50ff]) ebenfalls dargestellt (vgl. auch Abbildung 3.5a).

Forderungen an das Referenz-Metaschema

Bei Beschreibungen nach dem Aufgabengliederungsparadigma steht also die Darstellung der Aufgaben und ihrer Untergliederung, evtl. nach unterschiedlichen Aufgabentypen, im Mittelpunkt. Das Referenz-Metaschema für die Sprachen des Aufgabengliederungsparadigmas sollte somit Konzepte zur Modellierung von Aufgaben mit ihren Verrichtungs- und Objektkomponenten, von Aufgabengliederungen und von Gliederungskriterien bereitstellen.

3.3 Visuelle Modellierungssprachen der Aufbausicht

Aus der Aufbausicht werden die Organisationseinheiten beschrieben, die mit der Umsetzung der Aufgaben betraut sind. *Organisationseinheiten* fassen einen oder mehrere Aufgabenträger zusammen, die bestimmte Aufgaben zu erfüllen haben und hierzu mit den nötigen Kompetenzen ausgestattet sind. Organisationseinheiten werden in Stellen, Abteilungen und Stellenmehrheiten unterschieden. *Stellen* bezeichnen hierbei von konkreten Personen unabhängige Zusammenfassungen von Aufgaben, die von einer Person bewältigt werden können. *Abteilungen* sind Zusammenfassungen von Stellen, in denen einer Stelle besondere Leitungs- und Koordinationsaufgaben bezogen auf die restlichen Stellen übertragen werden. Zusammenfassungen gleichrangiger Stellen ohne Leitungsstelle werden als *Stellenmehrheit* bezeichnet. (zur Diskussion und Einordnung des Stellenbegriffs vergleiche auch [Kosiol, 1976, S.89], [Schwarz, 1980])

Zur Konkretisierung von Stellen (vgl. auch [Grochla, 1982, S. 329], [Schmidt, 1989, S. 268ff], [Lehner et al., 1991, S. 264f]) sind die Stellenaufgaben und die hierzu gewährten Kompetenzen und Befugnisse festzulegen. Hinzu kommt die Einbettung der Stelle in die Aufbauorganisation durch Angabe des Stellenrangs (z. B. Gruppenleiter oder Abteilungsleiter) und der Festlegung der fachlichen und disziplinarischen Über- und Unterstellungsverhältnisse sowie der aktiven und passiven Vertretungsverhältnisse. Die Zusammenarbeit mit anderen Organisationseinheiten ist ebenfalls festzulegen. Dieses umfaßt u. a. auch die Kommunikations- und Informationsbeziehungen zu anderen Stellen und die Mitwirkung in Ausschüssen oder Arbeitskreisen. Darüber hinaus sind auch die Anforderungen an den Stelleninhaber festzulegen.

Je nach Leitungsbefugnissen werden Stellen in Ausführungsstellen, Leitungsstellen und in Stabsstellen unterschieden. *Ausführungsstellen* haben keine Leitungsbefugnisse, *Leitungsstellen* oder *Instanzen* sind mit Leitungs- und Weisungsbefugnissen ausgestattet. Stellen, die Leitungsstellen beraten und unterstützen, aber keine direkten Weisungsbefugnisse gegenüber den Ausführungsstellen besitzen, werden als *Stabstellen* bezeichnet.

Die visuellen Modellierungsprachen des *Stellengliederungsparadigmas* heben hierbei die Einbettung der Stellen in die Aufbauorganisation hervor. Diese bezieht sich insbesondere auf die Über- und Unterstellungsverhältnisse und die Bildung von Abteilungen. Kommunikations- und Informationsbeziehungen zwischen Stellen und Abteilungen werden mit den Darstellungsmitteln des *Kommunikationsparadigmas* beschrieben.

3.3.1 Visuelle Modellierungsprachen des Stellengliederungsparadigmas

Die Einbettung der Stellen in die Aufbauorganisation erfolgt durch Festlegung der Leitungsbeziehungen zwischen den Stellen. Unterschieden werden hierbei *Einlinien-Systeme*, bei denen jeder Stelle höchstens eine Instanz vorgesetzt ist, und *Mehrlinien-Systeme*, in denen einer Stelle auch mehrere Leitungsstellen übergeordnet sein können. Das Einlinien-System beruht auf dem von [Fayol, 1919, S. 21] postulierten Prinzip der „*Einheit der Auftragserteilung*“, durch das Leitungs- und Weisungsbefugnisse eindeutig festgelegt werden. Das Mehrlinien-System geht auf das Funktionsmeistersystem von [Taylor, 1919, S. 132f] zurück, in dem für spezielle Teilbereiche der Aufgabenerledigung unterschiedliche Leitungsinstanzen (Spezial- oder Funktionsmeister) vorgesehen sind.

Zur Beschreibung der Weisungs- und Leitungsbeziehungen zwischen Stellen werden Organigramme, Abteilungsgliederungspläne, Organisationspläne, Organisationsschaubilder, Stellenbeschreibungen, Stellengliederungspläne, Stellenpläne und Strukturpläne verwendet.

Notationelle Grundform: Organigramm

Zur Darstellung von Einlinien-Systemen werden ähnlich wie für Aufgabengliederungspläne baumartige Diagrammformen verwendet. Die Knoten symbolisieren hierbei die Stellen und die Kanten die Leitungsbeziehungen, die ebenfalls von oben nach unten bzw. von links nach rechts gelesen werden.

Die Darstellung der Stellen erfolgt durch unterschiedliche graphische Formen für Instanzen, Stabsstellen und Ausführungsstellen. [Kosiol, 1976, S. 173ff] notiert Leitungsstellen durch Rechtecke, Ausführungsstellen durch Dreiecke und Stabsstellen durch Ovale. Stellenmehrheiten werden für alle drei Stellentypen durch Doppellinien (und optionaler Angabe der Stellenanzahl) ausgezeichnet. In der Praxis hat sich das Dreieck zur Darstellung der Ausführungsaufgaben aufgrund der begrenzten Annotationsfähigkeit nicht durchgesetzt [Blum, 1980a]. Ausführungsstellen werden daher ebenfalls durch Rechtecke notiert.

Beispiel 3.3 (Organigramm eines Krankenhauses)

Abbildung 3.6 stellt einen Ausschnitt eines Organigramms zur Beschreibung der Aufbauorganisation eines Krankenhauses [Schumm et al., 1995] dar. Die Gesamtleitung des Krankenhauses erfolgt durch den Verwaltungsrat, der hier als Stellenmehrheit aus fünf Stellen besteht. Diesem sind der *ärztliche Direktor*, der *Pflegedienstleiter* und der *Verwaltungsdirektor* unterstellt.

Der *ärztliche Direktor* koordiniert, unterstützt durch eine mehrfach besetzte Stabsstelle *Sekretärin*, die medizinischen Bereiche, denen jeweils *Chefärzte* vorstehen. Diese Bereiche bestehen aus mehreren Ärzten, wobei die *Oberärzte* gegenüber den *Ärzten* Leitungsfunktionen übernehmen. Den *Chefärzten* sind weiter *Abteilungsleiter* diverser Funktionen mit ihren *Mitarbeitern* unterstellt. Der *Pflegedienstleiter* leitet den pflegerischen Betrieb. Er wird ebenfalls durch eine mehrfach besetzte Stabsstelle *Sekretärin* unterstützt. Dem *Pflegedienstleiter* unterstehen mehrere Abteilungen, die durch *Abteilungsleiter* geleitet werden. Den *Abteilungsleitern* sind jeweils *Stationsleitungen* und diesen wiederum *Pflegekräfte* in einer Stellenmehrheit unterstellt. In ähnlicher Form ist auch der Verwaltungsbereich

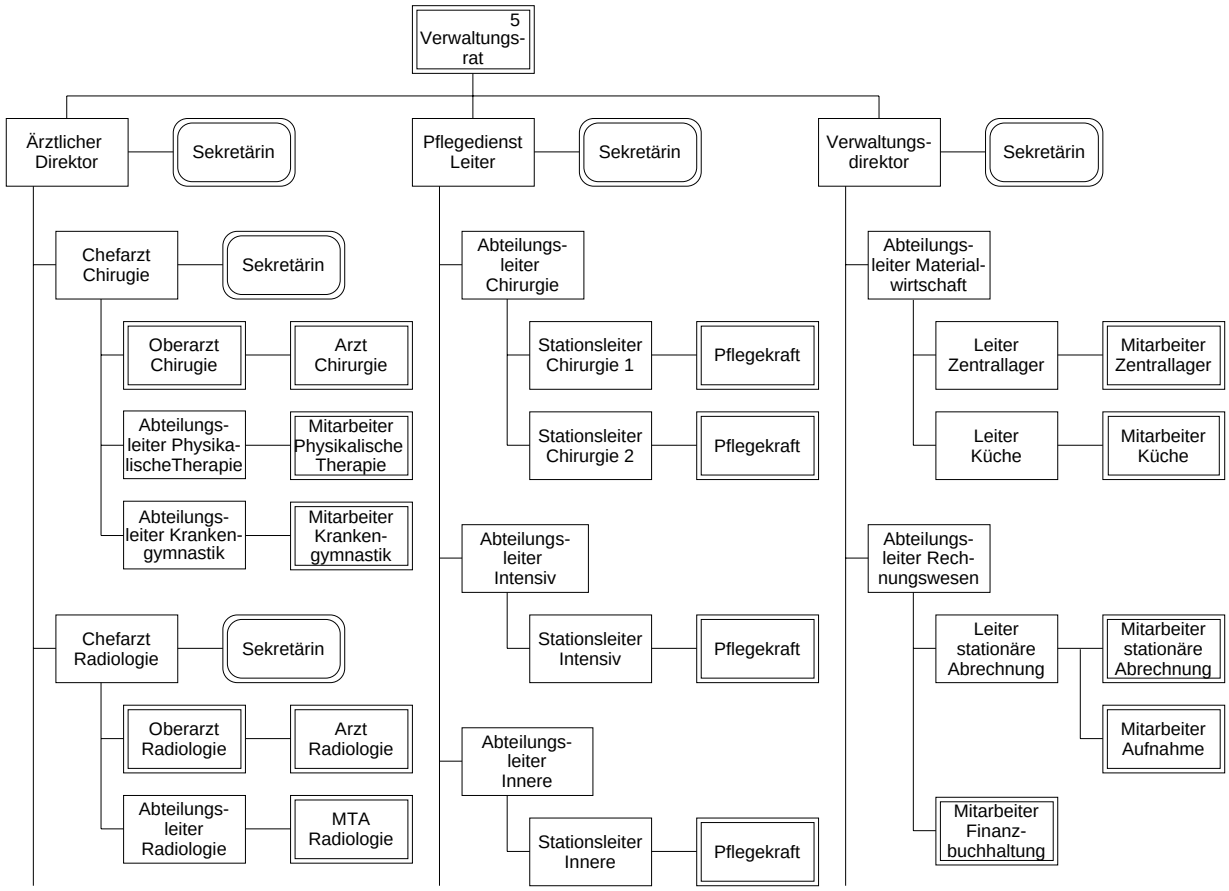


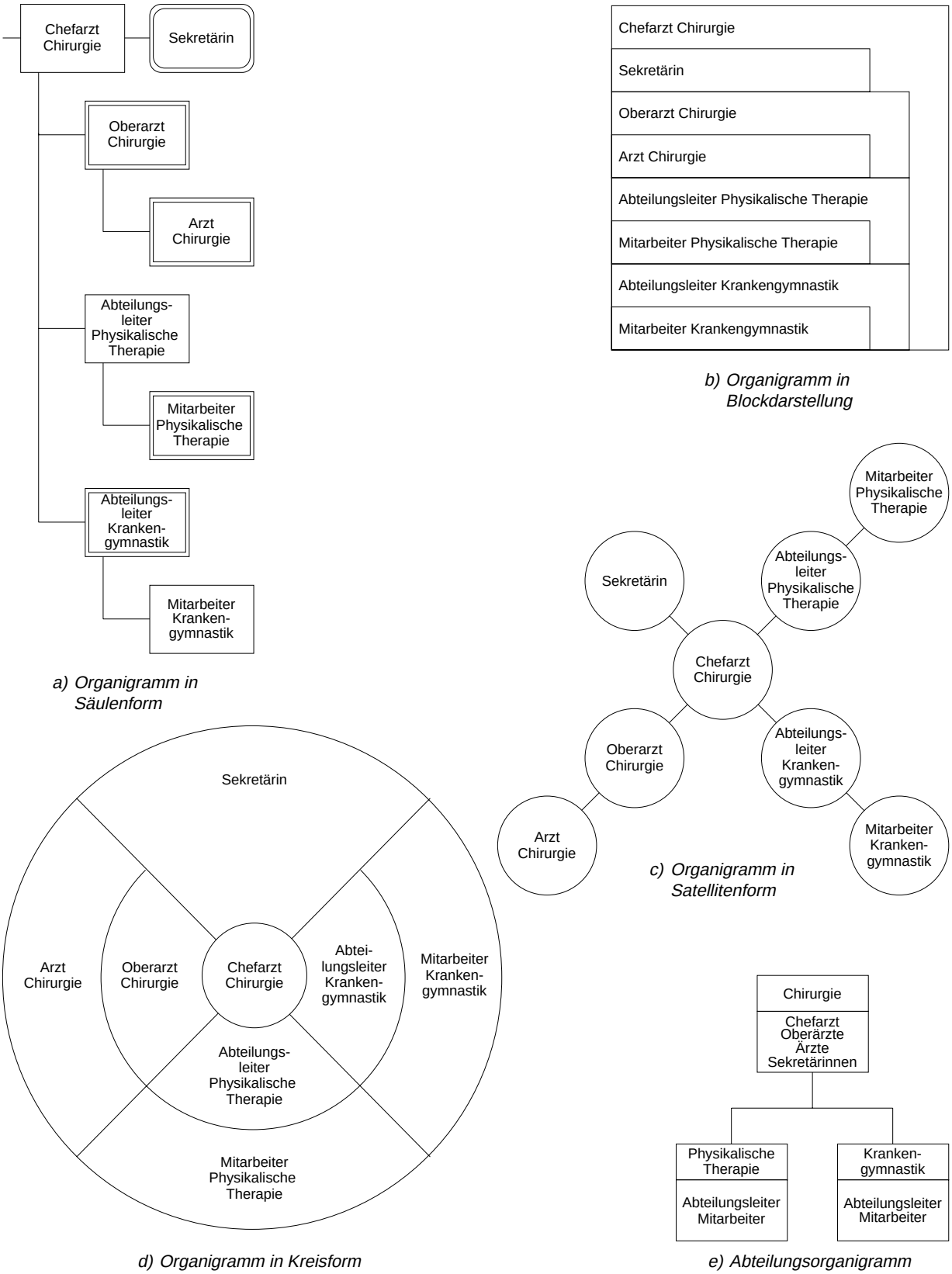
Abbildung 3.6: Organigramm

strukturiert. Dem *Verwaltungsdirektor* unterstehen die durch *Abteilungsleiter* koordinierten Verwaltungseinheiten. Auch der *Verwaltungsdirektor* wird durch *Sekretärinnen* unterstützt. Den *Abteilungsleitern* sind weitere *Leiter* und *Mitarbeiter* unterstellt. □

Notationelle Varianten

Da die Darstellung von Einlinien-Systemen durch jede baumartige Notationsform erfolgen kann, finden auch hier die schon für Aufgabengliederungspläne eingeführten Darstellungsvarianten Verwendung (vgl. Abbildung 3.5). Organigramme werden häufig in Pyramiden- oder Terrassendarstellungen notiert. Diese werden sowohl horizontal als auch vertikal ausgerichtet. Daneben werden auch Blockdarstellungen evtl. unter Verwendung von Rasterblättern oder listenartige Darstellungen verwendet.

Weitere Darstellungsvarianten für Einlinien-Systeme sind in Abbildung 3.7 am Beispiel des Teilorganigramms der Chirurgie aus Abbildung 3.6 skizziert. Organigramme in *Säulenformen* entsprechen für die ersten Gliederungsebenen den Organigrammen in Pyramidenform. Meist ab der zweiten oder dritten Stufe werden die Stellen unterhalb ihrer überstellten Instanz angeordnet. Es entstehen dann nebeneinander liegende Spalten oder Säulen für klar identifizierbare Teilbereiche der Aufbauorganisation. Abbildung 3.7a beschreibt die Chirurgie durch eine solche Säule.



a) Organigramm in Säulenform

b) Organigramm in Blockdarstellung

c) Organigramm in Satellitenform

d) Organigramm in Kreisform

e) Abteilungsorganigramm

Abbildung 3.7: Organigramme für Einliniensysteme, notationelle Varianten

Eine *listenartige Form der Blockdarstellung* wird in Abbildung 3.7b verwendet. Jede Stelle wird hierbei linksbündig in einer eigenen Zeile vermerkt, wobei die Zeilen für untergeordnete Stellen immer kürzer werden. Gegenüber der reinen Blockform (vgl. Abbildung 3.5e) erlaubt diese Notationsform anhand der sich rechts ergebenden leeren Spalten einen guten Überblick über alle, einer Instanz auch indirekt unterstellten Stellen. Diese Darstellungsform tritt auch in Spaltenformen auf. Jede Stelle wird hierbei in einer Spalte dargestellt. Werden die nachgeordneten Stellen kreisförmig um die Leitungsstelle angeordnet, erhält man ein *Satelliten-* oder *Sonnenorganigramm* (vgl. Abbildung 3.7c). Anstelle rechteckiger Blöcke verwenden Organigramme in *Kreisform* Kreissegmente (vgl. Abbildung 3.7d).

Aus Platzgründen bietet es sich bei Organigrammen an, Mischformen zu verwenden. So wird auch in Abbildung 3.6 zunächst eine vertikale Pyramidenform verwendet, die in eine säulenartige Darstellung mit „Säulen“ für die Medizin, die Pflege und die Verwaltung übergeht. Innerhalb der Säulen erfolgt die Darstellung durch horizontale Terrassendarstellungen. Überblicksdarstellungen und Bewertungen der einzelnen Organigrammformen bezüglich ihrer Ausdrucksmächtigkeit und Übersichtlichkeit finden sich z. B. in [Hub / Fischer, 1977] und in [Blum, 1980a, Blum, 1980b].

Abteilungsorganigramme. In den bisher beschriebenen Notationsformen von Organigrammen in Abbildung 3.6 und 3.7a-d werden Leitungsbeziehungen zwischen Stellen beschrieben. Häufig wird durch Organigramme aber auch die Gliederung von Abteilungen in Unterabteilungen herausgestellt (vgl. z. B. [Nordsieck, 1962, Abb. 81], [Kosiol, 1976, S. 175], [Jordt / Gscheidle, (o.J.), Lehrbrief 3]). Abbildung 3.7e stellt ein solches *Abteilungsorganigramm* für die *Chirurgie* dar, die sich in die Unterabteilungen *physikalische Therapie* und *Krankengymnastik* gliedert. Hier sind neben den Abteilungsnamen auch noch die in der Abteilung zusammengefaßten Stellen einschließlich der Instanz (in Fettdruck) aufgeführt. Beziehungen zwischen den Stellen werden in diesem Beispiel nicht abgebildet. Varianten der Abteilungsorganigramme notieren auch Leitungsbeziehungen zwischen den Stellen innerhalb einer Abteilung oder verzichten vollständig auf eine Darstellung der Stellen.

Organigramme für Mehrliniensysteme. In Mehrliniensystemen ist die Leitung einer Stelle nicht eindeutig geregelt. Einer Stelle sind evtl. mehrere verschiedene Instanzen bezogen auf unterschiedliche Zusammenhänge [Taylor, 1919, S. 132] weisungsbefugt. Zur Beschreibung von Mehrlinien-Systemen wird auch auf die Organigrammformen für Einliniensysteme zurückgegriffen. Durch diese Einlinien-Organigramme werden häufig disziplinarische Unter- bzw. Überstellungen dargestellt. Weitere, dann eher funktionsbezogene Weisungsbeziehungen werden durch zusätzliche Beziehungsgeflechte zwischen den jeweiligen Stellen angedeutet.

Beispiel 3.4 (Mehrlinien-Organigramme)

Abbildung 3.8 skizziert sowohl die disziplinarischen als auch die funktionsbezogenen Weisungsbeziehungen für die Chirurgie einschließlich der Stationen. Der *Abteilungsleiter Chirurgie*, der *Stationsleiter* und die *Pflegekräfte* sind der *Pflegedienstleitung* disziplinarisch unterstellt. Sie erhalten aber auch Anweisungen durch das medizinische Personal, das direkt oder indirekt dem *Ärztlichen Direktor* unterstellt ist. □

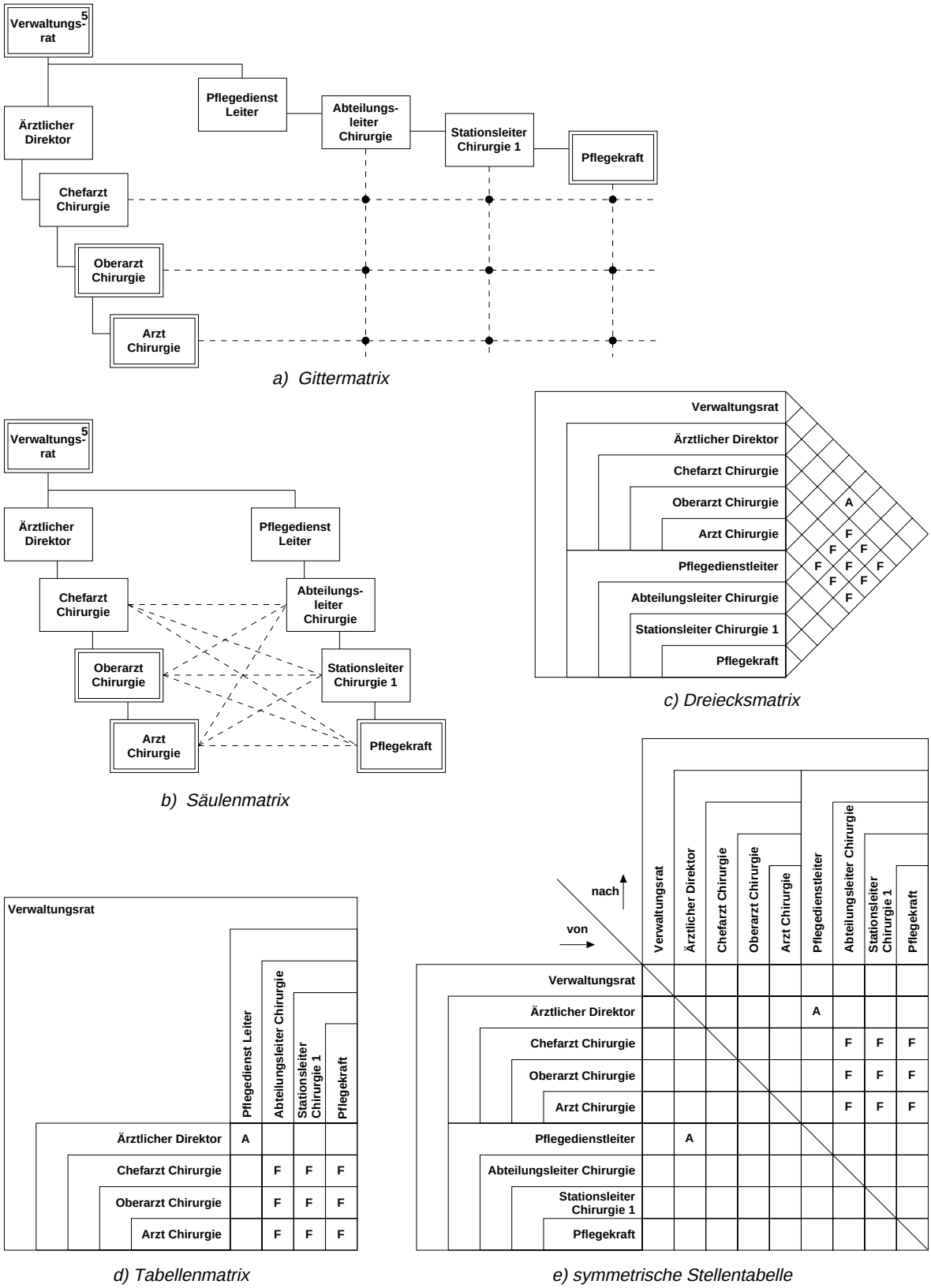


Abbildung 3.8: Organigramme für Mehrlinien-Systeme

Zur Darstellung von Systemen mit zwei Leitungsdimensionen werden Teilorganigramme so zueinander ausgerichtet, daß die zweite Leitungsdimension übersichtlich notiert werden kann. Durch ein horizontal ausgerichtetes Teilorganigramm des pflegerischen Bereichs und ein vertikal ausgerichtetes Teilorganigramm des medizinischen Bereichs wird in Abbildung 3.8a eine *Gittermatrix* aufgespannt, an deren Schnittpunkten die zusätzlichen Weisungsbeziehungen zwischen den jeweiligen Stellen notiert werden. In der *Säulenmatrix* in Abbildung 3.8b werden die Teilorganigramme des medizinischen und pflegerischen Bereichs parallel zueinander dargestellt. Die zweite Leitungsdimension wird hier durch zusätzliche Linien direkt eingezeichnet. Matrizenartige Darstellungen dieser beiden Leitungsdimensionen werden in Abbildung 3.8c und d dargestellt. Die jeweiligen Teilorganigramme sind hier in *Blockdarstellung* notiert, so daß in den Schnittpunkten der Zeilen und Spalten zu den einzelnen Stellen die Art der Weisungsbeziehung notiert werden kann. Hier können u. a. fachliche Weisungsbeziehungen (F) z. B. zwischen medizinischem und pflegerischem Personal und Abstimmungsbeziehungen (A) z. B. zwischen dem *ärztlichen Direktor* und dem *Pflegedienstleiter* unterschieden werden. Die Richtung der Weisungsbeziehungen ist in diesen Darstellungen, bezogen auf die erste Leitungsdimension, den Darstellungen der Einliniensysteme zu entnehmen. Die Richtung der zweiten Leitungsdimension ist dagegen häufig nicht eindeutig erkennbar. Eine eindeutige Ausrichtung der zweiten Leitungsdimension wird in Abbildung 3.8e sichtbar, in der zwei identische vertikal bzw. horizontal ausgerichtete Organigramme in Blockdarstellung so zu einer *symmetrischen Stellentabelle* überlagert sind, daß die Beziehungen zwischen den Stellen in beiden Richtungen notiert werden können.

Zur Darstellung von mehr als zwei Leitungsdimensionen können insbesondere die Darstellungen als Gitter- oder Säulenmatrix um weitere Dimensionen ergänzt werden. Graphische Darstellungen für Organigramme mit mehr als zwei Dimensionen werden jedoch schnell sehr unübersichtlich (vgl. [Blum, 1980b, Bild 12ff]), so daß diese Zusammenhänge kaum graphisch dargestellt werden.

Funktionendiagramme. Während in Organigrammen Leitungs- und Weisungsbeziehungen zwischen Stellen sowie hierarchische Abteilungsgliederungen dargestellt werden, beschreiben Funktionendiagramme [Menzl / Nauer, 1974], [Hub / Fischer, 1977, S. 53] die Zuordnung von Aufgaben zu diesen Stellen. Hierzu werden Organigramme und Aufgabengliederungspläne (vgl. Kapitel 3.2.1) einander gegenübergestellt. Funktionendiagramme bilden daher die jeweiligen Stellen- und Aufgabengliederungen gemeinsam ab und stellen somit ein integriertes Beschreibungsmittel der Aufgaben- und der Aufbausicht dar. Wesentlicher Beschreibungsaspekt ist jedoch die Festlegung der Zuständigkeiten zur Erledigung einzelner Aufgaben durch hierarchisch gegliederte Stellen, so daß Funktionendiagramme hier als Beschreibungsmittel der Aufbausicht und des Stellengliederungsparadigmas zugeordnet werden.

Zur Darstellung von Funktionendiagrammen bieten sich jeweils orthogonal zueinander gestellte, blockartige Notationsformen der Organigramme und der Aufgabengliederungspläne an. Diese spannen dann eine Tabelle auf, in der für jede Stelle und für jede Aufgabe die Art der Aufgabenerledigung notiert werden kann (vgl. Abbildung 3.9). Die Art der Aufgabenerledigung, die in diesem Zusammenhang auch als Funktion (vgl. z. B. [Hub / Fischer, 1977, S. 58f]) bezeichnet wird, beschreibt den Anteil der Aufgabenträger an der Aufgabenerfüllung. Diese Klassifizierung erfolgt entlang der Rang- und Phasengliederungen der Aufgaben. Unterschieden werden Zustän-

digkeiten bezogen auf die Entscheidung, die Planung, die Anordnung, die Ausführung und die Kontrolle der Aufgabenausführung (vgl. z. B. [Menzl / Nauer, 1974], [Schmidt, 1989, S. 283ff], [Lehner et al., 1991, S. 269], [Jordt / Gscheidle, (o.J.), Lehrbrief 4, S. 19]. Eine übliche Klassifikation der Funktionen ist in der Legende zu Abbildung 3.9 dargestellt. In den Zeilen eines Funktionendiagramms kann dann die Zerlegung der Aufgaben in Funktionen und deren Aufteilung auf Aufgabenträger abgelesen werden. Die Spalten geben die Aufgaben und Funktionen der einzelnen Stellen an.

G = Gesamtzuständigkeit
EV = Entscheidungsvorbereitung
E = Entscheidung
EM = Mitentscheidung
EK = Kollektiventscheidung
EN = Entscheidung im Normalfall
EG = Entscheidung in Grundsatzfragen
EW = Entscheidung in wichtigen Fällen
EA = Entscheidung in Ausnahmefällen
P = Planung
A = Ausführung
AM = Mitwirkung bei der Ausführung
K = Kontrolle
KE = Ergebniskontrolle
KV = Verfahrenskontrolle

	Verwaltungsrat	Ärztlicher Direktor	Chefarzt Chirurgie	Oberarzt Chriurgie	Arzt Chriugie	Pflegedienstleiter	Abteilungsleiter Chirurgie	Stationsleiter Chriurgie 1	Pflegekraft
Therapie durchführen									
therapeutische Maßnahmen planen		G							
pflegerische Maßnahmen anordnen			EG	EW	EN		EG	EW	
Medikamente anordnen			EG	EW	EN				
OP anordnen			EG	EW	EN				
therapeutische Maßnahmen durchführen		G				G			
Patient medizinisch versorgen			A	A	A				AM
Patient pflegen							P	P	A
Patient für OP vorbereiten					A				A

Abbildung 3.9: Funktionendiagramm

Beispiel 3.5 (Funktionendiagramm der Stationen (Ausschnitt))

Die Verteilung der Aufgaben einer Station im Krankenhaus [Schumm et al., 1995] auf unterschiedliche Aufgabenträger wird im Funktionendiagramm aus Abbildung 3.9 skizziert. Entscheidungen über die *Anordnung pflegerischer Maßnahmen* treffen in graduellen Abstufungen *Mediziner* und *Pflegepersonal.* Die *Anordnung von Medikamenten* und

Operationen ist ausschließlich dem *medizinischen Personal* vorbehalten. Das *medizinische Personal* ist ebenfalls für die *medizinische Versorgung* der Patienten verantwortlich, wird aber hierbei durch die *Pflegekräfte* unterstützt. Die *Pflege der Patienten* wird durch den *Abteilungsleiter* und den *Stationsleiter* geplant und durch die *Pflegekräfte* ausgeführt. Die *Vorbereitung der Patienten* für Operationen wird sowohl von *Ärzten* und *Pflegekräften* ausgeführt. Die *Gesamtverantwortung* für die Planung und Durchführung der Behandlungen liegt beim *Ärztlichen Direktor* und dem *Pflegedienstleiter.* □

Die Darstellung der unterschiedlichen Funktionen erfolgt wie in Abbildung 3.9 zumeist durch Buchstabenkürzel. Es ist aber auch üblich, geometrische Muster zur Unterscheidung der Funktionen zu verwenden (z. B. [Nordsieck, 1962, S. 25], [Grochla, 1982, S. 310], [Jordt / Gscheidle, (o.J.), Lehrbrief 4, S. 19]), die überlagert auch Funktionskombinationen ausdrücken können.

Stellenbeschreibungen. Organigramme und Funktionendiagramme stellen mit der Stellenhierarchie und der Verteilung der Aufgaben auf die Stellen jeweils nur besondere Aspekte der Aufbauorganisation dar. Eine umfassendere Beschreibung aller Aspekte erlauben Stellenbeschreibungen [Grochla, 1982, S. 328], [Schmidt, 1989, S. 267ff], [Lehner et al., 1991, S. 264]. In Stellenbeschreibungen werden textuell alle wesentlichen Eigenschaften der Stellen aufgeführt. Neben den durch Stellen auszuführenden Aufgaben und der Unter- und Überstellungsbeziehungen, können in Stellenbeschreibungen u. a. auch die Kompetenzen, Befugnisse, Stellvertretungsregelungen, Kommunikationsbeziehungen und die Anforderungen an Stelleninhaber zusammenfassend dargestellt werden. Organigramme und Funktionendiagramme ergänzen häufig diese textuellen Beschreibungen.

Forderungen an das Referenz-Metaschema

Wesentliche Konzepte des Stellengliederungsparadigmas sind die Leitungs-, Stabs- und Ausführungsstellen, die zwischen diesen vorliegenden Weisungs- und Leitungsbeziehungen sowie die den einzelnen Stellen zugeordneten Aufgaben. Das Referenz-Metaschema für die visuellen Modellierungssprachen des Stellengliederungsparadigmas muß somit Konzepte zur Beschreibung der verschiedenen Stellenformen, der Leitungs- und Weisungsbeziehungen und der Aufgabenzuordnungen für unterschiedliche Funktionen bereitstellen. Darüber hinaus sind auch Zusammenfassungen der Stellen zu Stellenkomplexen (Abteilungen, Ausschüssen etc.) und weitere Eigenschaften der Stellen wie z. B. Kompetenzen, Befugnisse, Vertretungsregelungen und Anforderungen zu berücksichtigen.

3.3.2 Visuelle Modellierungssprachen des Kommunikationsparadigmas

Die Kommunikation zwischen Organisationseinheiten wird mit den Beschreibungsmitteln des Kommunikationsparadigmas modelliert. Beschreibungsgegenstand sind neben den an der Kommunikation beteiligten Stellen und Abteilungen auch die Kommunikationshäufigkeiten, die Kommunikationsdauer und die verwendeten Kommunikationsmedien (vgl. [Grochla, 1982, S. 313], [Schmidt, 1989, S. 286])

Die Beschreibung der Kommunikationsbeziehungen zwischen Stellen und Abteilungen erfolgt durch Kommunigramme, durch Kommunikationsdiagramme, durch Kommunkationsgraphen, durch Kommunikationstabellen und durch Kooperationsbilder.

Notationelle Grundform: Kommunigramm

Zur Darstellung von Kommunikationsbeziehungen werden graphbasierte Darstellungsmittel verwendet, bei denen die Knoten jeweils die Kommunikationspartner und die Kanten die Kommunikationsbeziehungen darstellen. Graduelle Unterschiede in der Kommunikation, wie z. B. unterschiedliche Kommunikationshäufigkeiten oder -dauern, werden durch Kantenattributierungen oder durch visuelle Unterscheidung der Kantendarstellung hervorgehoben.

Beispiel 3.6 (Kommunigramm)

Das Kommunigramm aus Abbildung 3.10 beschreibt die Kommunikationsbeziehungen zwischen der *Chirurgischen Station*, den medizinischen Funktionen *Physikalische Therapie* und *Krankengymnastik*, den *Ärzten* und den Verwaltungsabteilungen *Aufnahme* und *stationäre Abrechnung*.

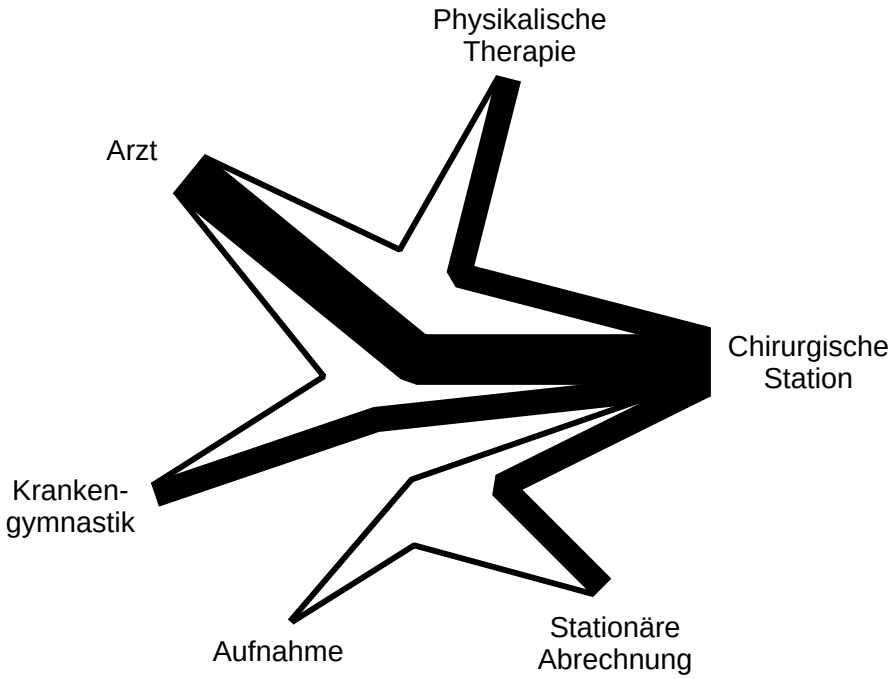


Abbildung 3.10: Kommunigramm (Kommunikationshäufigkeit)

Die unterschiedlichen Strichstärken der Kanten zwischen den einzelnen Abteilungen symbolisieren hierbei unterschiedliche Kommunikationshäufigkeiten. Dickere Strichstärken stellen hierbei häufigere Interaktionen dar. Aus dem Kommunigramm in Abbildung 3.10 wird ersichtlich, daß z. B. mehr Interaktionen zwischen der *Chirurgischen Station* und den *Ärzten* stattfinden als zwischen der *Chirurgischen Station* und der *Aufnahme*. Die Richtung einer Informationsübermittlung wird durch diese Notation nicht abgebildet. □

Notationelle Varianten

Auch zur Darstellung von Kommunikationsbeziehungen werden alternative Darstellungsformen verwendet. Aufbauend auf Organigrammen in Blockdarstellung können Kommunigramme als Adjazenzmatrizen dargestellt werden. Abbildung 3.11 spannt eine solche *Adjazenzmatrix in Dreiecksform* auf. Das links dargestellte Organigramm ist hierbei auf die relevanten Abteilungen (einschließlich einer „Abteilung“ für die Ärzte) beschränkt. Ebenso werden auch *symmetrische Kommunikationsmatrizen* ähnlich Abbildung 3.8e verwendet. In den Feldern solcher Matrizen oder Tabellen können dann die Kommunikationseigenschaften notiert werden. In Abbildung 3.11a wird z. B. durch Pfeilsymbole die Kommunikationsrichtung (↑↓ : bidirektional; ↑ , ↓ : unidirektional) dargestellt.

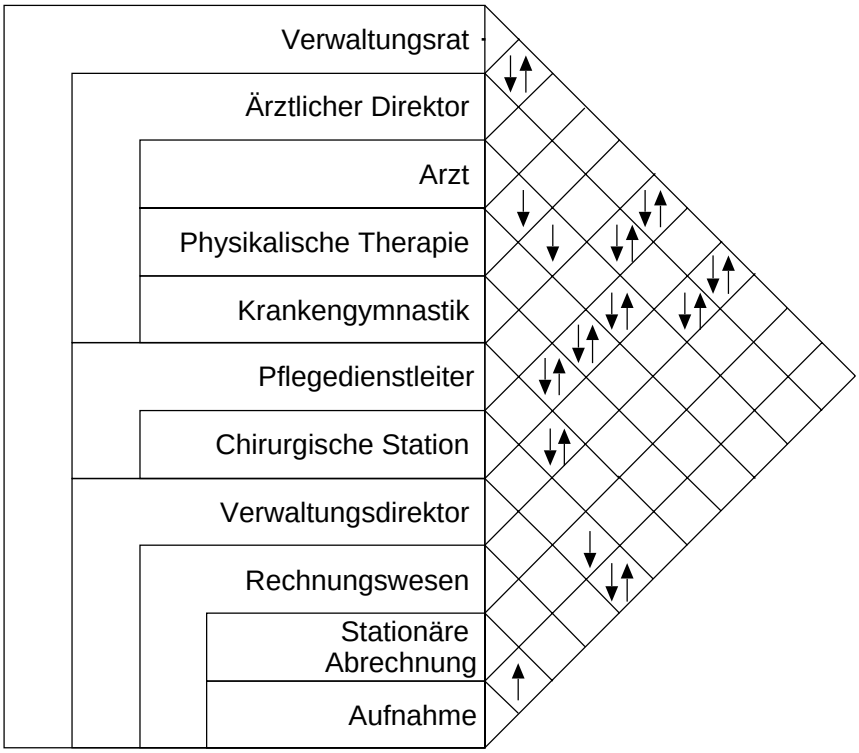


Abbildung 3.11: Kommunikationsmatrix (Dreiecksform)

Der Kommunikationsgraph in Abbildung 3.12 beschreibt durch die gerichteten Kanten ebenfalls die Kommunikationsrichtung. Durch die Strichstärken der Kanten werden analog zu Abbildung 3.10 verschiedene Kommunikationshäufigkeiten unterschieden. In ähnlicher Form kann auch die Kommunikationsdauer dargestellt werden.

Kooperationsbilder. Während durch die bisher skizzierten Notationsformen für Kommunigramme die Kommunikationhäufigkeit oder -dauer beschrieben wurde, wird durch Kooperationsbilder [Krabbel et al., 1996], [Floyd et al., 1997] das zur Kommunikation verwendete Medium herausgestellt.

Das Kooperationsbild in Abbildung 3.13 unterscheidet durch Piktogramme die Kommunikation durch Telefon, durch übermittelte Dokumente und durch persönlich übermittelte Dokumen-

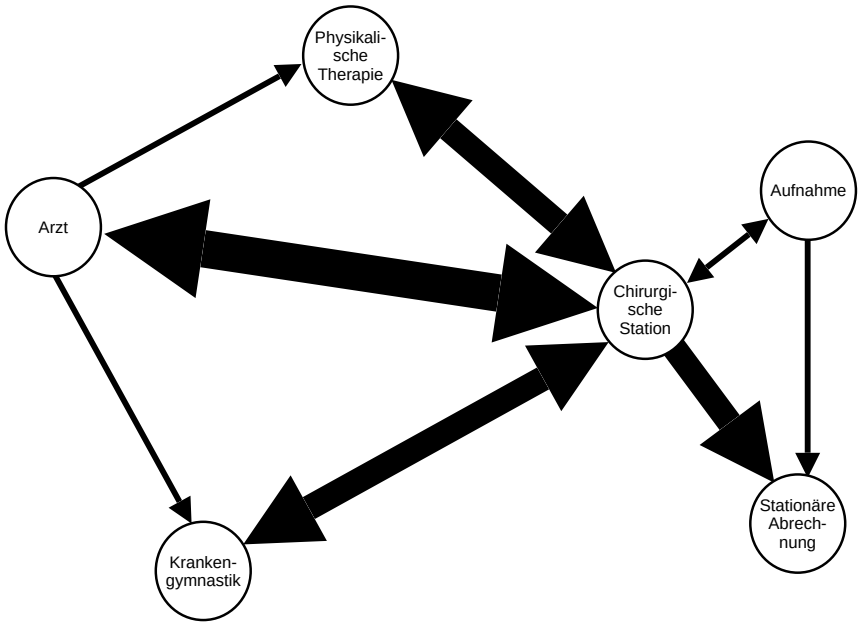


Abbildung 3.12: Kommunikationsgraph

te. Gerichtete Interakionen können in Kooperationsbildern zwischen Abteilungen (Rechtecke), zwischen Stellen (Ovale) und zwischen Bereichen außerhalb des modellierten Systems (Wolken) dargestellt werden. Eingesetzt werden Kooperationsbilder sowohl zur Erarbeitung der Kommunikationszusammenhänge als auch zur Dokumentation in der Anforderungsermittlung [Krabbel et al., 1997].

Forderungen an das Referenz-Metaschema

Durch die visuellen Modellierungsprachen des Kommunikationsparadigmas wird die Interaktion zwischen verschiedenen Kommunikationspartnern in bezug auf die Kommunikationshäufigkeit, die Kommunikationsdauer und die hierzu verwendeten Kommunikationsmedien beschrieben. Das Referenz-Metaschema dieser Sprachen muß somit Konzepte zur Beschreibung der miteinander interagierenden Stellen, Abteilungen und externen Partnern, der quantitativen Kommunikationseigenschaften und der Kommunikationsmedien bereitstellen.

3.4 Visuelle Modellierungssprachen der Prozeßsicht

Beschreibungsschwerpunkt der Prozeßsicht ist die Darstellung sowohl *logischer* als auch *zeitlicher Aspekte der Aufgabenbearbeitung*.

Logische Zusammenhänge in der Aufgabenbearbeitung beziehen sich auf die funktionalen Eigenschaften eines Systems. Beschrieben wird hierbei nicht, *wann* oder *wie* eine Berechnung erfolgt oder ein Prozeß ausgeführt wird, sondern *was* berechnet bzw. *was* getan wird. Funktionale Beschreibungen stellen das nach außen sichtbare Verhalten eines Systems dar. Hierzu werden die einzelnen Prozesse als Funktionen aufgefaßt, deren Eingaben in Ausgaben transformiert werden.

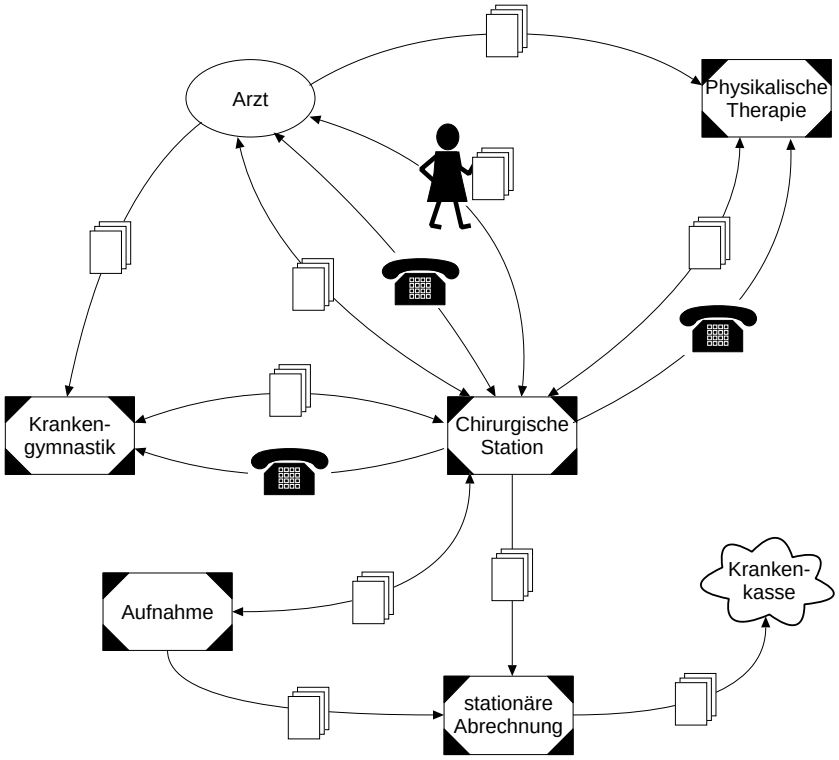


Abbildung 3.13: Kooperationsbild

Die logischen Zusammenhänge zwischen den einzelnen Prozessen werden durch die hierzwischen ausgetauschten Daten und Gegenstände beschrieben. Zur Darstellung dieser funktionalen oder logischen Systemeigenschaften werden die Beschreibungsmittel des *Datenflußparadigmas* verwendet.

Die zeitliche Betrachtung der Aufgabenbearbeitung umfaßt die Untersuchung der Dynamik des zu beschreibenden Systems. Betrachtet wird hier das innere Verhalten eines System, d. h. *wann*, in welcher zeitlichen oder logischen Reihenfolge die Prozesse des Systems bearbeitet werden. Hierbei interessiert insbesondere der Kontrollfluß, d. h. die Reihenfolge in der die einzelnen Teilprozesse oder Operationen ausgeführt werden. Zur Beschreibung dieser Kontrollflußaspekte gibt es mehrere Paradigmen. Durch die visuellen Sprachen des *Zustandsübergangsparadigmas* wird der Zustandswechsel eines einzelnen Objekts oder eines ganzen Systems über seine Lebenszeit beschrieben. Das dynamische Verhalten eines Systems kann auch durch Abfolgen von Prozessen und/oder Ereignissen beschrieben werden. Diese Beschreibungsmittel folgen dem *Netzparadigma*, das nahezu beliebige Kontrollflüsse zwischen Prozessen und Ereignissen abbildet. Die Modellierungssprachen des *Kontrollflußparadigmas* erlauben dagegen ausschließlich reguläre Strukturierungen der Kontrollflüsse durch Sequenzen, Iterationen und Verzweigungen.

3.4.1 Visuelle Modellierungssprachen des Datenflußparadigmas

Durch die visuellen Modellierungssprachen des Datenflußparadigmas werden die funktionalen Beziehungen zwischen Daten oder Gegenständen, die von einem System benötigt und erzeugt werden, beschrieben. Das Gesamtsystem wird hierbei als eine funktionale Komponente (Akti-

vität, Prozeß) aufgefaßt, die in weitere funktionale Komponenten zerlegt werden kann. Das von außen sichtbare Verhalten dieser funktionalen Komponenten wird durch die Angabe der Beziehungen zur Umwelt beschrieben. Mit den Beschreibungsmitteln des Datenflußparadigmas wird das zu modellierende System als ein Netz funktionaler Komponenten dargestellt, die über den Austausch von Daten oder Gegenständen miteinander interagieren. Diese Darstellungsmittel beschreiben den potentiell möglichen Daten- oder Materialfluß durch das System. Hierbei wird der Kontrollfluß in der Regel *nicht* modelliert.

Zur Beschreibung dieser funktionalen Eigenschaften einer Organisation oder eines Softwaresystems werden Datenflußdiagramme, Bonapart Prozeßmodelle, Bubble-Charts, Datenflußpläne, Funktionszuordnungsdiagramme, SADT-Aktivitätsdiagramme, SADT-Datendiagramme und Verkehrsschaubilder verwendet. Datenflußdiagramme, die einen ersten Blick auf das zu modellierende System erlauben, werden auch als Kontextdiagramme oder Anwendungsfalldiagramme (Use-Case-Diagramme) bezeichnet.

Notationelle Grundform: Datenflußdiagramm

Die Darstellung von Datenflußdiagrammen erfolgt durch Graphen, bei denen die funktionalen Systemelemente durch Knoten und die Datenflüsse durch Kanten dargestellt werden. In den heute gebräuchlichen Datenflußdiagrammen der (modernen) strukturierten Analyse [DeMarco, 1978], [Yourdon, 1989] werden neben den Prozessen und Datenflüssen auch Datenspeicher aufgeführt. Diese Speicher dienen zur Modellierung von persistenten Daten oder zwischengelagerten Gegenständen.

Der Einstieg in eine Datenflußmodellierung erfolgt häufig über ein spezielles Datenflußdiagramm, das als *Kontextdiagramm* bezeichnet wird. Kontextdiagramme beschreiben die Einbettung des zu modellierenden Systems in die Systemumgebung. Hierzu wird das gesamte System als ein Prozeß aufgefaßt, der mit externen Prozessen über Datenflüsse interagiert. Diese externen Prozesse, die wieder eigene, hier nicht weiter betrachtete, Systeme sind, stellen die Schnittstelle des zu modellierenden Systems zu einer Umwelt dar.

Für die weitere Modellierung werden Prozesse verfeinert. Sie werden hierzu z. B. analog zur Aufgabengliederung (vgl. Kapitel 3.2.1) oder entlang der inzidenten Datenflüsse (vgl. [Yourdon, 1989, S. 375]) wiederholt in Teilprozesse zerlegt. Das Zusammenspiel dieser Prozesse wird dann in einem neuen Datenflußdiagramm beschrieben. In Anlehnung an [Miller, 1956] fordern [Ross, 1977] und [Yourdon, 1989, S. 160], daß ein Prozeß in drei bis sechs Unterprozesse zerlegt werden soll. Diese Zerlegungen werden solange durchgeführt, bis Elementarprozesse erreicht sind, die durch wenige Zeilen Text ausreichend beschrieben werden können [Yourdon, 1989, S. 168]. Weitere Konkretisierungen der Prozesse, die dann konkretes Vorgehen und nicht das nach außen sichtbare Verhalten der Elementarprozesse beschreiben, erfolgen in der (modernen) strukturierten Analyse z. B. durch Entscheidungstabellen, Nassi-Shneiderman-Diagramme oder Pseudocode entlang des Kontrollflußparadigmas (vgl. Kapitel 3.4.4). In der Unified Modeling Language (UML) [Rumbaugh et al., 1999, S. 64] erfolgt die Konkretisierung von Anwendungsfällen, die als Prozesse aufgefaßt werden können, z. B. durch Statecharts des Zustandsübergangsparadigmas (vgl. Kapitel 3.4.2), durch Aktivitätsdiagramme des Netzparadigmas (vgl. Kapitel 3.4.3), durch Sequenzdiagramme und Kollaborationsdiagramme des Objekt-Interaktionsparadigmas (vgl. Kapitel 3.5.2) oder durch informale, textuelle Beschreibungen.

Bei der Verfeinerung eines Prozesses durch weitere Datenflußdiagramme ist sicherzustellen, daß alle in den verfeinerten Prozeß eingehenden bzw. alle aus ihm ausgehenden Datenflüsse mit den in die Verfeinerung eingehenden bzw. mit den aus ihr ausgehenden korrespondieren. Ebenso darf die Verfeinerung keine weiteren ein- bzw. ausgehenden Datenflüsse besitzen. Zur Verfeinerung der Datenflüsse können diese in weitere Teildatenflüsse verzweigt werden. Die weitere Konkretisierung der Datenflüsse erfolgt durch die Beschreibungsmittel des Objekt-Beziehungsparadigma (vgl. Kapitel 3.5.3).

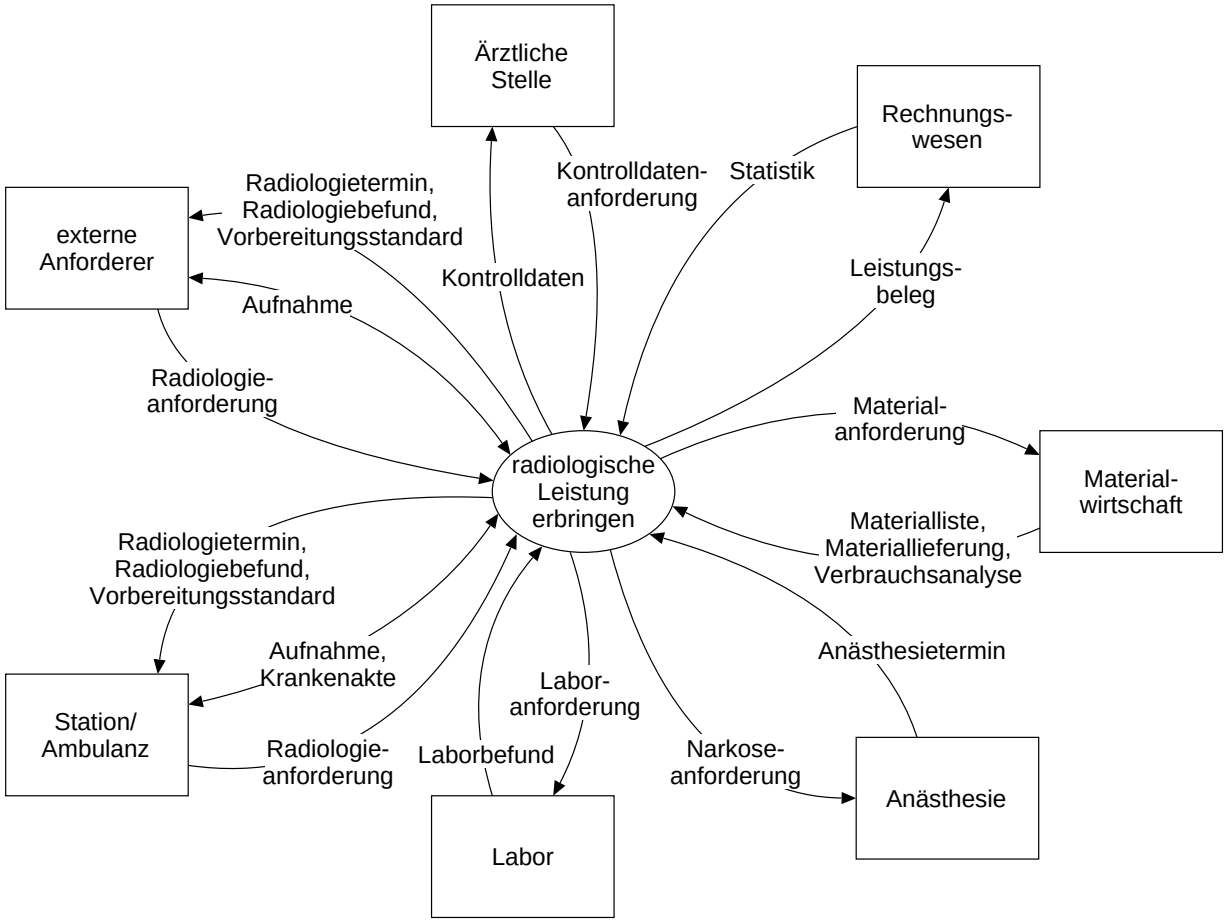


Abbildung 3.14: Kontextdiagramm

Zur Unterscheidung von Prozessen, Speichern, Schnittstellen und Datenflüssen werden in konkreten Notationen unterschiedliche graphische Formen verwendet. Prozesse werden in Anlehnung an [DeMarco, 1978] meist durch Ellipsen, Speicher durch zwei parallele Linien, Schnittstellen durch Rechtecke und Datenflüsse durch Pfeile dargestellt, die jeweils durch einen Bezeichner annotiert sind. Zur Bezeichnung der Datenflüsse, Speicher und Schnittstellen werden hierzu meist Substantive im Singular und für Prozesse Verbalphrasen verwendet. In Beispiel 3.7 ist die Radiologie eines Krankenhauses (in Ausschnitten) als Datenflußdiagramm [Schumm et al., 1995], [Winter / Ebert, 1997] modelliert.

Beispiel 3.7 (Datenflußmodellierung der Radiologie)

Abbildung 3.14 beschreibt das Kontextdiagramm der radiologischen Abteilung eines Krankenhauses. Die Radiologie wird hierbei als Prozeß *radiologische Leistung erbrin-*

gen aufgefaßt, der von unterschiedlichen Schnittstellen (Abteilungen, externe Partner etc.) Daten und Gegenstände erhält bzw. an diese gibt. Datenflußbeziehungen bestehen zu den Verwaltungsabteilungen (*Rechnungswesen* und *Materialwirtschaft*) und zu medizinisch/pflegerischen Abteilungen (*Anästhesie, Labor, Station/Ambulanz*) innerhalb des Krankenhauses sowie zu Partnern außerhalb des Krankenhauses (*externe Anforderer* und *Ärztliche Stelle*).

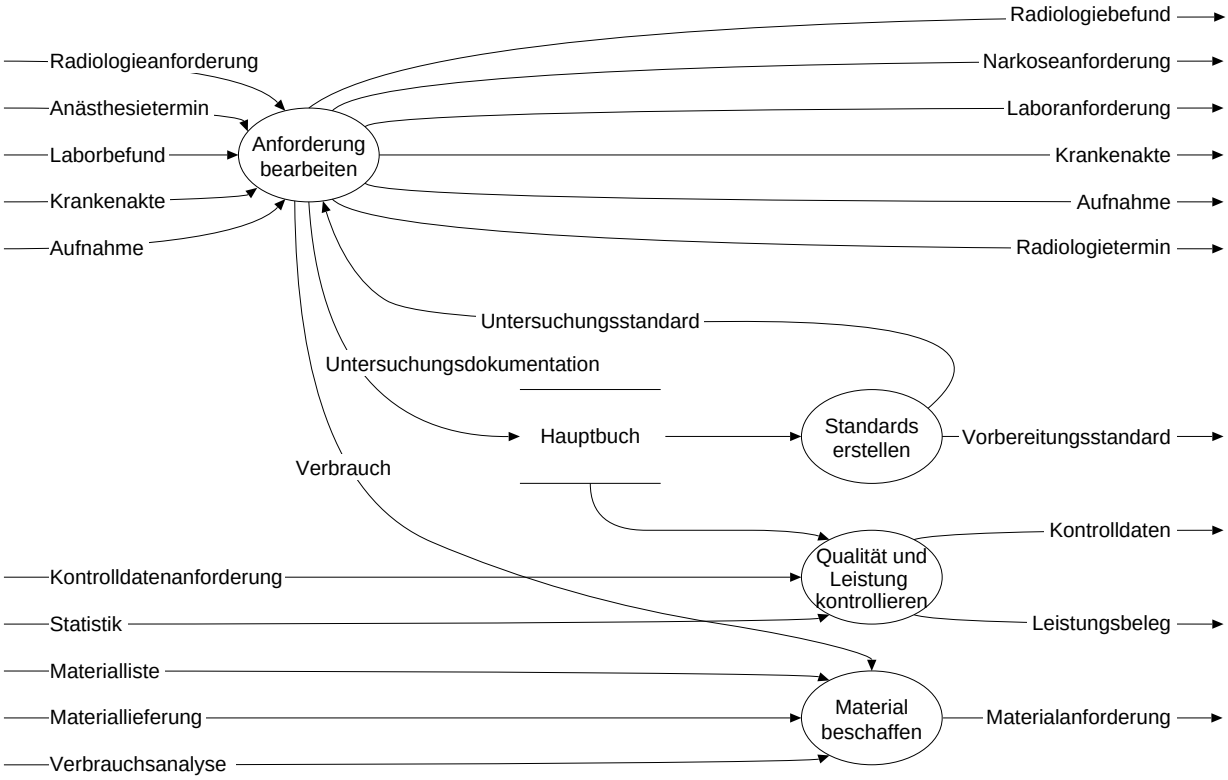


Abbildung 3.15: Datenflußdiagramm

Der Prozeß *radiologische Leistung erbringen* wird in Abbildung 3.15 durch die Prozesse *Standards erstellen, Anforderung bearbeiten, Qualität und Leistung kontrollieren* und *Material beschaffen* konkretisiert (vgl. auch die Aufgabengliederung in Beispiel 3.2). Die in den Prozeß *radiologische Leistung erbringen* eingehenden bzw. die aus ihm ausgehenden Datenflüsse finden sich auch in der Verfeinerung als ein- und ausgehende Datenflüsse wieder, so daß dieser Prozeß und die Zusammenfassung seiner Teilprozesse nach außen dasselbe Verhalten aufweisen.

Im Prozeß *radiologische Leistung erbringen* werden im Speicher *Hauptbuch* die diversen radiologischen Untersuchungen protokolliert. Diese Aufzeichnungen bilden die Grundlage zur Festlegung von Standards im Prozeß *Standards erstellen* und zur Qualitätskontrolle im Prozeß *Qualität und Leistung kontrollieren.* Ein Beispiel für eine Datenflußbeziehung ist der Datenfluß *Verbrauch* zwischen den Prozessen *Anforderung bearbeiten* und *Material beschaffen.* Hierdurch wird der Materialverbrauch der einzelnen Untersuchungen an die interne Beschaffung übermittelt, die z. B. neue *Materialanforderungen* an die Materialwirtschaft stellen kann. □

Notationelle Varianten

Datenflußorientierte Beschreibungsmittel sind schon sehr lange zur Modellierung von funktionalem Systemverhalten im Einsatz, so daß sich über die Zeit auch hier mehrere Darstellungsformen herausgebildet haben. Einige auf den ersten Blick verschiedene Beschreibungsmittel aus unterschiedlichen Modellierungsepochen sind in den Abbildungen 3.16 bis 3.18 skizziert.

Die verschiedenen Beschreibungsmittel des Datenflußparadigmas unterscheiden sich lediglich durch ihre konkrete Notation. So werden in der Datenflußmodellierung nach [Gane / Sarson, 1979], [Gane, 1990] Prozesse durch Rechtecke mit abgerundeten Ecken und Speicher durch rechts offene Rechtecke dargestellt. In SADT-Aktivitätsdiagrammen [Ross, 1977] oder in Datenflußplänen [Chapin, 1970], [DIN 66001, 1983] werden Prozesse durch Rechtecke beschrieben. Datenflüsse werden sowohl durch Kurven als auch durch rechtwinklige (evtl. abgerundete) Linienzüge notiert.

Die Darstellungsvarianten nach [Ross, 1977] und [Gane / Sarson, 1979] erlauben gegenüber der notationellen Grundform nach [DeMarco, 1978] auch die Beschreibung der menschlichen oder technischen Handlungsträger eines Prozesses. In der Notation nach [Gane / Sarson, 1979] werden diese direkt in den Prozeßdarstellungen notiert. SADT beschreibt die Handlungsträger (Mechanismen) durch Substantive, die mit der Prozeßdarstellung durch einen Pfeil verbunden sind. In Bonapart Prozeßmodellen [Pro Ubis, 1999b, S. 37ff] können Prozesse ebenfalls um Handlungsträger ergänzt werden. Werden die Handlungsträger auf Stellen innerhalb der Organisation bezogen, lassen sich hierdurch Querbezüge zur Aufbausicht (vgl. Kapitel 3.3) herstellen.

Echtzeit-Datenflußdiagramme. Eine Erweiterung der Grundform der Datenflußdiagramme um Mittel zur Beschreibung von Echtzeitsystemen nehmen [Ward / Mellor, 1985, S. 47ff] vor. Hierbei werden zusätzlich zu den Datenflüssen, Prozessen und Speichern noch Ereignisflüsse, Kontrollprozesse und Ereignisspeicher eingeführt. Ereignisflüsse zwischen Prozessen steuern deren Ausführung. Kontrollprozesse sind ausschließlich zu Kontrolldatenflüssen inzident und dienen zur Modellierung von Steuerungskomponenten. Diese Kontrollprozesse werden durch die Mittel des Zustandsübergangsparadigmas (vgl. Kapitel 3.4.2) weiter konkretisiert. Ereignisspeicher dienen ausschließlich zur Zwischenspeicherung von Ereignissen. Die Darstellung dieser Kontrollelemente erfolgt durch unterbrochene Linien.

Datenflußpläne. *Datenflußpläne nach DIN 66001* [DIN 66001, 1983], die auch als Verkehrsschaubilder (vgl. z. B. [Grochla, 1982, S. 315]) bezeichnet werden, können als eine frühe Form der Beschreibungsmittel des Datenflußparadigmas aufgefaßt werden. In diesen Diagrammen werden die Daten neben Prozessen und Speichern als eigenständige Dinge dargestellt. Datenflußkanten verbinden dann Prozeßdarstellungen mit Datendarstellungen in der jeweiligen Datenflußrichtung.

Gegenüber den heute gebräuchlichen Datenflußdiagrammen werden in Datenflußplänen verschiedene Arten der Trägermedien für Daten (z. B. Dokumente, allgemeine Datenträger), verschiedene Arten der Speicherung (Lochkarte, Lochstreifen, Magnetband, Trommelspeicher, Hauptspeicher) und verschiedene Arten der Verarbeitung (allgemeine Verarbeitung, manuelle Verarbeitung) unterschieden. Für diese Konzepte werden jeweils unterschiedliche Piktogramme (vgl. [DIN 66001, 1983], [Schmidt, 1989, S. 323]) verwendet.

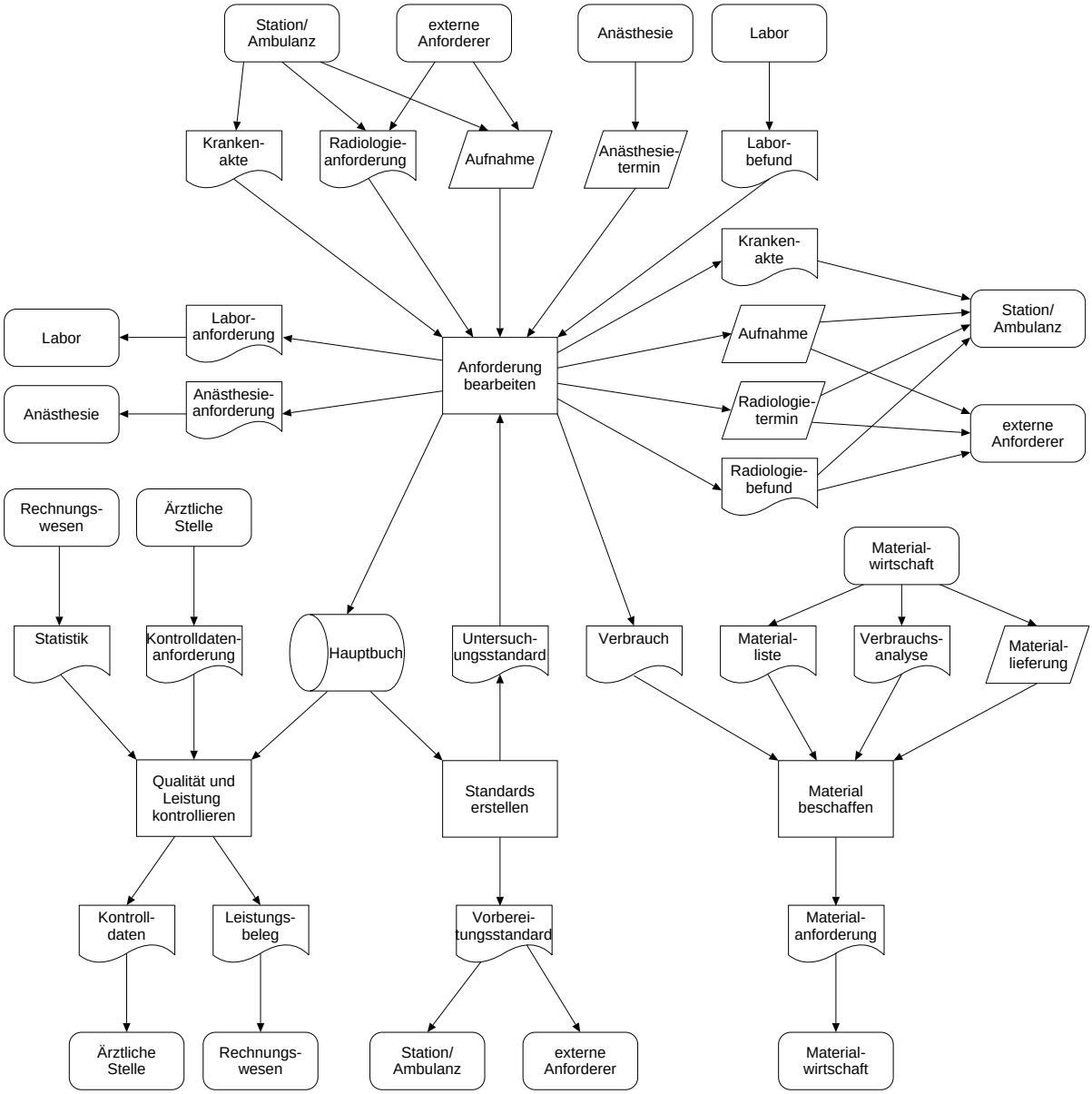


Abbildung 3.16: Datenflußplan

Abbildung 3.16 beschreibt den Prozeß *radiologische Leistung erbringen* als Datenflußplan. Hierbei werden die Teilaktivitäten als allgemeine Verarbeitungsschritte aufgefaßt, die durch Rechtecke notiert werden. Daten, die zwischen diesen ausgetauscht werden, sind i. allg. Dokumente (z. B. *Krankenakte* oder *Radiologieanforderung*). Diese werden durch Piktogramme, die einen Ausriß aus einem Papierdokument symbolisieren, beschrieben. Allgemeine bzw. nicht weiter konkretisierte Datenträger werden durch Rauten (z. B. *Aufnahme*) notiert. In Datenflußplänen wird der Fluß dieser Datenträger durch das System modelliert. Gerichtete Kanten symbolisieren die Erzeugung bzw. Verwendung der Daten. Schnittstellen zur Systemumwelt werden in Datenflußplänen durch Ovale notiert. Da Datenflußpläne i. allg. von oben nach unten oder von links nach rechts gelesen werden, sind z. B. die Schnittstellen *Station/Ambulanz* oder *Materialwirtschaft* jeweils getrennt als Daten liefernde und Daten erhaltene Interaktionspartner modelliert.

SADT-Diagramme. Eine Variante der Datenflußdiagramme der (modernen) strukturierten Analyse stellen *SADT-Aktivitätsdiagramme* dar. Gegenüber den Notationen von [DeMarco, 1978], [Gane / Sarson, 1979] und [Yourdon, 1989] werden in SADT umfangreiche Vorgaben zur übersichtlichen Darstellung der Datenflußdiagramme gemacht (vgl. [Ross, 1977], [Marca / McGowan, 1988]). Die Prozesse eines Datenflußdiagramms sind hierbei z. B. treppenartig von links oben nach rechts unten anzuordnen und Datenflußbeziehungen sind durch rechtwinklige Linienzüge zu beschreiben. Für die Modellierung von SADT-Diagrammen ist es wesentlich, an welcher Stelle der Prozeßdarstellung Kanten inzident sind. Eingehende Datenflüsse werden grundsätzlich an der linken Seite und ausgehende Datenflüsse an der rechten Seite einer Prozeßdarstellung notiert. Kontrolldaten, die in der Prozeßbearbeitung nicht verändert werden und nur steuernden Charakter haben, gehen von oben und Verbindungen zu Handlungsträgern von unten in die Prozeßdarstellung ein. Speicher werden in SADT nicht verwendet. Wie auch die Datenflußdiagramme der (modernen) strukturierten Analyse erlaubt auch SADT die Verfeinerung der Datenflußdiagramme.

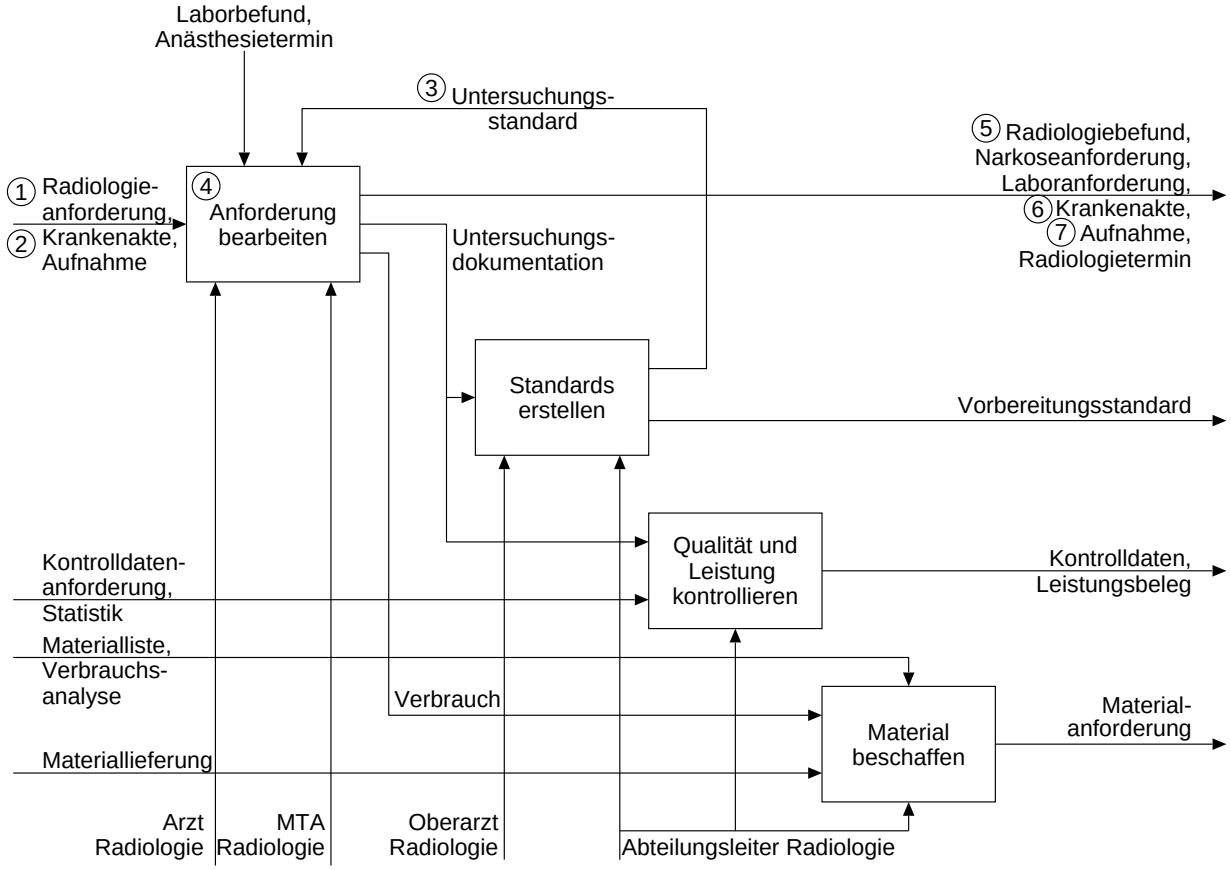


Abbildung 3.17: SADT-Aktivitätsdiagramm

In Abbildung 3.17 ist der Prozeß *radiologische Leistung erbringen* als SADT-Aktivitätsdiagramm beschrieben. Für die eingehenden Datenflüsse ist hierbei erkennbar, ob sie durch den Prozeß umgewandelt werden oder ob sie lediglich steuernden Charakter haben. So kontrollieren beispielsweise *Laborbefund, Anästhesietermin* und *Untersuchungsstandard* den Prozeß *Anforderung bearbeiten,* während *Radiologieanforderung, Krankenakte* und *Aufnahme* weiter bearbeitet

werden. In diesem Diagramm ist auch erkennbar, durch welche Mitarbeiter die einzelnen Aktivitäten ausgeführt werden.

Beschreibungsmittel des Datenflußparadigmas modellieren üblicherweise keine Kontrollflußstrukturen. SADT erlaubt aber durch *Sequentialisierungen* [Balzert, 1988, S. 120] Ablauffolgen herauszustellen. Diese Sequenzen, die durch Markieren von Datenflüssen und Prozessen mit Zahlen notiert werden, beschreiben Szenarien von Aktivierungen einzelner Aktivitäten in Abhängigkeit der ein- und ausgehenden Datenflüsse. Die in Abbildung 3.17 herausgestellte Sequenz

1. *Radiologieanforderung,*
2. *Krankenakte,*
3. *Untersuchungsstandard,*
4. *Anforderung bearbeiten,*
5. *Radiologiebefund,*
6. *Krankenakte,*
7. *Aufnahme*

beschreibt ein Standardszenario zur Durchführung einer radiologischen Untersuchung.

Neben den Aktivitätsdiagrammen, die die Prozesse in den Vordergrund stellen, werden in SADT auch *Datendiagramme* eingeführt, in denen funktional zusammenhängende Daten durch Knoten notiert werden. Die Prozesse, die diese Daten verändern oder erzeugen, werden als von links in den Datenknoten eingehende Kanten gezeichnet. Steuernde Aktivtäten (z. B. Änderungsdienste) werden analog zu den Aktivtätendiagrammen als von oben eingehende Kanten dargestellt. Aus den Datenknoten ausgehende Kanten beschreiben die lesenden Aktivitäten. Der von unten eingehende Mechanismuspfeil wird zur Darstellung des Speichermediums verwendet. SADT-Datendiagramme stellen eine zu den SADT-Aktivitätsdiagrammen komplementäre Notation bereit.

Anwendungsfalldiagramme. Auch in den objektorientierten Methoden zur Systemanalyse werden Beschreibungsmittel des Datenflußparadigmas verwendet. So setzt die Object Modeling Technique (OMT) [Rumbaugh et al., 1991] Datenflußdiagramme der strukturierten Analyse zur Beschreibung der Systemdynamik ein. In der Unified Modeling Language (UML) [Booch et al., 1999, S. 266] können Aktivitätsdiagramme (vgl. die Beschreibungsmittel des Netzparadigmas in Kapitel 3.4.3) um datenflußähnliche Objektflüsse ergänzt werden. *Anwendungsfalldiagramme* der UML, die aus Object-Oriented Software Engineering (OOSE) [Jacobson et al., 1993, S. 126] übernommen wurden, betrachten ebenfalls das zu modellierende System in seinem Systemkontext. In *Anwendungsfällen* ist die nach außen sichtbare Funktionalität des Systems zusammengefaßt [Rumbaugh et al., 1999, S. 488], so daß Anwendungsfälle als Systemprozesse betrachtet werden können. Im Mittelpunkt der Beschreibung durch Anwendungsfalldiagramme stehen die wesentlichen Prozesse und deren Interaktionsbeziehungen zur Systemumwelt, die durch Akteure repräsentiert werden. Die Anwendungfälle bzw. die hiermit assoziierten Prozesse werden in Aktivtätsdiagrammen des Netzparadigmas (vgl. Kapitel 3.4.3) näher beschrieben.

Im Gegensatz zu den Kontext- und Datenflußdiagrammen der (modernen) strukturierten Analyse werden in Anwendungsfalldiagrammen die zwischen der Systemumwelt und den Anwendungsfällen ausgetauschten Daten *nicht* beschrieben. Anwendungsfalldiagramme stellen nur das Vorhandensein einer (Interaktions-) Beziehungen zwischen Akteuren und Anwendungsfällen heraus.

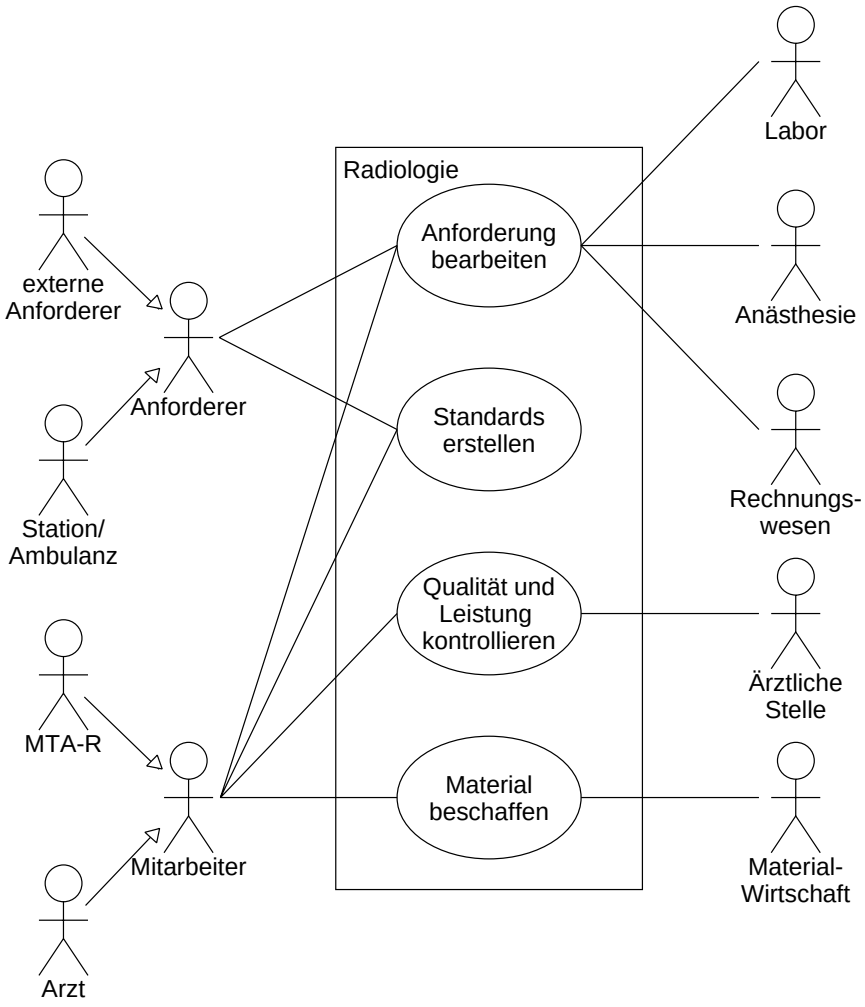


Abbildung 3.18: Anwendungsfalldiagramm

In Anwendungsfalldiagrammen werden vier Beziehungsarten unterschieden [Rumbaugh et al., 1999, S. 65]. Das Vorliegen einer Interaktionsbeziehung zwischen einem Akteur und einem Prozeß wird durch Assoziationen beschrieben. Assoziationen können auch durch Kardinalitäten annotiert sein [OMG, 1999, S. 3-88]. Sowohl zwischen Akteuren als auch zwischen Anwendungsfällen erlaubt die UML Generalisierungsbeziehungen. Extend-Beziehungen zwischen Anwendungsfällen stellen eine implementationsnähere Variante der Generalisierung auf Basis der Delegation dar. Include-Beziehungen sind ebenfalls nur zwischen Anwendungsfällen zugelassen. Sie werden verwendet, um Teilfunktionalität mehreren Anwendungsfällen bereitzustellen. Extend- und Include-Beziehungen werden durch gestrichelte Pfeile dargestellt, die jeweils durch „`extend`“ oder „`include`“ markiert sind. Anwendungsfalldiagramme beschreiben voneinander unabhängige, zentrale Prozesse eines Systems, so daß zwischen diesen keine Assoziationen erlaubt sind [Rumbaugh et al., 1999, S. 489].

Ein Anwendungsfalldiagramm für die *Radiologie* ist in Abbildung 3.18 skizziert. Die Akteure zur Darstellung der Schnittstellen zur Systemumwelt werden durch Strichmännchen notiert. Hierbei wurden *externe Anforderer* und *Station/Ambulanz* zu *Anforderer* generalisiert. Die *Mitarbeiter* (*MTA-R* und *Ärzte*)), durch die Anwendungsfälle bearbeitet werden, sind hier ebenfalls

als Akteure modelliert. Mögliche Interaktionsbeziehungen zwischen den Akteuren und den durch Ellipsen dargestellten Anwendungsfällen werden durch Linien notiert.

Forderungen an das Referenz-Metaschema

Die visuellen Modellierungssprachen des Datenflußparadigmas beschreiben das von außen sichtbare Verhalten eines Systems. Hierzu wird das System durch Prozesse modelliert, die mit der Systemumwelt interagieren. Querbezüge zwischen Prozessen und Schnittstellen werden sowohl durch Datenflüsse als auch durch Interaktionsbeziehungen hergestellt. Das Referenz-Metaschema muß daher Konzepte zur Darstellung der funktionalen Systembestandteile, zur Darstellung der Daten- und Interaktionsbeziehungen und zur Darstellung der Schnittstellen zur Systemumwelt abbilden.

Neben der Einbettung eines Systems in seine Umwelt erlauben die Beschreibungsmittel des Datenflußparadigmas auch die Konkretisierung des funktionalen Verhaltens durch Verfeinerungen. Hierzu muß das Referenz-Metaschema auch Konzepte zur Verfeinerung von Prozessen durch weitere Datenflußbeschreibungen bereitstellen. Diese Verfeinerungskonzepte erfordern auch eine Berücksichtigung der Strukturierung der durch die Datenflüsse ausgetauschten Daten. Daneben sollten auch Konzepte aus Ergänzungen und Erweiterungen der Datenflußbeschreibungen wie z. B. Speicher, Mechanismen oder Szenarien abgebildet werden.

3.4.2 Visuelle Modellierungssprachen des Zustandsübergangsparadigmas

Zur Beschreibung reaktiver Systeme werden die Modellierungssprachen des Zustandsübergangsparadigmas verwendet. Die Darstellung des Systemverhaltens erfolgt hierbei entlang der Zustände, die Objekte oder Systeme einnehmen können. Übergänge zwischen diesen Zuständen werden durch Ereignisse ausgelöst, die entweder außerhalb des Systems oder im modellierten System selbst eintreten können. Sowohl Zuständen als auch Zustandsübergängen können Aktionen zugeordnet werden, die vom System in den entsprechenden Situationen ausgeführt werden. Darstellungen des Zustandsübergangsparadigmas dienen somit zur zustandsübergangsorientierten Kontrollflußmodellierung.

Der Modellierung entlang des Zustandsübergangsparadigmas liegt die Betrachtung des Systems durch endliche, deterministische Automaten mit Ausgabe [Hopcroft / Ullmann, 1979, S. 42] zugrunde. Notiert werden diese Automaten durch Statecharts, durch (hierarchische) Automaten, durch Zustandsautomaten, durch Zustands-(übergangs)-Diagramme, durch Zustands-(übergangs)-Matrizen und durch Zustandsübergangstabellen.

Notationelle Grundform: Statecharts

Modellierungen reaktiver Systeme durch Automaten werden durch Graphen dargestellt. Die Knoten beschreiben die Systemzustände und die Kanten die Zustandsübergänge. Hierbei sind die Knoten durch einen Zustandsbezeichner und die Kanten durch das den Übergang auslösende Ereignis markiert. Beschreibungen durch solche Zustandsübergangsgraphen werden aufgrund der modellierten Komplexität jedoch schnell sehr unübersichtlich, so daß Zustandsübergangsgraphen um zusätzliche Strukturierungsmittel zu Statecharts ergänzt wurden [Harel, 1987], [Harel,

1988]. Diese Strukturierungsmittel erlauben die Zusammenfassung und Verfeinerung von Zuständen (Depth), die Beschreibung von Parallelabläufen (Orthogonality) und die Synchronisation von Teilsystemen durch Nachrichtenaustausch (Broadcast). Zur Darstellung der Statecharts werden Hypergraphen verwendet, die Venn-Diagramm-ähnlich notiert werden. Abbildung 3.19 skizziert exemplarisch ein Statechart zur Modellierung eines Dialysesystems.

Durch die Verwendung der zusätzlichen Strukturierungsmittel erfolgt die Modellierung eines reaktiven Systems durch mehrere Teilautomaten. Der Zustand des Gesamtsystems ergibt sich hierbei aus der Summe der Zustände der jeweiligen Teilautomaten. Um die strukturierten Zustände der einzelnen Teilautomaten begrifflich vom Zustand des Gesamtsystems zu trennen, werden die strukturierten Zustände im folgenden als *Blob* bezeichnet. Zustandsübergänge beschreiben daher auch den Übergang zwischen Blobs. Diese Übergänge werden durch Pfeile notiert, die mit den die Übergänge auslösenden Ereignissen annotiert sind. Übergänge können zusätzlich zum Eintreffen des Ereignisses noch von Prädikaten abhängig gemacht werden. Diese „Guards“ werden in eckigen Klammern den Ereignissen vorangestellt.

Statecharts verwenden atomare und zusammengesetzte Blobs, die durch Ovale dargestellt werden und mit einem Bezeichner markiert sind. Bei den zusammengesetzten Blobs werden *Xor-Blobs* (sequential substates), die, zu einem Teilautomaten zusammengefaßte Blobs enthalten, und *And-Blobs* (concurrent substates) unterschieden, die sich aus mindestens zwei zueinander orthogonal stehenden Teilautomaten zusammensetzen. Die Teilautomaten der And-Blobs werden durch unterbrochene Linien voneinander getrennt.

Xor-Blobs ermöglichen die Zusammenfassung mehrerer Blobs zu einem Teilautomaten. Wie auch Automaten können Xor-Blobs einen Start-Blob besitzen, der durch einen unmarkierten Pfeil dargestellt wird, der von einem schwarzen Punkt ausgeht. Die Unified Modeling Language [Booch et al., 1999, S. 331] verwendet daneben noch optionale End-Blobs, die durch einen von einem Kreis umgebenen schwarzen Punkt notiert werden. Der Zustand des Gesamtsystems wird jeweils durch genau einen dieser Blobs beeinflußt. Zustandsübergänge in einen Xor-Blob können sich sowohl auf den Gesamtblob als auch auf die in ihm enthaltenen Komponenten beziehen. Nach einem Übergang in einen Xor-Blob befindet sich der Teilautomat entweder im Start-Blob des Xor-Blobs oder bei Verwendung eines „History“-Mechanismus (durch ein „H“ markiert) im zuletzt besuchten Blob. Bei Zustandsübergängen in eine Komponente des Xor-Blobs befindet sich der Teilautomat in dieser Komponente. Zustandsübergänge aus einem Xor-Blob können ebenfalls sowohl von dem Xor-Blob selbst als auch von den Komponenten-Blobs ausgehen. In beiden Fällen trägt der Xor-Blob nicht mehr zum Gesamtzustand des Systems bei.

And-Blobs dienen zur Modellierung von parallelem Systemverhalten. Zustandsübergänge sind hier grundsätzlich ebenfalls zu und aus dem And-Blob sowie zu und aus seinen Komponenten-Blobs erlaubt. Hierbei ist aber sicherzustellen, daß keine Konflikte auftreten und der Automat deterministisch bleibt. Ausführliche Diskussionen und Lösungstrategien für diese Konflikte finden sich in [Harel, 1987] und [Ebert, 1993]. Zustandsübergänge zwischen zueinander parallel liegenden Teilautomaten sind generell nicht erlaubt. Der Systemzustand eines And-Blobs bestimmt sich aus der Kombination der jeweiligen Zustände der Teilautomaten.

Wird das Statechart als Moore-Automat aufgefaßt, können den Blobs auch Aktionen zugeordnet werden. [Booch et al., 1999, S. 295ff] erlaubt hierbei Aktionen, die bei Eintritt in den Blob, während des Aufenthalts im Blob und bei Verlassen des Blobs ausgeführt werden. Diese Aktionen werden durch die Schlüsselworte „`entry`“, „`do`“ und „`exit`“ unterschieden. Die Betrachtung des

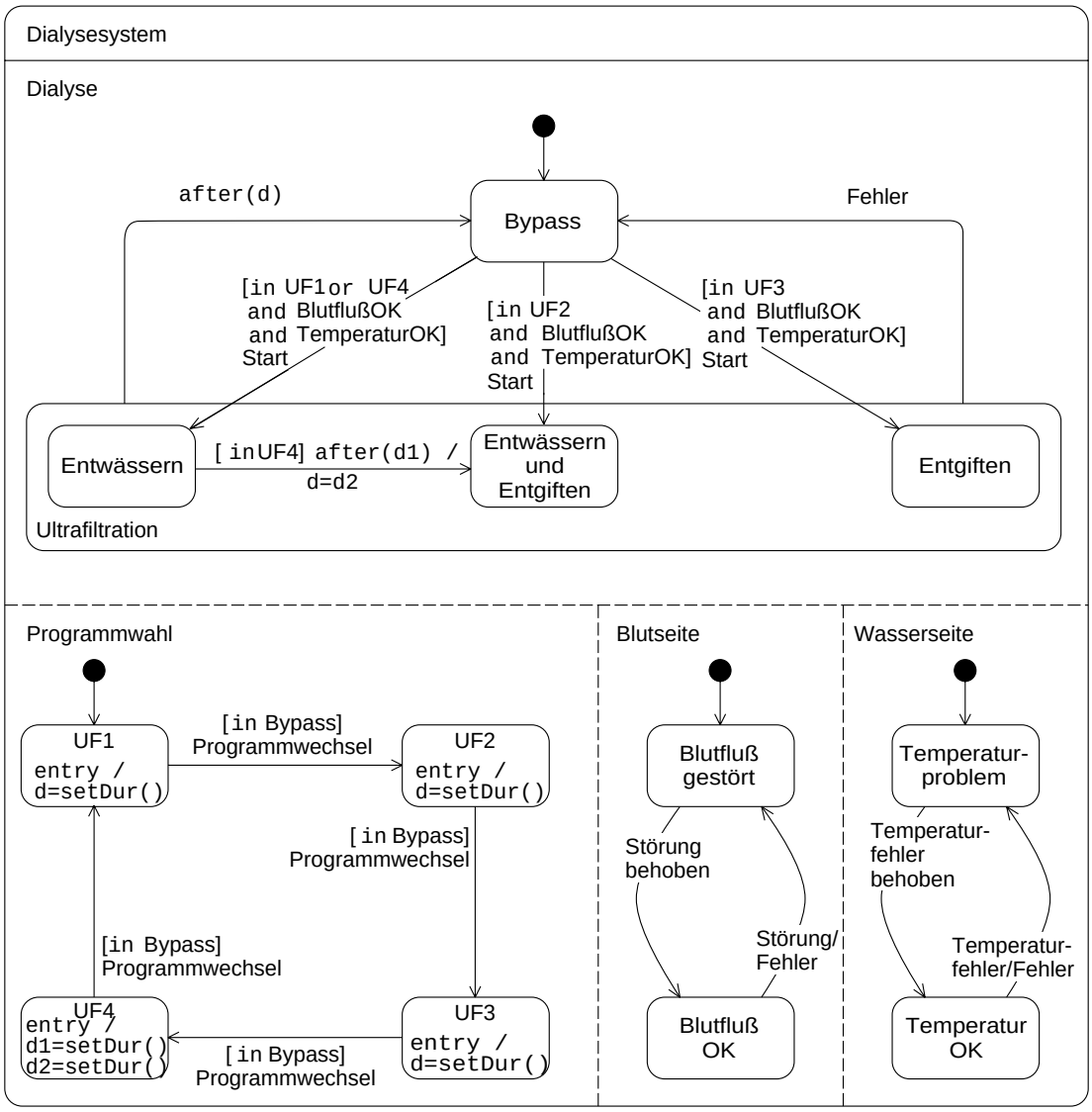


Abbildung 3.19: Statechart

Statecharts als Mealy-Automat ermöglicht Aktionen an den Zustandsübergängen. Diese werden durch einen „/“ von der Ereignisdarstellung getrennt. Zur Synchronisation von parallelen Automaten können diese Ereignisse auch weitere Ereignisse auslösen (Broadcast), auf die das System zeitgleich reagiert. Die Unified Modeling Language [Booch et al., 1999, S. 287ff] verwendet sowohl Moore- und Mealy-Automaten als auch Kombinationen aus beiden.

Beispiel 3.8 (Statechart eines Dialysesystems)

Abbildung 3.19 enthält ein Statechart zur Modellierung eines Dialysesystems. Dieses (extrem vereinfachte) Dialysesystem basiert auf vier Xor-Blobs, die in dem And-Blob *Dialysesystem* zusammengefaßt sind. Der Blob *Dialyse* stellt die Grundfunktionalität des Systems mit Ultrafiltrationsmodi zum *Entwässern*, zum *Entwässern und Entgiften* und zum *Entgiften* sowie einem *Bypass*-Modus bereit. Im Blob *Programmauswahl* erfolgt die Vorwahl von vier Dialyseprogrammen (je eines für die drei Ultrafiltrationsverfahren und eines

für eine Kombinationsbehandlung mit den Teilphasen *Entwässern* und *Entwässern und Entgiften*). Die beiden restlichen Automaten skizzieren am Blutfluß und an der Temperatursteuerung exemplarisch Überwachungseinrichtungen auf der *Blutseite* und der *Wasserseite.*

Der Zustand des Systems bestimmt sich aus den jeweiligen Zuständen dieser Teilsysteme. Bei Aktivierung des Dialysesystems befinden sich diese in ihren Startblobs. Durch das Ereignis *Programmwechsel* können verschiedene Dialyseprogramme ausgewählt werden. Diese Auswahl ist jedoch nur möglich, falls sich der *Dialyse*-Automat im *Bypass*-Blob befindet. Dieses wird durch das Prädikat *[`in` Bypass]* modelliert. Mit Eintreten in einen der vier Programmblobs (*UF1-4*) wird auch die Dauer für die jeweilige Behandlung durch die „`entry`"-Aktion `d=setDur()` gesetzt.

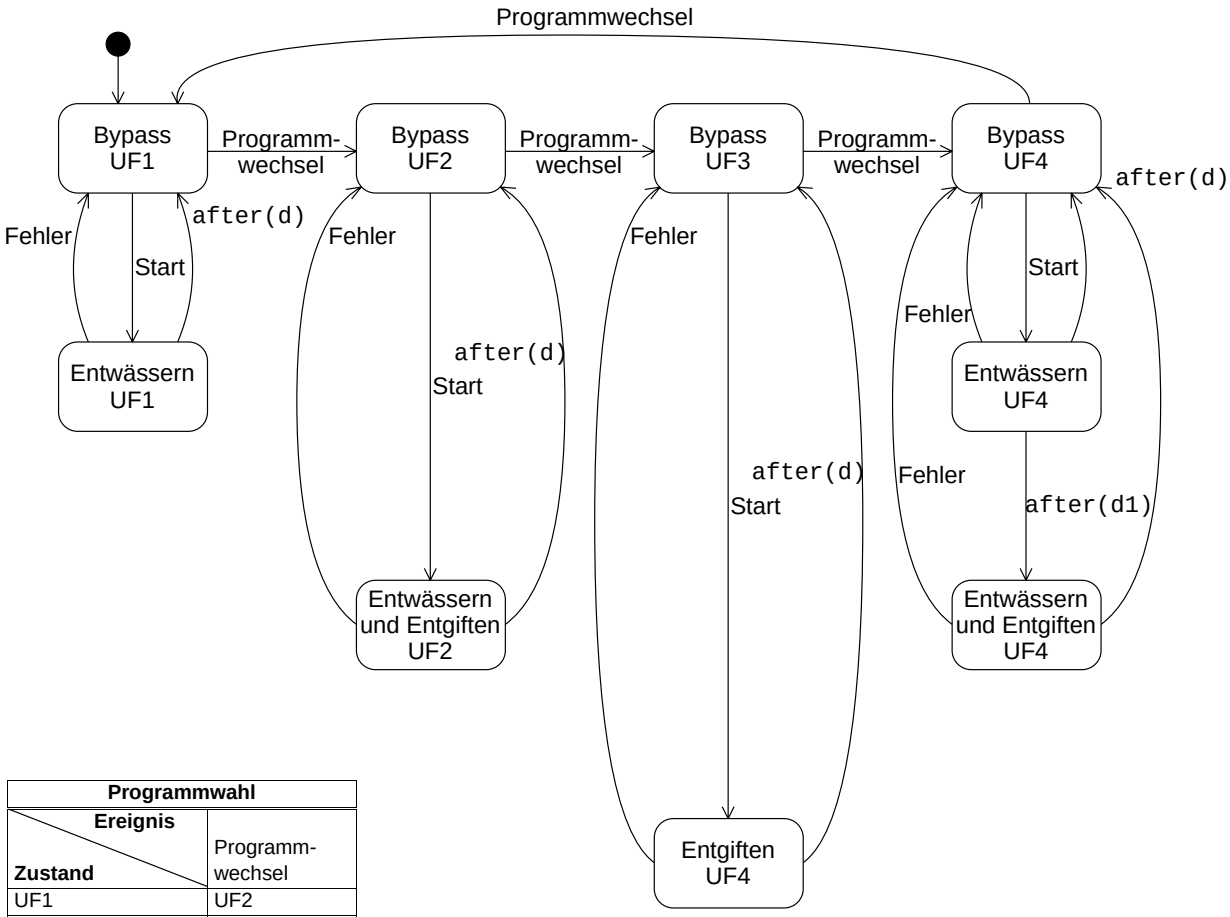
Ausgehend von dem *Bypass*-Blob wechselt der *Dialyse*-Blob, nach dem Ereignis *Start* in das durch den Zustand des *Programmauswahl*-Systems angegebene Ultrafiltrationsverfahren. Diese Übergänge sind nur zugelassen, falls auch die Überwachungsautomaten keine Probleme signalisieren. Ist das Programm *UF4* vorgewählt, wird nach der Beendigung der Entwässerung (*`after(d1)`*) in den Blob *Entwässern und Entgiften* gewechselt. Dieser Übergang löst gleichzeitig auch eine Aktion `d=d2` zum Setzen der Behandlungsdauer für die zweite Phase der Behandlung aus. Aus dem Xor-Blob *Ultrafiltration* erfolgt der Rücksprung zum *Bypass*-Blob, falls die voreingestellte Behandlungsdauer abgelaufen (*`after(d)`*) oder das *Fehler*-Ereignis eingetreten ist.

Die beiden Blobs *Blutseite* und *Wasserseite* befinden sich während der Durchführung einer Dialyse in den Blobs *BlutflußOK* bzw. *TemperaturOK*. Im Fehlerfall wird hier jeweils in einen Fehler-Blob gewechselt. Dieses *Fehler*-Ereignis wird durch den Broadcast-Mechanismus an den hierzu parallelen *Dialyse*-Xor-Blob gemeldet, wodurch eine Behandungsunterbrechung ausgelöst wird. □

Notationelle Varianten

Konkrete Notationen zur Visualisierung von Statecharts oder hierarchischen Automaten unterscheiden sich nur geringfügig. Mit Ausnahme weniger Erweiterungen wie z. B. die Ergänzung um Endzustände und die Zuordnung verschiedener Aktionen an Blobs, verwendet die Unified Modeling Language [Booch et al., 1999, S. 287] die gleiche Notation wie sie auch bereits in [Rumbaugh et al., 1991, S. 87ff] genutzt wurde. Kleinere semantische Unterschiede zwischen den Dialekten von [Harel, 1987] und der UML werden in [OMG, 1999, S. 2-150] herausgestellt. Eingeschränkt werden Statecharts im Dialekt nach Booch verwendet. Hier werden nur And-Blobs, aber keine Xor-Blobs unterstützt [Booch, 1994, S. 208]. Alternative Notationen unterscheiden sich hiervon i. allg. nur in ihrer Ausdrucksmächtigkeit. In Abbildung 3.20 sind einige dieser Varianten aufgeführt.

Zustandsautomaten. Im Gegensatz zu Statecharts werden unter *Zustandsautomaten* oder *Zustandsübergangsdiagrammen* flache Automaten, ohne Verwendung zusammengesetzter Blobs verstanden. Abbildung 3.20a skizziert das Zustandsübergangsdiagramm, das den Blobs *Dialyse* und *Programmauswahl* aus Abbildung 3.19 entspricht. Statecharts können generell in „flache"



a) Zustandsübergangsdiagramm

Programmwahl	
Ereignis / **Zustand**	Programm-wechsel
UF1	UF2
UF2	UF3
UF3	UF4
UF4	UF1

Blutseite		
Ereignis / **Zustand**	Störung	Störung behoben
Blutfluß gestört		BlutflußOK
BlutflußOK	Blutfluß gestört	

Wasserseite		
Ereignis / **Zustand**	Temperatur-fehler	Temperatur-fehler behoben
Temperaturproblem		TemperaturOK
TemperaturOK	Temperatur-problem	

b) Zustandsübergangsmatrix

Dialyse			
Zustand	**Ereigniss**	**Aktion**	**Folgezustand**
Bypass	[(in UF1 or UF4) and BlutdruckOK and TemperaturOK] Start		Entwässern
Bypass	[in UF2 and Blut-druckOK and Tempe-raturOK] Start		Entwässern und Entgiften
Bypass	[in UF3 and Blut-druckOK and Tempe-raturOK] Start		Entgiften
Entwässern	[in UF4] after(d)	d = d2	Entwässern und Entgiften
Ultrafiltration	Fehler		Bypass

c) Zustandsübergangstabelle

Abbildung 3.20: Statecharts, notationelle Varianten

Zustandsübergangsdiagramme übertragen werden. Die (maximale) Anzahl der Zustände eines Zustandsübergangsdiagramms ergibt sich hierbei aus dem Produkt der Anzahl der Zustände der parallel auszuführenden Teilautomaten. Inzidente Übergänge an einem And-Blob sind auf alle in diesem enthaltenen Zustände zu übertragen. Wie in Abbildung 3.20a können aber bei Berück-

sichtung der die Übergänge überwachenden Prädikate nicht mögliche Zustandsübergänge und Zustände eingespart werden.

Zustandsübergangsmatrizen und -tabellen. Neben den graphischen Graph- oder Hypergraph basierten Darstellungsformen werden auch tabellarische Notationen (z. B. [Partsch, 1998, S. 103ff]) verwendet. *Zustandsübergangsmatrizen* spannen eine Matrix aus Zuständen und Ereignissen auf und notieren in den Matrixfeldern die Folgezustände des Systems. Diese Form der Darstellung beschreibt lediglich die Beziehungen zwischen Zuständen und Ereignissen. Prädikate und Aktionen werden in dieser Notationform nicht berücksichtigt. Erweiterungen durch zusätzliche bzw. umfangreichere Matrizen sind denkbar, werden aber ebenfalls sehr schnell unübersichtlich. Abbildung 3.20b faßt die Zustandsübergangsmatrizen der Teilautomaten *Programmauswahl*, *Blutseite* und *Wasserseite* zusammen. In *Zustandsübergangstabellen* kann das Zusammenspiel von Zuständen, Ereignissen, Aktionen und Folgezuständen tabellarisch beschrieben werden. In Abbildung 3.20c ist eine solche Zustandsübergangstabelle für den Automaten des *Dialyse*-Blobs aus Abbildung 3.19 dargestellt, in dem auch die Prädikate berücksichtigt sind.

Forderungen an das Referenz-Metaschema

Die Beschreibung reaktiver Systeme durch die visuellen Modellierungssprachen des Zustandsübergangsparadigmas basiert auf der Darstellung der System- bzw. Teilsystemzustände und den hierzwischen möglichen Zustandsübergängen. Übergänge werden durch Ereignisse ausgelöst, die evtl. durch Prädikate überwacht werden. Sowohl Zuständen als auch Übergängen können Aktionen zugeordnet sein. Das Referenz-Metaschema muß daher Konzepte zur Abbildung von Zuständen, Zustandsübergängen, Ereignissen, Prädikaten, Aktionen und deren Querbezüge bereitstellen. Daneben muß es Hilfsmittel zur Strukturierung von Zustandsautomaten durch Kompositionen und Parallelbearbeitung enthalten.

3.4.3 Visuelle Modellierungssprachen des Netzparadigmas

Zur Modellierung dynamischer Organisations- und Softwareaspekte stellen die Sprachen des Netzparadigmas das Aufeinanderfolgen von *Prozessen* und/oder *Ereignissen* heraus. Zu bearbeitende Prozesse[4] werden hierbei durch ein Netz von Teilprozessen und Ereignissen und den hierzwischen vorliegenden Folgebeziehungen beschrieben. Prozesse werden hierbei als zeitverbrauchende Geschehnisse verstanden, während Ereignisse zeitpunktbezogene Zustände beschreiben (vgl. auch [DIN 69900, 1987]).

[4] Insbesondere in der Terminologie der Geschäftsprozeßmodellierung wird der Begriff „Prozeß“ oder „Prozeßkette“ (vgl. z. B. [Scheer, 1992], [Langer et al., 1997]) für eine in sich geschlossene Abfolge mehrerer Teilaktivitäten verwendet. Diese Teilaktivitäten werden auch als „Funktion“ bezeichnet (vgl. z. B. [Scheer, 1994, S. 19]). Funktionen können weiterverfeinert werden und nehmen dann die Rolle der Prozesse auf einer tieferen Modellierungsstufe ein. In den folgenden Betrachtungen wird diese (künstliche) Unterscheidung nicht gemacht. Prozesse, Funktionen, Vorgänge, Aktivitäten, Transitionen etc. werden unter den Begriff „Prozeß“ zusammengefaßt.

Die graphische Darstellung der Sprachen des Netzparadigmas basiert auf Graphen, in denen die Knoten Ereignisse oder Prozesse beschreiben, die durch Kontrollflußkanten miteinander verbunden sind. Ergänzt werden diese in einigen Dialekten um zusätzliche Knoten, die Operatoren über den Kontrollflüssen abbilden.

Wesentliches Modellierungselement ist die direkte Aufeinanderfolge von Prozessen und/oder Ereignissen. Daneben können mittels Operatoren Kontrollflüsse verzweigt bzw. parallelisiert und wieder vereinigt werden. Die Grundbausteine zur Kontrollflußverzweigung und -vereinigung sind in Abbildung 3.21 zusammengefaßt (vgl. auch [Schmidt, 1989, S. 301ff], [Keller et al., 1992, S. 14], [Scheer, 1994, S. 50f]).

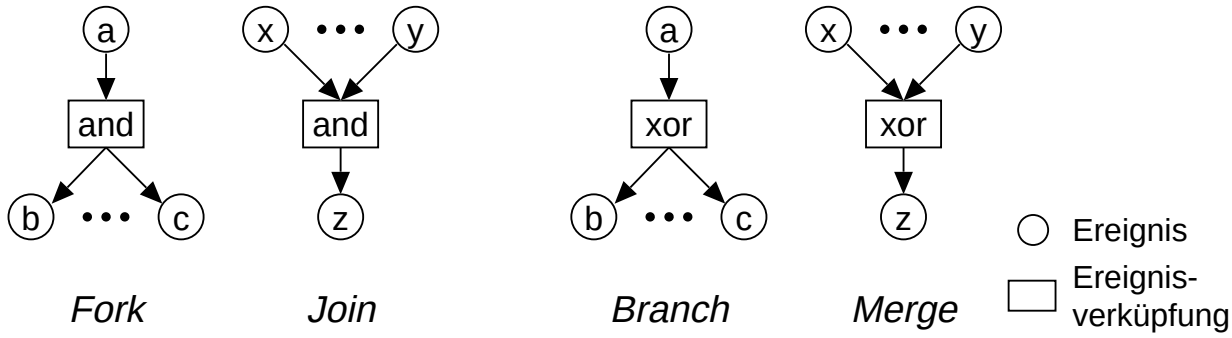


Abbildung 3.21: Grundbausteine zur Kontrollflußverknüpfung

Die Modellierung von Parallelabläufen wird durch *and*-Verzweigungen ($Fork$) eingeleitet. Auf die Verzweigung folgen dann voneinander unabhängige Prozeßfolgen, die durch and-Vereinigungen ($Join$) wieder zusammengeführt werden. Analog werden durch xor-Verzweigungen ($Branch$) exklusive, alternative Prozeßfolgen eingeleitet, die durch xor-Vereinigungen ($Merge$) wieder zusammengeführt werden können. Iterationen von Prozeßfolgen werden in den Sprachen des Netzparadigmas ebenfalls auf die Grundformen der xor-Verzweigungen und -Vereinigung zurückgeführt.

Vielfach wird in den Modellierungssprachen des Netzparadigmas, die diese Ereignis-Verknüpfungen verwenden, *nicht* explizit gefordert, daß Verzweigungen und Vereinigungen zueinander balanciert sein müssen (z. B. [DIN 66001, 1983], [Scheer, 1994]). Solche Modellierungen führen unweigerlich zu „Spaghetti“-Modellierungen, deren Semantik kaum noch faßbar ist. Aus diesem Grund wird in moderneren Modellierungsansätzen die Balancierung als besserer Modellierungsstil empfohlen (z. B. [Booch et al., 1999, S. 264]), oder es werden balancierte Verzweigungs-Vereinigungs-Paare als eigenständige Modellierungskonstrukte (z. B. [Schmidt, 1989, S. 305]) eingeführt.

Die visuellen Modellierungssprachen, die dem Netzparadigma folgen, können in vier Gruppen eingeteilt werden. Aktivitätsdiagramme, Aufgabenfolgepläne, Ablaufkarten, Folgestrukturen, Folgepläne und Programmablaufpläne dienen in erster Linie zur Darstellung von Prozeßfolgen ohne (visuelle) Berücksichtigung der Ereignisse. Ereignisgesteuerte Prozeßketten, Stimulus-Response-Folgen und Vorgangskettendiagramme ergänzen diese Notationsformen u. a. um die obligatorische Darstellung der die Prozesse auslösenden bzw. der durch die Prozesse verursachten Ereignisse. Aufgrund ihrer formalen Basis ermöglichen Petri-Netze (S/T-Netze, B/E-Netze, Pr/T-Netze, gefärbte Netze etc.) umfangreiche Analysen der auf aktiven und passiven Ele-

menten basierenden Modelle. Unterstützung bei der zeitlichen Einplanung von Prozeßabläufen bieten Netzpläne (Vorgangsknotennetzpläne, Vorgangspfeilnetzpläne, Ereignisknotennetzpläne, Balkendiagramme).

Notationelle Grundform: Aktivitätsdiagramm[5]

Zur Beschreibung des Kontrollflusses zwischen Prozessen (hier Aktivitäten) verwendet die Unified Modeling Language (UML) [Booch et al., 1999] Aktivitätsdiagramme.

Die betrachteten Prozesse werden in Ovalen notiert, die durch Pfeile zur Beschreibung des Kontrollflusses miteinander verbunden sind. Der Startpunkt einer solchen Prozeßfolge wird durch ein Startereignis und das Ende durch ein oder mehrere Endereignisse angezeigt. Diese Ereignisse werden analog zu den Start- und Endzuständen der Statecharts durch einen schwarzen Punkt bzw. einen schwarzen Punkt mit Kreis (vgl. Kapitel 3.4.2) notiert. Zur Strukturierung von Prozeßmodellierungen können Prozesse in Aktivitätsdiagrammen auch verfeinert werden [Booch et al., 1999, S. 261].

Zur Interaktion mit Objekten außerhalb des betrachteten Prozesses werden in Aktivitätsdiagrammen spezielle Prozesse zum Auslösen und zum Empfangen von externen Ereignissen (Nachrichten) verwendet. Prozesse zum Senden von Ereignissen werden durch Rechtecke mit einer aus dem Rechteck zeigenden Spitze und Prozesse zum Empfangen von Ereignissen werden durch Rechtecke mit einer in das Rechteck zeigenden Spitze notiert. In diesen Symbolen sind die jeweiligen Ereignisse vermerkt [Rumbaugh et al., 1999, S. 240], [OMG, 1999, S. 3-147].

Aktivitätsdiagramme unterstützen die Modellierung von Parallelabläufen und von Verzweigungen. Parallele Prozeßabläufe werden durch dicke, horizontale Linien (Synchronisationsbalken) voneinander getrennt ($Fork$) und wieder zusammengeführt ($Join$). Die Verzweigung des Kontrollflusses in exklusive Alternativen ($Branch$) und deren Zusammenführung ($Merge$) wird durch Rauten beschrieben. In die Symbole für $Fork$ und $Branch$ geht jeweils ein Kontrollflußpfeil ein und mindestens zwei Kontrollflußpfeile aus. $Join$- und $Merge$-Symbole verhalten sich entsprechend umgekehrt. An die Raute zur Beschreibung der Verzweigung kann das Entscheidungskriterium angetragen werden; die ausgehenden Kontrollflußkanten werden mit den jeweiligen Alternativen annotiert. Nach [Rumbaugh et al., 1999, S. 240] kann auch auf die Darstellung durch Rauten verzichtet werden. Die Kontrollfluß-Pfeile verbinden dann direkt die Prozeßdarstellungen.

Die Zuordnung von Prozessen zu Organisationseinheiten oder zu prototypischen Objekten wird durch Aktivitätsdiagramme ebenfalls unterstützt. Hierdurch ermöglichen Aktivitätsdiagramme auch die Darstellung von Querbezügen zu den Beschreibungsmitteln des Stellengliederungsparadigmas (vgl. Kapitel 3.3.1) und des Objekt-Instanzparadigmas (vgl. Kapitel 3.5.1). Das Aktivitätsdiagramm wird hierzu in Spalten (engl. swimlanes) untergliedert, die jeweils die Prozesse der einzelnen Leistungserbringer (Organisationseinheiten oder Objekte) enthalten.

[5] Als notationelle Grundform des Netzparadigmas sind auch die Ereignisgesteuerten Prozeßketten, die in der Geschäftsprozeßmodellierung weit verbreitet sind, oder auch Petri-Netz-Varianten, z. B. Prädikat/Transitions-Netze denkbar. Die Entscheidung wurde zugunsten der Aktivitätsdiagramme getroffen, weil sie das entsprechende Beschreibungsmittel des kommenden Quasi-Modellierungsstandards UML sind.

Neben der Darstellung des Kontrollflusses bieten die Aktivitätsdiagramme der UML auch die Möglichkeit Objektflußbeziehungen zu beschreiben. Objekte werden auch hier durch Rechtecke dargestellt, die den unterstrichenen Objektbezeichner enthalten. Diese sind durch unterbrochen gezeichnete Pfeile, die in Objektflußrichtung gerichtet sind, mit den erzeugenden und verbrauchenden Prozessen verbunden.

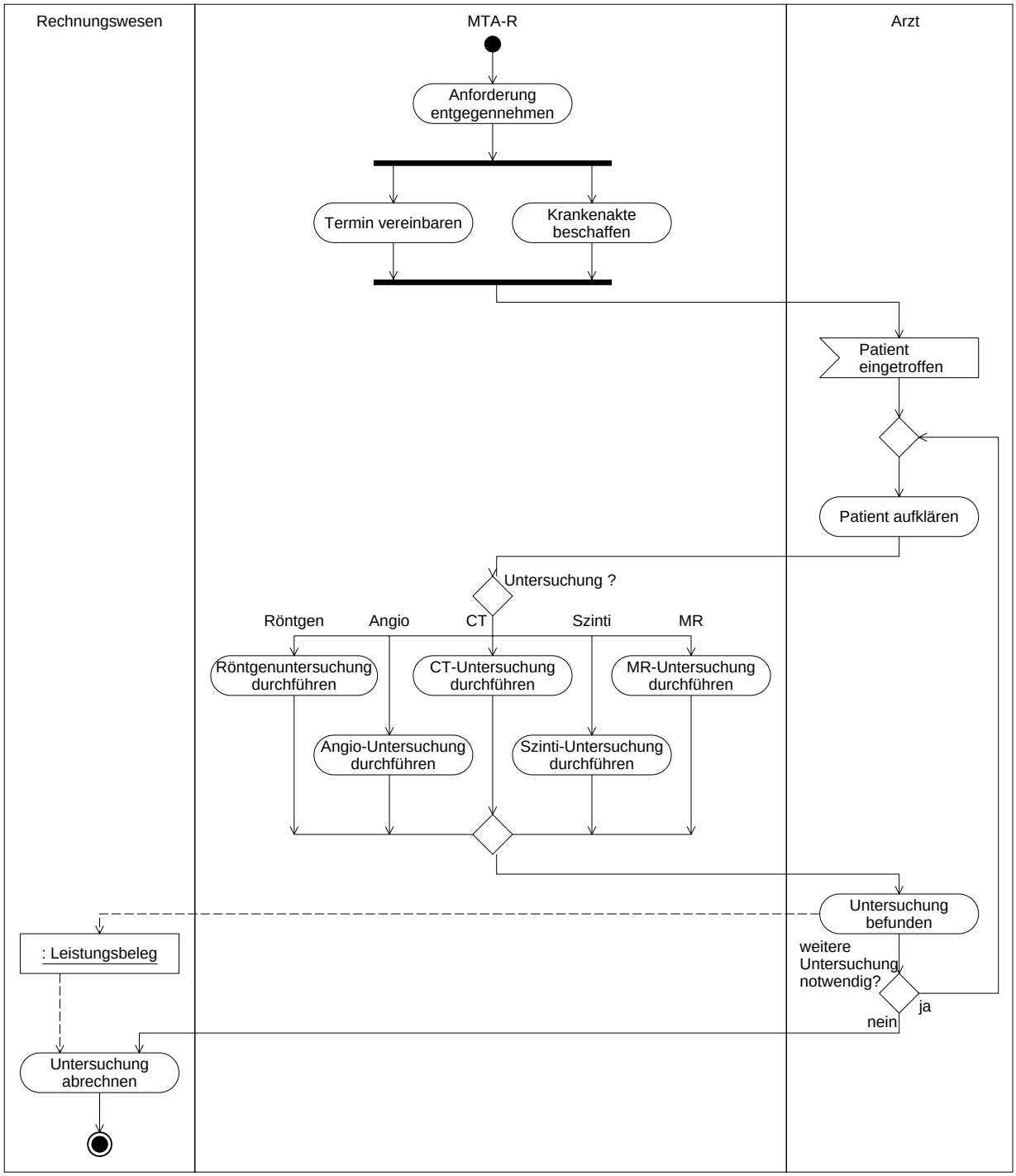


Abbildung 3.22: Aktivitätsdiagramm

Beispiel 3.9 (Aktivitätsdiagramm einer radiologischen Untersuchung)

An der Durchführung radiologischer Untersuchungen sind Medizinisch-Technische Assistenten der Radiologie (*MTA-R*), *Ärzte* und zur Abrechnung das *Rechnungswesen* beteiligt. Im Aktivitätsdiagramm in Abbildung 3.22 sind die Teilprozesse der radiologischen Untersuchung auf die entsprechenden Aufgabenträger aufgeteilt.

Nachdem die Untersuchungsanforderung vorliegt (*Anforderung entgegennehmen*), ist ein Untersuchungstermin zu vereinbaren und die Krankenakte des Patienten anzufordern. Die Prozesse *Termin vereinbaren* und *Krankenakte beschaffen* können nebenläufig bearbeitet werden. Trifft der Patient zur Untersuchung ein (vgl. Ereignis *Patient eingetroffen*), erfolgt durch den *Arzt* die Information des Patienten über die vorgesehene Untersuchung (*Patient aufklären*). Nach Auswahl der Untersuchungsmethode wird durch den *Medizinisch-Technischen Assistenten* die Untersuchung durchgeführt. Anschließend wird durch den *Arzt* der Befund erstellt (*Untersuchung befunden*). Ist die Untersuchung abgeschlossen, wird durch das *Rechnungswesen* die *Untersuchung abgerechnet*. Hierzu wird vom *Arzt* ein *Leistungsbeleg* an das *Rechnungswesen* gegeben. Sind zusätzliche Untersuchungen nötig, wird mit der Aktivität *Patient aufklären* eine weitere Maßnahme eingeleitet. □

Notationelle Varianten

Wie die datenflußorientierten Beschreibungsmittel sind auch die Sprachen zur Beschreibung der Kontrollflußabhängigkeiten zwischen Prozessen seit langem bewährte Beschreibungsmittel zur Organisations- und Softwaremodellierung. Entsprechend haben sich auch hier verschiedene Notationsformen herausgebildet. *Programmablaufpläne*, *Blockschaubilder* oder *Verkehrsschaubilder* [Chapin, 1970], [Grochla, 1982, S. 315ff], [DIN 66001, 1983] modellieren Prozesse durch Folgen, Verzweigungen, Zusammenführung und Parallelbearbeitungen von Kontrollflüssen. Zur Vermeidung von Spaghetti-Modellierungen in Programmablaufplänen werden Programmablaufpläne gelegentlich auch auf ausschließlich reguläre Kontrollstrukturen (vgl. das Kontrollflußparadigma in Kapitel 3.4.4) eingeschränkt. Hierzu wurde z. B. ein eigenständiges Beschreibungsmittel für Schleifen eingeführt, das die Konstruktion aus Verzweigung und Zusammenführung ersetzt.

Ähnlich wie in Aktivitätsdiagrammen werden in *Rasterdarstellungen* [Schmidt, 1989, S. 306ff], [Lehner et al., 1991, S. 270] Prozesse in vorgefertigte Tabellen eingetragen, bei denen in Spalten die Aufgaben der verschiedenen Leistungsträger chronologisch notiert sind. Von der sequenziellen Kontrollflußabfolge abweichende Strukturen lassen sich in diesen Tabellen jedoch nur sehr unübersichtlich durch Pfeile zwischen einzelnen Tabellenfeldern notieren. Ebenfalls ausschließlich auf die Beschreibung sequenzieller Ablauffolgen beschränkt sind (moderne) *Ablaufkarten* [Schmidt, 1989, S. 308ff], [Lehner et al., 1991, S. 270] oder *Arbeitsablaufschaubilder* [Grochla, 1982, S. 320]. Prozeßfolgen werden hier in einer Liste notiert, bei der die Prozesse noch zusätzlich nach den Verrichtungskomponenten „Bearbeitung“, „Transport“, „Kontrolle“, „Verzögerung“, und „Lagerung“ durch graphische Symbole oder Kennbuchstaben klassifiziert sind. In diesen tabellarischen Notationsformen können den einzelnen Prozessen auch die Handlungsträger zugewiesen werden (vgl. [Nordsieck, 1962, S. 133]).

Zur Untersuchung von Prozeßstrukturen auf der Basis von aktiven und passiven Komponenten haben sich weitere, nahezu eigenständige Beschreibungs- und Analysemittel herausgebildet, die daher eine eigene Betrachtung sinnvoll machen. Ereignisgesteuerte-Prozeßketten, Petri-Netze und Netzpläne werden in den folgenden Abschnitten kurz skizziert.

Ereignisgesteuerte Prozeßketten. Zentrales Beschreibungsmittel der Geschäftsprozeßmodellierung nach dem Ansatz der Architektur integrierter Informationssysteme (ARIS) [Scheer, 1992] sind *Ereignisgesteuerte Prozeßketten (EPK)* [Keller et al., 1992], [Staud, 1999]. Im Gegensatz zu Aktivitätsdiagrammen werden in Ereignisgesteuerten Prozeßketten Ereignisse im Prozeßablauf explizit herausgestellt. Sowohl Ereignisse als auch Prozesse können in Ereignisgesteuerten Prozeßketten verfeinert werden. Zur Modellierung von Verzweigungen bieten sie $Branch$- und $Merge$-Operatoren sowohl für die exklusive (xor) als auch die nicht-ausschließende Alternativenbildung (or) an. Ebenso wird die Modellierung paralleler Kontrollflüsse durch $Fork$- und $Join$-Operatoren unterstützt. Als weitere Kontrollflußoperatoren verwenden Ereignisgesteuerte Prozeßketten auch Kombinationen von Vereinigungen bzw. Parallelisierungen und deren Zusammenführungen.

In der Einführung der Ereignisgesteuerten Prozeßketten [Keller et al., 1992] werden durch diese Operatoren jeweils Ereignisse mit Prozessen oder Prozesse mit Ereignissen verknüpft. In den grundlegenden Arbeiten [Keller et al., 1992], [Scheer, 1994] zu den Ereignisgesteuerten Prozeßketten und zum ARIS-Ansatz [Scheer, 1992] wird deren Syntax nicht eindeutig eingeführt. [Scheer, 1994, S. 50] gibt lediglich wenige Beispiele möglicher Ablaufstrukturen an. Aus diesen Beispielen kann abgeleitet werden, daß Ereignisgesteuerte Prozeßketten als gerichtete, bipartite Hypergraphen von Ereignissen und Prozessen notiert werden, deren binäre Hyperkanten sequenziellen Kontrollfluß und deren nicht-binäre Hyperkanten Kontrollfluß-Operatoren beschreiben.

Notiert werden die Ereignisse durch Sechsecke und die Prozesse durch Ovale, die jeweils mit einem Bezeichner annotiert sind. Die Operatoren zur Kontrollflußverknüpfung werden durch einen horizontal in der Mitte geteilten Kreis dargestellt. Durch die logischen Operatoren $\wedge$, $\vee$, und $\underline{\vee}$ (xor) wird im oberen Halbkreis angezeigt, wie die eingehenden Kontrollflüsse vereinigt werden und im unteren Halbkreis, wie sie verzweigt werden. Die Kontrollflüsse zwischen Ereignissen, Prozessen und Operatoren werden durch Pfeile beschrieben. Verzweigungskriterien sind an den Operatoren zur Kontrollflußverzweigung nicht vorgesehen. Eine Forderung nach Balancierung der Kontrollflußoperatoren wird in den Arbeiten zu Ereignisgesteuerten Prozeßketten nicht aufgestellt. Abbildung 3.23 zeigt die dem Aktivitätsdiagramm aus Abbildung 3.22 entsprechende Modellierung einer radiologischen Untersuchung durch eine Ereignisgesteuerte Prozeßkette.

Eine alternative Darstellung der Ereignisgesteuerten Prozeßketten erlauben *Vorgangskettendiagramme* [Brombacher, 1991], [Scheer, 1994, S. 59] und *erweiterte Ereignisgesteuerte Prozeßketten*. Diese Beschreibungsmittel ermöglichen die integrierte Darstellung der Prozeßmodelle mit den benötigten und erzeugten Daten und ihrer Einbettung in die jeweilige Organisationsstruktur. Hierzu werden in Vorgangskettendiagrammen Tabellen u. a. mit Spalten für Ereignisse, Funktionen, Daten und Organisationseinheiten verwendet, in die entsprechende graphische Symbole eingetragen werden. Kontrollflüsse, Datenflüsse und organisatorische Zusammenhangsbeziehungen werden durch zusätzliche Pfeile in diesen Tabellen notiert. Darstellungen dieser Art erlauben zwar ein umfassendes Bild der betrachteten Prozeßkette, werden jedoch durch die Einschränkung auf tabellarische Darstellungen und die Überfrachtung mit Modellierungskonstrukten schnell

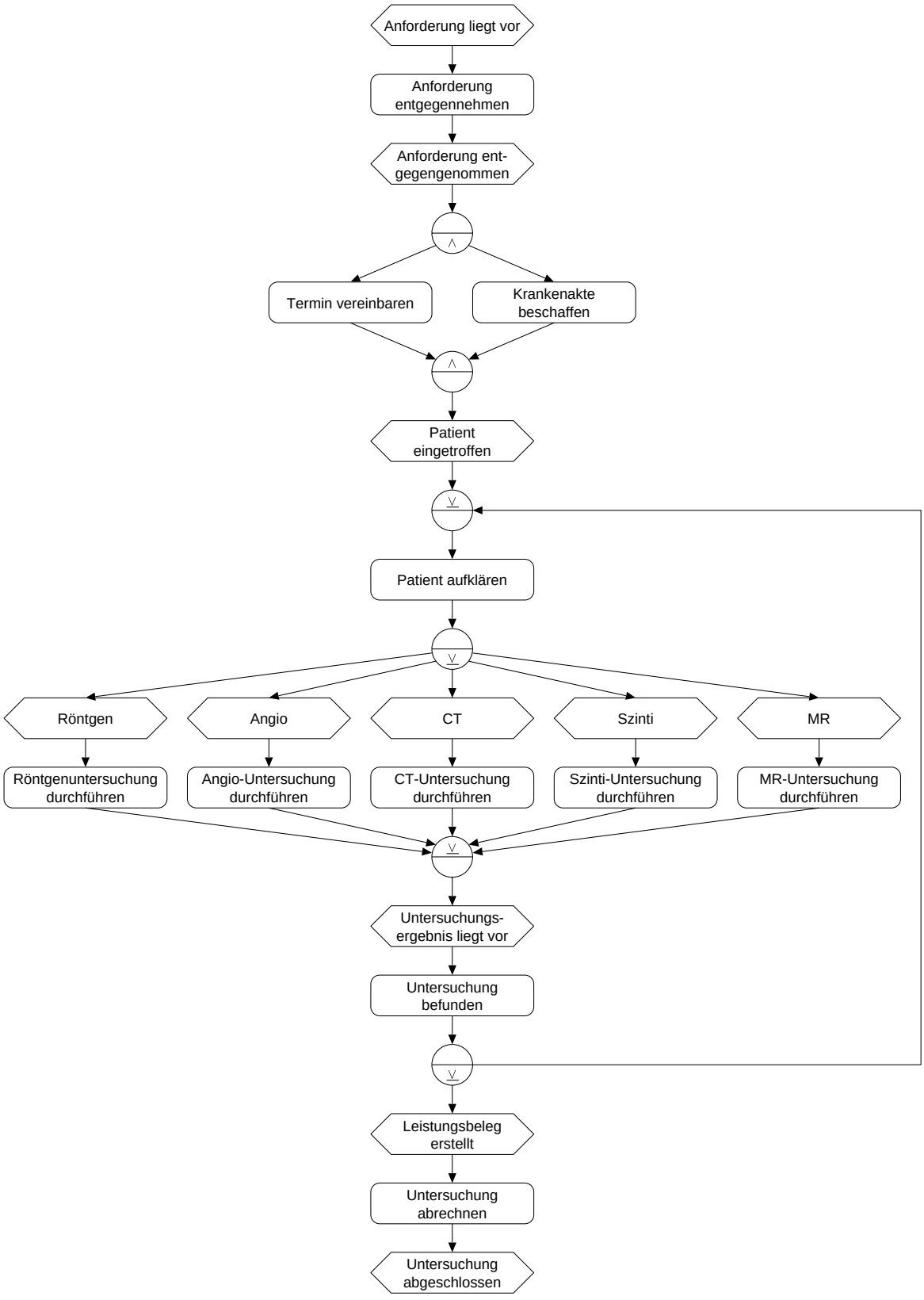


Abbildung 3.23: Ereignisgesteuerte Prozeßkette

sehr unübersichtlich. Im Gegensatz zu Vorgangskettendiagrammen erfolgt daher die Prozeßdarstellung in erweiterten Ereignisgesteuerten Prozeßketten (vgl. z. B. [Staud, 1999, S. 45ff]) durch Graphen, deren Knoten nicht auf Tabellenspalten aufgeteilt sind. In beiden Notationsformen werden Daten durch Rechtecke und Organisationseinheiten durch Ellipsen notiert.

Petri-Netze. *Petri-Netze* [Petri, 1962], [Reisig, 1992], [Baumgarten, 1996] wurden zur Beschreibung und Analyse von nebenläufigen und nicht-deterministischen Prozessen entwickelt. Die Modellierung durch Petri-Netze erfolgt durch aktive und passive Komponenten, zwischen denen ein Fluß notiert wird. Durch *Stellen*[6] werden passive Komponenten wie z. B. Zustände oder Bedingungen modelliert. Aktive Komponenten, die Zustandsveränderungen bewirken, werden als *Transitionen*[7] bezeichnet. Stellen und Transitionen werden in Petri-Netzen als gleichgewichtige Modellierungskonzepte aufgefaßt.

Die graphische Darstellung von Petri-Netzen basiert auf bipartiten Graphen, bei denen die Knoten entweder Transitionen oder Stellen modellieren (*Netzgraphen*). Transitionen werden durch Rechtecke und Stellen durch Kreise notiert, die jeweils um Bezeichner außerhalb oder innerhalb der geometrischen Symbole ergänzt werden können. Der Fluß zwischen Stellen und Transitionen wird in Netzgraphen durch gerichtete Kanten beschrieben. Der Modellierung mit Petri-Netzen liegt die Anschauung zugrunde, daß Transitionen neue Objekte erzeugen und diese auf Stellen ablegen. Die klassische Petri-Netz-Theorie abstrahiert von konkreten Objekten. Stellen enthalten nicht unterscheidbare *Marken*, die von den Transitionen erzeugt oder verbraucht werden. Hierbei kann modelliert werden, daß Stellen nur eine begrenzte Anzahl von Marken besitzen dürfen, und daß von Transitionen eine feste Anzahl von Marken entnommen bzw. erzeugt werden kann. Zur Darstellung dieser Kapazitätsangaben werden Stellen mit Stellenkapazitäten und Flußbeziehungen mit Kantengewichten annotiert. Der Systemzustand eines Petri-Netzes ergibt sich aus den Markierungen seiner Stellen. Zustandsänderungen werden durch *Schaltvorgänge* der Transitionen ausgelöst. Eine Transition ist dann aktiv, wenn ihre vorgelagerten Stellen gemäß der Kantengewichte ausreichend Marken bereitstellen und die nachgelagerten Stellen ausreichend Kapazität zur Aufnahme der erzeugten Marken bieten. Ob eine Transition aktiv ist, wird ausschließlich von der direkten Umgebung der Transition bestimmt (Lokalität). Sind mehrere Transitionen gleichzeitig aktiv, können sie nebenläufig schalten.

In Petri-Netzen können zur Strukturierung sowohl Stellen als auch Transitionen verfeinert werden (vgl. [Baumgarten, 1996, S. 59ff]). Stellen werden hierbei durch stellenberandete Teilnetze und Transitionen durch transitionenberandete Teilnetze verfeinert.

Während durch Netzgraphen lediglich die statischen Zusammenhänge zwischen Prozessen (Transitionen) und Ereignissen (Stellen) beschrieben werden, kann durch die Ergänzung um eine Anfangsmarkierung und die Berücksichtigung des Schaltverhaltens des Netzes auch das Verhalten des Prozeßmodells untersucht werden. Die Petri-Netz-Theorie stellt hierzu eine Vielzahl

[6] Der Stellenbegriff der Petri-Netze darf nicht verwechselt werden mit dem Stellenbegriff der Aufbausicht. Soweit die Bedeutung aus dem Kontext ersichtlich ist, wird der Begriff „Stelle“ für beide Zusammenhänge verwendet.

[7] Da in den Grundformen der Petri-Netze von zeitliche Aspekten abstrahiert wird, erfordert die Bearbeitung innerhalb einer Transition keine Zeit. Transitionen werden z. B. in den eher zustandsorientiert aufgefaßten Bedingungs/Ereignis-Netzen als *Ereignisse* bezeichnet, deren Eintreffen eine Zustandsveränderung auslöst. Betrachtet man Petri-Netze jedoch prozeßorientiert und vergleicht sie mit den anderen Beschreibungsmitteln des Netzparadigmas, entsprechen Transitionen jedoch den Prozessen während die Stellen den Ereignissen der Aktivitätsdiagramme oder der Ereignisgesteuerten Prozeßketten entsprechen.

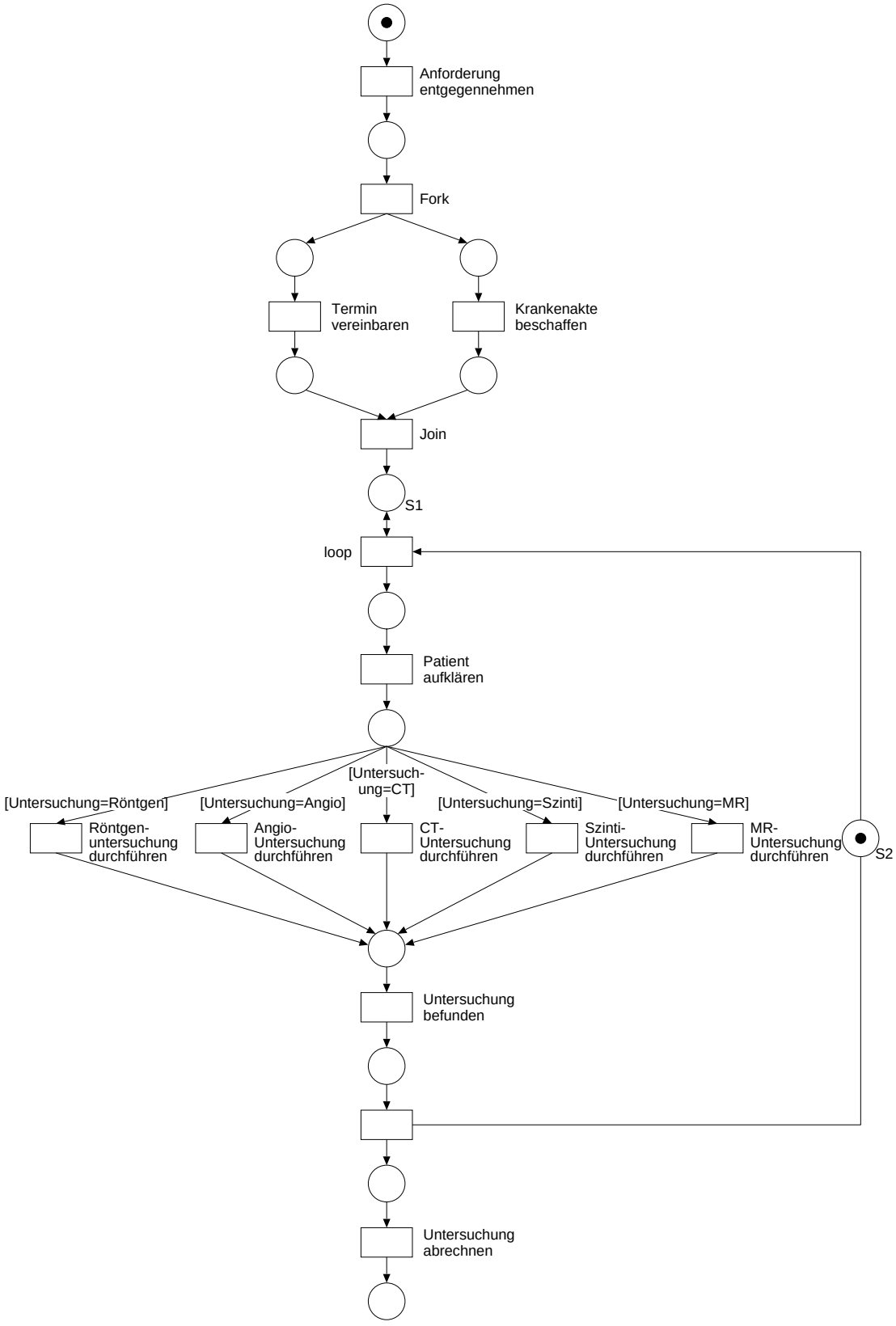


Abbildung 3.24: Bedingungs/Ereignis-Netz

von Analysen (u. a. auf Termination, Lebendigkeit, Verklemmung, Erreichbarkeit) [Baumgarten, 1996, S. 129ff] bereit und erlaubt auch eine Simulation des Prozeßablaufs.

Zur Modellierung mit Petri-Netzen wurden viele verschiedene Petri-Netz-Typen eingeführt, die sich in den Typen der Stellenmarkierungen, in den Anforderungen an die Kantengewichte und in den Schaltregeln der Transitionen unterscheiden. In *Bedingung/Ereignis-Netzen* (B/E-Netze) [Petri, 1962], [Baumgarten, 1996, S. 111] wird für die Stellen generell eine Stellenkapazität von 1 gefordert. Anonyme, untypisierte Marken zeigen die Gültigkeit einer durch Stellen modellierten Bedingung an. *Stellen/Transitions-Netze* (S/T-Netze) [Baumgarten, 1996, S. 77] verwenden ebenfalls anonyme Marken. Sie gestatten aber beliebige Stellenkapazitäten und Kantengewichte. Höhere Petrinetze wie *Prädikat/Transitions-Netze* [Genrich / Lautenbach, 1981], [Marx, 1998, S. 21ff] oder die hieraus abgeleiteten *Gefärbten Petri-Netze* [Jensen, 1998] verwenden unterschiedlich typisierte, individuelle Marken. Transitionen dieser Netztypen können nur dann schalten, wenn auf den adjazenten Stellen entsprechende Marken liegen bzw. Platz für erzeugte Marken bereit steht. An den Transitionen werden Schaltregeln über Variablen notiert, die durch Kantenanschriften an die Marken der Stellen gebunden werden. Die Schaltregeln bestehen aus Schaltbedingungen, die der Vorbedingung der Transition entsprechen (Guards) und Schaltwirkungen, durch die die Nachbedingungen beschrieben werden. Einen Überblick über grundlegenden Petri-Netzvarianten sowie über weitere Netzvarianten gibt auch [Baumgarten, 1996, S. 256ff].

In Abbildung 3.24 ist der Prozeß der radiologischen Untersuchung aus Beispiel 3.9 durch ein (erweitertes) Bedingungs/Ereignis-Netz, einschließlich seiner Startmarkierung beschrieben. Die Parallelbearbeitung der Prozesse *Termin vereinbaren* und *Krankenakte beschaffen* wird durch Einführung der zusätzlichen Transitionen *Join* und *Fork* modelliert. Die Verzweigung zur Durchführung der ausgewählten Untersuchungen wird durch *Wächter* (Guards) überwacht. Diese stellen sicher, daß nur genau eine der fünf Untersuchungs-Transitionen schalten kann. Zur Modellierung des Zusammenflusses der abweisenden Schleife ist sicherzustellen, daß die Transition *loop* sowohl beim ersten Durchlaufen, als auch bei den Iterationen schalten kann. Hierzu ist die Stelle *S2* in der Anfangsmarkierung bereits mit einer Marke belegt. Verbrauchte Marken werden nach Schalten von *loop* auf Stelle *S1* zurückgelegt.

Netzpläne. Die Planung, Steuerung und Überwachung von Abläufen kann ebenfalls auf Beschreibungsmittel des Netzparadigmas zurückgeführt werden. Auch die Verfahren der Netzplantechnik [DIN 69900, 1987], [Schwarze, 1994] basieren auf der Modellierung von zeitverbrauchenden Prozessen (in der Netzplan-Literatur i. allg. als Vorgänge bezeichnet), Ereignissen und den hierzwischen vorliegenden, ausschließlich schleifenfreien, Folgebeziehungen.

Bei der Projektplanung mit Netzplänen werden *vorgangsorientierte Netzpläne*, bei denen Prozesse und deren Folgebeziehungen betrachtet werden und *ereignisorientierte Netzpläne*, die die Ereignisse und deren Folgebeziehungen in den Vordergrund stellen, unterschieden. Vorgangsorientierte Netzpläne werden je nach der Darstellung der Prozesse weiter in Vorgangsknotennetze und in Vorgangspfeilnetze unterteilt.

Prozesse werden in *Vorgangsknotennetzen* durch Knoten beschrieben; die gerichteten Kanten modellieren deren Aufeinanderfolgen. Die Verwendung von Vorgangsknotennetzen geht auf die *Metra-Potential-Method (MPM)* [Roy, 1962], [DIVO, 1966] zurück, die zunächst zur Terminplanung und Überwachung beim Bau von Kernkraftwerken eingesetzt wurde.

In *Vorgangspfeilnetzen* werden Prozesse durch gerichtete Kanten modelliert, die zwischen Knoten verlaufen, die nicht näher betrachtete Anfangs- und Endereignisse der Prozesse beschreiben. Die Abbildung der Prozesse durch Kanten erfordert zur Modellierung nebenläufiger Prozesse und deren Synchronisation die Verwendung zusätzlicher Scheinprozesse, deren Bearbeitung keine Zeit erfordert. Diese Scheinprozesse erschweren sowohl die Erstellung wie auch das Verstehen dieser Netzplanvarianten. Vorgangspfeilnetze wurden im Rahmen der Entwicklung der *Critical-Path-Method (CPM)* [Kelley, 1961] als Planungs- und Überwachungsinstrument für das Chemie-Unternehmen Du Pont de Nemour & Co. eingeführt.

In ereignisorientierten Netzplänen werden Ereignisse als Knoten modelliert (*Ereignisknotennetze*), die Kanten beschreiben lediglich das Aufeinanderfolgen von Ereignissen und betrachten die Prozesse nicht. Ereignisknotennetze sind das Modellierungsmittel der *Program Evaluation and Review-Technique (PERT)* [PERT, 1958], [Malcolm et al., 1959], die im Auftrag der U. S.-Navy zur Koordination der Entwicklung von Waffensystemen entwickelt wurden.

Notiert werden Netzpläne durch gerichtete Graphen, bei denen die Knoten mit dem Ereignis- oder Prozeßbezeichner markiert werden. Die Folgebeziehungen werden durch gerichtete Kanten beschrieben. Prozeßknoten bzw. Prozeßpfeile werden häufig noch mit ihrer Dauer attributiert.

Durch die Folgebeziehungen können in Vorgangsknoten- und Ereignisknotennetzen auch (zeitliche) Abstände zwischen den Prozessen bzw. Ereignissen z. B. zur Modellierung von Warte- und Liegezeiten beschrieben werden. Hierzu werden die Kanten entsprechend mit der Wartezeit attributiert. In Vorgangsknotennetzen können sich diese Wartezeiten jeweils auf den Beginn oder das Ende der beiden adjazenten Prozesse beziehen. *Normalfolgen* beschreiben den zeitlichen Abstand zwischen dem Ende des ersten und dem Anfang des zweiten Vorgangs. *Anfangsfolgen* und *Endfolgen* modellieren jeweils die Abstände zwischen den Prozeßanfängen und -enden. Die Folgebeziehung zwischen Anfang des ersten und Ende des zweiten Prozesses wird als *Sprungfolge* bezeichnet.

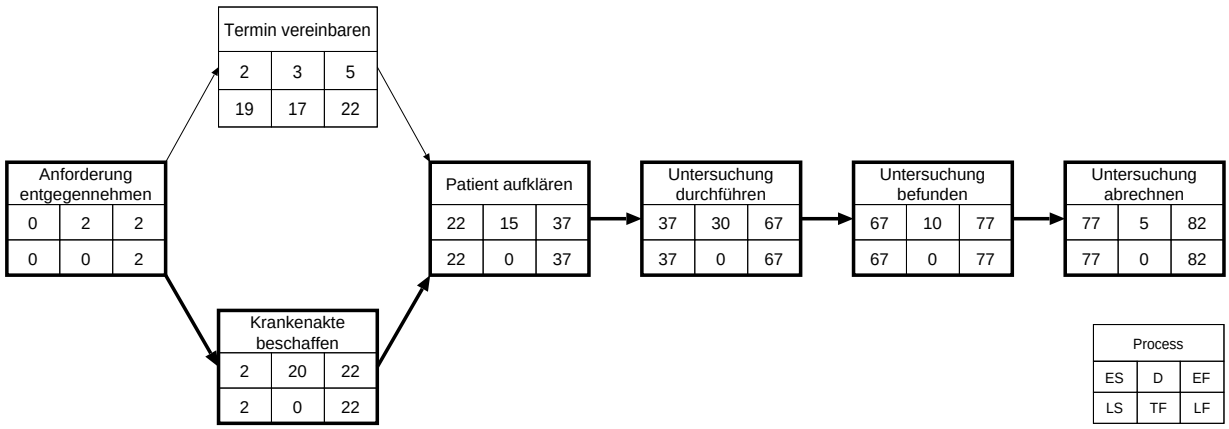


Abbildung 3.25: Vorgangsknotennetzplan

Abbildung 3.25 zeigt den Vorgangsknotennetzplan zur Durchführung der Radiologischen Untersuchung aus Beispiel 3.9, bei dem alle Prozesse zueinander in Normalfolge stehen. Da durch die Netzplantechnik lediglich sequenzielle und parallele Prozeßfolgen beschrieben (und analysiert) werden können, mü's hier die Schleife zur Durchführung einer weiteren Untersuchung und die Untersuchungsalternativen entfallen.

Die Netzplantechnik erlaubt es, Aussagen über das zeitliche Verhalten eines Prozesses zu machen. Aus der Zeitdauer (duration, *D*) der einzelnen Prozesse, der Folgebeziehungen und deren Verzögerung kann u. a. der zeitliche Aufwand des Gesamtprozesses oder die zeitliche Einordnung der Prozesse und Ereignisse errechnet werden. Für Prozesse kann bestimmt werden, zu welchem Zeitpunkt sie frühestens beginnen (earliest start time, *ES*) oder enden (earliest finish time, *EF*). In ähnlicher Weise können auch, ausgehend von einem Endzeitpunkt des Gesamtprozesses, für jeden Prozeß der späteste Zeitpunkt an dem die Prozeßbearbeitung beginnen (latest start time, *LS*) oder enden (latest finish time, *LF*) muß, errechnet werden um diesen Endtermin zu erreichen. Für Ereignisse können analog früheste (earliest time, *ET*) und späteste (latest time, *LT*) Zeitpunkte für deren Eintreffen bestimmt werden. Aus diesen Zeitpunkten kann weiter errechnet werden, um welchen Zeitraum ein Prozeß oder das Eintreffen eines Ereignis verzögert werden kann, ohne den angestrebten Endtermin zu gefährden. Diese Pufferzeit (total float, *TF*) berechnet sich aus der Differenz der jeweiligen spätesten und frühesten Zeitpunkte. Prozesse, deren Pufferzeit gleich 0 ist, werden *kritische Prozesse* genannt, weil ihre Verzögerung auch den Gesamtprozeß verzögert. Ein Pfad zwischen Anfang und Ende des Gesamtprozesses, der ausschließlich kritische Vorgänge enthält, wird als *kritischer Pfad* bezeichnet. Weitere Maße und Analysemöglichkeiten werden z. B. in [Zimmermann, 1971], [Schwarze, 1994] definiert.

Für den einfachen Netzplan aus Abbildung 3.25 wurden die frühesten und spätesten Zeitpunkte sowie die Pufferzeiten bestimmt und in den Prozeßdarstellungen notiert. Hierzu wurde davon ausgegangen, daß der Gesamtprozeß zum Zeitpunkt 0 beginnt und zum frühest möglichen Zeitpunkt (82) endet. In diesem Beispiel müssen außer dem Prozeß *Termin vereinbaren* alle anderen Prozesse genau zu ihren geplanten Zeitpunkten begonnen bzw. beendet werden. Diese bilden den kritischen Pfad.

Neben den graphbasierten Darstellungsformen der Vorgangs-Knoten-, der Vorgangs-Pfeil und der Ereignisknotennetzpläne werden in der Netzplantechnik auch tabellenartige Beschreibungsmittel verwendet. In *Balkendiagramme* oder *Ganttcharts* werden die Prozesse in einer Spalte untereinander geschrieben. In einer weiteren Spalte wird zu jedem Prozeß ein Balken, dessen Länge der Prozeßdauer entspricht, in ein Zeitraster eingetragen. Ergänzt werden können weitere Spalten z. B. zur Festlegung der beteiligten Stellen oder der benötigten Maschinen. In klassischen Balkendiagrammen werden die Folgebeziehungen zwischen den Prozessen nicht notiert, so daß die Abhängigkeiten der Teilprozesse nicht veranschaulicht werden können. Werkzeuge zur Unterstützung des Projektmanagement verwenden daher *erweiterte Balkendiagramme*, bei denen vom Balkenende Verbindungslinien zum Beginn der Balken direkt abhängiger Prozesse gezogen werden können.

Forderungen an das Referenz-Metaschema

Wesentliche Modellierungskonstrukte der Sprachen des Netzparadigmas sind Prozesse, Ereignisse und Folgebeziehungen zwischen diesen. Das Referenz-Metaschema muß daher Möglichkeiten zur Abbildung dieser Konzepte bereitstellen. Für die Folgebeziehungen sind auch die verschiedenen Möglichkeiten der Verzweigung und Zusammenführung der Kontrollflüsse zu berücksichtigen. Modellierungen in den Sprachen des Netzparadigmas nehmen häufig auch Bezug auf Konzepte anderer Paradigmen (z. B. Aufgabenträger oder Datenbezüge). Daher sind auch im Referenz-Metaschema entsprechende Querbezüge vorzusehen.

3.4.4 Visuelle Modellierungssprachen des Kontrollflußparadigmas

Die Beschreibungsmittel des Kontrollflußparadigmas verfolgen ebenso wie die Sprachen des Netzparadigmas (vgl. Kapitel 3.4.3) die Darstellung der Ablaufstrukturen in Organisationen und Softwaresystemen. Im Gegensatz zu den Sprachen des Netzparadigmas, die nahezu beliebige Folgestrukturen zulassen, erlauben die visuellen Modellierungssprachen des Kontrollflußparadigmas ausschließlich die Beschreibung *regulär strukturierte* Abläufe.

In den Diskussionen um die strukturierte Programmierung (vgl. u. a. [Dijkstra, 1968], [Knuth, 1974]) stellte sich heraus, daß die Programmentwicklung bis auf wenige Ausnahmen ausschließlich auf den regulären Kontrollstrukturen der Sequenz, der Iteration und der Verzweigung aufsetzen sollte. Für die visuellen Sprachen zur Ablaufmodellierung ergab sich hieraus die Forderung nach Beschreibungsmitteln, die ausschließlich diese Modellierungsmittel unterstützen und insbesondere schwer nachvollziehbare, beliebige Folgestrukturen verbieten.

Im Mittelpunkt der Beschreibung durch die Sprachen des Kontrollflußparadigmas stehen auch hier *Aktivitäten* (Aktionen, Anweisungen, Prozesse etc.) die mittels *Sequenzenbildung*, *Verzweigungen* und *Iterationen* zu komplexeren Aktivitäten zusammengefaßt werden. Einige Sprachvarianten erlauben darüber hinaus noch die Komposition von Aktivitäten durch *Parallelbearbeitung* und bieten Möglichkeiten zur Verfeinerung von Aktivitäten. Diese Konzepte werden durch Nassi-Shneiderman-Diagramme alias Struktogramme, durch Entscheidungstabellen, durch Jackson-Diagramme, durch Pseudo Code oder durch Warnier-Orr-Diagramme unterstützt.

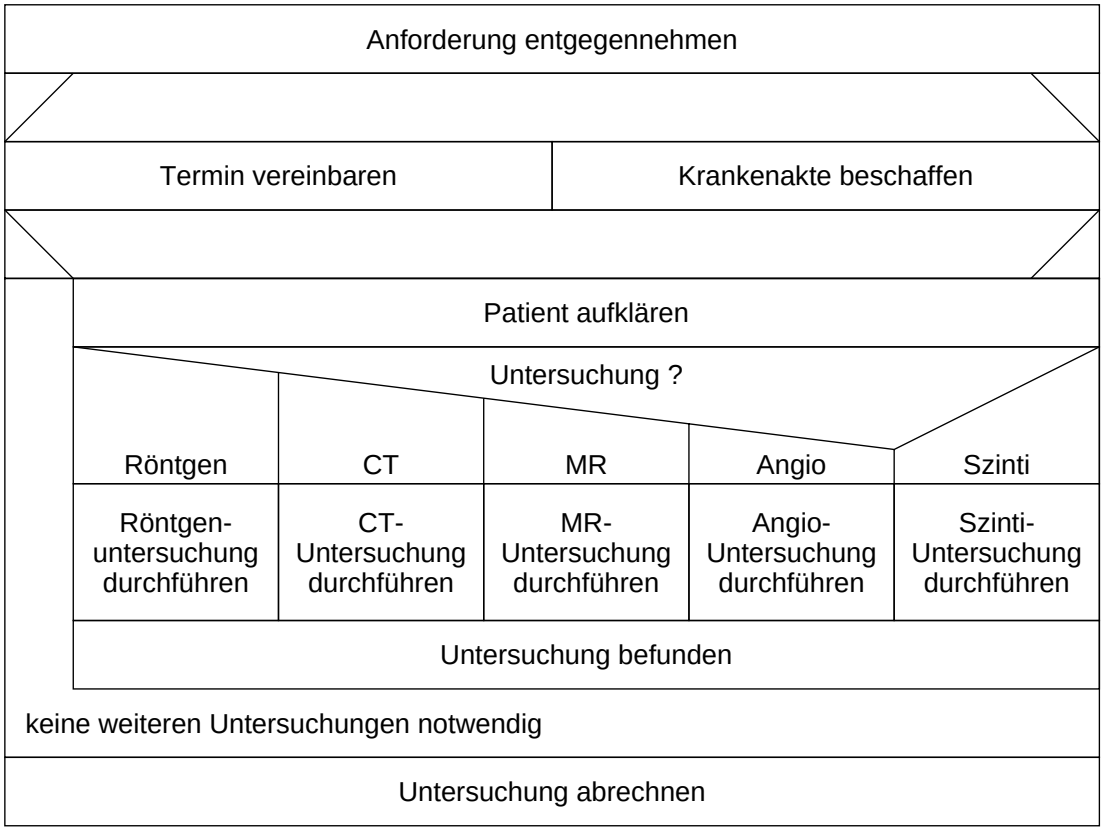


Abbildung 3.26: Nassi-Shneiderman-Diagramm

Notationelle Grundform: Nassi-Shneiderman-Diagramm

Zur Modellierung regulärer Strukturen durch Nassi-Shneiderman-Diagramme [Nassi / Shneiderman, 1973], [DIN 66261, 1985] werden elementare Aktivitäten in mit dem Aktivitätenbezeichner attributierten Rechtecken notiert. Die Komposition von Aktivitäten erfolgt in weiteren Rechtecken, so daß durch die Ineinanderschachtelung der Rechtecke die Blockstrukturierung der atomaren und zusammengesetzten Aktivitäten sichtbar wird. Zur Darstellung der Sequenzen werden die Rechtecke direkt untereinander angeordnet. In Verzweigungen wird das Auswahlkriterium in einem Dreieck notiert unter dem die Alternativen nebeneinander gestellt werden. Iterationen werden durch ein die iterierte Aktivität umgebendes Rechteck dargestellt, das zusätzlich das Iterationskriterium enthält. Parallel zu bearbeitende Aktivitäten werden ebenfalls nebeneinander aufgeführt. Die Parallelbearbeitung wird durch umgebende Trapeze kenntlich gemacht. Abbildung 3.26 zeigt die Modellierung der radiologischen Untersuchung aus Beispiel 3.9 als Nassi-Shneiderman-Diagramm.

Notationelle Varianten

Alternative Darstellungen zur Beschreibung regulär strukturierter Folgebeziehungen, die die baumartige Struktur der Zerlegung der Gesamtaktivität in Sequenzen, Verzweigungen, Iterationen, Parallelbearbeitungen und atomare Aktivitäten auch optisch herausstellen, bieten auch *Jackson-Diagramme* [Jackson, 1975], [Balzert, 1996a, S. 127] und Warnier-Orr-Diagramme [Warnier, 1974] [Schulz, 1988, S. 137].

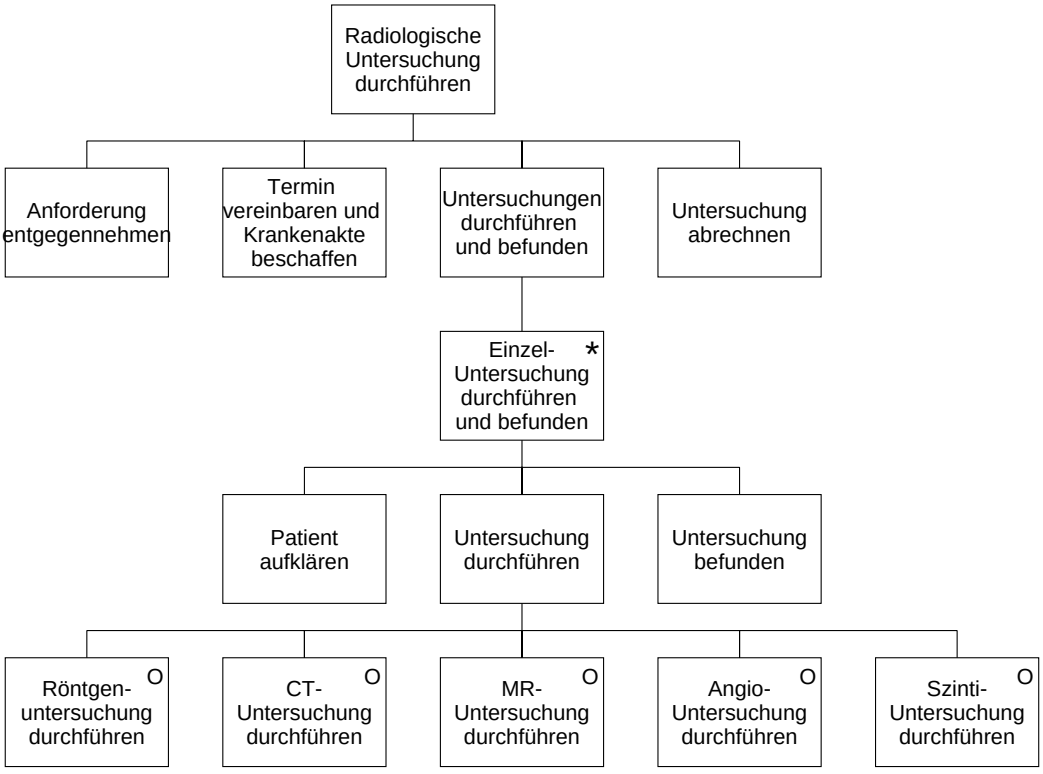


Abbildung 3.27: Jackson-Diagramm

Sowohl in Jackson-Diagrammen (vgl. Abbildung 3.27) als auch in Warnier-Orr-Diagrammen (vgl. Abbildung 3.28) werden für zusammengesetzte Aktivitäten eigene graphische Repräsentationen verwendet, die mit ihren Komponenten durch Kanten bzw. geschweifte Klammern verbunden sind. Die verschiedenen Arten der Komposition von Aktivitäten werden durch Annotationen in den Knoten der Jackson-Diagramme („O“ für Alternativen und „$*$“ für Iterationen) herausgestellt. Iterationshäufigkeiten werden an den Klammern der Warnier-Orr-Diagramme und Alternativen durch „$\oplus$“ zwischen den Aktivitäten notiert. Beide Darstellungsformen bieten keine Mittel zur Modellierung von Parallelabläufen. Jackson- und Warnier-Orr-Diagramme wurden sowohl zur Beschreibung regulär strukturierter Prozeßabläufe als auch zur Modellierung regulärer Datenstrukturen entwickelt. Als Mittel zur Datenbeschreibung sind sie daher auch dem Objekt-Beziehungsparadigma (vgl. Kapitel 3.5.3) zuzuordnen.

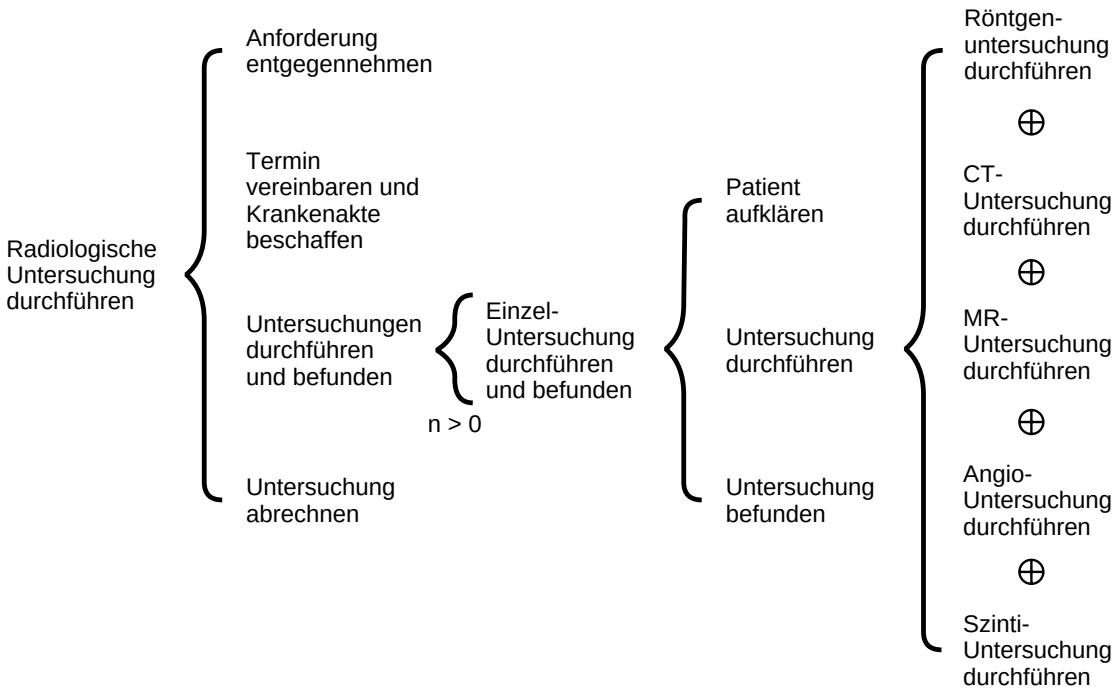


Abbildung 3.28: Warnier-Orr-Diagramm

Neben graphischen Darstellungen werden auch textuelle Notationen zur regulären Ablaufbeschreibung verwendet. Ein an imperative Programmiersprachen angelehnter Pseudo-Code mit blockstrukturierenden Sprachmitteln für Sequenzen, Iterationen, Verzweigungen und Parallelbearbeitungen wird in Abbildung 3.29a verwendet. *Entscheidungstabellen* [Schmidt, 1989, S. 327ff], [Balzert, 1996a, S. 222ff] können als tabellarische Darstellungen des Kontrollflußparadigmas aufgefaßt werden, die ausschließlich die Alternativenbildung abbilden. Eine Entscheidungstabelle zur Auswahl der Untersuchungsalternativen der radiologischen Untersuchungen ist in Abbildung 3.29b aufgeführt. Hierin ist auch angezeigt, daß eine angiographische Untersuchung[8] auch eine Röntgen-Untersuchung umfaßt.

Zur Unterstützung der regulär strukturierten Ablaufmodellierung können auch Programmablaufpläne [Chapin, 1970], [DIN 66001, 1983] eingesetzt werden. Diese stellen u. a. auch Mittel zur

[8] Röntgenographische Darstellung von Blutgefäßen mit Hilfe injizierter Kontrastmittel.

```
begin
Anforderung entgegennehmen;
beginPar
      Termin vereinbaren || Krankenakte beschaffen
endPar;
repeat
      Patient aufklären ;
      case Untersuchung
            Röntgen :
              Röntgenuntersuchung durchführen;
            Angio :
              Angio-Untersuchung durchführen;
            CT :
              CT-Untersuchung durchführen;
            Szinti :
              Szinti-Untersuchung durchführen;
            MR :
              MR-Untersuchung durchführen;
      endcase;
      Untersuchung befunden;
until alle notwendigen Untersuchungen durchgeführt;
Untersuchung abrechnen;
end
```

a) PseudoCode

	R1	R2	R3	R4	R5
Röntgen	J	N	N	N	N
CT	N	J	N	N	N
MR	N	N	J	N	N
Angio	N	N	N	J	N
Szinti	N	N	N	N	J
Röntgenuntersuchung durchführen	X			X	
CT-Untersuchung durchführen		X			
MR-Untersuchung durchführen			X		
Angio-Untersuchung durchführen				X	
Szinti-Untersuchung durchführen					X

b) Entscheidungstabelle

Abbildung 3.29: Pseudo-Code und Entscheidungstabelle

regulären Ablaufstrukturierung bereit, fordern deren Verwendung jedoch nicht und lassen beliebige Folgestrukturen zu, so daß dieses Beschreibungsmittel unter das Netzparadigma (vgl. Kapitel 3.4.3) eingeordnet wurde.

Forderungen an das Referenz-Metaschema

Die Beschreibung von Ablaufstrukturen durch die Sprachen des Kontrollflußparadigmas erfolgt entlang der Teilaktivitäten, die mittels Sequenzen, Alternativen, Iterationen und Parallelbearbeitung regulär strukturiert werden. Diese Konzepte muß das Referenz-Metaschema bereitstellen.

3.5 Visuelle Modellierungssprachen der Objektsicht

In der Objektsicht werden die statischen Systemaspekte betrachtet. Hierzu werden Organisationen oder Softwaresysteme ausgehend von den *Objekten* beschrieben, die dieses System bilden. Objekte (oder Entities) [Chen, 1976], [Rumbaugh et al., 1999] stellen wesentliche, unterscheidbare Dinge innerhalb des Systems dar, die eine eigenständige Identität aufweisen. Neben der Objektidentität können Objekte weiter durch Attribute und Operationen charakterisiert werden. Attribute fassen Informationen über Objekte zusammen und definieren hierüber deren Zustand. Operationen beschreiben Handlungen oder Transformationen, die Objekte ausführen können. Sie charakterisieren das nach außen sichtbare Verhalten der Objekte. Das Zusammenspiel der Objekte wird durch hierzwischen vorliegende *strukturelle Beziehungen* oder *Interaktionsbeziehungen* herausgestellt. Strukturelle Beziehungen können durch Attribute konkretisiert werden.

Die Beschreibung eines Systems kann sowohl entlang konkreter Objekte und deren strukturellen Beziehungen bzw. deren Interaktionsbeziehungen auf Instanzenebene als auch auf einer schematischen Ebene erfolgen. Bei der Betrachtung auf schematischer Ebene werden Mengen gleichar-

tiger Objekte bzw. gleichartiger Beziehungen zu Klassen zusammengefaßt. Die Systembeschreibung abstrahiert dann von konkreten Objekten und Beziehungen und stellt die Zusammenhänge generell für alle möglichen Instanzen der modellierten Klassen heraus.

Die Darstellung statischer, struktureller Objektzusammenhänge auf Instanzenebene erfolgt durch die Sprachen des *Objekt-Instanzparadigmas*. Durch die Beschreibungsmittel des *Objekt-Interaktionsparadigmas* werden die Interaktionsbeziehungen entlang der zwischen Objekten ausgetauschten Nachrichten dargestellt. Visuellen Modellierungssprachen zur Beschreibung von Objektstrukturen auf Schemaebene stellen die Sprachen des *Objekt-Beziehungsparadigmas* bereit.

3.5.1 Visuelle Modellierungssprachen des Objekt-Instanzparadigmas

Zur Modellierung der statischen Sicht eines Systems entlang konkreter oder anonymer Systembestandteile werden die Sprachen des Objekt-Instanzparadigmas verwendet. Im Gegensatz zu den Beschreibungsmitteln des Objekt-Interaktionsparadigmas (vgl. Kapitel 3.5.2), mit denen die Interaktionen zwischen den Objekten des Systems dargestellt werden, beschreiben die Sprachen des Objekt-Instanzparadigmas einen möglichen Zustand des Systems zu einem festen Zeitpunkt. Hierzu werden die Objekte des Systems mit ihren Attribut-Eigenschaften und ihren Beziehungen skizziert.

Die Beschreibung struktureller Systemzusammenhänge entlang konkreter oder anonymer Objekte und Beziehungen erfolgt durch Objektdiagramme oder Instanzdiagramme.

Eingesetzt werden diese Sprachen auch als Grundlage zur Erstellung von Beschreibungen des Objekt-Beziehungsparadigmas (vgl. Kapitel 3.5.3) oder zur Darstellung und Erklärung solcher Modellierungen. Die Sprachen des Objekt-Instanzparadigmas beschreiben hierzu exemplarische Zusammenhänge, die mit den Beschreibungsmitteln des Objekt-Beziehungsparadigmas schematisch beschrieben werden.

Notationelle Grundform: Objektdiagramm

Die Darstellung von Modellierungen nach dem Objekt-Instanzparadigma erfolgt durch attributierte und typisierte Graphen. Knoten beschreiben hierbei konkrete oder anonyme Objekte und Kanten stellen die strukturellen Beziehungen zwischen den Objekten heraus. Zur Darstellung ähnlicher Objekte bzw. Beziehungen können diese typisiert werden. Attributeigenschaften der Objekte und Beziehungen werden in Attribut-Wertpaaren notiert.

Der zur Modellierung des Referenz-Metaschemas in Kapitel 6 verwendete *EER/GRAL*-Ansatz zur graphbasierten Konzeptmodellierung stellt auch ein visuelle Beschreibungsmittel zur Darstellung struktureller Instanzzusammenhänge bereit. Dieses Beschreibungsmittel wird im folgenden als notationelle Grundform des Objekt-Instanzparadigmas vorgestellt. Eine Einführung in die Formalisierung dieses Ansatzes findet sich in Kapitel 5.2.1.

Objekte werden durch Rechtecke mit abgerundeten Ecken dargestellt. Im oberen Teil des Rechtecks können optional Objektbezeichner und Objekttyp notiert werden. Im unteren Teil können Attribut-Wertpaare zur näheren Charakterisierung des Objekts aufgeführt werden. Die Beziehungen zwischen den Objekten werden durch Pfeile notiert, die ebenfalls mit einem Bezeich-

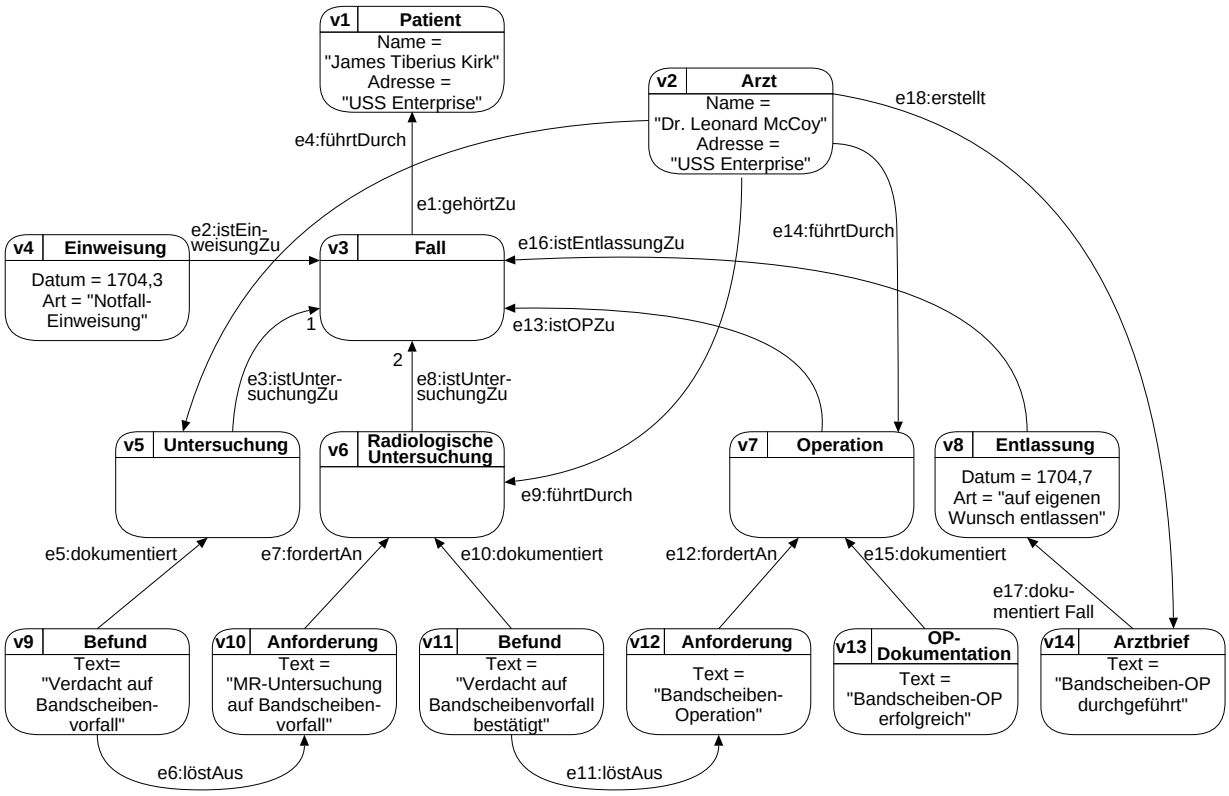


Abbildung 3.30: Objektdiagramm

ner, ihrer Typangabe und einer Attributierung versehen sein können. Attribute zu Beziehungen werden in einem Oval notiert. Ist die Reihenfolge der inzidenten Kanten zu einem Knoten von Bedeutung, so kann diese Anordung durch Nummerierung der Kanten notiert werden. Schlingen treten in solchen Reihenfolgen zweimal auf, nämlich zum einen als aus dem Knoten ausgehende und zum anderen als in den Knoten eingehende Kante.

Beispiel 3.10 (Objektdiagramm eines medizinischen Falls)

In Abbildung 3.30 ist ein *EER/GRAL*-Objektdiagramm zu Beschreibung eines konkreten medizinischen Falls dargestellt, der einen Krankenhausaufenthalt eines Patienten zwischen Aufnahme und Entlassung zusammenfassend dokumentiert (vgl. [Haines, 1996]). Der hier modellierte *Fall* (v3) bezieht sich auf den *Patienten* „James Tiberius Kirk“ (v1) und umfaßt zwei *Untersuchungen* (v5, v6) und eine *Operation* (v7) der Bandscheibe. Die Reihenfolge der beiden Untersuchungen ist hier durch Anordnung der *IstUntersuchungZu*-Kanten ausgedrückt. Die *Untersuchungen* werden durch *Befunde* Verdacht auf Bandscheibenvorfall (v9) und dessen Bestätigung (v11) nach einer *Radiologischen Untersuchung* dokumentiert. Diese Befunde *lösen* die ebenfalls dokumentierte *Operation* der Bandscheibe aus. Durchgeführt werden diese Behandlungen von dem *Arzt* „Dr. Leonard McCoy“ (v2). □

Notationelle Varianten

Objektdiagramme, wie sie beispielsweise in der Objekt Modeling Technique (OMT) [Rumbaugh et al., 1991, S. 21ff] oder in der Unified Modeling Language (UML) [Booch et al., 1999, S. 195ff] eingeführt werden, unterscheiden sich von der Darstellung in Abbildung 3.30 nur geringfügig. In OMT werden die Objekte ebenfalls durch Rechtecke mit abgerundeten Ecken dargestellt, die den jeweiligen Objekttyp (fettgedruckt, in Klammern) und die Attributwerte enthalten. Beziehungen werden in OMT-Instanzdiagrammen durch ungerichtete Linien notiert, die aber ebenfalls mit dem Beziehungstyp annotiert sein können. Gegenüber *EER/GRAL*-Objektdiagrammen, die ausschließlich binäre Beziehungen zwischen Objekten erlauben, unterstützen sowohl OMT-Instanzdiagramme wie auch UML-Objektdiagramme die Darstellung von Beziehungsinstanzen beliebiger Arität. Diese werden durch Hyperkanten notiert, die durch eine Raute ausgezeichnet sind. In UML-Objektdiagrammen werden Objekte durch Rechtecke notiert, in dem die Objekttypen unterstrichen dargestellt werden (vgl. auch die Notation der Interaktionsdiagramme, Kapitel 3.5.2). Attribute werden in UML-Objektdiagrammen analog zu *EER/GRAL*-Objektdiagrammen durch Attribut-Wertepaare beschrieben. Die Beschreibung von Beziehungsinstanzen erfolgt analog zu OMT durch ungerichtete Linien, die jedoch i. allg. nicht bezeichnet sind.

Forderungen an das Referenz-Metaschema

Mit den visuellen Sprachen des Objekt-Instanzparadigmas werden mögliche strukturelle Zusammenhänge der ein System ausmachenden Objekte und deren Beziehungen beschrieben. Das Referenz-Metaschema muß zur Abbildung dieser Beschreibungsmittel die Konzepte zur Modellierung von Objekten, Beziehungen und deren Typ- und Attributeigenschaften bereitstellen. Da durch die Modellierungssprachen des Objekt-Instanzparadigmas auch die Typen der modellierten Objekte und Beziehungen notiert werden können, muß das Referenz-Metaschema auch diese Querbezüge zur Schema-Ebene abbilden.

3.5.2 Visuelle Modellierungssprachen des Objekt-Interaktionsparadigmas

Die Modellierungssprachen des Objekt-Interaktionsparadigmas dienen zur Beschreibung vonn Interaktionen zwischen Objekten. Objekte, die konkrete oder anonyme Instanzen von Klassen, Schnittstellen oder Komponenten beschreiben, interagieren miteinander durch den Austausch von Nachrichten [Booch et al., 1999, S. 243].

Eingesetzt werden diese Beschreibungsmittel zur Konkretisierung von Anwendungsfällen und zur Beschreibung der Ablauffolge von Methodenaufrufen. Während durch die Anwendungsfalldiagramme des Datenflußparadigmas (vgl. Kapitel 3.4.1) die Einbettung eines Anwendungsfalls in seine Systemumgebung beschrieben wird, wird der Anwendungsfall durch die Beschreibungsmittel des Objekt-Interaktionsparadigmas exemplarisch in seinem inneren Verhalten modelliert. Hierzu wird das System aus der Sicht der Instanzen der Systemkomponenten dargestellt. Gegenüber den Sprachen des Objekt-Instanzparadigmas (vgl. Kapitel 3.5.1), die mögliche statische

Zusammenhänge beschreiben, betonen die Beschreibungsmittel des Interaktionsparadigmas das dynamische Verhalten der Objekte. Dieses wird durch Folgen von Nachrichten, die zwischen den Objekten ausgetauscht werden, szenarioartig beschreiben.

Zur Darstellung dieser Zusammenhänge werden Sequenzdiagramme, Ereignisflußdiagramme, Ereignispfaddiagramme, Interaktionsdiagramme, Kollaborationsdiagramme, Message Sequence Charts und Objektdiagramme verwendet. Interaktionsdiagramme werden insbesondere in objektorientierten Modellierungsansätzen ([Rumbaugh et al., 1991, S. 173ff], [Booch, 1994], [Booch et al., 1999, S. 243ff]) eingesetzt. Sie gehen auf die Beschreibung von Szenarien in Object-Oriented Software-Engineering (OOSE) [Jacobson et al., 1993, S. 215ff] zurück. Der Begriff Interaktionsdiagramm wird in der Unified Modeling Language (UML) als Oberbegriff für Kollaborationsdiagramme und Sequenzdiagramme verwendet. [Booch, 1994, S. 217ff] bezieht diesen Begriff ausschließlich auf Sequenzdiagramme. Als Message Sequence Charts [ITU, 1996], [Rudolph et al., 1996] werden diese Diagrammformen im Rahmen der SDL (Specification and Design Language) [ITU, 1988], [Braek / Haugen, 1993] zur Beschreibung verteilter Systeme verwendet.

Notationelle Grundform: Sequenzdiagramm

Durch Sequenzdiagramme, die bei [Rumbaugh et al., 1991] als Ereignispfaddiagramme (Event-Trace) bezeichnet werden, wird die zeitliche Abfolge der Interaktionen zwischen Objekten tabellenartig beschrieben. Die für den modellierten Anwendungsfall relevanten Objekte werden ähnlich zu Tabellenüberschriften oberhalb einer vertikalen Linie notiert. Diese Linie symbolisiert die Lebenslinie des Objekts und repräsentiert die Existenz des Objekts in der Zeit. Nachrichten, die zwischen zwei Objekten ausgetauscht werden, werden durch Pfeile zwischen den jeweiligen Lebenslinien notiert. Die Reihenfolge, in der die Nachrichten ausgesendet bzw. empfangen werden, ist an den Positionen auf den Lebenslinien abzulesen, die hierzu als Zeitachsen aufgefaßt werden.

Objekte werden durch Rechtecke visualisiert. Zur Unterscheidung der Objektdarstellungen von Klassendarstellungen werden in Sequenzdiagrammen die Objektnamen unterstrichen. Objektnamen bestehen aus einem optionalen Objektbezeichner und dem Objekttyp. Wird auf den Objektbezeichner verzichtet, wird durch das Objektsymbol ein anonymes Objekt beschrieben.

Die Pfeile zur Darstellung der ausgetauschten Nachrichten sind durch den Namen der Nachricht und optionaler Übergabeparameter markiert. [Rumbaugh et al., 1999, S. 336] unterscheidet vier verschiedene Nachrichtentypen: Pfeile mit einem Winkelsymbol als Pfeilspitze beschreiben einen *einfachen Kontrollfluß*, der Aktivierungsfolgen der einzelnen Objekte modelliert. *Aufrufe von Methoden* werden durch Pfeile mit gefüllten Spitzen beschrieben. Methodenaufrufe können hierbei auch hintereinander an mehrere Objekte gesendet werden. Weitere (synchrone) Methodenaufrufe eines Objekts sind aber erst nach vollständiger Bearbeitung dieser Folgen von Methodenaufrufe erlaubt. Spezielle Nachrichten werden für Konstruktoren und Destruktoren verwendet. Hierzu werden die Stereotypen „`create`“ und „`destroy`“ verwendet. Nachrichten zum Methodenaufruf korrespondieren mit *Rückgabe-Nachrichten*, die durch gestrichelte Pfeile dargestellt werden. Je nach Modellierungskontext kann durch eine Rückgabe auch das wertliefernde Objekt zerstört werden. Das Zerstören eines Objekts wird jeweils durch ein „X“ auf der Lebenslinie modelliert. *Asynchrone Nachrichtenbeziehungen* werden durch Pfeile mit einseitiger Pfeilspitze dargestellt.

Die Übermittlung von Nachrichten kann auch von Bedingungen (Guards) abhängig gemacht werden. Ebenso werden auch Iterationen von Nachrichten unterstützt. Die hierzu nötigen Bedingungen werden vor dem Bezeichner der jeweiligen Nachricht notiert[9].

Ein Beispiel für ein Sequenzdiagramm, das eine radiologische Untersuchung beschreibt, ist in Abbildung 3.31 dargestellt.

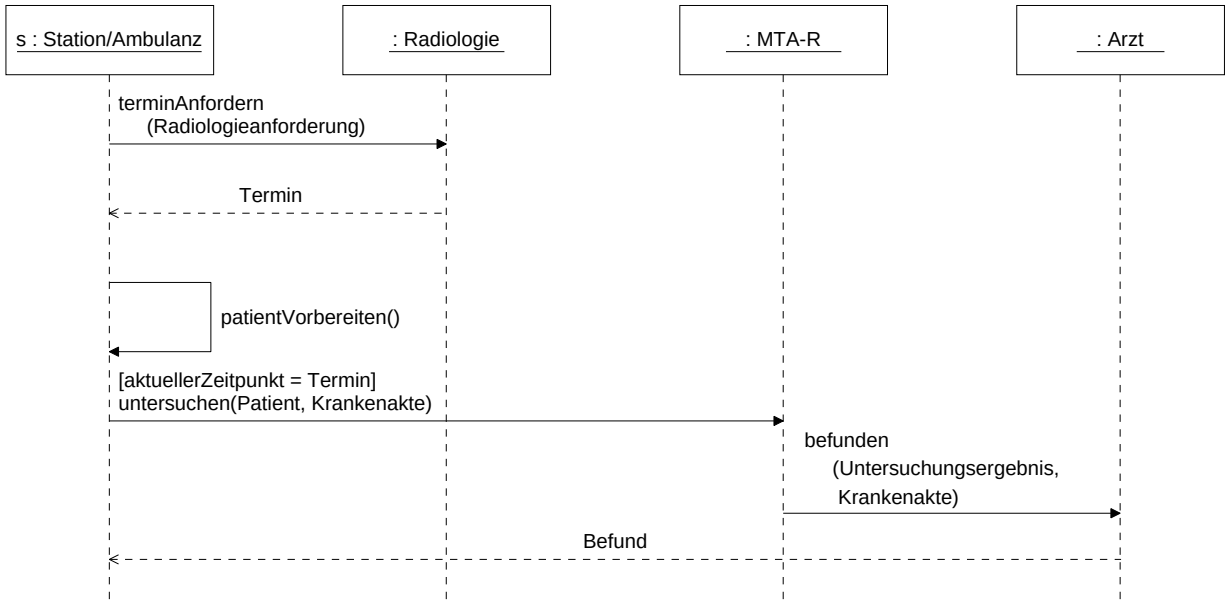


Abbildung 3.31: Sequenzdiagramm

Beispiel 3.11 (Sequenzdiagramm einer radiologischen Untersuchung)

Das Sequenzdiagramm in Abbildung 3.31 skizziert den Anwendungsfall *Anforderung bearbeiten* bezogen auf eine radiologische Untersuchung aus Abbildung 3.18. Hierzu interagieren die *Station/Ambulanz* s, sowie die anonymen Objekte *Radiologie*, *MTA-R* und *Arzt*. Das Objekt s fordert bei der *Radiologie* zunächst einen *Termin* an. Anschließend ist der Patient vorzubereiten. Hierzu schickt sich s selbst die Nachricht *patientVorbereiten()*. Ist der zuvor ermittelte Untersuchungszeitpunkt eingetroffen, wird die Untersuchung des Patienten ausgelöst. Das Absenden der Nachricht *untersuchen(Patient, Krankenakte)* wird über das Prädikat [*aktuellerZeitpunkt=Termin*] überwacht. Nach Erstellen der Röntgenaufnahmen werden diese durch einen *Arzt befundet* und der *Befund* an die anfordernde *Station/Ambulanz* übermittelt. □

Notationelle Varianten

Darstellungsvarianten für Sequenzdiagramme unterscheiden sich auch hier lediglich in ihrer konkreten Syntax und in ihrer Ausdrucksfähigkeit. So notiert [Fowler / Scott, 1998, S. 108] asynchrone Nachrichtenübermittlungen durch Pfeile mit halbausgefüllten Pfeilspitzen. [Rumbaugh

[9] Eine konkrete Notation hierfür wird in UML nicht vorgesehen [Rumbaugh et al., 1999, S. 338], Prädikate für Guards und Iterationen können hier durch beliebige Textfragmente angegeben werden.

et al., 1991, S. 174] verwendet in den Ereignispfaddiagrammen der Object Modeling Technique (OMT) ausschließlich einfache Nachrichten zur Beschreibung der Aktivierungsfolgen von Objekten. Auch kennt OMT keine Guards und Iterationen.

Kollaborationsdiagramme. Während bei der Modellierung mit Hilfe von Sequenzdiagrammen die zeitliche Reihenfolge des Nachrichtenaustauschs zwischen den Objekten besonders hervorgehoben wird, stellen *Kollaborationsdiagramme* [Rumbaugh et al., 1999, S. 203ff], Ereignisflußdiagramme [Rumbaugh et al., 1991, S. 175ff] und Objektdiagramme [Booch, 1994, S. 208ff]) eher die strukturellen Zusammenhänge zwischen den interagierenden Objekten in den Mittelpunkt der Betrachtung.

In Kollaborationsdiagrammen wird der Interaktionszusammenhang zwischen Objekten durch ungerichtete Graphen beschrieben, in denen die Objekte die Knoten bilden. Zwei Objekte sind dann zueinander adjazent, wenn zwischen ihnen eine Interaktionsbeziehung besteht. Zur Darstellung der jeweils ausgetauschten Nachrichten werden die Kanten durch in Datenflußrichtung ausgerichtete Pfeile, die analog zu den Sequenzdiagrammen mit den Nachrichten markiert sind, annotiert. Die Modellierung der Aktivierungssequenzen der Nachrichten erfolgt analog zur Sequenzialisierung der SADT-Aktivitätendiagramme (vgl. Seite 62) durch Nummerierung der Nachrichten.

Kollaborationsdiagramme und Sequenzdiagramme beschreiben semantisch äquivalente Informationen. Durch die graphische Anordnung wird jedoch in Sequenzdiagrammen der zeitliche Ablauf und in Kollaborationsdiagrammen der strukturelle Zusammenhang zwischen den Objekten deutlicher herausgestellt. Das Kollaborationsdiagramm, das dem Sequenzdiagramm aus Abbildung 3.31 entspricht, ist in Abbildung 3.32 dargestellt.

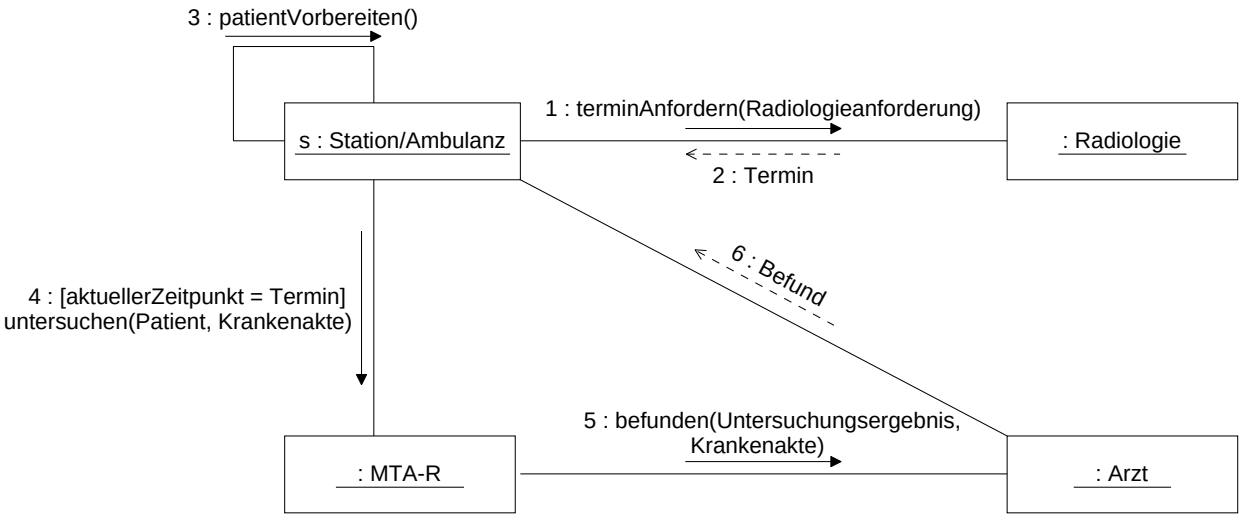


Abbildung 3.32: Kollaborationsdiagramm

Wie auch für Ereignispfaddiagramme der OMT unterscheidet sich die konkrete Notation der Ereignisflußdiagramme [Rumbaugh et al., 1991, S. 175] geringfügig von der Darstellung der Kollaborationsdiagramme. In Ereignisflußdiagrammen werden die Nachrichten durch gemäß ihrer Datenflußrichtung gerichtete Kanten beschrieben. Diese sind aber auch hier auf einfache, kontrollflußartige Nachrichtenbeziehungen eingeschränkt. Anstelle von Rechtecken zur Darstel-

lung der Objekte werden in Objektdiagrammen nach [Booch, 1994] Wolkensymbole verwendet. Dieser Dialekt unterstützt ferner die graphische Unterscheidung verschiedener Synchronisationen (einfach, synchron, asynchron, Abbruch im Fehlerfall) der ausgetauschten Nachrichten und Sichtbarkeiten der interagierenden Objekte (global, lokal, als Parameter).

Forderungen an das Referenz-Metaschema

Beschreibungsinhalt der visuellen Modellierungssprachen des Objekt-Interaktionsparadigmas ist die Darstellung von Interaktionsszenarien. Diese Interaktionsszenarien beziehen sich auf den Austausch von Nachrichten zwischen Objekten. Das Referenz-Metaschema muß daher für diese Modellierungsmittel Konzepte zur Beschreibung der Szenarien und der hieran beteiligten Objekte und Nachrichten bereitstellen. Ebenfalls sind auch verschiedene Nachrichtentypen zu berücksichtigen.

3.5.3 Visuelle Modellierungssprachen des Objekt-Beziehungsparadigmas

Im Gegensatz zum Objekt-Instanzparadigma (vgl. Kapitel 3.5.1) wird durch die Sprachen des Objekt-Beziehungsparadigmas die statische Struktur eines Systems auf schematischer Ebene beschrieben.

Die visuellen Sprachen des Objekt-Beziehungsparadigmas gehen auf die Arbeiten von [Chen, 1976] zur Datenmodellierung zurück. Wesentliche Modellierungskonstrukte dieses Ansatzes sind *Objekt*- und *Beziehungsklassen*. Objektklassen (Entitätstypen) beschreiben Mengen ähnlicher Dinge, die im zu modellierenden System existieren. Gleichartige Beziehungen zwischen Objekten werden in Beziehungsklassen (Beziehungstypen) zusammengefaßt. Kardinalitäten geben an, wie viele Objekte einer Klasse mit Objekten anderer Klassen in Beziehung stehen können bzw. dürfen. Weitere Eigenschaften von Objekt- und Beziehungsklassen werden durch Attributschemata beschrieben.

Im Rahmen der semantischen Datenmodellierung (vgl. z. B. [Hull / King, 1987], [Smith / Smith, 1977]) wurden diese Modellierungskonstrukte um Mittel zur regulären Strukturierung der Modelle zu erweiterten Objekt-Beziehungsdiagrammen ergänzt. Diese Erweiterungen umfaßten Konstrukte zur Spezialisierung bzw. Generalisierung von Objekt- und Beziehungsklassen, zur Aggregation und zur Gruppierung. Aggregationen dienen zur Beschreibung von „besteht-aus-“ oder „Teil-Ganzes“-Beziehungen, die als eigenständige Objektklassen betrachtet werden können. Gruppierungen beschreiben Zusammenfassungen von Objekten derselben Objektklasse in einer eigenständigen Klasse. Solche Zusammenfassungen werden heute meist als spezielle, durch Kardinalitäten eingeschränkte, Aggregationen aufgefaßt.

Die Ansätze der objektorientierten Modellierung (vgl. z. B. [Rumbaugh et al., 1991], [Booch, 1994], [Booch et al., 1999]) erweiterten die Beschreibungsmittel des Objekt-Beziehungsparadigmas um Operationen. Operationen beschreiben das nach außen sichtbare Verhalten von Objekten und können als Dienste, die die Objekte bereitstellen, aufgefaßt werden. Die Operationen, die die Objekte einer Klasse besitzen, werden durch ihre Signaturen beschrieben, die das Ein- und Ausgabeverhalten dokumentieren.

Notiert werden statische Systemstrukturen entlang des Objekt-Beziehungsparadigmas u. a. durch Klassendiagramme, (erweiterte) Entity-Relationship-Diagramme, (erweiterte) Objekt-Beziehungsdiagramme, IDEF1X-Datenmodelle und NIAM-Informationsstrukturdiagramme. Verkürzte Darstellungen erfolgen durch Datenlexika und Jackson-Diagramme. Zusätzliche Anforderungen an statische Systemstrukturen können häufig nicht mit den graphischen Mitteln der Sprachen des Objekt-Beziehungsparadigmas ausgedrückt werden. Hierzu werden diese Sprachen um Zusicherungssprachen wie z. B. die Graph Specification Language (*GRAL*) [Franzke, 1997] oder die Object Constraint Language (OCL) [OMG, 1999, S. 6-1ff] ergänzt.

Notationelle Grundform: Klassendiagramme

Auch die Darstellungsmittel des Objekt-Beziehungsparadigmas basieren auf Graphen bzw. Hypergraphen. Durch Knoten dieser Graphen werden die Objektklassen und durch Kanten die Beziehungsklassen modelliert. Modellierungsansätze, die Beziehungsklassen beliebiger Arität zulassen, notieren diese durch Hyperkanten. Die Modellierung von beliebigen Beziehungsstrukturen kann ohne Informationsverlust in Modelle mit ausschließlich binären Beziehungsklassen transformiert werden. Hierzu werden n-äre Beziehungsklassen durch eine Kett-Objektklasse und n binäre Beziehungsklassen dargestellt. Modellierungen, die nur binäre Beziehungsklassen verwenden, und daher durch einfache Linien notiert werden können, werden als übersichtlicher eingestuft [Ebert / Engels, 1994], so daß beispielsweise in NIAM-Informationsstruktur-Diagrammen [Verheijen / van Bekkum, 1982] und in *EER/GRAL*-Klassendiagrammen [Ebert et al., 1996b] nur die Modellierung binärer Beziehungen unterstützt wird.

Als notationelle Grundform des Objekt-Beziehungsparadigmas wird der in dieser Arbeit verwendete *EER/GRAL*-Dialekt vorgestellt. Eine ausführliche Einführung in die formale Grundlage und die Beschreibungsmittel von *EER/GRAL* findet sich in Kapitel 5.2.

Objektklassen werden in *EER/GRAL*-Klassendiagrammen durch Rechtecke beschrieben, die durch Klassenbezeichner annotiert sind. Binäre Beziehungsklassen werden durch Linien zwischen den in Beziehung gesetzten Objektklassen dargestellt, die ebenfalls durch Klassenbezeichner ausgezeichnet sind. Die Leserichtung dieser Beziehungen wird durch ein Winkelsymbol auf der Beziehungslinie angezeigt.

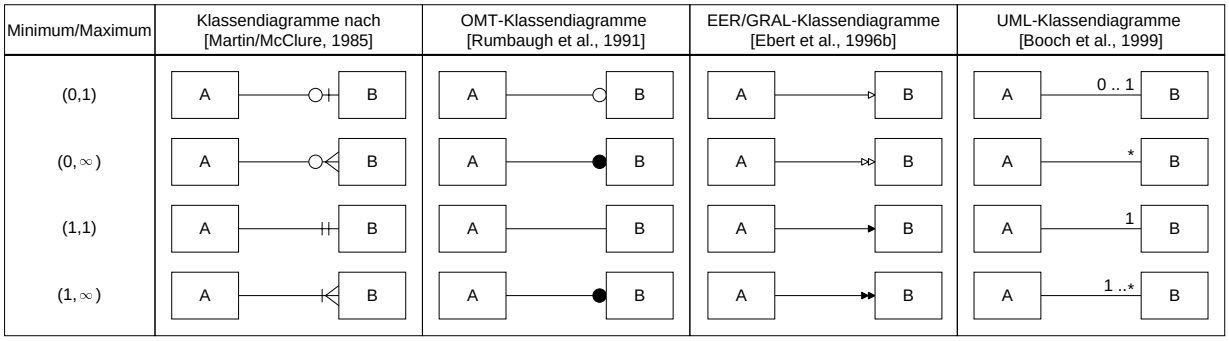


Abbildung 3.33: Notation für Kardinalitäten

Die Angabe von Kardinalitäten erfolgt heute üblicherweise durch Minimum-Maximum-Paare, die die Mindest- und die Maximalanzahl der Beziehungen angeben, die ein Objekt der entsprechenden Klasse eingehen kann. Zur Darstellung häufig vorkommender Kardinalitäten wer-

den in einigen Sprachen des Objekt-Beziehungsparadigmas graphische Notationen verwendet, die in Abbildung 3.33 (bezogen auf ausschließlich binäre Relationen) zusammengefaßt sind. In *EER/GRAL* wird zur Modellierung der Kardinalitäten eine Pfeilnotation verwendet.

Attributschemata zu Objekt- und Beziehungsklassen werden durch Angabe der Attributbezeichner und -wertebereiche in Ovalen notiert, die mit den Klassendarstellungen verbunden sind. Alternativ können Attributschemata zu Objektklassen auch direkt in den Rechtecken notiert werden. Generalisierungen von Objektklassen werden in *EER/GRAL* durch Ineinanderschachteln der Objektklassen beschrieben. Eine generalisierte Objektklasse enthält hierbei ihre Spezialisierungen. Diese Notationsform erlaubt eine kompakte und übersichtliche Beschreibung von Systemstrukturen, die durch tiefe Generalisierungshierarchien geprägt sind. Abstrakte Objekt- bzw. Beziehungsklassen, d. h. Klassen zu denen keine direkten Instanzen existieren, werden durch Schraffuren bzw. unterbrochen gezeichnete Linien notiert. Die Darstellung von Aggregationen erfolgt durch auf der Spitze stehende Quadrate an den Repräsentationen der Aggregate. Die Komponenten der Aggregation sind mit diesen Quadraten durch Linien, die die Aggregationsbeziehung notieren, verbunden. Wie auch Beziehungstypen sind diese Aggregationsbeziehungen mit einem Bezeichner versehen, können ausgerichtet und durch Kardinalitäten eingeschränkt werden.

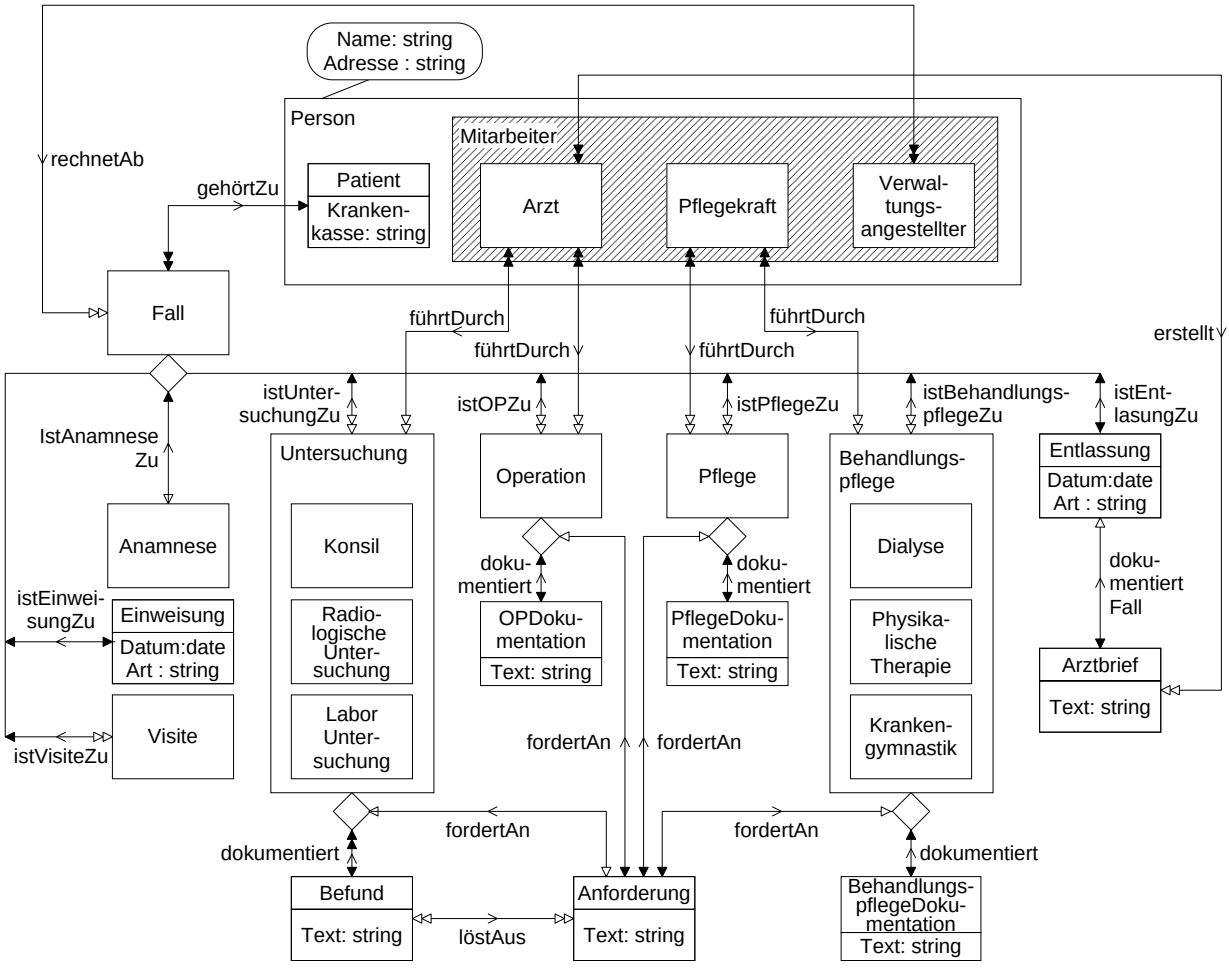


Abbildung 3.34: Objekt-Beziehungsdiagramm

Beispiel 3.12 (Klassendiagramm für medizinische Fälle)

In Beispiel 3.10 wurde ein Objektdiagramm zur Beschreibung eines konkreten medizinischen Falls dargestellt. Dieses Diagramm ist eine mögliche Instanz des *EER/GRAL*-Klassendiagramm aus Abbildung 3.34.

Ein *Fall*, der den Aufenthalt eines Patienten im Krankenhaus beschreibt [Haines, 1996], ist hier als Aggregation von einer *Einweisung*, einer optionalen *Anamnese*, beliebig vielen *Visiten, Untersuchungen, Operationen, Pflegen* und *Behandlungspflegen* sowie einer *Entlassung* modelliert. *Untersuchungen, Operationen, Pflegen* und *Behandlungspflegen* werden jeweils durch *Anforderungen* ausgelöst und durch unterschiedliche *Dokumentationen* bzw. *Befunde* dokumentiert. *Untersuchungen* und *Behandlungspflegen* sind weiter spezialisiert.

An *Fällen* sind mehrere *Personen* beteiligt. Durch eine Generalisierung werden hier *Patienten*, Krankenhaus-*Mitarbeiter* und weitere nicht näher klassifizierte Personen unterschieden. *Mitarbeiter* sind hierbei als abstrakte Klasse modelliert, die in *Ärzte*, *Pflegekraft* und *Verwaltungsangestellte* zerfällt. Je nach Beteiligung am *Fall* stehen diese *Personen* in unterschiedlichen Beziehungen zu den Komponenten des *Falls*. □

Notationelle Varianten

Zur Beschreibung nach dem Objekt-Beziehungsparadigma existieren ebenfalls mehrere visuelle Sprachen, die sich sowohl in ihrer Modellierungsmächtigkeit als auch in ihren konkreten Darstellungsformen unterscheiden. Die folgenden Abschnitte geben einen kurzen Überblick über weitere Beschreibungsmittel des Objekt-Beziehungsparadigmas, die die Entwicklung dieser Sprachen zu den heute verwendeten *Klassendiagrammen* wesentlich geprägt haben.

Datenlexika. *Datenlexika* (vgl. [Yourdon, 1989, S. 188f]) oder *Jackson-Bäume* [Jackson, 1975] und Warnier-Orr-Diagramme [Warnier, 1974] zur Datenmodellierung (vgl. auch die Beschreibungsmittel des Kontrollparadigmas in Kapitel 3.4.4), in denen ausschließlich reguläre Strukturen zur Beschreibung von Datenstrukturen verwendet werden, werden ebenfalls dem Objekt-Beziehungsparadigma zugeordnet. Die Modellierung der Datenstrukturen durch Alternativen entspricht der Generalisierung, die durch Sequenzen der Aggregation und die Modellierung durch Iterationen der Gruppierung. Beziehungsartige Zusammenhänge der Datenstrukturen werden mit diesen Mitteln jedoch nicht beschrieben.

Entity-Relationship-Diagramme. Die klassische visuelle Notation zur Beschreibung von *Entity-Relationship-Diagrammen* nach [Chen, 1976] unterstützt lediglich die Darstellung von Objektklassen durch Rechtecke und von Beziehungsklassen durch Rauten. Graphische Formen zur Darstellung von Attributierungen, Aggregationen und Generalisierungen werden nicht angeboten. Folglich wird in Abbildung 3.35, die einen Ausschnitt der Modellierung zu Beispiel 3.12 zeigt, auch auf die Darstellung von Attributen verzichtet. Die Aggregationsbeziehungen sind durch „normale“ Beziehungstypen modelliert. In Erweiterungen dieser Notation (vgl. z. B. [Vossen, 1994, S. 70ff], [Elmasri / Navathe, 2000]) wurden Ovale zur Darstellung von Attributschemata und durch „ISA“ markierte Rauten zur Darstellung von Generalisierungen eingeführt.

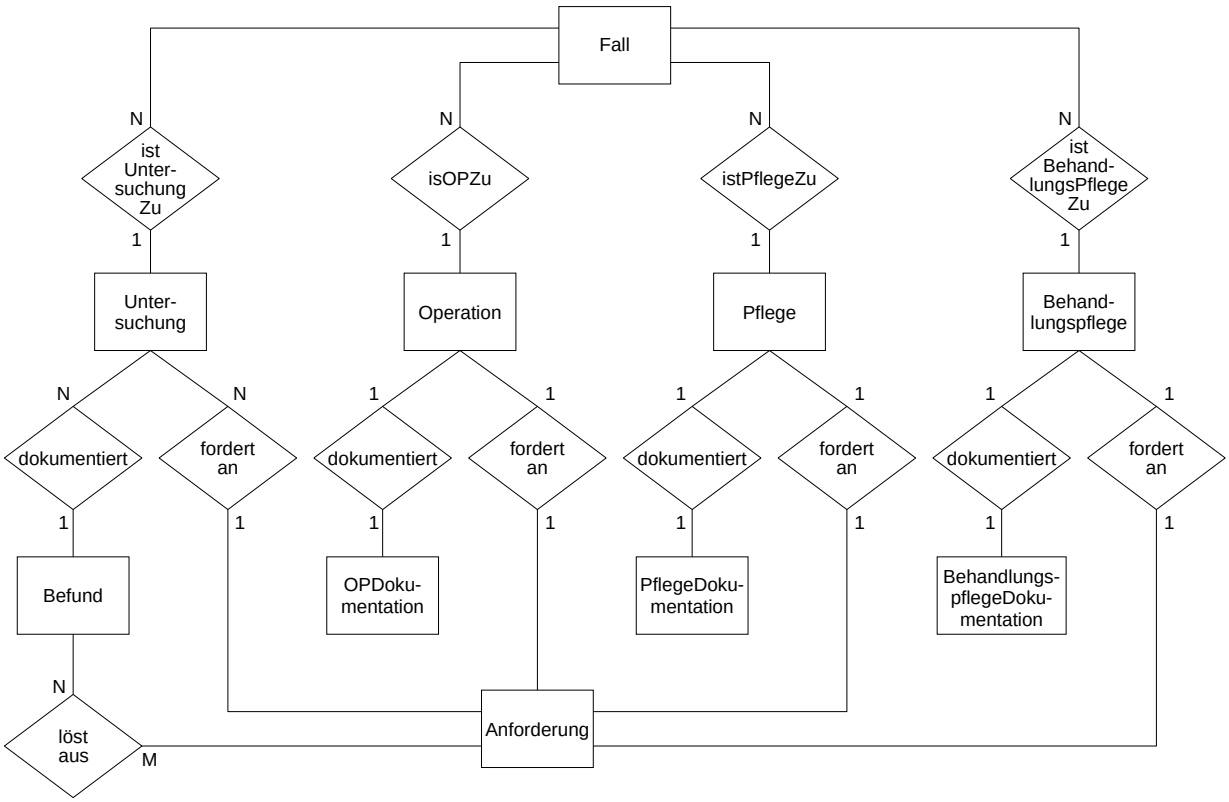


Abbildung 3.35: Entity-Relationship-Diagramm

Kardinalitäten werden in der Notation nach [Chen, 1976] nur in den Formen $1:N$, $M:N$ und $1:1$ notiert. Die Zeichen 1 bzw. N oder M an einer Objektklasse zeigen an, daß dieses Objekt in höchstens einer bzw. in beliebig vielen Instanzen der adjazenten Beziehungsklasse enthalten sein kann.

NIAM-Informationsstruktur-Diagramme. *Informations-Struktur-Diagramme* der Nijssens Information Analysis Method (NIAM) [Verheijen / van Bekkum, 1982], [Laender / Flynn, 1993] verwenden ausschließlich binäre Beziehungsklassen, in denen die Beziehungen jeweils aus Sicht der beteiligten Objektklassen benannt werden. Die Beziehungsklassen werden hierzu durch zwei nebeneinander plazierte Rechtecke notiert, die mit den jeweiligen Objektklassen verbunden sind und die Rolle dieser Objekte in der Beziehung modellieren. Zur Darstellung von Objektklassen werden in NIAM Kreise verwendet, die mit dem Klassenbezeichner versehen sind. Ein eigenständiges Konstrukt zur Modellierung von Attributen existiert in NIAM nicht. Die Attributwertebereiche werden hier ebenfalls durch Objektklassen modelliert, die mit der attritutierten Klasse über Beziehungen verbunden sind. Sind die Attributwertebereiche druckbare Standardtypen (lexikalische Objekte) wird der Kreis unterbrochen gezeichnet. NIAM erlaubt auch die Modellierung von Generalisierungen der Objektklassen durch gerichtete Pfeile zur Oberklasse.

Die Kardinalitäten der Beziehungstypen werden in NIAM durch Doppelpfeile über den Beziehungstypdarstellungen notiert. Ein über beiden Rollensymbolen notierter Doppelpfeil zeigt eine $M:N$-Beziehung, Doppelpfeile über jedem Rollensymbol modellieren eine $1:1$- und ein einzelner Doppelpfeil über einem Rollensymbol beschreibt eine $1:N$-Beziehung. Neben diesen

Invarianten stellt NIAM auch graphische Konstrukte zur Beschreibung von Einschränkungen an die Objektmengen, die an einer Beziehung beteiligt sind, bereit. Hierzu werden die entsprechenden Rollensymbole miteinander verbunden und in einem Kreis mit der Art der Einschränkung markiert. In Abbildung 3.36, die einen Ausschnitt des Klassendiagramms aus Beispiel 3.12 in NIAM zeigt, wird dieser Mechanismus zur Modellierung der Aggregation ausgenutzt. Die Objektklasse *Fall* besteht aus Zusammenfassungen von *Untersuchungen, Operationen, Pflegen* und *Behandlungspflegen.*

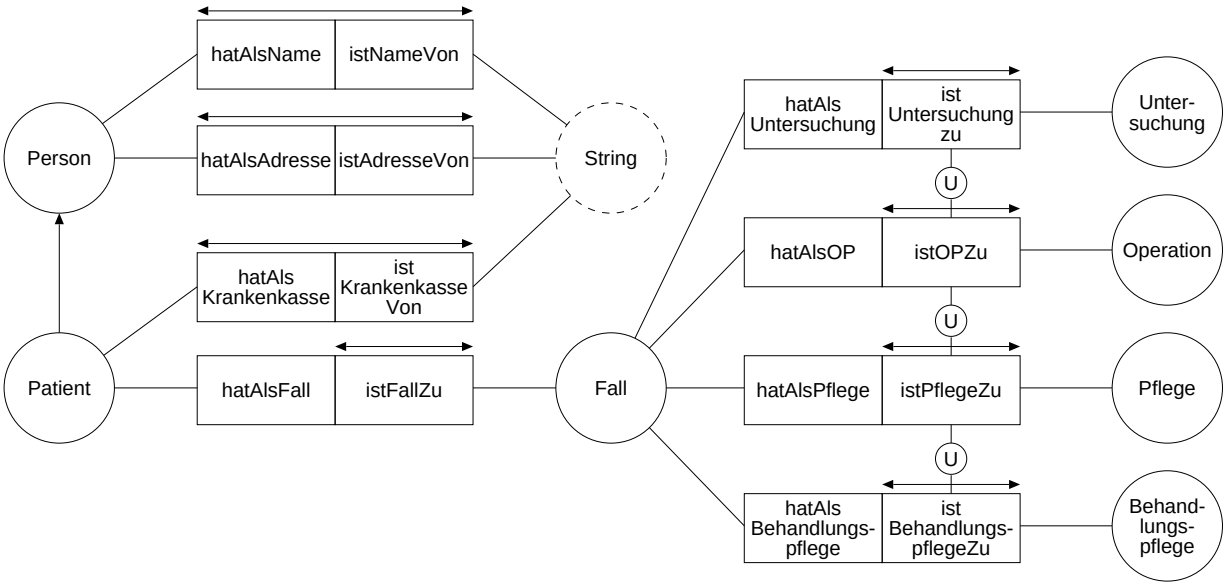


Abbildung 3.36: NIAM-Informationsstruktur-Diagramm

Generische Semantische Modelle. *Generische Semantische Modelle (GSM)* [Hull / King, 1987] erweitern die klassischen Entity-Relationship-Diagramme um Konstruktoren für Objektklassen. Diese Konstruktoren basieren ebenfalls auf regulären Strukturen. Zusammengesetzte Objektklassen werden in GSM durch Kreise modelliert, die zur Darstellung von Aggregationen durch ein Kreuz und zur Darstellung von Gruppierungen durch einen Stern markiert sind. Von diesen Objektklassen zeigt eine gerichtete, unterbrochene Linie zu den jeweiligen Komponenten, die ebenfalls durch zusammengesetzte Objektklassen, aber auch atomare, nicht weiter zerlegte Objektklassen beschrieben sein können. Atomare Objektklassen werden durch Dreiecke symbolisiert. Wie auch in NIAM werden in GSM Attribute ebenfalls durch Objektklassen modelliert. Die Darstellung von lexikalischen Objektklassen, die als dritte Art der Objektklassen aufgefaßt wird, erfolgt in GSM durch Ovale. Generalisierungen werden durch einen Doppelpfeil, der zur Generalisierung hin ausgerichtet ist, notiert. Die Spezialisierungen werden hierbei ebenfalls als konstruierte Objektklassen eingeordnet und durch Kreise beschrieben.

Sowohl die Darstellung von Attributzuordnungen als auch die Beschreibung von Beziehungen erfolgt in GSM durch gerichtete (Hyper-) Kanten. Diese Bezüge zwischen zwei Objektklassen werden hierbei jeweils als Attribute einer Objektklasse aufgefaßt, die durch lexikalische, durch atomare oder auch durch komplexe Objektklassen beschrieben sein können. Die Kanten sind jeweils zu den Attributen hin ausgerichtet.

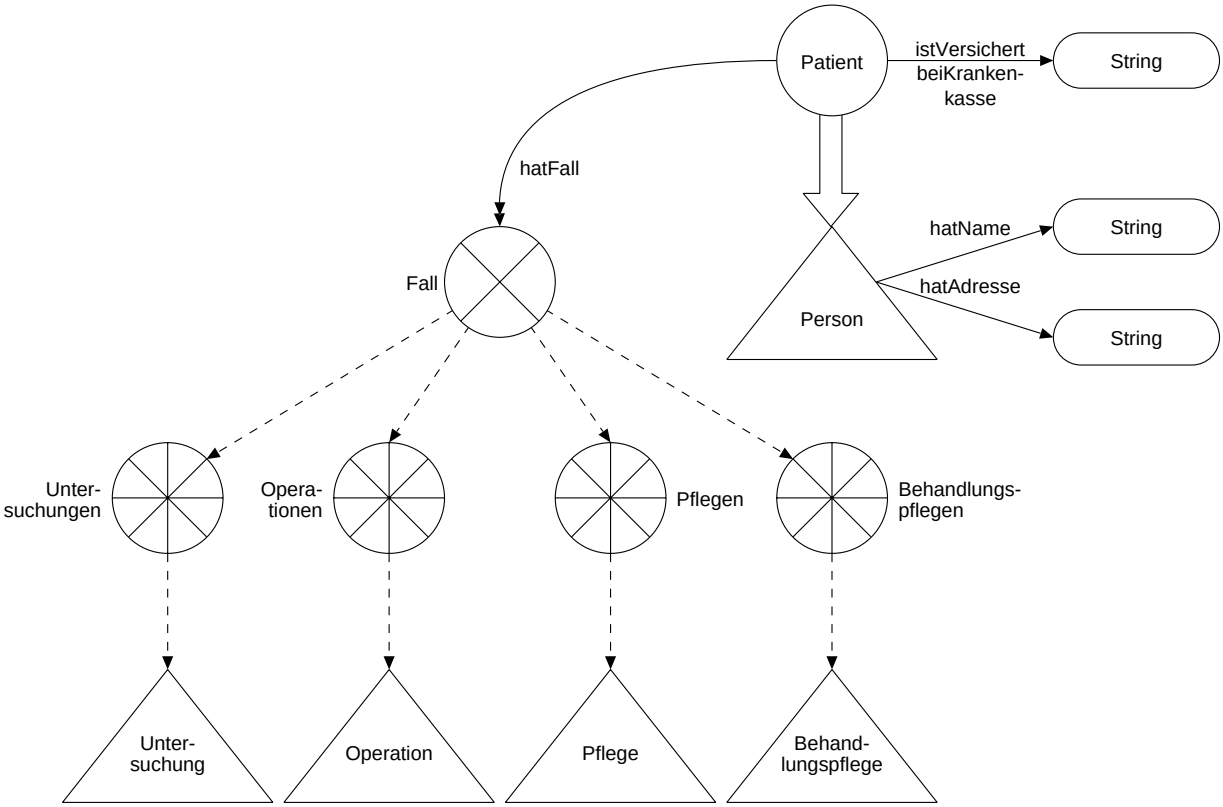


Abbildung 3.37: Generisches Semantisches Modell

Der Ausschnitt der Modellierung des medizinischen Falls aus Abbildung 3.36 ist in Abbildung 3.37 mit den Mitteln der generischen semantischen Modelle dargestellt. Die deutliche Unterscheidung von Aggregationen und Gruppierungen erfordert bei der GSM-Modellierung des *Falls* zunächst Gruppierungen der *Untersuchung*, der *Operation*, der *Pflege* und der *Behandlungspflege*, die erst anschließend aggregiert werden können. Zur Beschreibung mengenwertiger Attribute stellt GSM noch ein weiteres, der Gruppierung entsprechendes Sprachkonstrukt durch eine doppelte Pfeilspitze bereit. So wird dem *Patient* eine Menge von *Fall*-Attributen zugeordnet. Über die Gegenrichtung dieser Beziehung wird jedoch keine Aussage gemacht.

UML-Klassendiagramme. Objektorientierte Modellierungsmethoden (z. B. [Rumbaugh et al., 1991] [Booch, 1994]) stellen die Beschreibungsmittel des Objekt-Beziehungsparadigmas als zentrales Modellierungswerkzeug heraus. *Klassendiagramme* objektorientierter Modellierungssprachen unterstützen wie die bisher betrachteten Sprachen die Beschreibung von Objektklassen und deren Beziehungen sowie die Konzepterweiterungen durch Generalisierung, Gruppierung und Aggregation. Darüber hinaus stellen Klassendiagramme Darstellungsmittel zur Modellierung des nach außen sichtbaren Verhaltens der Objektklassen bereit. Klassendiagramme sind somit eine Verallgemeinerung der in erster Linie auf die Datenbeschreibung ausgerichteten Entity-Relationship-Diagramme um die Berücksichtigung von Verhaltensaspekten (vgl. [Booch et al., 1999, S. 110]).

In Abbildung 3.38 ist der medizinische Fall aus Beispiel 3.34 als Klassendiagramme der Unified Modeling Language (UML) [Booch et al., 1999], [Rumbaugh et al., 1999] modelliert.

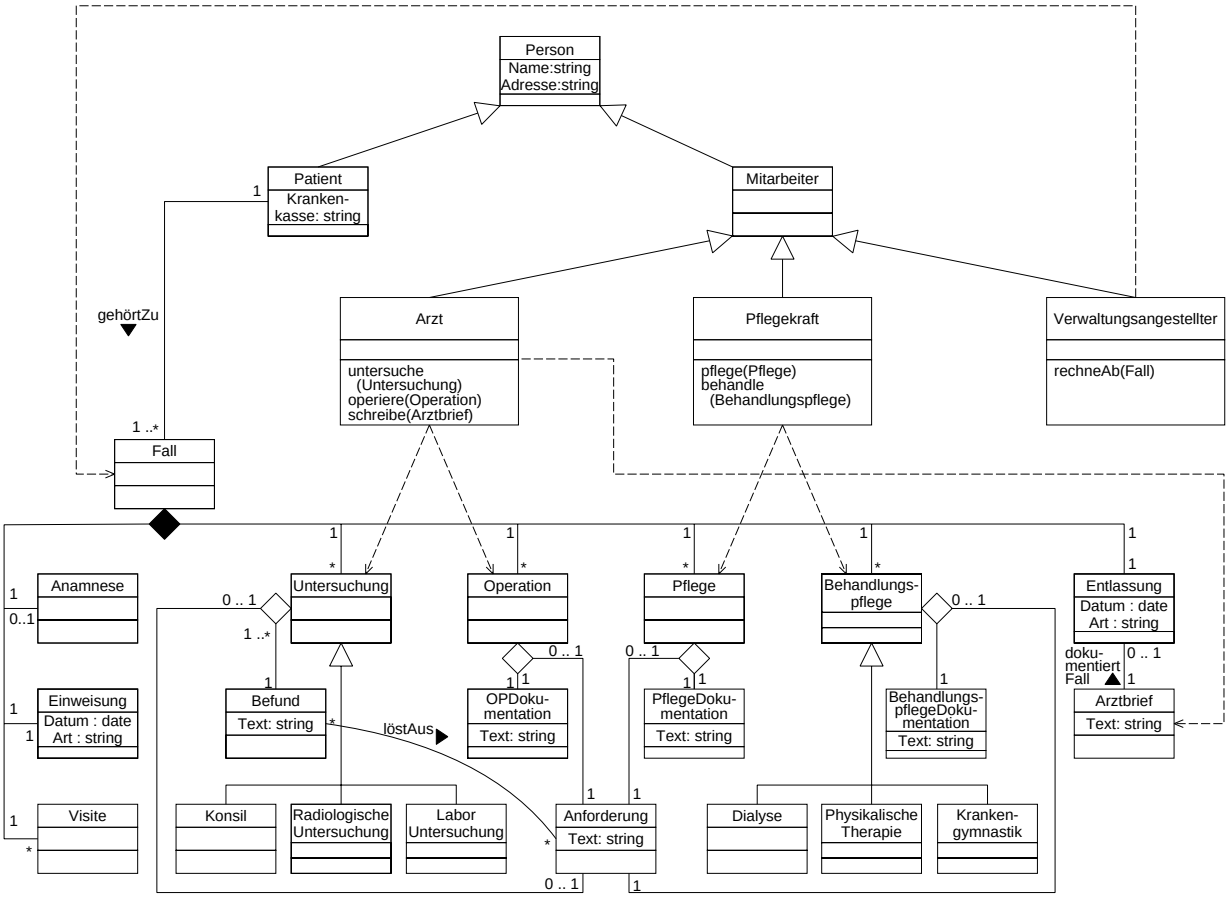


Abbildung 3.38: UML-Klassendiagramm

Wie auch in *EER/GRAL*-Klassendiagrammen und in klassischen Entity-Relationship-Diagrammen werden in UML-Klassendiagrammen die Objektklassen durch Rechtecke notiert, die neben Bezeichnern und Attributschemata auch die nach außen sichbaren Operationen enthalten. Diese Operationen werden durch Angabe ihrer Signatur einschließlich evtl. benötigter Modifikatoren beschrieben. In Abbildung 3.38 sind die Dienste notiert, die von *Ärzten, Pflegekräften* und *Verwaltungsangestellten* angeboten werden. Zur Durchführung von Untersuchungen besitzen *Ärzte* u. a. die Methode *untersuche(Untersuchung)*, die mit der durchzuführenden Untersuchung parametrisiert ist.

Binäre Beziehungstypen werden auch hier durch Linien notiert, die optional mit einem Bezeichner versehen sein können. Beziehungstypen beliebiger Arität werden in Anlehnung an die Notation von [Chen, 1976] durch Hyperkanten, die mit einer kleinen Raute markiert sind, dargestellt. UML-Klassendiagramme unterscheiden mehrere Arten von Beziehungsklassen (vgl. hierzu [Booch et al., 1999, S. 137ff]). Strukturelle Zusammenhänge zwischen Objektklassen werden durch Assoziationen (Associations) modelliert, die durch Linien notiert werden. Die Leserichtung der Bezeichner von Assoziationen kann ähnlich zur *EER/GRAL*-Klassendiagrammen durch ein ausgefülltes Dreieck neben dem Bezeichner festgelegt werden. Abhängigkeitsbeziehungen oder Nutzt-Beziehungen (Dependencies) zwischen zwei Klassen, die z. B. anzeigen, daß Operationen einer Klasse Objekte der anderen als Parameter benötigen, werden durch gerichtete, unterbrochene Linien dargestellt. Diese sind jeweils zu der benötigten Objektklasse hin ausge-

richtet. Aufgrund der Methode *untersuche(Untersuchung)* ist z. B. die Klasse *Arzt* von der Klasse *Untersuchung* abhängig. Dieser Bezug ist durch eine Abhängigkeitsbeziehung in Abbildung 3.38 dargestellt. Weitere Arten von Beziehungen werden durch textuelle Annotationen unterschieden. Die UML-Klassendiagramme stellen kein eigenständiges graphisches Element zur Beschreibung von Attributschemata zu Beziehungsklassen bereit. Attributschemata zu Beziehungsklassen sind durch eigene Objektklassen-Symbole zu beschreiben, die mit den Beziehungsklassen verbunden sind.

Aggregationen werden analog zu *EER/GRAL* durch auf der Spitze stehende Quadrate notiert. Bei der Aggregation unterscheidet UML zwei Varianten. Nicht ausgefüllte Quadrate beschreiben beliebige Zusammenfassungen von Komponenten zu einem Ganzen. Hierbei können die Komponenten jedoch auch als eigenständige Objekte existieren. Ist die Existenz einer Komponente an die Existenz des Aggregats gekoppelt (Komposition), wird dieses durch ein ausgefülltes Quadrat notiert. Alternativ kann die Komposition auch durch Schachtelung der Komponenten in der Aggregat-Darstellung notiert werden.

Kardinalitäten zu Beziehungstypen und Aggregationen werden durch Minimum-Maximum-Paare notiert. Die konkrete Darstellungsform kann Abbildung 3.33 entnommen werden. Die an Beziehungen oder Aggregationen beteiligten Objekte können auch ähnlich wie in NIAM durch ihre Rollen beschrieben werden. Hierzu werden die Rollenbezeichner an die Enden der Linien zur Beschreibung des Beziehungszusammenhangs notiert. Wie auch für Assoziationen kann auch für Aggregationen und Kompositionen die Leserichtung angegeben werden.

Generalisierungsbeziehungen sowohl zwischen Objektklassen als auch zwischen Beziehungsklassen werden durch (Hyper-) Kanten beschrieben. Durch nicht ausgefüllte Dreiecke, die zur Generalisierung zeigen, werden die Generalisierungen von den Spezialisierungen unterschieden.

Die verschiedenen objektorientierten Methoden verwenden unterschiedliche graphische Symbole zur Beschreibung von Klassendiagrammen. Im Rahmen der Vereinheitlichung der Notation zur Unified Modeling Language hat sich hier die Notation der Object Modeling Language (OMT) [Rumbaugh et al., 1991] durchgesetzt. Wesentliche Sprachbestandteile der UML-Klassendiagramme wurden aus OMT übernommen. Lediglich die Darstellung der Kardinalitäten wurde gegenüber OMT verändert (vgl. Abbildung 3.33) und die Dreiecke zur Beschreibung der Generalisierungsbeziehungen sind zur Generalisierung „gerutscht“.

Forderungen an das Referenz-Metaschema

Die Darstellung statischer Systemzusammenhänge entlang des Objekt-Beziehungsparadigmas basiert auf der Modellierung von Objekt- und Beziehungsklassen. Im Referenz-Metaschema sind daher Konzepte zur Beschreibung dieser Klassen einschließlich deren Attributschemata bereitzustellen. Zur weiteren Konkretisierung der Beziehungsklassen muß das Referenz-Metaschema auch die Darstellung von Kardinalitäten erlauben. Die Strukturierung der Modellierungen des Objekt-Beziehungsparadigmas kann durch Generalisierungen von Objekt- und Beziehungsklassen und Aggregationen bzw. Gruppierungen unterstützt werden. Diese Modellierungskonstrukte sind ebenfalls durch das Referenz-Metaschema abzubilden.

3.6 Einordnung in die Zielsetzung dieser Arbeit

Die Betrachtung organisatorischer und softwaretechnischer Systeme kann aus der Aufgabensicht, der Aufbausicht, der Prozeßsicht und der Objektsicht erfolgen. Innerhalb dieser Sichten folgt die Modellierung organisatorischer und softwaretechnischer Sachverhalte verschiedenen Darstellungsparadigmen. Ausgehend von diesen Sichten und Paradigmen wurde in Kapitel 3.1 ein Klassifikationsschema zur Einordnung der visuellen Sprachen der Organisations- und Softwaretechnik entwickelt.

Dieses Klassifikationsschema stellt ein umfassendes Hilfsmittel zur Untersuchung der u. a. in funktions- oder prozeßorientierten Methoden zur Organisationsmodellierung oder in strukturierten oder objektorientierten Methoden zur Softwareentwicklung eingesetzten visuellen Modellierungssprachen zur Verfügung. In den vorangegangenen Kapiteln wurden die konkreten Notationsformen der wesentlichen Beschreibungsmittel des Requirements-Engineerings der Organisations- und Softwaretechnik entlang der in diesem Klassinfikationsschema festgelegten Beschreibungsparadigmen vorgestellt. Das in Abbildung 3.2 auf Seite 37 zusammenfassend dargestellte Klassifikationsschema hat sich hierbei als valides und ausreichendes Instrument zur Einordnung der Beschreibungsmittel erwiesen.

Zentrales Ziel dieser Arbeit ist die integrierte Darstellung der verschiedenen viuellen Modellierungssprachen in einem Referenz-Metaschema. Die Beschreibungsmittel eines Paradigmas unterscheiden sich lediglich in ihren konkreten Darstellungsformen. Konzeptionell werden durch die Sprachen eines Paradigmas jeweils gleiche Sachverhalte dargestellt. Zur Entwicklung des Referenz-Metaschemas kann folglich von konkreten Notationen abstrahiert werden. Zur integrierten Darstellung der Beschreibungssmittel aus Organisations- und Softwaretechnik ist es ausreichend, im Referenz-Metaschema nur die wesentlichen Konzepte der einzelnen Paradigmen abzubilden. Ausgehend von den in den Kapiteln 3.2 bis 3.5 erarbeiteten Anforderungen erfolgt die Entwicklung des Referenz-Metaschemas für visuelle Modellierungssprachen in Kapitel 6 ebenfalls entlang des hier eingeführten Klassifikationsschemas.

4 Modelle, Referenzmodelle und Metamodelle

Betrachtungsgegenstand dieser Arbeit ist die konzeptionelle Modellierung der in Kapitel 3 eingeführten visuellen Sprachen zur Darstellung von Organisationen und Softwaresystemen. Hierzu wird ein Modell dieser Sprachen erstellt, durch das die jeweils dargestellten Modellierungskonzepte und deren Beziehungen abbildet. Im folgenden wird dieses Modell in den Kontext der Modellbildung (Kapitel 4.1), der Referenzmodellierung (Kapitel 4.2) und der Metamodellierung (Kapitel 4.3) eingeordnet.

4.1 Modelle

Modelle können als Darstellungen von Zusammenhängen einer Diskurswelt aufgefaßt werden, die wesentliche Bestandteile in den Vordergrund stellen und von unwesentlichen abstrahieren. Als Hilfmittel zur Darstellung und zum Nachvollziehen vielfältiger Zusammenhänge werden Modelle in den unterschiedlichsten Bereichen eingesetzt.

Modelle der *Mathematik und Logik* repräsentieren Strukturen zu einer Theorie, durch die diese interpretiert wird und die die formulierten Axiome erfüllen. Die Menge der ganzen Zahlen mit der Addition stellt ein solches Modell für die Gruppentheorie dar. Unter dem in der Mathematik verwendeten Modellbegriff werden also Repräsentanten einer Theorie verstanden. In den Natur-, Ingenieurs- und Geisteswissenschaften wird der Modellbegriff eher auf evtl. vorweggenommene Replikationen von Realitätsausschnitten [Dörner, 1993, S. 337] bezogen:

- In der *Architektur* stellen Modelle in Form von Bauplänen Arbeitsanweisungen zur Erstellung von Bauwerken dar. Zeichnungen oder Darstellungen aus Holz, Plastik und Ähnlichem modellieren Bauwerke mit dem Ziel, das spätere Aussehen eines Gebäudes frühzeitig zu visualisieren.
- In der *Physik* werden Modelle entwickelt, um beobachtbare Naturerscheinungen zu beschreiben, um ermittelte physikalische Gesetze zu deuten und um Voraussagen über den Ablauf neuer Experimente (u.a. auch zur Modellvalidierung) zu machen. Als Beispiele hierfür können die Atommodelle angesehen werden, die jeweils das zum Zeitpunkt der Erstellung bekannte Wissen über Aufbau und Zusammenhang von Atomen darlegen.
- Modelle in den *Sozialwissenschaften* befassen sich mit den Reaktionen eines Realitätsausschnitts auf bestimmte Außeneinflüsse und mit seinem Verhalten in der Zeit. Solche

Modelle finden ihre Anwendung z. B. zur Vorhersage der Bevölkerungsentwicklung oder zur Beurteilung möglicher Folgen von Gesetzesvorlagen.

- Durch die in Kapitel 3 eingeführten visuellen Modellierungssprachen werden Modelle der *Organisationstheorie* und der *Softwaretechnik* notiert. Diese Modelle abstrahieren von — im jeweiligen Problemkontext — unwesentlichen Aspekten und werden zur Kommunikation über die modellierten Zusammenhänge z. B. bei der Organisationsgestaltung oder bei der Entwicklung von Softwaresystemen verwendet. Hierbei werden Modelle auch zur Schaffung eines gemeinsamen Begriffsverstehens eingesetzt.

Modelle im Sinn der Natur-, Ingenieurs- und Geisteswissenschaften finden ihre Anwendung im Zusammenhang mit Systemen aus einer realen oder abstrakten Diskurswelt, die durch diese Modelle veranschaulicht werden. Ein Modell wird jedoch nicht nur von dem modellierten System selbst geprägt; seine Ausgestaltung und Interpretation wird auch von der Zielsetzung des Modellerstellers bzw. des Modellanwenders beeinflußt [Apostel, 1960], [Schütte / Rotthowe, 1998]. Hieraus wird in [Apostel, 1960] eine allgemeine Definition des Modellbegriffs abgeleitet. Ein System A, welches weder direkt noch indirekt mit einem System B interagiert und von einem Subjekt S verwendet wird, um Erkenntnisse über das System B zu gewinnen, ist hiernach für dieses Subjekt unter einer festgelegten Zielsetzung ein Modell des Systems B. [Becker / Vossen, 1996a] fassen — hieran angelehnt — ein Modell für Geschäftsprozesse als *„ein immaterielles und abstraktes Abbild der Realwelt für die Zwecke des Subjekts“* auf. Eine Diskussion des Modellbegriffs, bei der u. a. auch noch der Kenntnisstand des Modellanwenders in die Modellinterpretation einfließt, findet sich auch in [Troitzsch, 1990].

> Ein ***Modell*** ist ein zielgerichtetes Abbild eines Systems, das zum einen ähnliche Beobachtungen und Aussagen ermöglicht wie dieses System und zum anderen diese Realität durch Abstraktion auf die jeweils problembezogen relevanten Aspekte vereinfacht.

Modelle erlauben eine deutliche und strukturierte Beschreibung der Konzepte eines Systems durch Einschränkung auf die wesentlichen Problemaspekte. Hierdurch kann u. a. auch das Verstehen und Nachvollziehen komplexer Zusammenhänge unterstützt werden.

Modell-Taxonomie

Abhängig von Modellierungszusammenhängen, Modellierungsaspekten und Modellverwendung werden verschiedene Modelltypen unterschieden (vgl. hierzu auch [Lehner, 1995]). Eine ausführliche Klassifikation von Modelltypen vor dem Hintergrund der sozialwissenschaftlichen Modellbildung, die auch auf das im folgenden entwickelte Modell der visuellen Modellierungssprachen für Organisationen und Softwaresysteme angewandt werden kann, findet sich in [Troitzsch, 1990, S. 12ff].

Nach dem *Bildbereich* können konkrete und symbolische Modelle unterschieden werden [Brinkkemper, 1990], [Gigch, 1991]. *Konkrete Modelle* zeichnen sich dadurch aus, daß Dinge aus der modellierten Welt direkt (Realmodelle) oder nach Konstruktion (ikonische Modelle) als Modell

übernommen werden. *Symbolische Modelle*, die in einer symbolischen Form beschrieben werden, werden weiter in *Verbalmodelle*, bei denen Modelle in natürlicher Sprache dargestellt werden und in *formale Modelle* unterschieden, die durch formale (mathematische) Sprachen notiert werden [Troitzsch, 1990].

Nach *Art der Zusammenhänge* zwischen den modellierten Objekten unterscheidet [Troitzsch, 1990, S. 15] *statische Modelle*, bei denen sich die Merkmale der modellierten Objekte und Beziehungen im Verlauf der Zeit nicht ändern und *Prozeßmodelle*, bei denen zeitliche Änderungen einbezogen werden.

Die Unterscheidung nach dem *Modellierungszweck* bietet ein weiteres aussagekräftiges Unterscheidungsmerkmal. Hierbei ist zu beachten, daß mit einer Modellierung nur selten genau ein Ziel verfolgt wird. Konkrete Modelle können durchaus auch mehreren der folgenden Modellklassen zugehörig sein. Mit *Deskriptionsmodellen* soll in erster Linie eine Unterstützung bei der Kommunikation über modellierte Sachzusammenhänge geboten werden. In diese Modellklasse fallen auch solche Modelle, die im Rahmen der Softwareerstellung (Implementationsmodelle) sowohl zur Kommunikation zwischen den hieran beteiligten Personen, als auch als Implementations-Hilfsmittel verwendet werden. *Didaktische* oder *tutorielle Modelle* werden vor dem Hintergrund der Vermittlung der Modellinhalte erstellt. Simulationen werden *Simulationsmodelle* zu Grunde gelegt. Sie haben auch einen sehr engen Bezug zu formalen Modellen, da Simulationsmodelle in der Regel auch formal zu beschreiben sind.

4.2 Referenzmodelle

Eine ausgewiesene Form von Modellen bilden Referenzmodelle. Der Begriff „Referenz“ leitet sich vom lateinischen „refero“: auf etwas zurückführen, beziehen auf, nach etwas beurteilen, ab. Mit dem Zusatz „Referenz“ werden solche Dinge ausgezeichnet, die als Bezug oder Empfehlung geeignet sind. ***Referenzmodelle*** bezeichnen folglich solche Modelle, die als Bezug i. w. S. dienen.

In Begriffsklärungen aus der Literatur wird dieser Bezug häufig auf die Verwendung von Referenzmodellen als Modellierungsmittel zur Erstellung spezieller Modelle bezogen. So versteht [Scheer, 1997] unter einem Referenzmodell ein Modell, „*das als Ausgangspunkt für die Entwicklung auf konkrete Aufgabenstellungen bezogener Problemlösungen dienen kann.*“ [Seubert, 1997] beschreibt Referenzmodelle als „*vorgefertigte Lösungsrahmen*“, die eine Basis für den „*gezielten und ökonomischen Aufbau von [. . .] Konzeptionen*“ bieten. Ähnliche Begriffsbildungen nehmen [Raue, 1996], [Hars, 1994] und [Schütte, 1996] vor. Auch hier steht die Verwendung von Referenzmodellen als Bezug für den Entwurf anderer Modelle im Vordergrund.

Neben dieser Eigenschaft stellen u.a. [Marent, 1995], [Becker / Schütte, 1997] und [Rosemann / Schütte, 1997] den Empfehlungscharakter von Referenzmodellen heraus. Referenzmodellen wird hier ein „*normativer Charakter*“ zugesprochen, da durch sie für eine „*abgebildete Klasse von Problemstellungen*“ Gestaltungsempfehlungen gegeben werden.

Auch das Referenzmodell für Workflow-Managementsysteme [Hollingsworth, 1994] wurde als Grundlage zur Erstellung spezieller Modelle entwickelt. Zusätzlich wurden jedoch Anforderungen an den Bildbereich des Referenzmodells formuliert. So soll das Referenzmodell für

Workflow-Managementsysteme deren „*charakteristischen Merkmale, deren Terminologie und deren Komponenten*“ beschreiben. Ebenfalls auf die Beschreibung wesentlicher Konzepte bezieht [ECMA, 1991] den Begriff Referenzmodell. Referenzmodelle sind hiernach „*konzeptionelle und funktionale Rahmen*“ zur Beschreibung und zum Vergleich komplexer Systeme.

Ausgehend von Fragestellungen der Unternehmensmodellierung versteht [Marent, 1995] unabhängig vom Verwendungszweck unter einem Referenzmodell ein „*konzeptionelles Geschäftsbereichsmodell für ein idealtypisches Unternehmen*“. Ähnlich allgemein faßt auch [Scholz-Reiter, 1990] den Referenzmodellbegriff. Referenzmodelle werden hier als Modelle verstanden, deren wesentliche Ausprägungen „*generell gültig*“ und „*von individuellen Besonderheiten freigehalten*“ sind.

Gemeinsam ist diesen Begriffsbildungen eine Abgrenzung zwischen dem Referenzmodell und den hierzu in bezug gesetzten speziellen Modellen. *Spezielle Modelle* beschreiben einen Sachzusammenhang aus einer auf ein konkretes Modellierungsziel hin ausgerichteten Sicht. Sie sind in ihrem Detaillierungsgrad ebenfalls speziell auf den zu beschreibenden Problemkontext hin zugeschnitten. *Referenzmodelle* sind dagegen nicht auf ein konkretes Problem bezogen, sondern werden als problemübergreifende Modellierung verstanden. Sie beschreiben die wesentlichen Merkmale des Modellierungskontexts eher allgemeingültig.

> Ein ***Referenzmodell*** ist ein ausgewiesenes Modell, das *charakteristische Eigenschaften* einer Klasse gleichartiger Systeme beschreibt und als *Bezugspunkt* für *spezielle Modelle* dient.

In dieser Begriffsbildung wird lediglich gefordert, daß Referenzmodelle als Bezugspunkte für spezielle Modelle dienen. Über die Qualität der Referenzmodelle wird keine Aussage gemacht. Im Gegensatz zu den zuvor genannten Begriffsbildungen wird *nicht* gefordert, daß ein Referenzmodell eine idealtypische oder normative Modellierung mit Empfehlungscharakter ist. Hierdurch wird erreicht, daß grundsätzlich jedes Modell als Referenzmodell aufgefaßt werden kann (vgl. auch [Lehner, 1995, S. 126]). Ein Modell kann jedoch erst im Zusammenspiel mit einem speziellen Modell zum Referenzmodell werden. Nach dieser Auffassung können auch weniger gelungene und schlechte Modelle als Referenz, z. B. zur Darstellung von ungeeigneten Modellierungsansätzen, die in einer neuen Modellierung nicht wieder verfolgt werden sollen, betrachtet werden.

Der Referenzmodellbegriff ist den Begriffen Standard, Norm und Entwurfsmuster gegenüberzustellen [Winter / Ebert, 1997]. *Standards* legen durch allgemeine Akzeptanz Begriffe und Eigenschaften von Systemen fest und liefern so eine allgemeine Richtschnur. Standards (oder de facto-Normen) entstehen in der Regel ohne vorgegebenen Plan [Tanenbaum, 1990], werden aber in ihrem Anwendungsbereich weitgehend akzeptiert. Eine planmäßig durchgeführte Festlegung von Begriffen und Eigenschaften durch autorisierte Normierungsinstitute (z. B. DIN, ANSI, ISO) wird als *Norm* (oder de jure-Norm) bezeichnet. Standards und Normen können somit auch als Referenzmodelle aufgefaßt werden, die sich jedoch in ihrem Verbindlichkeitsgrad unterscheiden. Als eine besondere Form von Referenzmodellen können auch *Entwurfsmuster* oder *Design Pattern* aufgefaßt werden. Während Referenzmodelle durch nahezu vollständige Modelle eines Anwendungsbereichs eher etabliertes Modellwissen im Großen widerspiegeln, beschreiben Design Patterns [Gamma et al., 1994] Modellwissen im Kleinen. Sie beschreiben einfache und

elegante Lösungen für wiederkehrende Probleme, die sich in der Entwurfspraxis bewährt haben. Diese Referenzen dienen nicht als genereller Bezugspunkt zur Modellierung eines vollständigen Problembereichs, sondern sind vielmehr als Referenzbausteine zur Modellierung aufzufassen. [Gamma, 1996] ordnet Entwurfsmuster auch als „Mikroarchitektur“ ein, die zur Gesamtarchitektur eines Systems beiträgt.

4.2.1 Anforderungen an Referenzmodelle

Da Referenzmodelle auch Modelle sind, können auch Anforderungen, die für die Qualität von Modellen formuliert wurden, auf Referenzmodelle übertragen werden. Eine solche Übertragung zeigen beispielsweise [Becker / Schütte, 1997] und [Rosemann / Schütte, 1997], die die generellen Modellierungsgrundsätze aus [Becker et al., 1995] auf Referenzmodelle anwenden. Qualitätsanforderungen an Konzeptmodelle diskutieren [Lindland et al., 1994] unter syntaktischen, semantischen und pragmatischen Aspekten. In [Krogstie et al., 1995] wird dieser Ansatz erweitert. Qualitätsmerkmale von Modellen werden hier ausgehend von Beziehungen zwischen der Modelldomäne, der Modellinterpretation durch die Modellverwender, dem Kenntnisstand der Modellverwender, den Modellierungsmitteln und dem Modell selbst klassifiziert und durch Metriken beschrieben. Zu einigen Merkmalen werden Mittel zur Qualitätverbesserung kurz skizziert. Einen Überblick über Anforderungen an Datenmodelle geben [Moody / Shanks, 1994]. Hier sind auch Hinweise zur Bestimmung einer qualitativen Bewertung dieser Kriterien genannt. Generelle Eigenschaften von Modellierungen werden in [Brinkkemper, 1990, S. 42ff] diskutiert. Die in diesen Arbeiten aufgeführten Anforderungen an Modelle können in ähnlicher Form auch auf Referenzmodelle übertragen werden (vgl. auch [Heym, 1995, S. 107ff]).

Anforderungen an (spezielle) Modelle sind noch um solche zu ergänzen, die den Referenzcharakter einer Modellierung unterstreichen. Nicht alle Qualitätsanforderungen, die an spezielle Modellierungen gestellt werden, sind direkt auf Referenzmodelle übertragbar. Anforderungen an spezielle Modelle können durchaus im Konflikt zu Anforderungen an Referenzmodelle stehen. Auch andere Anforderungen an Eigenschaften von Referenzmodellen sind untereinander nicht notwendig widerspruchsfrei. Zur Einschätzung eines Referenzmodells ist ein geeigneter Mix an Merkmalsausprägungen heranzuziehen. Eine Übersicht zu den Abhängigkeiten zwischen den Ausprägungen unterschiedlicher Qualitätskriterien findet sich ebenfalls in [Moody / Shanks, 1994].

In den nachfolgenden Abschnitten werden Anforderungen an Referenzmodelle hinsichtlich der Kriterien *Allgemeingültigkeit, Korrektheit, Vollständigkeit, Anwendbarkeit* und *Erweiterbarkeit* diskutiert.

Allgemeingültigkeit

Eine wesentliche Eigenschaft, die Referenzmodelle von speziellen Modellen unterscheidet, ist der Anspruch der Referenzmodelle *allgemeiner* und *umfassender* zu sein, so daß sie auch in verschiedenen speziellen Modellierungskontexten eingesetzt werden können. Hiermit eng verbunden ist die Wahl des geeigneten *Detaillierungsgrades* der Referenzmodellierung. Der Detaillierungsgrad einer Referenzmodellierung sollte nicht zu konkret gewählt sein, da sie dann nicht mehr auf vielfältige Problemstellungen anwendbar ist und somit der Forderung nach Allgemeingültigkeit nicht mehr entsprochen wird. Der Detaillierungsgrad eines Referenzmodells sollte aber

auch nicht zu abstrakt sein, da zu allgemeine Modelle die Anwendung auf konkrete Anwendungsbereiche kaum unterstützen (vgl. auch [Marent, 1995]). Die Forderung von [Scheer, 1997], wonach ein Referenzmodell soweit spezifiziert sein soll, daß es ohne Veränderung als spezielles Modell verwendbar ist, sollte nur als ein möglicher Spezialfall für Referenzmodelle aufgefaßt werden. Eine Referenzmodellierung sollte vielmehr so bemessen sein, daß sie durch einfache Spezialisierungen und wenige Ergänzungen und Änderungen auf spezielle Probleme angewandt werden kann. Dieses kann durchaus auch durch die Angabe von Modellierungsvarianten [Raue, 1996] unterstützt werden.

Korrektheit

Wie auch Modelle sollten Referenzmodelle korrekt sein. Bei der Korrektheit von Modellen wird die syntaktische und die semantische Korrektheit unterschieden [Zamperoni / Löhr-Richter, 1993], [Becker et al., 1995]. Ein Modell ist *syntaktisch korrekt*, wenn seine Beschreibung den syntaktischen Regeln des verwendeten Beschreibungsmittels entspricht. Die syntaktische Korrektheit kann entlang eines Metamodells (vgl. Kapitel 4.3) nachgewiesen werden. Die *semantische Korrektheit* eines Modells ist im Zusammenhang mit dem zu modellierenden Gegenstandsbereich zu untersuchen. Ein Modell gilt dann als semantisch korrekt, wenn es unter Berücksichtigung des Modellierungsziels mit der modellierten Realität „übereinstimmt“. Eine Validierung der semantischen Korrektheit einer Referenzmodellierung kann nur durch Akzeptanz des Modells durch die Anwender vor dem Hintergrund ihrer Anwendungsziele erfolgen.

Vollständigkeit

Für Modelle wird neben der Korrektheit auch die *Vollständigkeit* gefordert. Ein Modell gilt als vollständig, wenn alle für die Zielsetzung der Modellierung relevanten Elemente im Modell enthalten sind. Gleichzeitig sollte ein Modell auch ausschließlich relevante Objekte enthalten (vgl. das Kriterium der Relevanz in [Becker et al., 1995]). Auf Referenzmodelle läßt sich die Forderung nach Vollständigkeit und Relevanz nur eingeschränkt übertragen. Da Referenzmodelle eher problemübergreifend und allgemein angelegt sind, kann ein Referenzmodell kaum vollständig sein und alle relevanten Modellaspekte abbilden. Ein vollständiges Referenzmodell, das ausschließlich problemrelevante Dinge abbildet, ist in der Regel zu speziell, um als Bezugspunkt für verschiedene problembezogene Modellverwendungen zu dienen. Die Forderung aus [Scheer, 1997], wonach ein Referenzmodell soweit vollständig sein muß, daß mindestens ein Anwendungsfall denkbar ist, für das es unverändert als spezielles Modell eingesetzt werden kann, steht daher in deutlichem Widerspruch zur Forderung nach Allgemeingültigkeit. [Hars, 1994] überträgt die Forderung nach Vollständigkeit für Referenzmodelle daher in die Forderung nach *Nützlichkeit*. Ein Referenzmodell muß hierbei mindestens potentiell für die Erstellung mehrerer Modelle verwendbar sein. Überträgt man die Betrachtungen zur Allgemeingültigkeit auch auf die Vollständigkeit, sollte ein Referenzmodell soweit vollständig sein, daß möglichst wenige Ergänzungen zur Verwendung des Referenzmodells zur Lösung konkreter Probleme nötig sind. Die Ableitung problembezogener Modelle aus einem Referenzmodell sollte nach festen Regeln erfolgen. Referenzmodelle sollten daher auch um *Regeln zur Ableitung* ergänzt werden (vgl. [Hars, 1994]). Auch wenn die Forderung nach Vollständigkeit für Referenzmodelle nur teilweise erfüllt werden kann, sollte durch Anwendung dieser Regeln die Ableitung vollständiger, spezieller Modelle ermöglicht werden.

Anwendbarkeit

Referenzmodelle sollten auch *einfach anwendbar* sein. Hierzu sollten sie leicht *vermittelbar und nachvollziehbar* sein (vgl. auch den Grundsatz der Klarheit in [Rosemann / Schütte, 1997]). Die Einfachheit eines Modells ist geprägt von seiner Komplexität, d. h. von der Anzahl der modellierten Objekte und deren Beziehungen [Moody / Shanks, 1994]. Durch geeignete Strukturierung und systematischen Modellaufbau kann diese Komplexität reduziert werden. Insbesondere sollten ähnliche Zusammenhänge auch ähnlich modelliert werden. Unterstützt wird die Nachvollziehbarkeit von komplexen Modellen auch dadurch, daß gut abgrenzbare Teilbereiche separat und unabhängig beschrieben und genutzt werden können.

Die Anwendbarkeit eines Referenzmodells hängt im besonderen Maß auch von seinem Anwendungskontext ab. Während ein Referenzmodell bei der Verwendung als Hilfsmittel zum Vergleich eher (genau) eine *idealtypische* Modellierung darstellen sollte, kann ein Referenzmodell, das eher als Ausgangspunkt zur Erstellung spezieller Modelle Anwendung findet, auch mehrere *Modellalternativen* für unterschiedliche Anwendungszusammenhänge bereitstellen [Rosemann, 1996, S. 35]. Die Bereitstellung von Modellvarianten erhöht einerseits die Anwendbarkeit des Referenzmodell auf unterschiedliche Problemstellungen [Raue, 1996], erschwert anderseits aber auch dessen Anwendung, da mehrere Alternativen zu berücksichtigen sind. Daher sollte ein Referenzmodell eher wenige Alternativen bereitstellen. Zur Beurteilung der Güte eines Referenzmodells unter dem Anwendbarkeitsaspekt ist folglich auch die Adaption des Referenzmodells an die spezielle Aufgabenstellung zu beachten [Becker / Schütte, 1997].

Auch sollte die Anwendbarkeit des Referenzmodells *nachgewiesen* werden. Der Vorschlag, bei der Beschreibung von Design Patterns mindestens zwei unterschiedliche Verwendungen aus verschiedenen Anwendungskontexten zu nennen [Gamma et al., 1994, S. 7], sollte auch für Referenzmodelle übernommen werden.

Erweiterbarkeit

Referenzmodelle spiegeln den aktuellen Stand innerhalb ihres Modellierungskontexts wider. Da dieser einer dynamischen Entwicklung unterworfen ist, müssen Referenzmodelle auch die Anpassung an zukünftige Entwicklungen und Erkenntnisse erlauben. Referenzmodelle sollten daher *offen* und *flexibel* [Moody / Shanks, 1994] angelegt sein, um sie leicht an geänderte Anforderungen anpassen zu können. Dem gegenüber steht die Forderung, daß ein Referenzmodell auch in einem gewissen Maß *robust* [Rosemann / Schütte, 1997] sein sollte, so daß nicht jede Änderung der Rahmenbedingungen zu einer Überarbeitung des Referenzmodells führt.

Eng mit der Erweiterbarkeit eines Modells ist auch das Zusammenspiel mit anderen Modellen in benachbarten Kontexten zu sehen. So sollte auch ein Referenzmodell soweit *integrationsfähig* [Moody / Shanks, 1994] sein, daß es mit Referenzmodellen angrenzender Bereiche kombinierbar ist.

> Die skizzierten Eigenschaften von Referenzmodellen sind wesentlich von der Akzeptanz und Einschätzung durch die Modellanwender geprägt. Diese Eigenschaften müssen für Referenzmodelle *durch die Verwendung ständig nachgewiesen* werden.

4.2.2 Beispiele für Referenzmodelle

Für eine Vielzahl von Systemen unterschiedlicher Art wurden Modelle entwickelt, denen Referenzeigenschaften zugesprochen werden. Hierunter fallen z. B. Modelle, die *Referenz-Architekturen* beschreiben. Im besonderen Maß werden Referenzmodelle auch im Bereich der Modellierung und Untersuchung von *Informationssystemen* verwendet. In den folgenden Abschnitten werden daher einige Beispiele für Referenz-Architekturen und für Referenzmodelle für Informationssysteme skizziert, um generelle Anwendungsbereiche für Referenzmodelle aufzuzeigen.

Referenz-Architekturen

Referenz-Architekturen beschreiben Systeme durch Angabe ihrer wesentlichen Komponenten und deren Zusammenhänge. Beispiele für Referenz-Architekturen sind das *OSI-Referenzmodell*, das *ECMA-Referenzmodell* und das *Workflow-Referenzmodell*.

OSI-Referenzmodell

Das OSI-Referenzmodell [Day / Zimmermann, 1983] beschreibt die Kommunikation zwischen offenen Rechnersystemen (open systems interconnection). In diesem Modell wird in sieben Schichten die Verbindung zwischen Rechnersystemen von der Übertragung einzelner Bits bis zum Transfer von Dateien und der Kommunikation zwischen unterschiedlichen Terminaltypen festgelegt. Hierbei wird jedoch nur der Leistungsumfang jeder einzelnen Schicht definiert. Diese Schichten bilden die Hauptkomponenten einer Architektur für die Kommunikation offener Rechnersysteme. Das Grundmodell wurde als ISO-Norm ISO 7498 festgeschrieben und bildet den Ausgangspunkt für die Entwicklung spezieller Normen für die einzelnen Schichten. Daher sollte das OSI-Referenzmodell auch als *Norm* aufgefaßt werden.

Neben der Verwendung als Grundlage eines Normierungsprozesses und der Festlegung grundsätzlicher Konzepte der Kommunikation zwischen Rechnersystemen dient das OSI-Referenzmodell auch als Grundlage zur Schulung. So ist beispielsweise das Lehrbuch zu Computernetzwerken [Tanenbaum, 1990], welches auch eine ausführliche Beschreibung des OSI-Referenzmodells enthält, nach dieser Referenz gegliedert.

ECMA-Referenzmodell

Ein Referenzmodell für Software-Entwicklungsumgebungen liefert das ECMA-Referenzmodell [ECMA, 1991] der European Computer Manufacturers Association. In diesem Modell werden die Komponenten eines erweiterungsfähigen Rahmens für eine Software-Entwicklungsumgebung und die von ihnen bereitgestellten Dienste beschrieben. Dieser Rahmen enthält Komponenten zur Datenhaltung, zur Datenintegration, zur Task-Verwaltung, zur Kommunikation mit dem Anwender und zur Kommunikation zwischen den Komponenten. Innerhalb einer Softwareentwicklungsumgebung können die Dienste dieser Komponenten durch eingebundene Werkzeuge genutzt werden. Durch das Referenzmodell wird die Software-Entwicklungsumgebung in unabhängige Komponenten zerlegt,

die durch festgelegte Dienste miteinander und mit weiteren Werkzeug-Komponenten interagieren. Daher sollte auch hier eher von einer *Referenz-Architektur* gesprochen werden (vgl. auch [Kelter, 1988]).

Das ECMA-Referenzmodell wurde im Rahmen des Standardisierungsverfahrens für PCTE (Portable Common Tool Environment, vgl. [Kelter, 1989]) entwickelt, stellt aber ein von PCTE unabhängiges Modell dar, das auch für weitere Standardisierungsvorhaben genutzt werden soll. Entlang dieses Modells wurde PCTE, das eine Schnittstellenspezifikation von Basisfunktionen zur Konstruktion von Software-Entwicklungsumgebungen darstellt, mit anderen Standards gleicher Zielsetzung verglichen. Neben der Unterstützung bei der Standardisierung und Einordnung von PCTE und der Ableitung anderer Softwarearchitekturen soll das ECMA-Referenzmodell auch zur Ausbildung von Systementwicklern im Umgang mit Softwareentwicklungsrahmen eingesetzt werden.

Workflow-Referenzmodell

Die Workflow Management Coalition [Hollingsworth, 1994] definierte das Workflow-Referenzmodell. Dieses legt die Terminologie der Workflow Management Coalition fest (ein ausführliches Glossar befindet sich in [WfMC, 1996]) und beschreibt eine Architektur für Workflow-Managementsysteme. Ähnlich zum ECMA-Referenzmodell wird diese Architektur durch Angabe der wesentlichen Komponenten zur Verwaltung und Ausführung von Workflows, zur Modellierung von Workflows, zur Interaktion mit Benutzern, zum Auslösen externer Anwendungen ohne Benutzerinteraktion und zur Systemadministration durch Angabe ihrer Eigenschaften und Schnittstellen beschrieben. Ebenfalls umfaßt diese Architektur eine Schnittstelle zu anderen Workflow-Managementsystemen. Das Workflow Referenzmodell ist somit als *Referenz-Architektur* inklusive einer *Referenz-Terminologie* aufzufassen.

Ausgangspunkt für die Erstellung dieses Referenzmodells war die Feststellung, daß bereits viele Produkte mit teilweise unterschiedlicher Schwerpunktsetzung im Workflow-Kontext verfügbar waren, diese jedoch nicht oder nur sehr schwer kombiniert werden konnten. Wesentliche Anforderung war es daher, die Interoperabilität zwischen Workflow-Managementsystemen und zwischen hierbei verwendeten Anwendungssystemen zu ermöglichen. Aus diesem Grund gibt das Referenzmodell auch einen groben Überblick über eine solche Schnittstelle (workflow application programme interface). Details wurden in weiteren Berichten der Workflow Management Coalition veröffentlicht[1]. Das hier vorgestellte Referenzmodell dient somit auch als Ausgangspunkt für weitere Standardisierungsvorhaben innerhalb der Workflow Management Coalition.

Referenzmodelle für Informationssysteme

Im Bereich der *Informationssysteme* werden Referenzmodelle sowohl für verschiedene Branchen und für verschiedene Geschäftsprozesse als auch vor dem Hintergrund unterschiedlicher Anwendungskontexte erstellt. Hierzu werden in der Regel visuelle Modellierungssprachen, wie sie in Kapitel 3 skizziert wurden, eingesetzt.

[1] Siehe hierzu `http://www.wfmc.org` (17.09.1999).

Im folgenden werden überblicksartig Ansätze und Anwendungsbereiche der Referenzmodellierung von Informationssystemen beschrieben. Diese beziehen sich auf *betriebliche Informationssysteme* generell, für die bereits in den 60er Jahren erste Referenzmodelle entwickelt wurden, und auf aktuelle Schwerpunkte der Referenzmodellierung in der *öffentlichen Verwaltung* und im *Gesundheitswesen*.

Referenzmodelle für betriebliche Informationssysteme

Ein erster Ansatz zur Referenzmodellierung betrieblicher Informationssysteme wurde mit dem *Kölner Integrationsmodell (KIM)* [Grochla, 1974] bereits 1965 begonnen. Hierin wurden die wichtigsten betrieblichen Aufgaben im Kontext der Informationsverarbeitung erfaßt und abstrahiert von individuellen betrieblichen Eigenschaften beschrieben.

Dieses Grundmodell war auch als Basis für die Ableitung weiterer Modelle für andere Wirtschaftszweige und zur Erstellung branchenspezifischer Modelle und deren Verfeinerungen konzipiert. Das Kölner Integrationsmodells ist datenflußorientiert durch Aufgabenlisten, Kanallisten (zur Datenbeschreibung) und Datenflußdiagramme beschrieben.

Ein heute weit verbreiteter Ansatz zur Modellierung von Referenzmodellen für Informationssysteme basiert auf der *Architektur integrierter Informationssysteme (ARIS)* [Scheer, 1992] (vgl. auch Seite 121). Referenzmodelle für industrielle Unternehmensprozesse wie z. B. der Logistik, der Produktentwicklung, des Rechnungswesens und der Personalwesen sind in [Scheer, 1994] beschrieben. In ähnlicher Form liegen auch Referenzmodelle für unterschiedliche Branchen u. a. für die Papierindustrie, die chemische Industrie und den Maschinenbau vor[2]. Ein auf die Prozesse in Handelsunternehmen bezogenes Referenzmodell wird in [Schütte, 1996] vorgestellt. Dominierendes Beschreibungsmittel der ARIS-Referenzmodelle sind Ereignisgesteuerte Prozeßketten, die durch Funktionsbäume, Objekt-Beziehungsdiagramme und Organigramme ergänzt werden und somit diese Informationssysteme aus Aufgaben-, Aufbau-, Prozeß- und Objektsicht beschreiben.

Anwendung finden auf ARIS aufbauende Referenzmodelle sowohl als Ausgangspunkt zur Erstellung spezieller Modelle als auch als Vergleichsmaßstab für vorliegende Modellierungen z. B. zur Restrukturierung des Informationssystems oder zur Auswahl von Software-Unterstützungsmitteln.

Einen Überblick über betriebliche Referenzmodelle vor dem Hintergrund des Computer Integrated Manufacturing gibt auch [Scholz-Reiter, 1990, S. 109ff]. Einige Beispiele für Modellierungen, denen (von den jeweiligen Autoren) Referenzeigenschaften zugesprochen werden, sind in [Lehner, 1995, S. 126] genannt. Verschiedene Unternehmensmodelle und Ansätz zur Unternehmensmodellierung werden auch in [Frank, 1994, S. 135ff] überblicksartig dargestellt.

Referenzmodelle für die öffentliche Verwaltung

Für die kommunale Verwaltung wurde 1994 durch die Städte Mannheim und Mainz mit dem Kommunalen Rechenzentrum Niederrhein (KRZN) in Moers das *Kommunale Informationsmodell KIM* [KGSt, 1995] als Grundlage zur Entwicklung einer Übersicht über das

[2] Einen groben Überblick über (kommerzielle) auf ARIS basierende Referenzmodelle bietet auch `http://www.ids-scheer.de/produkte/toolset/refmod.htm` (17.09.1999).

in den Kommunalverwaltungen vorhandene Informationspotential erstellt. In diesem Datenmodell werden die Daten, die für alle Kommunalverwaltungen gleichermaßen von Bedeutung sind, erfaßt und strukturiert. Hierzu wurde zunächst ein verwaltungsübergreifendes „Kommunales Top-Informationsmodell“ auf hohem Abstraktionsniveau erstellt, das dann zu fachbereichsbezogenen bzw. anwendungsbezogenen Informationsmodellen verfeinert wurde. Das Kommunale Informationsmodell ISMONE - Mainz/Mannheim, das [KGSt, 1995] als Anhang beigefügt ist, stellt das Ergebnis der Modellkonsolidierung des Top-Informationsmodells mit weiteren kommunalen Datenverarbeitungszentralen dar. Dieses sehr grobgranulare Datenmodell wird durch ein Objekt-Beziehungsdiagramm beschrieben. Jeder Objekttyp ist glossarartig durch eine Definition und kleinere Beispiele näher charakterisiert.

Dieses Modell kann als *Referenz-Datenmodell* aufgefaßt werden, da aufgrund der Konsolidierung durch mehrere Kommunale Rechenzentren auch seine Anwendbarkeit in weiteren Kommunalverwaltungen nachgewiesen ist. Darüber hinaus dient es auch als Grundlage zur Erstellung weiterer, dann verfeinerter Datenmodelle für einzelne Fachbereiche und Anwendungen.

Die Kommunale Gemeinschaftsstelle empfiehlt das Kommunale Informationsmodell als Grundlage zur Entwicklung der speziellen Datenmodelle der einzelnen Kommunalverwaltungen [KGSt, 1995, S. 13]. Neben der strukturierten Beschreibung der Informationsstrukturen in der kommunalen Verwaltung dient das Modell auch zur Begriffsvereinheitlichung und zur besseren Koordination des Datenaustauschs zwischen einzelnen Kommunalverwaltungen. Gemeinsam mit Funktions- und Verfahrensmodellen bietet es auch eine Entscheidungsgrundlage zur Auswahl von Anwendungssoftware.

Ein *Referenzmodell zur IT-gestützten Vorgangsbearbeitung* [Engel, 1996] [KoopA ADV, 1997, Kapitel 4] wurde im Rahmen des Handlungsleitfadens „IT-gestützte Vorgangsbearbeitung“ der gleichnamigen Arbeitsgruppe des Kooperationssausschuß ADV Bund/Länder/Kommunaler Bereich entwickelt. Es stellt die organisatorischen Zusammenhänge im Rahmen der Vorgangsbearbeitung innerhalb der öffentlichen Verwaltung idealtypisch dar. Zur Einordnung und Definition der hier verwendeten Begriffe enthält der Handlungsleitfaden ein ausführliches Glossar [KoopA ADV, 1997, Kapitel 2]. Den Hauptteil bildet ein Funktionenmodell, in dem die Phasen und Teilprozesse typischer Prozeßverläufe der Vorgangsbearbeitung dargestellt werden. Die Darstellung erfolgt durch Ereignisgesteuerte Prozeßketten. Das Referenzmodell wird anhand von zwei hieraus abgeleiteten Modellierungen von Verwaltungsvorgängen validiert. Ergänzt wird das Referenzmodell noch durch eine hiermit verträgliche Ziel- oder Referenzarchitektur zur IT-gestützten Vorgangsbearbeitung [KoopA ADV, 1997, Kapitel 5]. Diese umfaßt Komponenten zur Bürokommunikation, zum Schriftgutmanagement und zur Vorgangssteuerung, die durch ihre Grundfunktionalität und partiell durch ihre Schnittstellen näher erklärt sind. Die Komponente zur Vorgangssteuerung wurde entlang der Architektur für Workflow-Managementsysteme des Workflow-Referenzmodells erstellt.

Das Funktionenmodell des Referenzmodells zur IT-gestützten Vorgangsbearbeitung beschreibt einen idealtypischen Vorgang der öffentlichen Verwaltung durch Abstraktion auf wesentliche Teilvorgänge. Seine Anwendbarkeit wurde sowohl anhand von abgeleiteten, speziellen Modellen als auch durch die abgeleitete Architektur nachgewiesen. Gemein-

sam mit dem Glossar kann es daher als ein *Referenzmodell* eingeordnet werden, daß auch den in Kapitel 4.2.1 angesprochenen Anforderungen genügt. Das Referenzmodell enthält zusätzlich ein objektorientiertes Datenmodell (Referenz-Schema) [Engel, 1995], in dem die grundlegenden Konzepte zur Beschreibung der Vorgangsbearbeitung wie z. B. Vorgänge, Dokumente und organisatorische Einheiten sowie die hierzwischen möglichen Beziehungen eingeführt werden. Dieses Schema stellt im Sinn der in Kapitel 4.3 eingeführten Terminologie ein Meta-Datenmodell zur Beschreibung von Verwaltungsvorgängen dar.

Anwendung findet das Referenzmodell als theoretische Grundlage des Leitfadens. Ebenfalls dient es durch das Glossar zur Schaffung einer terminologischen Basis. Darüber hinaus soll es auch die Präzisierung von Anforderungen an informationstechnische Umgebungen für die Vorgangsbearbeitung unterstützen. Hierbei wird es einerseits zur Formulierung dieser Anforderungen aber auch zum Vergleich angebotener Lösungen eingesetzt. Die Verwendung des Referenzmodells für die Vorgangsunterstützung zur Schwachtellenanalyse und Verwaltungsreorganisation wird in [Engel, 1996] beschrieben.

Referenzmodelle im Gesundheitswesen

Neben der öffentlichen Verwaltung bietet auch das Gesundheitswesen ein aktuelles Betätigungsfeld der Referenzmodellierung. Hier beschäftigen sich sowohl Wissenschaftler als auch Softwareersteller, Unternehmensberater und Praktiker in Krankenhäusern mit der Erstellung von Referenzmodellen für Krankenhausinformationssysteme [Winter / Ebert, 1997], [Winter et al., 1999]. Diese Arbeiten beziehen sich sowohl auf Referenzmodelle für krankenhausweite Datenmodelle als auch auf Referenzmodelle für Abläufe in Krankenhäusern.

Ein *Referenzdatenmodell für Krankenhäuser* wird in [Bihr / Seelos, 1997] vorgestellt. Notiert wird dieses Datenmodell durch Objekt-Beziehungsdiagramme. Anwendung soll dieses krankenhausweite Datenmodell bei der Integration verschiedener, heterogener Anwendungskomponenten für den administrativen, medizinischen und pflegerischen Bereich finden. Ebenfalls soll es als Grundlage zur Datenmigration aus Altsystemen dienen.

Das *Datenmodell Bundeswehrkrankenhaus* [Imhoff et al., 1996] [Imhoff / Paczkowski, 1997] wurde als Referenz und Rahmenkonzept für die Krankenhäuser der Bundeswehr erstellt. Trotz des Namens, der historisch bedingt ist, war es das Ziel dieser Studie, ein Informations- und Prozeßmodell der unmittelbar patientenbezogenen Informationsprozesse zu erstellen. Dieses Modell basiert auf der zuvor skizzierten ARIS-Architektur und wird aus der Aufgabensicht durch Funktionsbäume, aus der Prozeßsicht durch Prozeßketten und aus der Objektsicht durch Objekt-Beziehungsdiagramme beschrieben. Das Modell umfaßt neben dem Datenmodell auch Prozeßmodelle für elf Krankenhausbereiche sowie ein verfeinertes Modell der Orthopädie. Ebenfalls enthält es ein umfangreichen Glossar. Wesentliche Zielsetzung zur Erstellung dieses Modells war neben der Erhebung und Dokumentation und der hieraus resultierenden Konzeption eines optimierten Sollmodells auch die Entwicklung einer Basis zur Ableitung fachlicher Feinkonzepte. Das Modell wurde zunächst im Bundeswehrkrankenhaus Koblenz erstellt und anschließend als Referenzmodell für die Bundeswehr in zwei weiteren Häusern validiert und festgeschrieben.

4.2.3 Zusammenfassung: Anwendungsbereiche von Referenzmodellen

Aus diesen Beispielen lassen sich die wesentlichen *Anwendungsbereiche für Referenzmodelle*, die in ähnlicher Form auch für das in Teil II entwickelte Modell der visuellen Sprachen gelten, ableiten.

Wie auch spezielle Modelle, dienen Referenzmodelle zunächst als ein generelles *Beschreibungsmittel*, durch das wesentliche Konzepte des Modellierungskontextes verdeutlicht werden.

Referenzmodelle liefern einen Beitrag zur *Terminologiebildung*, da hierdurch nicht nur wesentliche Konzepte benannt sondern diese Konzepte auch näher erklärt und in ihren Kontext eingeordnet werden. Hierdurch eignen sie sich auch als *Schulungsmittel*, da mit ihrer Hilfe eine einheitliche Begriffswelt vermittelt werden kann. Aufgrund des *Normierungscharakters* von Referenzmodellen kann auch ein gemeinsames Verstehen der modellierten Zusammenhänge z. B. in Diskussionsrunden heterogener Gruppen erreicht werden.

Durch Referenzmodelle werden auch Rahmen beschrieben, die Ausgangspunkt von *Standardisierungs-* oder *Normungsvorhaben* sind. Komponenten dieser Referenzmodelle oder Referenzarchitekturen beschreiben zu standardisierende Bereiche und deren Schnittstellen.

Ein wesentlicher Anwendungsbereich von Referenzmodellen liegt in ihrem Zusammenspiel mit speziellen Modellen. Als *Modellierungsmittel* dienen Referenzmodelle als Grundlage zum Entwurf und zur Weiterentwicklung spezieller Modelle. Die Erhebung und Dokumentation eines speziellen Modells erfolgt dann durch *Abgleich mit dem Referenzmodell*. Zu modellieren sind dann nur noch die Abweichungen zum Referenzmodell.

Als *Vergleichsmaßstab* dienen sie auch zur Beurteilung vorhandener Modelle. Im Vergleich mit einem Referenzmodell, das für einen Aspekt als besonders geeignet erscheint, können so beispielsweise Ansätze für Modelloptimierungen ermittelt werden.

Anforderungen an Softwaresysteme können ebenfalls aufgrund von Referenzmodellen für den zu unterstützenden Bereich erstellt und präzisiert werden. Solche Modelle dienen dann auch als Grundlage zur *Auswahl* der in Frage kommenden Softwaresysteme. Hierbei wird dann das Organisationsmodell mit dem Softwaremodell verglichen. Software-Referenzmodelle dienen darüber hinaus auch als Hilfsmittel zur *Konfiguration* der einzuführenden Software, indem diese Software entlang des Referenzmodells an die konkreten Anforderungen angepaßt wird.

Einen zusammenfassenden Überblick über die Anwendungsbereiche von Referenzmodellen bietet Abbildung 4.1.

- Beschreibung
- Terminologiebildung
- Schulung und Ausbildung
- Normierung und Standardisierung
- Modellierung
- Vergleich
- Anforderungsdefinition
- Auswahl und Konfiguration

Abbildung 4.1: Anwendungsbereiche von Referenzmodellen

4.3 Metamodelle

Betrachtungsgegenstand dieser Arbeit sind die in Kapitel 3 vorgestellen visuellen Sprachen zur Modellierung organisatorischer Zusammenhänge. Es wird ein Modell der Beschreibungsmittel dieser Methoden erstellt. Während bei der Organisationsmodellierung im Sinn der Modelldefinition nach [Apostel, 1960] die betrachtete Organisation die Rolle des modellierten Systems und die Darstellungen durch die Beschreibungsmittel die Rolle des Modells einnehmen, treten nun diese Darstellungen in die Rolle des modellierten Systems. Modellierungsziel ist folglich ein *Modell von Modellen*.

Durch solche Modelle werden Eigenschaften und Anforderungen an die Modelle einer Realität beschrieben. Im bezug zur modellierten Realität, wird dieses Modell der Modellierung als ***Metamodell*** bezeichnet. Metamodellierung kann folglich als eine Modellbildung aufgefaßt werden, die eine Abstraktionsebene höher als die „normale“ Modellierung angesiedelt ist [Gigch, 1991]. In ähnlicher Form können auch Metamodelle wieder Betrachtungsgegenstand einer Modellierung sein. Meta-Metamodelle beschreiben dann Abstraktionen von Metamodellen.

Nach einer eher sprachbezogenen Auffassung des Metamodell-Begriffs (vgl. [Strahringer, 1996]) werden durch Metamodelle die Modellierungssprachen, die zur Notation der Modelle verwendet werden, näher charakterisiert. Ähnlich beziehen auch [Frank, 1994, S. 171f] und [Frank, 1997b] Metamodelle auf die Beschreibung der zur Modellierung verwendeten Sprachen. [Blaha, 1992] und [Rumbaugh, 1995] verstehen Metamodelle generell als Modelle zur Beschreibung anderer Modelle. Im Speziellen beziehen sich Metamodelle für Modellierungsmethoden auch hier auf die zur Modellierung verwendeten Konzepte und die konkrete Notation des Modellierungsmittels.

Ein umfassenderer Metamodellbegriff wird im Method-Engineering verwendet. Metamodellierung ist hier ein wesentliches Hilfsmittel sowohl bei der Entwicklung als auch bei der Bewertung von Beschreibungs- und Entwicklungsmethoden wie sie z. B. für den Entwurf von Informationssystemen oder für die Analyse von Organisationsstrukturen eingesetzt werden. Hierbei sind die Methoden, die hierbei verwendeten Techniken und die zur Unterstützung eingesetzten Werkzeuge zu entwerfen und anzupassen [Brinkkemper, 1996].

Im Gegensatz zu der eher sprachbezogenen Verwendung des Metamodellbegriffs umfaßt der Metamodellbegriff hier auch Aspekte des Modellierungsvorgehens. In diesen Metamodellen werden die charakteristischen Eigenschaften der Methoden und Techniken durch die hierbei verwendeten Modellierungskonzepte, deren Repräsentation und ihre Verwendung [Tolvanen et al., 1996] beschrieben. Metamodelle legen hier sowohl das Vorgehen bei der Erstellung einer Modellierung als auch die Darstellung der Modellierungsergebnisse (vgl. [Brinkkemper et al., 1989], [Rosemann, 1996]) fest.

[Brinkkemper, 1990], [Brinkkemper, 1996] und [Tolvanen, 1998, S. 66ff] geben einen guten Überblick über die Metamodell-Terminologie aus Sicht des Method-Engineerings. Eine ausführliche Diskussion des Metamodell-Begriffs sowohl aus Sicht der Modellierungssprachen als auch aus Sicht des Modellierungsprozesses findet sich auch in [Strahringer, 1996, S. 9ff].

> Ein ***Metamodell*** ist die konzeptionelle Beschreibung einer Modellierung, die sowohl die verwendeten Modellierungskonzepte als auch ihre Verwendung verdeutlicht.

Nach dieser Begriffsbildung beschreiben Metamodelle zwei Modellierungsaspekte. Bei der Betrachtung der *dynamischen* Aspekte einer Modellierungsmethode steht das Vorgehen zur Erhebung und Dokumentation eines Modells im Vordergrund. Dieses wird in einem *Meta-Aktivitätsmodell* durch Beschreibungsmittel der Prozeßsicht dargestellt.

> Ein ***Meta-Aktivitätsmodell*** ist der Teil eines *Metamodells*, durch den das Modellierungsvorgehen beschrieben wird.

Die Methodenbetrachtung aus *statischer* Sicht stellt die bei der Modellierung erstellten Dokumente in den Mittelpunkt. Die zur Modellierung verwendeten Sprachmittel werden hierbei unabhängig von der konkreten Notation durch ihr *abstrakte Syntax* beschrieben. Hierzu werden die zu modellierenden Konzepte und deren Beziehungen in einem Datenmodell oder Schema der Methode dargestellt. Folglich wird eine solche Modellierung auch *Metaschema* oder *Meta-Datenmodell* genannt. Als Beschreibungsmittel bieten sich hier die Mittel zur Konzeptmodellierung der Objektsicht an. Die sprachbasierte Auffassung des Metamodellbegriffs verkürzt Metamodelle auf diese statischen Aspekte.

> Ein ***Metaschema*** (Meta-Datenmodell) ist der Teil eines *Metamodells*, durch den die zur Modellierung verwendeten Sprachmittel, unabhängig von ihrer konkreten Notation, durch ihre abstrakte Syntax beschrieben werden.

Die Verbindung zwischen Meta-Aktivitätenmodell und Metaschema wird durch *Dokumente* hergestellt. Diese beschreiben einerseits die Ergebnisse der Modellierungsaktivitäten des Meta-Aktivitätenmodells und werden andererseits durch die Objekte und Beziehungen des Metaschemas strukturell beschrieben [Brinkkemper et al., 1989].

Metamodelle, denen die in Kapitel 4.2 skizzierten Eigenschaften von Referenzmodellen zugesprochen werden, werden als ***Referenz-Metamodelle*** bezeichnet (vgl. z. B. auch [Österle / Gutzwiller, 1992] und [Rosemann / zur Mühlen, 1995]). Solche Referenz-Metamodelle beschreiben die charakteristischen Konzepte sowie deren Repräsentation und Verwendung für Modellierungssysteme.

4.3.1 Beispiele für Metamodelle

Als Ausgangspunkt für Diskussionen und Beschreibungen von Modellierungsmethoden in unterschiedlichen Bereichen wurden Metamodelle entwickelt. So existieren Metamodelle als Grundlage für die *Beschreibung von Informationssystemen* und hierauf aufbauende Analysen. *Entwurfsmethoden* für die Entwicklung von Softwaresystemen und zur Organisationsmodellierung werden ebenfalls durch Metamodelle beschrieben und verglichen.

Die folgenden Abschnitte geben einen Überblick über verschiedene Ansätze zur Metamodellierung in diesen Bereichen. Hieraus werden anschließend generelle Anwendungsbereiche für Metamodelle abgeleitet.

Metamodelle für die Beschreibung von Informationssystemen

Metamodelle für die Beschreibung von Informationssystemen setzen auf zwei Beschreibungsdimensionen auf. Neben Sichten auf organisatorische Zusammenhänge werden auch die Phasen des Software-Lebenszyklus (vgl. Abbildung 2.2) berücksichtigt. Beispiele solcher Metamodelle sind das *IFIP WG 8.1-Metamodell*, das *CC-RIM-Referenzmodell*, die *Architektur integrierter Informationssysteme (ARIS)* und das Metamodell der *Multi-Perspektivischen Unternehmensmodellierung (MEMO)*, die sich neben ihren Modellierungszielen auch in der Definition der Phasen und Sichten zur Bestimmung des Koordinatensystems leicht unterscheiden.

IFIP WG 8.1-Metamodell

Das IFIP WG 8.1-Metamodell [Olle et al., 1991] basiert auf Überlegungen der Arbeitsgruppe „Design and Evaluation of Informationsystems“ der „International Federation for Information Processing“, die im Rahmen mehrerer Konferenzen zum Thema „Information Systems Design Methodologies“ (z. B. [Olle et al., 1987]) zwischen 1985 und 1987 erarbeitet wurden. Informationssysteme werden im IFIP-Metamodell während der Analysephase und der Entwurfsphase betrachtet. Für diese Phasen werden Informationssysteme aus einer Daten-, einer Prozeß- und einer Verhaltenssicht konzeptionell untersucht.

Für die *Analysephase* werden aus der *Datensicht* die Komponenten zur Modellierung von Objekten und ihrer Beziehungen innerhalb eines Informationssystems betrachtet. Die *Prozeßsicht* bezieht sich im wesentlichen auf die abstrakte Syntax zur Modellierung von Geschäftsprozessen, von Kommunikationsbeziehungen zwischen den Prozessen und von organisatorischen Zuständigkeiten. Eine eigene Sicht zur Beschreibung Aufbauorganisatorischer Aspekte existiert in diesem Ansatz nicht. Konzepte zur Darstellung von Ereignissen und von Ereignisabfolgen sind die Betrachtungsschwerpunkte aus *Verhaltenssicht*.

In der *Entwurfsphase* werden eher implementationstechnische Aspekte betont. Aus *Datensicht* werden hier Datenhaltungskonzepte betrachtet, die auch in bezug zu den Konzepten der Analysephase gesetzt werden. Zentrales Konzept aus *Prozeßsicht* sind die Mittel zur Unterstützung der Geschäftsprozesse aus der Analysephase. Die *Verhaltenssicht* der Entwurfsphase betrachtet Systemereignisse und ihre Auswirkungen auf Konzepte der Daten- und Prozeßsicht.

Daneben werden generelle Querbezüge zwischen den Sichten jeder Phase durch zusätzliche *Integrationssichten* beschrieben. Die Integration zwischen Analyse- und Entwurfsphase erfolgt bei der Beschreibung der Konzeptmodellierungen der Entwurfsphase durch Rückgriff auf Konzepte der Analysephase. Die einzelnen Konzeptmodelle werden jeweils durch eine Liste der modellierten Konzepte (Objekt-Typen), einem Objekt-Beziehungsdiagramm und einer glossarartigen Beschreibung der Objekttypen notiert. Jedes Konzeptmodell wird darüber hinaus noch durch ein umfangreiches Beispiel erklärt. Als Beschreibungsmittel werden hier neben Objekt-Beziehungsdiagrammen ausschließlich tabellenartige Notationen verwendet.

Schwerpunkt der Betrachtung des IFIP-Modells sind in erster Linie die Konzepte zur Modellierung von Informationssystemen. Beziehungen zwischen diesen werden als Bestandteil der Konzepte betrachtet. Dynamische Modellierungsaspekte werden im IFIP-Modell,

außer durch die Gliederung nach den Phasen des Software-Lebenszyklus, nicht beschrieben. Das IFIP-Modell sollte daher eher als *Metaschema* aufgefaßt werden.

Intendiertes Ziel bei der Entwicklung des IFIP-Metaschemas war es, einen Rahmen zu schaffen, der das Nachvollziehen und Verstehen unterschiedlicher Ansätze zur Modellierung und softwaretechnischen Unterstützung von Informationssystemen ermöglicht und die konzeptionellen Querbezüge zwischen den verwendeten Techniken aufdeckt. In [Olle et al., 1991, Kapitel 6] werden hierzu vier exemplarische Modellierungsmethoden entlang der Konzepte des Metaschemas charakterisiert. Diese Modellierungsmethoden werden durch Folgen von Modellierungsschritten eingeführt. Die Ergebnisse der einzelnen Schritte werden jeweils durch die betrachteten Konzepte des Metaschemas näher beschrieben. Zur Einordnung dieser exemplarischen Modellierungsmethoden werden ihre (sehr einfachen) Meta-Aktivitätsmodelle mit dem Metaschema in Beziehung gesetzt. Die im IFIP-Metaschema notierten Konzepte zur Beschreibung von Informationssystemen bilden somit ein *Referenz-Metaschema* zur Untersuchung von Modellierungsmethoden.

CC-RIM-Referenzmodell

Im Rahmen des „Kompetenzzentrums Rechnergestütztes Informationsmanagement“ (CC-RIM) im Forschungsprogramm „Informationsmanagement 2000“ entstand zwischen 1989 und 1992 unter Federführung des Instituts für Wirtschaftsinformatik der Hochschule St. Gallen das CC-RIM-Referenzmodell für den Entwurf von betrieblichen, transaktionsorientierten Informationssystemen [Österle / Gutzwiller, 1992], [Gutzwiller, 1994]. Während im IFIP-Metamodell die statischen Aspekte der Modellierung von Informationssystemen im Vordergrund standen, werden im CC-RIM-Referenzmodell auch die dynamischen Aspekte von Modellierungsmethoden betrachtet. Als Bezugsrahmen dienen auch hier die Phasen des Software-Lebenszyklus und die Betrachtungssichten auf Informationssysteme. Entlang der Phasen werden Modelle für die Voruntersuchung, für den Entwurf eines Soll-Modells (Analysephase) und für die Erstellung der logischen Systemspezifikation (Entwurfsphase) erstellt. Diese Modelle bestehen aus Metadatenmodellen, die die abstrakte Syntax der Entwurfsdaten beschreiben, aus Dokumentationsmodellen, die die während des Entwurfsprozesses erzeugten Dokumente konzeptionell darstellen und aus Vorgehensmodellen, die die einzelnen Entwurfstätigkeiten im Ablauf modellieren.

In den *Metadatenmodellen* werden für die drei betrachteten Entwurfsphasen unterschiedliche Sichtenbildungen vorgenommen. Je nach Phase werden mehrere Sichten auf *Prozesse*, auf *Daten* und auf *Organisationsstrukturen* gebildet. Daneben existieren zusätzliche Sichten zur Integration dieser Teilsichten. Für die *Phase der Voruntersuchung* wird ein grobes Konzeptmodell zur Beschreibung der Geschäftsfunktionen und ihrer Kommunikations- und Datenflußbeziehungen aus Prozeßsicht und ein sehr grobes Konzeptmodell zur Beschreibung der zentralen Objekttypen aus Datensicht definiert. Diese Konzeptmodelle werden für die *Phase der Sollmodellierung* verfeinert und um Konzeptmodelle für eine eher kontrollflußorientierte Prozeßsicht und für eine Organisationssicht ergänzt. Wie auch im IFIP-Modell werden für die *logische Systemspezifikation* bzw. *Entwurfsphase* eher implementationsnähere Modellierungskonzepte untersucht. In den Prozeßsichten stehen Konzepte zur Modellierung von Systemapplikationen und des Dialogablaufs im Vordergrund. Aus Datensicht werden Datenbankkonzeptionen betrachtet. Neben einer Sicht zur Bildschirmgestaltung werden organisatorische Aspekte zur Stellenbildung, zur Verteilung der

Systemkomponenten und zur Sicherheit in weiteren Sichten konzeptionell abgebildet. Die Metadatenmodelle werden durch eine Liste der modellierten Konzepte, einem Objekt-Beziehungsdiagramm und einer glossarartigen Beschreibung der Konzepte notiert.

Objektstrukturen einzelner Dokumentarten, die Ergebnisse der Modellierung von Informationssystemen beschreiben, werden in *Dokumentationsmodellen* zusammengefaßt. Dieses erfolgt für jede Dokumentenart und Modellierungsphase durch Sichtenbildung auf die Metadatenmodelle. Für die *Voruntersuchung* werden z. B. Sichten für Kontextdiagramme, Datenflußdiagramme, grobe Objekt-Beziehungsdiagramme und Sichten für diverse Listen von Geschäftsfunktionen, Objekttypen und Datenbanken definiert. Sichten u. a. für Objekt-Beziehungsdiagramme, hierarchische Funktionsgliederungspläne, Datenflußdiagramme, Organigramme, textuelle Integritätsbedingungen, Zustandsübergangsdiagramme und für textuelle Beschreibungen von Geschäftsfunktionen, Objekttypen und Datenspeichern beschreiben mögliche Darstellungstechniken der *Analysephase*. Dokumentarten wie z. B. Datenbankstrukturdiagramme, Listen der Datenbanktabellen, Tabellenbeschreibungen, textuelle Integritätsbedingungen und Aufgaben-Stellen-Zuordnungen werden durch entsprechende Sichten für die *Entwurfsphase* aus dem Metadatenmodell abgeleitet.

Den dritten Teil der CC-RIM-Referenzmodelle bilden *Vorgehensmodelle*. Jede Phase wird hierzu in mehrere Unteraktivitäten hierarchisch gegliedert. Als Beschreibungsmittel dienen Ablaufdarstellungen, die neben den üblichen regulären Kontrollstrukturen (Sequenz, Verzweigung und Iteration) auch Parallelbearbeitungen verwenden. Zu jeder Aktivität werden neben einer textuellen Erklärung auch die benötigten Eingabe- und die erzeugten bzw. überarbeiteten Ergebnisdokumente angegeben. Diese Dokumenttypen sind in den Dokumentationsmodellen näher beschrieben und bilden somit die Verbindung zwischen dem Metadatenmodell und dem Vorgehensmodell.

Durch das CC-RIM-Referenzmodell wird eine konzeptionelle Grundlage zur Modellierung von Informationssystemen geschaffen. Modelle für Informationssysteme können für verschiedene Erstellungsphasen als Instanzen dieses Modells dargestellt werden. Ebenfalls wird das Vorgehen zur Erstellung dieser Modellinstanzen beschrieben. Die Metadatenmodelle und die Vorgehensmodelle des CC-RIM-Referenzmodells sind daher im Sinn der Begriffsbildung von Seite 116 als Metaschemata und Meta-Aktivitätsmodelle aufzufassen. Das CC-RIM-Referenzmodell ist folglich nicht als Referenzmodell sondern als *Metamodell* für die Modellierung von Informationssystemen zu betrachten.

Ausgehend von Beobachtungen, daß vorhandene Methoden und Werkzeuge für den Entwurf von Informationssystemen in der Praxis nicht für alle Problemzusammenhänge ausreichende Unterstützung anbieten, und daß auch vergleichende Beurteilungen und Einschätzungen vorhandener Methoden aufgrund der unterschiedlichen Terminologien und dem unterschiedlichen Aufbau nur sehr schwer möglich sind, wurde das CC-RIM-Referenzmodell im Verbund mehrerer Schweizer Unternehmen mit der Hochschule St. Gallen erstellt. Zielsetzung der Erstellung des CC-RIM-Referenzmodells war es, eine einheitliche, vollständige und widerspruchsfreie Beschreibung vorhandener Modellierungsmethoden zu schaffen. Hierdurch sollte eine Vereinheitlichung der Terminologie der Modellierungmethoden ermöglicht werden, auf deren Basis auch methoden- und werkzeugunabhängige Schulungen für Entwickler von Informationssystemen konzipiert werden kön-

nen. Das Modell sollte auch als Ausgangspunkt eines systematischen Methodenvergleichs dienen. Auf Basis von Metadatenmodellen zu diesem Modell wurden in [Färberböck et al., 1991] fünf Methoden zur Modellierung von Informationssystemen (Structured Analysis (SA) nach [DeMarco, 1978] und [Gane / Sarson, 1979], Integrierte Software Technologie (ISOTEC), Information Engineering Method (IEM/EY) nach [Ernst and Young, 1990], Information Engineering Method (IEM/JMA) nach [James Martin Associates, 1987] und [James Martin Associates, 1989] und Structured Systems Analysis and Design Method (SSADM) nach [Longworth / Nicholls, 1986]) miteinander verglichen. Hierzu wurden Metaschemata dieser Methoden aus dem CC-RIM-Metadatenmodell abgeleitet. Hiermit wurde auch die Referenzeigenschaft des CC-RIM-Metamodells nachgewiesen, so daß es in diesem Kontext als *Referenz-Metamodell* eingestuft werden kann.

Mit dem für das CC-RIM-Referenzmodell entwickelten Beschreibungsansatz wurden daneben weitere Metamodelle erstellt. So liegt beispielsweise in [Hess et al., 1995] ein Metaschema für den Prozeßentwurf und in [Derungs et al., 1996] ein Workflow Metaschema vor.

Architektur integrierter Informationssysteme

Die in [Scheer, 1992] und [Scheer, 1994, Teil A] eingeführte Architektur für integrierte Informationssysteme (ARIS) umfaßt ein umfangreiches Informationsmodell, durch das Techniken zur Entwicklung von Informationssystemen durch ihre abstrakte Syntax beschrieben werden. Analog zu den Ansätzen im IFIP-Metamodell und im CC-RIM-Referenzmodell werden auch nach dem ARIS-Ansatz Informationssysteme aus der *Datensicht*, der *Funktionssicht* (oder Prozeßsicht) und aus der *Organisationssicht* beschrieben. In Anlehnung an die Phasen des Softwarelebenszyklus der Erstellung von Informationssystemen, die hier durch ihre Ergebnisdokumente benannt werden, werden für jede Sicht Modelle auf Fachkonzeptebene, auf DV-Konzeptebene und auf Implementierungsebene erstellt. Das Fachkonzept entspricht hierbei im wesentlichen dem Ergebnis der Analysephase. Die Designphase ist bei [Scheer, 1992] in die Erstellung eines DV-Konzepts und in die Erstellung der technischen Implementierung unterteilt. Ergänzt wird auch hier eine Integrationssicht (*Steuerungssicht*), durch die Daten-, Funktions- und Organisationssichten der einzelnen Ebenen zu einem Gesamtmodell zusammengefaßt werden.

Die hieraus resultierende ARIS-Architektur legt einerseits die Sichten und Ebenen zur Beschreibung von Informationssystemen fest, bildet andererseits den konzeptionellen Rahmen zur Beschreibung des Informationsmodells, das dem ARIS-Ansatz zu Grunde liegt. Für jeden Baustein der Architektur werden die dort betrachteten Konzepte und Beziehungen einheitlich durch Objekt-Beziehungsdiagramme modelliert. In [Scheer, 1994, S. 17ff] werden die zur Darstellung dieser Zusammenhänge verwendeten Beschreibungstechniken ausführlicher beschrieben.

Auf *Fachkonzeptebene* beschreibt das ARIS-Informationsmodell aus *Funktionssicht* Unternehmensziele und Funktionen sowie deren Gliederungs- und Ablaufbeziehungen. Zur Beschreibung von Informationssystemen aus der Funktionssicht der Fachkonzeptebene werden u. a. Aufgabengliederungspläne (Funktionsbäume) und Ablauffolgediagramme eingesetzt. Die *Organisationssicht* der Fachkonzeptebene bezieht sich auf Organisationseinheiten (Stellen, Abteilungen) und deren Beziehungen. Als Beschreibungstechnik

für die Dokumentation eines Informationssystems werden hier Organigramme und Stellenbeschreibungen vorgeschlagen. Das Informationsmodell der *Datensicht* des Fachkonzepts beschreibt die Konzepte zur Darstellung der Objekte und deren Beziehungen in einem Informationssystem. Zur Modellierung solcher Strukturen wird auch hier ein Dialekt der Objekt-Beziehungsdiagramme verwendet. Das Konzeptmodell der *Steuerungssicht* des Fachkonzepts faßt die zunächst isoliert beschriebenen Teilmodelle der drei Teilsichten zusammen. Hierzu werden zwischen den wesentlichen Konzepten der Funktionssicht, der Organisationssicht und der Datensicht zusätzliche Beziehungen modelliert. Zur Darstellung der sichtenübergreifenden Zusammenhänge eines Informationssystems auf Fachkonzeptebene können u.a. Zuordnungsdiagramme zur Darstellung der Zusammenhänge zwischen organisatorischen Strukturen und Funktionen bzw. Daten, Funktionszuordnungsmatrizen, Datenflußdiagramme und Vorgangskettendiagramme genutzt werden.

Vor der Umsetzung des Fachkonzepts in die Implementierung sieht der ARIS-Ansatz die Ebene des *DV-Konzepts* vor, die das Fachkonzept des Informationssystems um implementationstechnische Aspekte, unabhängig von der konkreten Realisierung, ergänzt. Auf der Ebene des DV-Konzepts stehen aus *Funktionssicht* die Konzepte zur Beschreibung von Softwaremodulen, von Kontrollstrukturen und von Bildschirmmasken und Ausgabeformaten im Vordergrund. Zur Beschreibung dieser Aspekte eines Informationssystems sieht der ARIS-Ansatz ausschließlich semi-formale Beschreibungsmittel wie z. B. Strukturdiagramme, Nassi-Shneiderman-Diagramme, Entscheidungstabellen und Pseudo-Code vor; formale Spezifikationsmittel werden nicht berücksichtigt. Konzepte des Rechnernetzes, das die technische Plattform des Informationssystem bildet, sind der Betrachtungsschwerpunkt aus der *Organisationssicht* des DV-Konzepts. Hierbei werden sowohl einzelne Komponenten im Netz wie auch dessen Topologie berücksichtigt. Zur Darstellung der Netztopologie werden hierzu graphbasierte Darstellungen verwendet. Die *Datensicht* des DV-Konzepts bezieht sich auf die Umsetzung des Fachkonzepts in Datenbank-Managementsystemen. Im Konzeptmodell finden sich daher die Konzepte der Datendefintionssprachen. In der *Steuerungsicht* werden auch für das DV-Konzept die Modelle der Einzelsichten integriert. Das Informationsmodell der Steuerungssicht umfaßt daher u.a. Konzepte zur Beschreibung der Zugriffsrechte von Organisationseinheiten auf Programm-Module, zur Definition von Sichten auf das Datenmodell und zur Definition von Zugriffsmöglichkeiten sowohl von Modulen wie von Organisationseinheiten auf diese Sichten.

Das *Implementationsmodell* eines Informationssystems beschreibt dessen softwaretechnische Realisierung. In der *Funktionssicht* des Informationsmodells werden hierzu Konzepte zur Darstellung ausführbarer Programme und deren Zusammenfassung zu Programmbibliotheken modelliert. In der ARIS-Architektur umfaßt diese Sicht auch Konzepte für Software-Erstellungswerkzeuge wie Compiler und Programmgeneratoren. Die *Organisationssicht* des Implementationsmodells stellt Konzepte zur Modellierung der Zuordnung zwischen einzelnen Hard- und Software-Komponenten bereit. Als Darstellungsmittel dieser Zusammenhänge wird auch hier eine graphbasierte Darstellung verwendet. Aus *Datensicht* des Implementationsmodells wird die konkrete Realisierung der Datenstrukturen beschrieben. Im Informationsmodell werden hierzu Konzepte zur Modellierung von Datensätzen, Speichermedien und Zugriffspfade auf Daten bereitgestellt. In der *Steuerungssicht* werden auch für das Implementationsmodell die drei anderen Sichten integriert. Dieses erfolgt durch das Konzept der Reservierung, das Dateien, Programme und die Hardware-

Komponenten, auf denen die Dateien gespeichert, bzw. auf denen die Programme ausgeführt werden, zusammenfaßt.

Die Integration der Konzeptmodelle zum Fachkonzept, zum DV-Konzept und zur Implementierung erfolgt jeweils durch Rückbezug auf Konzepte, die bereits bei der Modellierung des jeweils abstrakteren Informationsmodells eingeführt wurden Das resultierende Informationsmodell wird in [Scheer, 1992, Abb. B.V.] in einem Diagramm abgebildet.

Die Erstellung der Modelle für Informationssysteme wird für die Fachkonzeptebene aus Funktionssicht, aus Organisationssicht, aus Datensicht und aus Steuerungssicht kurz diskutiert. Für den Ablauf der Modellerstellung für die Funktions- und Datensicht wird dieses durch einfache Aufgabengliederungen und Prozeßketten skizziert. Das Vorgehen zur Modellerstellung für das DV-Konzept und die Implementierung wird nicht näher betrachtet.

Das resultierende Informationsmodell der ARIS-Architektur beschreibt Konzepte und ihre Beziehungen, die der Modellierung von Informationssystemen zu Grunde gelegt werden können. Dieses Informationsmodell kann somit als *Metaschema* zur Beschreibung von Informationssystemen nach der ARIS-Methode aufgefaßt werden. Mit Ausnahme der rudimentären Vorgehensbeschreibung zur Modellierung des Fachkonzepts enthält die ARIS-Architektur kein Meta-Aktivitätsmodell.

Eingesetzt wird dieses Metaschema als theoretische Grundlage der auf ARIS basierenden Methode zur Modellierung von Informationssystemen. Daneben dient es auch als konzeptionelles Datenbankschema. Das ARIS-Informationsmodell beschreibt die Datenbankkonzeption des ARIS-Toolsets [IDS, 1995] zur Erstellung, Verwaltung und Analyse von Modellen für Informationssysteme nach der ARIS-Methode.

Multi-Perspektivische Unternehmensmodellierung

Ein Ansatz zur objektorientierten Unternehmesmodellierung wird mit MEMO (Multi Perspective Enterprise Modeling) in [Frank, 1994] vorgestellt und u.a. in [Frank, 1997a]) weiterentwickelt. In MEMO werden Unternehmen aus drei Sichten oder Perspektiven betrachtet. Während die Sichten bei den zuvor vorgestellten Ansätzen eher durch die Unterscheidung von statischen und dynamischen Aspekten der Unternehmen geprägt waren, sind die hier verwendeten Perspektiven entlang der Rollen der Beteiligten bestimmt. [Frank, 1994] unterscheidet hierbei die strategische Perspektive, die organisatorische Perspektive und die Informationssystem-Perspektive. Für jede dieser Perspektiven werden aber auch dynamische Aspekte (Prozeß) und statische Aspekte (Struktur) sowie die benötigten Resourcen und die jeweils verfolgten Ziele näher charakterisiert.

Aus der Sicht hochrangiger Führungskräfte werden in der *strategischen Perspektive* Unternehmensziele und -strategien modelliert. Unternehmen werden aus dieser Sicht, die in den bisher skizzierten Methoden keine Entsprechung hatte, in Anlehnung an den Wertketten-Ansatz nach [Porter, 1985] modelliert. Zur Beschreibung werden Primäraktivitäten einschließlich der sie unterstützenden Aktivitäten in einfachen Folgedarstellungen notiert (vgl. [Frank, 1997a, Fig. 2]). Die *organisatorische Perspektive* beschreibt die Sicht der für das operative Geschäft Verantwortlichen. Hier stehen insbesondere die aufbau- und ablauforganisatorischen Regelungen des Zusammenarbeitens im Unternehmen im Vordergrund. Zur Darstellung der dynamischen Aspekte wird die Sprache MEMO Process Modeling

Language (M-PML) [Frank, 1997a] verwendet, bei der Geschäftsprozesse durch ihre Aktivitäten und die hiervon generierten bzw. benötigten Dokumente in bipartiten Graphen beschrieben werden. Zur Darstellung der Aufbauorganisation werden auch hier Organigramme verwendet, die neben Leitungsbeziehungen auch Kommunikationsbeziehungen enthalten. Die Sicht der Entwickler und Betreiber von Informationssystemen wird in der *Informationssystem-Perspektive* behandelt. Dynamische Aspekte der Informationssystem-Perspekive werden durch eine Erweiterung von M-PML beschrieben, wobei die hier dargestellten Workflows als Verfeinerung der in der organisatorischen Perspektive beschriebenen Geschäftsprozesse aufgefaßt werden. Zu Beschreibung der Objektstrukturen wird die MEMO Object Modeling Language (M-OML) [Frank, 1998] des Objekt-Beziehungsparadigmas verwendet. Die enthält hierzu getrennte Darstellungsformen zur Beschreibung von Generalisierungen und zur Beschreibung von Beziehungsklassen zwischen Objektklassen.

Jede der drei Perspektiven ist durch ein in M-OML notiertes Metamodell konzeptionell beschrieben. Zur Integration der Teilmodelle (vgl. die Skizze in [Frank, 1997a, S. 12f]) werden zwischen den Objektklassen der Metamodelle der einzelnen Perspektiven zusätzliche Beziehungstypen definiert, eine Zusammenfassung gleicher bzw. ähnlicher Konzepte erfolgt jedoch nicht.

Zur Unterstützung der Unternehmensmodellierung mit MEMO existiert ein Werkzeug (MEMO-Center), bei dem die einzelnen Teilmodelle in aus dem MEMO-Metamodell abgeleiteten Masken textuell erfaßt werden. Darüber hinaus erlaubt MEMO-Center eine graphische Darstellung der Teilmodelle.

Schwerpunkt der Arbeiten an MEMO war die Erstellung eines Metamodells zur durchgängig objektorientierten und perspektivenübergreifenden Unternehmensmodellierung. Hierzu wurden in [Frank, 1994] umfassend motivierte Meta-Objektmodelle für die einzelnen Perspektiven definiert und zu einem integrierten Meta-Objektmodell zusammengefaßt. Dieses beschreibt die durch MEMO-Modelle abbildbaren Objekte einschließlich deren Beziehungen und stellt damit im Sinn der Begriffsfindung von Seite 116 ein *Metadatenmodell* dar. Das Vorgehen zur Erstellung einer MEMO-Modellierung wird in [Frank, 1994, S. 338ff] und in [Frank, 1997a, S. 14f] kurz durch Angabe der auch wiederholt ausführbaren, wesentlichen Modellierungsschritte skizziert.

Überblicksdarstellungen zu weiteren Ansätzen zur Metamodellierung von Informationssystemen finden sich auch in [Scheer, 1994, S. 29ff], [Gutzwiller, 1994, S. 15ff] und [Frank, 1994, S. 139ff].

Metamodelle für Entwurfsmethoden

Metamodelle dienen auch als Grundlage zur Beschreibung und Einordnung von Methoden und Techniken zum Softwareentwurf. In den folgenden Abschnitten zu *SOCRATES-Metamodellen*, zu *CDIF-Metamodellen*, zu *EER/GRAL-Metamodellen*, zu *KOGGE-Metamodellen* und zum *UML-Metamodell* werden verschiedene Ansätze zur Metamodellierung von Entwurfsmethoden und deren Anwendung skizziert.

SOCRATES-Metamodelle

Zielsetzung des SOCRATES-Projektes [Verhoef et al., 1991] war die Erstellung einer Architektur für CASE-Werkzeuge, die sowohl den Modellierungsprozeß (way of working) als auch die modellierten Dokumente (way of modelling) berücksichtigt. Hierzu wird in [Wijers et al., 1992] eine Modellierungstechnik entwickelt und formal beschrieben. Zur Beschreibung des „way of modelling“ werden die statischen Aspekte der Modellierungstechniken durch Objekt-Beziehungsdiagramme im NIAM-Dialekt [Verheijen / van Bekkum, 1982], ergänzt um zusätzliche Integritätsbedingungen in Prädikatenlogik, notiert. Der „way of working“ wird durch Ablaufdarstellungen der einzelnen Aktivitäten beschrieben. Diese Ablaufdarstellungen werden in [Wijers et al., 1992] formal eingeführt und u.a. auch durch ein NIAM-Metaschema charakterisiert. Methoden werden nach der SOCRATES-Modellierungstechnik somit sowohl durch ein Metaschemata als auch durch ein Meta-Aktivitätsmodell beschrieben. Daher können diese Modelle auch als *Metamodelle* betrachtet werden.

Entlang des SOCRATES-Ansatzes zur Metamodellierung werden in verschiedenen Arbeiten Modellierungsmethoden beschrieben und verglichen. Die SOCRATES-Metamodellierungsmethode wird in [Wijers et al., 1992] am Beispiel eines Metamodells für Jackson Structured Development (JSD) [Jackson, 1983] erläutert. Ein Metamodell der modernen strukturierten Analyse nach [Yourdon, 1989] wird in [Verhoef et al., 1991] diskutiert. SOCRATES-Metamodelle zu den drei Planungsansätzen für Informationssysteme Information Systems Planning (ISP) [Arthur Young International, 1988], Information Planning [Raet BV, 1988] und Strategic Data Planning [Martin, 1982] werden in [Brinkkemper et al., 1989] kurz skizziert. Diese Modelle wurden ausgehend von den Methodenbeschreibungen erstellt und miteinander verglichen. Die Ähnlichkeiten und Differenzen dieser Methodenbeschreibungen dienten als Ausgangspunkt zur Ableitung einer allgemeinen, verbesserten Methode, die als Vergleichsmaßstab für die drei Ausgangsmethoden genutzt wurde.

Ein ähnliches Vorgehen wird auch bei einem Vergleich von sechs objektorientierten Analyse- und Design Methoden [Goor et al., 1992] verfolgt. Ziel dieser Studie ist es, die Methoden Object-Oriented Analysis and Design (OOA/OOD) [Coad / Yourdon, 1991], Designing Object-Oriented Software (DOOS) [Wirfs-Brock et al., 1990], Object Modeling Technique (OMT) [Rumbaugh et al., 1991], Object-Oriented Design (OOD) [Booch, 1994], Object-Oriented Systems Analysis (OOSA) [Shlaer / Mellor, 1988] und Object-Oriented Systems Analysis and Design (OOAD) [Martin / Odell, 1992] nach einem festen Raster formal zu beschreiben.

Hierzu werden zunächst alle sechs Methoden u.a. durch die Hauptaktivitäten, eine glossarartige Beschreibung der Modellierungskonzepte und ihrer graphischen Notation charakterisiert. Die Modellierungsaktivitäten der Methoden werden durch sequentiell zu durchlaufende Modellierungsschritte beschrieben. Ergänzt werden die einzelnen Modellierungsschritte um die hierbei benötigten bzw. erzeugten Dokumente. Zur Beschreibung der abstrakten Syntax der Methoden werden auch hier Objekt-Beziehungsdiagramme nach NIAM, ergänzt um wenige, hier textuell notierte Einschränkungen, verwendet. Entlang dieser Metamodelle wurden die sechs Methoden entlang eines durchgehenden Beispiels und einer „Supermethode“ die durch eine hierarchisch gegliederte Aktivtätenliste und eine Liste der verwendeten Konzepte beschrieben wurde, miteinander verglichen. Diese “Su-

permethode“ wurde analog zu [Brinkkemper et al., 1989] aus den Metamodellen der Einzelmethoden abgeleitet, indem die allen Methoden gemeinsamen Aktivitäten und Konzepte zunächst übernommen und anschließend „optimiert“ wurden.

In [Brinkkemper et al., 1990] wird darüber hinaus gezeigt, wie mit Hilfe von SOCRATES-Modellen Modellierungsmethoden mit der Modellierungsunterstützung durch Werkzeuge abgeglichen werden können. Für die Methode System Development Methodology (SDM) [Turner et al., 1987] und die Information Engineering Workbench (IEW) [KnowledgeWare, 1987] wurden hierzu Metamodelle erstellt und miteinander verglichen.

Für den Methodenvergleich in [Brinkkemper et al., 1989] und [Goor et al., 1992] dienen die „Supermethoden“ als *Referenz-Metamodelle*. Im Gegensatz zum Vergleich der Methoden zur Modellierung betrieblicher Informationssysteme aus [Färberböck et al., 1991], bei dem die Metaschemata der untersuchten Methoden entlang des CC-RIM-Referenz-Metaschemas abgeleitet werden, werden in [Brinkkemper et al., 1989] und [Goor et al., 1992] die Referenz-Metamodelle aus den Modellen der untersuchten Methoden entwickelt. Bezogen auf die Erstellung der Metamodelle der „Supermethoden“ können in diesem Fall die Metamodelle der untersuchten Methoden als Referenz-Metamodelle aufgefaßt werden.

CDIF-Metamodelle

Das *CASE Data Interchange Format* (CDIF) [Ernst, 1997] [Flatscher, 1998] wurde 1987 und 1994 durch eine Standardisierungsgruppe der „Electronic Industries Alliance (EIA)“ entwickelt und durch die ISO als internationaler Standard [ISO/IEC DIS 15474, 1998], [ISO/IEC DIS 15475, 1998], [ISO/IEC DIS 15476, 1998] verabschiedet. Ausgangspunkt der Überlegungen zu CDIF war die Feststellung, daß CASE-Werkzeuge zwar ähnliche Modellierungsmethoden unterstützen, die erstellten Modelle jedoch nicht austauschbar waren. Mit CDIF sollte ein von Herstellern unabhängiges Austauschformat für Modelldaten zwischen CASE-Werkzeugen geschaffen werden.

Hierzu wurde in [EIA/IS-107, 1994][3] ein auf den Konzepten der Objekt-Beziehungsmodellierung basierendendes Meta-Metamodell zur Definitition von Metamodellen zur Beschreibung der auszutauschenden Daten definiert. Dieses wurde um eine Notation zum konkreten Austausch der Modelldaten ergänzt [EIA/IS-108, 1994], [EIA/IS-109, 1994], [EIA/IS-110, 1994].

Zur Erstellung der Metamodelle für CASE-Tool-Modelle wurden zwei grundlegenden Metamodelle vereinbart [EIA/IS-111, 1994], [EIA/IS-112, 1995], die in [EIA/IS-113, 1994] zur Beschreibung von Datenmodellen und in [EIA/IS-115, 1995] zur Beschreibung von Datenflußmodellen konkretisiert wurden. Weitere Metamodelle u. a. zur Beschreibung von Geschäftsprozeßmodellen, Modellen der objektorientierten Analyse oder von Zustandsübergangs-Modellen wurden erstellt [Flatscher, 1996].

Die Beschreibung der Metamodelle erfolgt durch einfache Objekt-Beziehungsdiagramme, die um umfangreiche Listen zur Darstellung der Generalisierungshierarchien und der Attributstrukturen der Objekt und Beziehungsklassen ergänzt werden. Die CDIF-Metamodelle

[3] Einen groben Überblick über die Famile der CDIF-Standards einschließlich Auszüge der einzelnen Standards finden sich auf `http://www.eigroup.org/cdif/online.html` (20.08.1999).

beschreiben somit CASE-Modelle entlang ihrer Konzepte und Beziehungen. Vorgehensaspekte der einzelnen Modellierungsansätze werden nicht betrachtet. Die CDIF-Metaschemata wurden mit dem Ziel entwickelt die jeweils betrachteten Modellierungsparadigmen möglichst vollständig abzubilden. Zur Anwendung dieser Modelle in konkreten CASE-Werkzeugen sieht CDIF Erweiterungsmöglichkeiten auf Basis des CDIF-Meta-Metamodells vor. Mit diesen Erweiterungsmöglichkeiten sind die CDIF-Modelle daher als Referenz-Metaschemata einzustufen.

Der in erster Linie durch ein Industrie-Konsortium entwickelte CDIF Standard zum Austausch von Modelldaten wird in einigen kommerziellen CASE-Werkzeugen eingesetzt. Bereits 1992 unterstützte der ORACLE-Designer/2000 den Austausch von Datenlexika mit Fremdwerkzeugen. Pardigm Plus unterstützt ebenfalls den Austausch von Modelldaten auf CDIF-Basis (vgl. auch [Flatscher, 1998, S. 15ff]).

EER/GRAL-Metamodelle

Der dieser Arbeit zu Grunde gelegte *EER/GRAL*-Ansatz [Ebert et al., 1996b] bildet u. a. auch die Basis zur Metamodellierung von Analyse- und Entwurfsmethoden. Mit *EER/GRAL*-Modellen wird die abstrakte Syntax der Modellierungsmittel durch erweiterte Objekt-Beziehungsdiagramme dargestellt. Zusätzliche Integritätsbedingungen werden in der auf der Spezifikationssprache *Z* [Spivey, 1992] basierenden Sprache *GRAL* [Franzke, 1997] formuliert (vgl. zum *EER/GRAL*-Ansatz auch Kapitel 5.2). Da mit *EER/GRAL*-Modellen Modellierungsmethoden nur durch ihre statischen Aspekte beschrieben werden, werden diese Modelle als *Metaschemata* eingeordnet.

Ausgangspunkt für die Erstellung der *EER/GRAL*-Modelle für objektorientierte Analyse- und Entwurfsmethoden war die Feststellung, daß diese Methoden in der Regel nur anhand ihrer graphischen Symbole und durch Beispielmodellierungen eingeführt werden. Eine formale Basis, die z. B. die Überprüfung einer vorliegenden Modellierung auf syntaktische Korrektheit erlaubt, existiert häufig nicht. Durch Modellierungsmethoden werden Systeme multiperspektivisch, aus unterschiedlichen Sichten beschrieben. Auch die Integration der hierbei verwendeten Techniken wird nur beispielhaft skizziert, so daß eine vollständige Beschreibung der Konzepte, einschließlich der Querbezüge über die einzelnen Modellierungstechniken hinaus, ebenfalls fehlt.

In [Ebert / Süttenbach, 1997a] und [Süttenbach / Ebert, 1997] werden *EER/GRAL*-Metaschemata für die Object Modeling Technique (OMT) nach [Rumbaugh et al., 1991] und für Object-Oriented Design with Applications (OOD) nach [Booch, 1994] definiert. Ausgehend von den graphischen Symbolen der in den Methoden verwendeten Beschreibungstechniken werden die einzelnen Modellierungsichten konzeptionell beschrieben. Diese Teilmodelle werden anschließend zu Gesamtmodellen der jeweiligen Methoden zusammengefaßt.

Ergebnis dieser Modellierungen sind Metaschemata, die die abstrakte Syntax der in beiden Methoden verwendeten visuellen Beschreibungstechniken über die jeweiligen Modellierungssichten integriert. Diese formalen Beschreibungen geben einen vollständigen Überblick über die in den Methoden verwendeten Konzepte und ihrer Zusammenhänge. Im Gegensatz zu den SOCRATES-Metamodellen zu OMT und OOD [Goor et al.,

1992], bei denen der Vergleich der Methoden im Vordergrund stand, bieten die *EER/GRAL*-Metaschemata aus [Ebert / Süttenbach, 1997a] und [Süttenbach / Ebert, 1997] auch aufgrund ihres Detaillierungsgrades eine formale Grundlage, gegen die konkrete Modellierungen nach diesen Ansätzen auf (syntaktische) Korrektheit überprüft werden können. Bei der Erstellung der *EER/GRAL*-Metaschemata konnten nicht alle Eigenschaften der modellierten Konzepte eindeutig aus den Methodenbeschreibungen abgeleitet werden. In diesen Fällen wurde in den Metamodellen eine mögliche, sinnvolle Interpretation vorgeschlagen. Diese *EER/GRAL*-Modelle sind daher auch als *Referenz-Metaschemata* in Diskussionen zur Schaffung einer allgemein akzeptierten Interpretation der Modellierungskonstrukte verwendbar.

Wie auch der CDIF-Ansatz basiert *EER/GRAL* auf einem Metamodell, das in *EER/GRAL* selbst formalisiert ist. Das Metaschema des EER-Anteils zur Modellierung nach dem Objekt-Beziehungsparadigma wird in [Ebert et al., 1999a] und das Metaschema des *GRAL*-Anteils zur Darstellung von Integritätsbedingungen wird in [Ebert et al., 1997a] beschrieben. Varianten dieses Metaschemas in [Ebert et al., 1997a] beschreiben die abstrakte Syntax der Spezifikationssprache Z [Spivey, 1992] und der Graph-Anfragesprache *GReQL* [Kamp, 1998]. Weiter dient es als Grundlage zur Festlegung der Semantik von *EER/GRAL*-Modellen in [Dahm et al., 1998b].

KOGGE Metamodelle

Zur Erstellung von Werkzeugen zur Modellierung nach einer vorgegebenen Methode wird eine Beschreibung dieser Methode benötigt. Methoden und Techniken, die durch das Werkzeug unterstützt werden sollen, werden hierzu durch Metamodelle beschrieben. Diese Werkzeugerstellung kann direkt durch Entwicklung auf Basis eines Metamodells erfolgen. Größere Flexibilität und Adaptivität bei der Werkzeugerstellung erlaubt die Verwendung von Meta-CASE-Werkzeugen, also von Werkzeugen zur Erstellung von Modellierungswerkzeugen. Während direkt erstellte Modellierungswerkzeuge i. allg. auf die Unterstützung der fest implementierten Methode eingeschränkt sind, erlauben Meta-CASE-Werkzeuge durch Änderung der Metamodelle, die hierzu als Werkzeugbeschreibungen aufgefaßt werden können, die Anpassung des Modellierungswerkzeugs an die (sich ändernden) spezifischen Anforderungen der Benutzer [Ebert, 1997]. Der *Koblenzer Generator für Graphische Entwurfsumgebungen (KOGGE)* ist ein solches Meta-CASE-Werkzeug [Ebert et al., 1997b], [Ebert et al., 1999b].

Mit *KOGGE* wird das Ziel verfolgt, Editoren für visuelle Sprachen und Methoden zu generieren. Hierzu wird für die in einer Modellierungsmethode verwendeten Techniken eine Werkzeugbeschreibung erstellt. Konkrete Modellierungswerkzeuge (*KOGGE*-Werkzeuge) entstehen durch Interpretieren der jeweiligen Werkzeugbeschreibungen durch ein statisches Basissystem. Eine *KOGGE*-Werkzeugbeschreibung besteht aus der Definition der Konzepte der Methode, der Definition der Menüstruktur des Werkzeugs und der Definition der Interaktionen mit dem Benutzer. In der Konzeptbeschreibung wird die abstrakte Syntax der in der Methode verwendeten Techniken, d. h. deren Sprachkonzepte und ihre Beziehungen notiert. Hierzu werden *EER/GRAL*-Modelle verwendet. Dieser Teil der Werkzeugbeschreibung umfaßt somit das *Metaschema der Methode*. Die Menüstruktur wird durch einen azyklischen Menügraphen beschrieben. Das Werkzeugverhalten, d. h. die Interaktion mit dem Benutzer wird durch Statecharts und eine Makrosprache definiert. Dynamische

Aspekte einer Modellierungsmethode sind in *KOGGE*-Werkzeugbeschreibungen nicht enthalten.

Zur Erstellung von Werkzeugbeschreibungen existiert ein eigenes *KOGGE*-Werkzeug (UrKOGGE), welches Editoren zur Erfassung und Manipulation von *EER/GRAL*-Modellen, Menügraphen, Statecharts und Programmtexten der Makrosprache bereitstellt. Die UrKOGGE basiert selbst auf einer *KOGGE*-Werkzeugbeschreibung, deren Metaschema-Anteil auf [Carstensen, 1996] zurückgeht. Dieses Metaschema integriert die Konzepte aller zur Erstellung von Werkzeugbeschreibungen verwendeten Modellierungstechniken. Darüber hinaus enthält das Metaschema aus [Carstensen, 1996] auch Konzeptmodellierungen für weitere Beschreibungsparadigmen, die jedoch in Werkzeug UrKOGGE keine Verwendung finden. Diese Modellierungen können jedoch als *Referenzen für Metaschemata* zu diesen Entwurfssprachen angesehen werden und flossen in dieser Funktion auch in die in Teil II beschriebenen Metamodelle ein.

KOGGE-Werkzeuge und somit auch Metaschemata existieren neben der UrKOGGE für Datenflußdiagramme [Drüke, 1996], für die objektorientierte Methoden BON [Ebert et al., 1997b], [Kölzer / Uhe, 1997] und die Unified Modeling Language. In Kapitel 9 wird skizziert, wie ein *KOGGE*-Werkzeug zur Organisationsmodellierung [Löcher / Pühler, 1997] entlang des in Teil II hergeleiteten Referenz-Metaschemas erstellt wurde.

UML Metamodell

Mit der Definition der *Unified Modeling Language (UML)*, [Booch et al., 1999], [Rumbaugh et al., 1999] wird das Ziel verfolgt, eine allgemeine, visuelle Sprache als Standard zur Beschreibung, Visualisierung und Konstruktion von objektorientierten Systemen bereitzustellen. Eine Ausschreibung für Vorschläge zur Standardisierung objektorientierter Modellierungssprachen erfolgte im Juni 1996 durch die Object Management Group (OMG).

Ausgehend von Erweiterungen zur Object Modeling Technique (OMT) [Rumbaugh et al., 1991] und zu Object-Oriented Design (OOD) [Booch, 1994] wurde seit Ende 1994 gemeinsam von G. Booch und J. Rumbaugh der erste UML-Entwurf (Unified Method) erstellt und 1995 präsentiert. Mit der Mitarbeit von I. Jacobson seit Herbst 1995 flossen auch die Use-Cases von aus Object-Oriented Software Engineering (OOSE) [Jacobson et al., 1993] in die Sprachentwicklung ein [Booch, 1996], [Booch et al., 1999]. Der resultierende Sprachentwurf wurde Ende 1996 der interessierten Öffentlichkeit zur Diskussion vorgelegt und gemeinsam mit den im UML Partner Consortium zusammengeschlossenen Firmen überarbeitet. Im Januar 1997 wurde die Version 1.0 mit fünf weiteren Vorschlägen, die zum Teil lediglich Ergänzungen zu UML darstellen [Frank / Prasse, 1997b], bei der OMG eingereicht. Die aus der Ergänzung um einige dieser Vorschläge hervorgegangene Sprachbeschreibung (Version 1.1) wurde am 14. November 1997 durch die Object Management Group akzeptiert. UML kann folglich als eine Weiterentwicklung und Verschmelzung von OMT, OOD, OOSE und weiteren Ansätzen verstanden werden, die bereits durch eine große Zahl von Anwendern im UML Partner Consortium getragen wird. Eine kritische Würdigung dieses Standardisierungsverfahrens zur Schaffung einer einheitlichen, allgemein akzeptierten, objektorientierten Modellierungssprache findet sich in [Frank / Prasse, 1997b].

Zur Beschreibung objektorientierter Systeme stellt auch die UML mehrere visuelle Sprachen [Booch et al., 1999], [OMG, 1999, S. 3-1ff] bereit[4]. Neben Use-Case-Diagrammen und Objekt-Beziehungsdiagrammen (Klassendiagrammen) werden u. a. Statecharts, Aktivitätsdiagramme und Interaktionsdiagramme als visuelle Darstellungsmittel verwendet. Wie generell für objektorientierte Modellierungsmethoden sind auch in UML Objekt-Beziehungsdiagramme eine wesentliche Modellierungstechnik, die auch zur Modellierung von Metamodellen eingesetzt werden kann. Für die Standardisierung von UML ist es daher naheliegend, die Methode auch mit UML-Mitteln zu beschreiben.

Die UML-Beschreibungen in [OMG, 1999, S. 2-1ff] umfassen sowohl die Festlegung der abstrakten Syntax als auch die Darstellung der Bedeutung dieser Sprachelemente. Zur Festlegung der abstrakten Syntax werden die Sprachkonstrukte der UML und ihre Beziehungen durch UML-Klassendiagramme definiert und glossarartig näher beschrieben. Zusätzliche Integritätsbedingungen an diese Konzeptmodellierung werden natürlichsprachlich und formal in der Object Constraint Language (OCL) [OMG, 1999, 6-1ff] formuliert. OCL entstammt der Modellierungsmethode Syntropy [Cook / Daniels, 1994] und wurde von IBM und ObjecTime als „*leicht les- und schreibbare*“ [OMG, 1999, S. 6-3ff] formale Sprache zur Geschäftsmodellierung entwickelt und ist seit Version 1.1 auch Bestandteil der UML. Während durch dieses Modell die Modellierungskonzepte der UML und die Zusammenhänge zwischen diesen beschrieben werden, ist das Vorgehen zur Erstellung einer UML-Modellierung nicht in der Dokumentation der Sprache enthalten. Dieses Modell beschreibt somit das *Metaschema für UML*, das aufgrund der verwendeten Modellierungssprache auch ein *Metaschema in UML* ist. Verwendung findet dieses Metaschema als Grundlage zur Festlegung der Semantik von UML. Die Bedeutung der einzelnen Sprachkonstrukte wird durch einen entsprechenden Ausschnitt des Metaschemas eingeleitet und natürlichsprachlich beschrieben. Das so entstandene Dokument zur Semantikbeschreibung [OMG, 1999, S. 2-1ff] stellt gemeinsam mit der Festlegung der konkreten Syntax in [OMG, 1999, S. 3-1ff] die wesentliche Beschreibung der Modellierungsmethode zur Diskussion der Unified Modeling Language im Rahmen des Standardisierungsverfahrens dar.

Weitere Metaschemata für objektorientierte Entwurfsmethoden als Grundlage zur Beurteilung dieser Methoden werden auch in [Strahringer, 1996, S. 121ff] vorgestellt. Ausschnitte aus Metaschemata zu Methoden, die im Rahmen der Standardisierung objektorientierter Methoden eingereicht wurden, finden sich in [Frank, 1997b].

4.3.2 Zusammenfassung: Anwendungsbereiche von Metamodellen

Wie die zuvor skizzierten Beispiele für Metamodelle zeigen, werden Metamodelle sowohl in ihrer Rolle als *spezielle Modelle* als auch in der Rolle als *Referenzmodelle* verwendet.

In ihrer Funktion als *spezielle Modelle* dienen Metamodelle als generelles Mittel zur *Beschreibung* und *Untersuchung* von Methoden und Techniken zur Modellierung. Durch die Darstellung der abstrakten Syntax der Methoden in Metaschemata und durch die Beschreibung des Modellierungsvorgehens in Meta-Aktivitätsmodellen eignen sie sich auch zur *Schulung* dieser

[4] Die jeweils aktuellen Sprachbeschreibungen zu UML sind über `http://www.rational.com/uml/` (14.09. 1999) abrufbar.

Modellierungsmethoden. Die Metaschemata bieten darüber hinaus auch die Grundlage zur Festlegung der innerhalb der Methode verwendeten *Terminologie*. Für den *Bau von Werkzeugen* zur Unterstützung einer vorgegebenen Methode liefern die Metaschemata auch das konzeptionelle Datenmodell. Metamodelle werden auch als Ausgangspunkt zur *Beschreibung der Semantik* der Sprachelemente von Modellierungsmethoden verwendet. Aufgrund der formalen Beschreibung der abstrakten Syntax einer Methode in einem Metamodell bildet dieses auch die Grundlage zur *Überprüfung der (syntaktischen) Korrektheit* von Modellen.

Im Zusammenspiel mit anderen Metamodellen kann Metamodellen auch *Referenz-Charakter* zugesprochen werden. Methodenbeschreibungen durch Metamodelle bieten eine aussagekräftige Grundlage zur *Diskussion der Modellierungskonzepte*. In diesem Zusammenhang werden Metamodelle z. B. auch als Ausgangspunkt zur *Standardisierung* verwendet. Ein wesentlicher Anwendungsbereich von Referenz-Metamodellen ist ihre Verwendung als Basis zum *Vergleich* der Konzepte und Vorgehensweisen unterschiedlicher Modellierungsmethoden und zur *Entwicklung* von Metamodellen zu Modellierungsmethoden. Solche Referenz-Metamodelle finden auch Anwendung zur *Einordnung* und zum *Vergleich von Modellierungswerkzeugen*. Referenz-Metamodelle werden auch zur *Methoden- und Technik-unabhängigen Schulung* von Konzepten zur Modellierung verwendet. Ebenso wird durch solche Referenzmodelle eine einheitliche und ebenfalls von konkreten Methoden und Techniken unabhängigen *Terminologie* festgelegt.

Anwendungsbereiche von Metamodellen

als spezielles Modell	**als Referenzmodell**
• Beschreibung und Untersuchung von Modellierungsmethoden • Schulung von Modellierungsmethoden • Festlegung der Terminologie der Modellierungsmethoden • Bau von Modellierungswerkzeugen • Ausgangspunkt zur Festlegung der Semantik von Modellierungsmethoden • Überprüfung der (syntaktischen) Korrektheit von Modellen	• Grundlage zur Diskussion und Standardisierung von Modellierungskonzepten • Festlegung einer methodenunabhängigen Terminologie • Methoden- und technikunabhängige Schulung • Grundlage zur Beurteilung von Modellierungsmethoden und -werkzeugen • Maßstab zum Vergleich von Metamodellen zu Modellierungsmethoden und -werkzeugen • Hilfsmittel zur Erstellung von Metamodellen zu Modellierungsmethoden und -werkzeugen

Abbildung 4.2: Anwendungsbereiche von Metamodellen

Die Anwendungsbereiche von Metamodellen sind in Abbildung 4.2 zusammengefaßt. Einen ausführlichen Überblick über Anwendungsbereiche von Metamodellen bieten auch [Brinkkemper, 1990, S. 33f] und [Strahringer, 1996, S. 49ff].

4.3.3 Metamodelle und Referenzmodelle

Zu einem Modell können sowohl Referenzmodelle als auch Metamodelle existieren. In diesem Abschnitt werden die unterschiedlichen Zusammenhänge zwischen Modellen, Referenzmodellen und Metamodellen anhand ihrer Einordnung in *Abstraktionsebenen* zusammenfassend dargelegt.

Bei der Betrachtung von Sprachen werden Objektsprachen und Metasprachen unterschieden (vgl. [Bußmann, 1990]). Während *Objektsprachen* solche Sprachen bezeichnen, die Betrachtungsgegenstand einer Untersuchung sind, bezeichnen *Metasprachen* oder *Syntaxsprachen* diejenigen Sprachen, in denen die Untersuchung erfolgt (vgl. auch [Carnap, 1968],[Mittelstraß, 1984]). Mit *Meta-Metasprachen* bezeichnet man folglich Sprachen zur Beschreibung der Metasprache einer Objektsprache. Diese Unterscheidung kann auf Modellierungssprachen und -methoden übertragen werden. Nach dem Abstraktionsgrad eines Modells von einer modellierten Realität können analog die *Modellebene*, die *Meta-Modellebene* und die *Meta-Meta-Modellebene* unterschieden werden.

Die erste Ebene faßt die Modelle der Diskurswelt in der *Modellebene* zusammen. Metamodelle sind in der zweiten Ebene (*Metamodellebene*) zusammengefaßt. Durch Metamodelle wird die Methode beschreiben, die der Modellierung zu Grunde liegt. Modelle sind daher als Instanzen zu Metamodellen aufzufassen. In ähnlicher Form sind Metamodelle Instanzen von Meta-Metamodellen, die die dritte Ebene (*Meta-Metamodellebene*) bilden. Zwischen Modell, Metamodell und Meta-Metamodell existiert folglich eine hierarchische Instanz-Schema-Beziehung.

Die Anzahl dieser Ebenen ist je nach Modellierungsansatz verschieden. Die Betrachtungen von [Gigch, 1991] und [Verhoef et al., 1991] basieren auf drei Abstraktionsebenen. [Hars, 1994, S. 12] betrachtet die Modell- und die Metamodell-Ebene, wobei jedoch die Modellebene noch weiter instanziert wird. Die Modellierungsansätze von CDIF [Flatscher, 1998, S. 32] und der Unified Modeling Language [OMG, 1999, S. 2-4] verfolgen einen vier-Ebenen Ansatz, bei denen Instanz-Ebene, Modellebene, Metamodellebene und Meta-Metamodellebene unterschieden werden. Der Gliederungsansatz kann auch auf beliebig viele Ebenen ausgedehnt werden (vgl. z. B. [Auramäki et al., 1987], [Mylopoulos et al., 1990]). In diesen Arbeiten werden in höheren Ebenen generell Meta-Modelle der niedrigeren Ebene betrachtet. Dieses führt zu einer unbegrenzten Hierarchie jeweils abstrakterer Modelle. Die hier vorliegende Arbeit basiert auf einer Strukturierung entlang der Modell-, der Metamodell- und der Meta-Metamodellebene. Die Ebene der Instanzmodelle bleibt hier unberücksichtigt.

Die Modell-, die Metamodell- und die Meta-Metamodellebene enthalten aus Sicht der jeweils niedrigeren Ebene Modelle. Innerhalb einer Ebene können Modelle als *Referenzmodelle* ausgezeichnet werden. Während die Instanz-Schema-Beziehung Modelle benachbarter Ebenen verbindet, bezieht sich Referenz-Beziehung auf Modelle einer Ebene. Beispielhaft sind diese Zusammenhänge zwischen Modellen, Metamodellen, Meta-Metamodellen und Referenzmodellen in Abbildung 4.3 zusammengefaßt. Instanz-Schema-Beziehungen sind hierbei in der Vertikalen und Referenz-Beziehungen in der Horizontalen notiert.

Innerhalb einer Ebene können durchaus mehrere Referenzmodelle existieren. Ein aus einem Referenzmodell abgeleitetes Modell kann wieder Referenz für die Ableitung oder den Vergleich anderer Modelle sein (vgl. die Modelle M_4, M_2 und M_1 in Abbildung 4.3). Ein Referenzmodell für Krankenhaus-Informationssysteme kann beispielsweise einerseits aus einem allgemeinen Re-

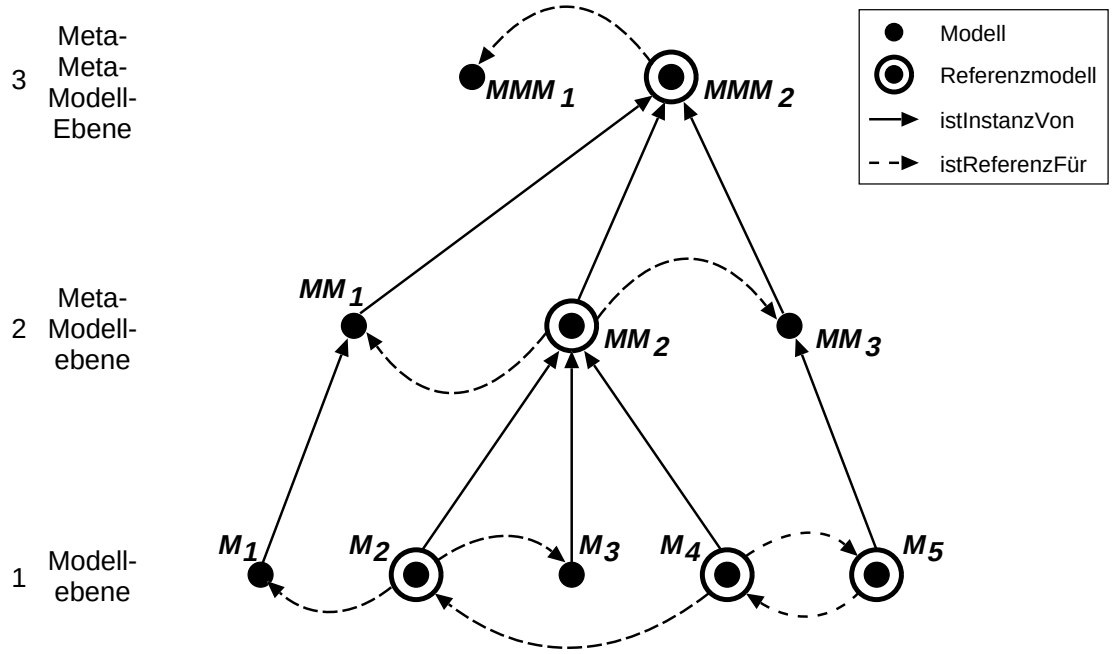


Abbildung 4.3: Metamodell und Referenzmodelle

ferenzmodell für Informationssysteme abgeleitet worden sein und andererseits wieder Referenz für eine spezielle Modellierung eines Krankenhaus-Informationssystemes sein.

Referenzbeziehungen können auch zyklisch sein. So kann ein Modell (M_4) beispielsweise als Referenz zur Erstellung eines Modells (M_5) herangezogen werden, das anschließend wieder als Referenz zur Bewertung des ersten Modells verwendet wird. Beispiele hierfür sind die SOCRATES-Modelle der „Supermethoden“ aus [Brinkkemper et al., 1989] und [Goor et al., 1992] und die hiermit verglichenen Methoden.

Es ist auch möglich, daß Metamodelle zu einem Referenzmodell und einem hierzu speziellen Modell unterschiedlich sind (vgl. die Modell-Metamodell-Paare M_1, MM_1 und M_2, MM_2). So könnte eine Instanz des UML-Metaschemas durchaus auch Ausgangspunkt einer *EER/GRAL*-Modellierung sein. Der Vergleich verschiedener Modellierungen ist jedoch einfacher, wenn die Modelle Instanzen desselben Metamodells sind.

In der hier vorgestellten Einordnung von Modellen, Referenzmodellen und Metamodellen wird deutlich zwischen Referenz- und Metamodell unterschieden. [Lehner, 1995, S. 126] macht diese Unterscheidung nicht. Metamodellen wird hier im Bezug zu Modellinstanzen auch Referenzcharakter zugesprochen in dem Sinn, daß „*Metamodelle [...] Referenzmodelle für alle daraus abgeleiteten Modelle darstellen*“. Dieser Auffassung wird hier nicht gefolgt, da hierdurch die klare Trennung zwischen dem Beibehalten der Abstraktionsebene bei der Referenzmodellierung und dem Wechsel der Abstraktionsebene bei der Metamodellierung verwischt würde. Ein vergleichende Diskussion von Referenz- und Metamodellen nimmt auch [Hars, 1994, S. 15ff] vor. Hier wird ebenfalls deutlich zwischen Meta- und Referenzmodellen unterschieden.

4.4 Einordnung in die Zielsetzung dieser Arbeit

Das in Kapitel 6 vorgestellte Referenz-Metaschema der visuellen Modellierungssprachen für Organisationen und Softwaresysteme kann auch entlang der in den Kapiteln 4.1–4.3 beschriebenen Begriffsbestimmungen und Anforderungen als Modell, als Referenzmodell und als Metamodell eingeordnet werden.

Modelle

Das Referenz-Metaschema der Beschreibungsmittel ist als *statisches Modell* einzustufen, da es die Konzepte und deren Beziehungen von Beschreibungssprachen unabhängig von zeitlichen und methodischen Aspekten notiert. Zur Modellbeschreibung wird der mathematisch fundierte *EER/GRAL*-Ansatz (vgl. Kapitel 5.2) verwendet, so daß das resultierende Modell zusätzlich *symbolisch* und *formal* ist. Nach dem Verwendungszweck ist das Referenz-Metaschema als *Deskriptionsmodell* zu klassifizieren, da durch die mit *EER/GRAL* vorgegebene Modellierungstechnik Konzepte und deren Beziehungen in den Vordergrund gestellt werden. Solche Konzeptmodelle unterstützen in besonderem Maß die Kommunikation über die modellierten Zusammenhänge. Darüber hinaus kann aus einem *EER/GRAL*-Modell auch nahtlos eine Implementation abgeleitet werden. Das Referenz-Metaschema ist auch als *didaktisches Modell* einsetzbar, da es durch die in der Modellierung explizit herausgearbeiteten Konzepte und deren Beziehungen eine einheitliche Vermittlung und Schulung der Beschreibungsmittel auf einem, von der konkreten Notationsform unabhängigen Niveau, unterstützt.

Referenzmodelle

Dem Referenz-Metaschema der visuellen Modellierungssprachen der Organisations- und Softwaretechnik kann auch Referenzcharakter zugesprochen werden. Dieses Modell wird mit dem Ziel erstellt, zum einen die *Begriffe*, die in den betrachteten Sprachen verwendet werden, einheitlich festzulegen und sie in ihrem Zusammenwirken zu beschreiben. Aufgrund der Modellierung entlang des *EER/GRAL*-Ansatzes (vgl. Kapitel 5.2), die insbesondere die Konzepte der Beschreibungsmittel in den Vordergrund stellt, eignet sich dieses Modell zur Methoden-unabhängigen *Ausbildung* der Beschreibungsmittel auf konzeptioneller Ebene. Wesentlicher Anwendungsbereich dieses Referenzmodells ist die Verwendung als *Vergleichs- und Modellierungsmittel*. Die Anwendbarkeit des Referenz-Metaschemas wird in Teil III begründet. Hierzu wird es zur Betrachtung der Beschreibungsmittel von Modellierungsmethoden für Organisationen (Kapitel 7) und Softwaresystemen (Kapitel 8) und zur Entwicklung eines Werkzeugs zur Organisationsmodellierung (Kapitel 9) eingesetzt.

Metamodelle

Modellierungsziel des Referenzmodells ist eine konzeptionelle, integrierte Beschreibung unterschiedlicher, strukturierter Modellierungstechniken, die zur Modellierung organisatorischer und softwaretechnischer Zusammenhänge verwendet werden. Vorgehensaspekte der Modellierungstechniken werden nicht betrachtet. Für die Beschreibungsmittel aus Kapitel 3 wird in Teil II ein

Metaschema entwickelt. Bezogen auf die drei Abstraktionsebenen aus Kapitel 4.3 befindet sich dieses Modell auf der Metamodell-Ebene.

Da dieses Metaschema auch Referenz sowohl für die Einordnung und Beschreibung von Modellierungsmethoden als auch für die Ableitung von Werkzeugen dient, ist es zusammenfassend als *Referenz-Metaschema* einzustufen.

Teil II

Modellbildung des Referenz-Metaschemas

Das Referenz-Metaschema der visuellen Modellierungssprachen für Organisationen und Softwaresysteme wird in den folgenden Kapiteln entlang des in Kapitel 3 entwickelten Klassifikationsschemas hergeleitet.

Die Erstellung des Referenz-Metaschemas erfolgt mit den Mitteln der graphbasierten Konzeptmodellierung. Hierzu wird zunächst in Kapitel 5 in die Konzeptmodellierung eingeführt. Ausgehend von seiner formalen, graphbasierten Basis wird der zur Modellierung verwendete *EER/GRAL*-Ansatz zur Konzeptmodellierung vorgestellt und in die Methoden und Techniken der Konzeptmodellierung eingeordnet.

Die Darstellung des Referenz-Metaschemas erfolgt in Kapitel 6. Zu jedem der in Kapitel 3 eingeführten Beschreibungsparadigmen werden in den Kapiteln 6.2.1 bis 6.5.3 *EER/GRAL*-Metaschemata vorgestellt. Durch Anwendung dieser Metaschemata zur Erstellung der speziellen Metaschemata wesentlicher Vertreter der einzelnen Sprachparadigmen, wird deren Referenz-Eigenschaft begründet. Die Integration dieser paradigmenbezogenen Referenz-Metaschemata zum integrierten Referenz-Metaschema der visuellen Modellierunssprachen für Organisationen und Softwaresysteme erfolgt in Kapitel 6.6.

Die Referenz-Eigenschaft dieses integrierten Referenz-Metaschemas wird in Teil III anhand der Beschreibungsmittel verschiedener Ansätze zur Organisations- und Softwaremodellierung nachgewiesen.

5 Graphbasierte Konzeptmodellierung

Die Erstellung des Referenz-Metaschemas folgt dem *EER/GRAL*-Ansatz zur graphbasierten Konzeptmodellierung. Nach einer kurzen Einführung in die konzeptionelle Modellierung in Kapitel 5.1 folgt in Kapitel 5.2 die Darstellung des in dieser Arbeit verwendeten Modellierungsansatzes einschließlich seiner formalen Basis.

5.1 Konzeptmodellierung

Unter Konzeptmodellierung versteht man eine Modellierung, bei der *Gegenstände* der zu modellierenden Realität durch Konzept- oder Begriffsgeflechte beschrieben werden. Ein *Konzept* faßt hierbei gedankliche Vorstellungen über die Eigenschaften und Merkmale des zu modellierenden Gegenstandes zusammen [DIN 2330, 1993]. Konzepte werden in Allgemeinbegriffe und Individualbegriffe unterschieden. Während Individualbegriffe konkrete, eindeutige Dinge der zu modellierenden Realität beschreiben, werden durch Allgemeinbegriffe Dinge mit gleichen Eigenschaften zusammengefaßt. Die Konzepte selbst werden durch *Symbole* oder Begriffsworte, deren Bildung und Verwendung willkürlich vereinbart sein kann, notiert. Die Zusammenhänge zwischen Gegenstand, Konzept und Symbol sind in Abbildung 5.1 in einem Konzeptmodell skizziert (vgl. hierzu auch das Bedeutungsdreieck in [Ogden / Richards, 1923, S. 14]).

Abbildung 5.1: Zusammenhang zwischen den Begriffen „Gegenstand“, „Konzept“ und „Symbol“

Wesentliches Merkmal der Konzeptmodellierung ist die Abbildung der Realität durch Konzepte, die ein möglichst allgemein akzeptiertes Verstehen der Dinge der Realität widerspiegeln. Durch die hierbei verwendeten Konzepte werden jeweils Eigenschaften gleichartiger Dinge zusammengefaßt. Hierbei werden sowohl Konzepte zur Beschreibung von Begriffen als auch Konzepte zur Beschreibung von Beziehungen (vgl. [Frege, 1891, S. 37]) verwendet. Die Konzeptualisierungen der Begriffe auf der Ebene der Allgemeinbegriffe werden *Objekttypen* oder *Objektklassen* und die Konzeptualisierungen der Beziehungen *Beziehungstypen* oder *Beziehungsklassen* genannt.

Diese Auffassung der Modellierung der realen Welt durch Entitäten oder Objekte und Beziehungen folgte auch [Chen, 1976] in seinem Beitrag zur Einführung der Objekt-Beziehungsmodellie-

rung: „*The entity-relationship model adopts the more natural view, that the real world consists of entities and relationships*“. Auch die Wissensrepäsentation folgt dieser Auffassung. In Schemata zur Beschreibung von Wissen wird der abzubildende Gegenstandsbereich als Sammlung von Individuen und hierzwischen vorliegenden Beziehungen aufgefaßt [Mylopoulos / Levesque, 1996]. Die Konzeptmodellierung wird von [Berztiss / Matjasko, 1995] ebenfalls als die Beschreibung einer Diskurswelt durch Objekte, deren Attribute und durch Beziehungen zwischen den Objekten, charakterisiert. Ähnlich definiert [Frank, 1994, S. 81] Konzeptmodelle als „*Abstraktion realweltlicher Gegenstände oder Sachverhalte, und [den] zwischen ihnen existierenden Beziehungen*“. Eine Einschränkung der Konzeptmodellierung auf Individual- oder auf Allgemeinbegriffe wird durch keine dieser Definitionen gefordert.

> ***Konzeptmodellierung*** ist die Abstraktion der Realität durch mentale Repräsentationen (Konzepte) der realen oder abstrakten Dinge der zu modellierenden Diskurswelt und den hierzwischen bestehenden Beziehungen verstanden. Diese Konzepte werden in einer symbolischen Form notiert.

Konzeptmodellierung, wie sie heute verwendet wird, geht auf Überlegungen der künstlichen Intelligenz- und der Softwaretechnikforschung in den Bereichen der Datenbanken und der Programmiersprachen zurück. Die Einflüsse aus Sicht der Datenbanken beziehen sich in erster Linie auf die Modellierung statischer Aspekte durch semantische Modelle. Der Beitrag der Programmiersprachen bezieht sich auf die Modellierung von Verhaltensaspekten und Datenstrukturen. Höhere Modellierungskonstrukte wie die Aggregation („partOf-Beziehungen“), die Gruppierung („memberOf-Beziehungen“), die Generalisierung bzw. die Spezialisierung („isA-Beziehungen“) und die Instanziierung werden als Beitrag der Überlegungen in der Wissensrepräsentation der Künstlichen Intelligenz betrachtet (vgl. [Rolland / Cauvet, 1992]). Diese Wurzeln werden auch in [Brodie et al., 1986], [Loucopoulos / Zicari, 1992] und [Kangassalo et al., 1995] einschließlich ausführlicher Anwendungsbeispiele beschrieben. Eine Abgrenzung von Konzeptmodellierung gegenüber der Wissensrepräsentation bzw. der semantischen Datenmodellierung skizziert [Mylopoulos, 1992]. Während in der Wissensrepräsentation die Modellierung von Wissensbasen eher von dem Hintergrund der automatischen Ableitung von Wissen geprägt ist, verfolgt die Konzeptmodellierung eher das Ziel, eine adäquate Form zur Beschreibung der Realität zur Schaffung eines gemeinsamen Verständnisses unter menschlichen Benutzern bereitzustellen. Die semantische Datenmodellierung faßt John Mylopoulos daher als Ergänzung der Konzeptmodellierung zur Umsetzung in einem Computersystem auf. Gegenüber der (reinen) Konzeptmodellierung sind bei der semantischen Datenmodellierung weitere, technisch bedingte Rahmenbedingungen zu beachten.

Anwendung finden Konzeptmodelle in erster Linie zur Beschreibung des Wissens über einen zu modellierenden Sachverhalt z. B. im Rahmen der Entwicklung von Informationssystemen (vgl. z. B. [Loucopoulos, 1992]). Ziel ist hierbei die formale Beschreibung sowohl der technischen wie auch sozialen Zusammenhänge des Informationssystems als Grundlage zur Kommunikation zwischen den Beteiligten und zur Schaffung eines gemeinsamen Verstehens über das betrachtete System [Mylopoulos / Levesque, 1996], [Mylopoulos, 1992], [Frank / Prasse, 1997a]. Bei der Entwicklung von Datenbanksystemen stellt die Erstellung von Konzeptmodellen die erste Phase des Datenbank-Entwurfs (vgl. z. B. [Batini et al., 1992]) dar. Die hier entwickelten konzeptionellen Schemata beschreiben Datenbankinhalte unabhängig von eingesetz-

ten Datenbank-Managementsystemen oder Programmiersprachen. Generell gilt für Konzeptmodellierungen, daß von Randbedingungen, die z. B. durch eine mögliche informationstechnische Umsetzung oder eine konkrete Darstellungsform gegeben sind, abstrahiert wird.

In dieser Arbeit wird die Konzeptmodellierung zur Erstellung eines Metadatenmodells der in Organisations- und Softwaretechnik eingesetzten visuellen Modellierungssprachen verwendet. Im Sinn der Konzeptmodellierung stellen sowohl die hier verwendeten Sprachmittel als auch die zwischen den Sprachmitteln vorliegenden Beziehungen *Konzepte* dar, die in ihrem Zusammenspiel zu beschreiben sind. Von Layout-Aspekten, d. h. von den symbolischen Formen der in den einzelnen Beschreibungsmitteln verwendeten Konzepten wird hierbei abstrahiert.

Ansätze zur Konzeptmodellierung

Zur Beschreibung von Konzeptmodellen auf Ebene der Allgemeinbegriffe bieten sich in erster Linie die visuellen Modellierungssprachen an, die dem Objekt-Beziehungsparadigma (vgl. Kapitel 3.5.3) folgen. So sind auch die im Kapitel 4.3 skizzierten Konzeptmodelle zur Metadatenmodellierung in verschiedenen Dialekten der Objekt-Beziehungsmodellierung dargestellt. Zur Beschreibung des IFIP WG 8.1 Metaschemas wurden einfache Datenstruktur-Diagramme [Olle et al., 1991, S. 61ff] eingesetzt, die Mittel zur Beschreibung der Objekttypen und binärer Beziehungstypen einschließlich deren Kardinalitäten bereitstellen. Die Metadatenmodelle des CC-RIM-Referenzmodells werden durch ASDM-Diagramme [Heym / Österle, 1993], [Gutzwiller, 1994, S. 24ff], [Heym, 1995, S. 15ff] notiert, die Konzepte verschiedener Typen u. a. zur Beschreibung von Spezialisierungen und Aggregationen und ebenfalls binäre Beziehungen mit Kardinalitäten enthalten. Attribute, die weiter strukturiert sein können, werden verbal beschrieben. Das ARIS-Informationsstrukturmodell ist in einer Erweiterung der von [Chen, 1976] vorgeschlagenen Notation dargestellt, die Ausdrucksmittel für Objekttypen, Beziehungstypen beliebiger Arität und Attribute bereitstellt. Die Erweiterungen in [Scheer, 1992, S. 51ff] beziehen sich auf Ergänzungen um die Generalisierung von Konzepten und um die Uminterpretation von Beziehungstypen in Objekttypen. Die Beschreibung des Metadatenmodells zur MEMO [Frank, 1994] erfolgt in erster Linie verbal. In graphischen Darstellungen werden darüber hinaus die Generalisierungs- bzw. Spezialisierungsbeziehungen zwischen den modellierten Konzepten sowie ausgewählte Beziehungen, die hier ebenfalls binär aufgefaßt werden, herausgestellt. Datenmodelle zu SOCRATES-Metamodellen [Verhoef et al., 1991] werden durch NIAM-Informationsstruktur-Diagramme [Verheijen / van Bekkum, 1982], [Wintraeken, 1985] notiert. Objekttypen, die weiter spezialisiert sein können, werden durch ebenfalls binäre Beziehungstypen verbunden. In Gegensatz zu den bisher genannten Modellierungsmitteln werden in NIAM noch die Rollen der Konzepte in einer Beziehung herausgestellt. Die Darstellung der CDIF-Metamodelle [Flatscher, 1998] erfolgt durch erweiterte, nicht attributierte Objekt-Beziehungsdigramme, die um umfangreiche Listen zur Attributierung der Objekt- und Beziehungsklassen ergänzt werden.

Weitere Ansätze zur Konzeptmodellierung nach dem Objekt-Beziehungsparadigma bieten auch die Notationen zur Beschreibung von Klassendiagrammen (vgl. z. B. [Rumbaugh et al., 1991, S. 25ff], [Booch, 1994, S. 158ff] oder [Rumbaugh et al., 1999, S. 41ff]) der objektorientierten Modellierungsmethoden. Gegenüber den bisher skizzierten Ansätzen zur Konzeptmodellierung stellen diese Beschreibungsmittel auch Konzepte zur Modellierung des Verhaltens der Klassen durch Methodendeklarationen bereit, die jedoch z. B. zur Modellierung des UML-

Metadatenmodells [OMG, 1999, S. 2-1ff] nicht verwendet wurden. Übersichten zu Ansätzen der Konzeptmodellierung auf Basis von Objekt-Beziehungsmodellierungen geben auch [Brinkkemper, 1990, S.36ff] [Hars, 1994, S. 110] und [Tolvanen, 1998, S. 81ff].

Nicht alle Ansätze zur Konzeptmodellierung beziehen sich ausschließlich auf die Ebene der Allgemeinbegriffe und verwenden Beschreibungsmittel des Objekt-Beziehungsparadigmas. So findet der in [Mylopoulos et al., 1990] eingeführte Konzeptmodellierungs-Ansatz *Telos* (τέλος) seine Wurzeln in der Wissensrepräsentation durch semantische Netze. Elementares Modellierungsmittel sind Propositionen. Diese beschreiben sowohl Objekte, Objektklassen als auch Attribute. Attribute sind hierbei als binäre, benannte Zuordnungen zwischen Propositionen aufzufassen. Telos-Konzeptmodelle werden durch Aggregationen, Generalisierungen und Instanziierungen (Klassifikationen) strukturiert. Im Gegensatz zur Objekt-Beziehungsmodellierung, bei der Klassen- und Objektmodellierungen klar getrennt sind, enthalten Wissensbasen sowohl Konzepte zur Beschreibung der Klassen als auch zur Beschreibung deren Instanzen. Die Konzeptmodellierung bezieht sich hier sowohl auf die Ebene der Individualbegriffe wie auch auf die Ebene der Allgemeinbegriffe. Neben einer textuellen Notation zur Beschreibung dieser strukturellen Eigenschaften können bei der Modellierung mit Telos in einer Zusicherungssprache Integritätsbedingungen und Ableitungsregeln in Prädikatenlogik erster Stufe formuliert werden. Mit ConceptBase [Jarke et al., 1995] liegt eine Implementation von Telos auf Basis objektorientierter Datenbanken vor. Diese Implementation erlaubt eine Partitionierung der in der Objektbasis enthalten Objekte in ein „Token layer“ zur Beschreibung der Instanzen und in ein „simple class layer“ zur Beschreibung der Schemainformation. Sie läßt darüber hinaus aber beliebig viele weitere „Meta-Ebenen“ zu.

Zur Konzeptmodellierung kann auch ausschließlich *Mengenlehre und Prädikatenlogik erster Stufe* verwendet werden. Ein Beispiel hierfür ist die Beschreibung von Petrinetzen in [Brinkkemper, 1990, S.36ff]. Die Petri-Netz-Konzepte „Stellen“, „Transitionen“ und „Flußkanten“ werden hierbei durch Mengen modelliert. Beziehungen zwischen den Konzepten werden durch Prädikate über diesen Mengen zur Beschreibung der Adjazenzbeziehungen zwischen Stellen und Transitionen über Flußkanten notiert. Zusätzliche Einschränkungen an die so modellierte Struktur werden in prädikatenlogischen Termen angegeben.

Die ebenfalls auf Mengenlehre und Prädikatenlogik aufbauende Spezifikationssprache Z [Spivey, 1992] wird in [Mišić / Moser, 1997] zu Konzeptmodellierung objektorientierter Systeme verwendet. Basistypen modellieren hier die grundlegenden Entitäten wie „Object“ und „Interface“. Beziehungen zwischen diesen werden durch Relationen beschrieben, die mit zusätzlichen Einschränkungen in Z-Schemata zusammengefaßt sind. In ähnlicher Form wird die Spezifikationssprache *GRAL* [Capellmann / Franzke, 1991], die als Erweiterung von Z um graphentheoretische Konstrukte aufgefaßt werden kann, in [Winter, 1992] zur Konzeptmodellierung für Beschreibungsmittel des Objekt-Beziehungsparadigmas verwendet. Sowohl die Entitäten wie auch die Beziehungen und die Attributstrukturen der Objekt-Beziehungsmodellierung werden hierzu durch Z-Schemata abgebildet. Diese Schemata gehen als Komponenten in ein Schema zur Spezifikation der Klasse der Objekt-Beziehungsdiagramme ein. Zusätzliche Einschränkungen an das Zusammenspiel der Komponentenschemata werden durch entsprechende Z-Prädikate formalisiert. Von diesem (Referenz-) Modell wurden spezielle Modelle zur Beschreibung der Entity-Relationship-Diagramme nach [Elmasri / Navathe, 2000], der generischen Semantischen Modelle nach [Hull / King, 1987] und der NIAM-Informationsstruktur-Diagramme nach [Verheijen / van Bekkum, 1982] abgeleitet. Die objektorientierte Z-Erweiterung Object$Z$ wird in [Saeki,

1995] zur Konzeptmodellierung von Entity-Relationship-Diagrammen, Klassendiagrammen und Statecharts verwendet. Wie in [Winter, 1992] und [Mišić / Moser, 1997] werden auch hier die einzelnen Konzeptmodelle durch Z-Schemata modelliert, die die entsprechenden Konzepte als Komponenten enthalten. Neben diesen statischen Modellaspekten wird auch das Verhalten der Modelle durch entsprechende Methoden-Schemata beschrieben, die mit den statischen Schemata zusammengefaßt sind. Die Integration der einzelnen Object$Z$-Konzeptmodelle erfolgt durch zusätzliche Schemata, die die hierzu nötigen Prädikate enthalten.

Grundsätzlich können zur Konzeptmodellierung sowohl die visuellen, semiformalen Beschreibungsmittel des Objekt-Beziehungsparagdimas als auch die formalen, auf Prädikatenlogik basierenden, Ansätze verwendet werden. Die rein visuellen Beschreibungsmittel reichen jedoch häufig nicht aus, um ausreichend detailierte Konzeptmodelle zu erstellen. Außer zur Darstellung von Kardinalitätsbeziehungen und von wenigen semantischen Bedingungen fehlen Modellierungskonstrukte zur Beschreibung zusätzlicher Einschränkungen. Insbesondere fehlen auch Mittel zur Beschreibung zusätzlicher Bedingungen, die für die Integration von Teilmodellen zu einem größeren Konzeptmodell benötigt werden. Formale Beschreibungsformen bieten dagegen die gewünschte Ausdruckskraft, sind aber wenig anschaulich. Zur Konzeptmodellierung bietet sich daher eine Kombination aus Objekt-Beziehungsdiagrammen zur Beschreibung einfacher struktureller Zusammenhänge und einer formalen Notation zur Beschreibung komplexer Einschränkungen an.

Zur Darstellung solcher einfacher strukturellen Zusammenhänge verwenden daher auch [Saeki, 1995] und [Mišić / Moser, 1997] neben den Z-Modellen Objekt-Beziehungsdiagramme. Ebenso schlägt [Brinkkemper, 1990, S.38ff] zur Konzeptmodellierung eine Kombination von NIAM-Informationsstruktur-Diagrammen und prädikatenlogischen Formeln vor. Zur Beschreibung des UML-Metamodells wurden ebenfalls sowohl UML-Klassendiagramme [Booch et al., 1999, S. 105ff] als auch zusätzliche formale Integritätsbedingungen in der Object Constraint Language (OCL) [OMG, 1999, S. 6.1ff] verwendet.

Wesentlich für das Zusammenwirken von formalen und semiformalen, visuellen Beschreibungsmitteln ist, daß beide Beschreibungsformen auf einer einheitlichen formalen Basis aufbauen. Insbesondere sollte die Semantik der semiformalen Beschreibungsmittel in einer mit dem verwendeten formalen Beschreibungsansatz kompatiblen Form definiert sein.

In dieser Arbeit wird zur Konzeptmodellierung der *EER/GRAL*-Ansatz [Ebert et al., 1996b] verwendet, der auch zur Beschreibung der *KOGGE*- und *EER/GRAL*-Metaschemata (vgl. Kapitel 4.3) eingesetzt wurde. Einfache strukturelle Aspekte werden hierbei durch erweiterte Objekt-Beziehungsdiagramme (extended Entity-Relationship-Diagramms; *EER*) beschrieben. Zur Beschreibung zusätzlicher Einschränkungen wird die Spezifikationssprache *GRAL* (Graph Specification Language) [Franzke, 1997] eingesetzt. Während *GRAL* als Erweiterung der prädikatenlogischen Sprache Z um Konstrukte zur einfachen Beschreibung von Graphen [Ebert / Franzke, 1995] zu betrachten ist, unterliegt dem verwendeten *EER*-Dialekt eine graphbasierte Semantik. Die Semantik des *EER*-Dialekts wurde in [Dahm et al., 1998a] in Z beschrieben. Gemeinsam mit der Software-Bibliothek *GraLab* [Dahm / Widmann, 1998] zur Bearbeitung von Graphen liegt mit *EER/GRAL* ein formal basierter und durchgängiger Modellierungs- und Implementationsansatz vor. Dieser wird im folgenden Kapitel eingeführt.

5.2 Konzeptmodellierung mit *EER/GRAL*

Die Grundlage der Konzeptmodellierung mit *EER/GRAL* bilden *Graphen*. Nach einer Einführung in die formale Basis der hier verwendeten *TGraphen* und in die Konzeptmodellierung mit *TGraphen* auf Ebene der Individualbegriffe in Kapitel 5.2.1 erfolgt in Kapitel 5.2.2 die Darstellung des *EER/GRAL*-Ansatzes zur Konzeptmodellierung auf Ebene der Allgemeinbegriffe.

5.2.1 *TGraphen*

Graphen sind ein ausdrucksstarkes Mittel zur Konzeptmodellierung, das in vielen Bereichen zur Beschreibung von Gegenständen und den zwischen ihnen bestehenden Beziehungen genutzt wird. Anwendung finden Graphen bei Modellierungen und hierauf aufsetzenden Analysen z. B. in Physik, in Chemie, in Netzwerktheorie, im Operations Research und in den Sozialwissenschaften. Aufgrund der einfachen bildlichen (graphischen) Darstellung von Zusammenhängen durch Graphen eignen sich diese auch als anschauliches Kommunikationsmittel.

Neben dieser Anschaulichkeit zeichnen sich Graphen auch als Mittel zur Konzeptmodellierung aus, weil sie auf formal eingeführten mathematischen Konstrukten basieren (vgl. z. B. [Harary, 1976] oder [Wilson, 1976]). Insbesondere stellen Graphen das mathematische Modell für beliebige Zusammenhänge dar, die durch binäre Relationen ausdrückbar sind. Darüber hinaus definieren Graphen eine Datenstruktur für die eine Vielzahl erprobter Algorithmen zur Analyse existieren (vgl. z. B. [Berge, 1976], [Even, 1979], [Ebert, 1981] oder [Mehlhorn, 1984]). Graphen erlauben somit sowohl eine *durchgängige Modellierung* als auch eine *softwaretechnische Realisierung* [Engels et al., 1992], [Ebert et al., 1996a].

Bei der Modellierung mit Graphen werden die Gegenstände der zu beschreibenden Realität durch *Knoten* und die (binären) Beziehungen zwischen diesen durch *Kanten* beschrieben. So werden beispielsweise in einem Graphen zur Beschreibung eines Straßennetzes markante Plätze und Kreuzungen durch Knoten und die hierzwischen verlaufenden Straßen durch Kanten beschrieben. Je nach Modellierungsziel werden unterschiedliche Graphtypen verwendet. Interessiert nur, ob es zwischen zwei Plätzen eine Straßenverbindung gibt, reicht zur Modellierung in der Regel ein *ungerichteter Graph* aus, in dem durch die Kanten lediglich die Verbindung angezeigt wird. Ist aber zusätzlich zu berücksichtigen, in welcher Fahrtrichtung Straßen befahren werden können, muß die Fahrtrichtung in den Kanten repräsentiert werden. In diesem Fall spricht man von *gerichteten Graphen*. Sollen in dem Straßennetz beispielsweise Plätze von besonderem touristischen Wert von einfachen Straßenkreuzungen unterschieden werden, kann dieses durch Typisierung der Knoten erfolgen. Ebenso können durch Kantentypen z. B. Autostraßen, Fahrradwege und Fußgängerwege unterschieden werden. Graphen, bei denen Knoten und Kanten zu verschiedenen Typen zusammengefaßt sind, werden *typisierte Graphen* genannt. Sollen weiter z. B. die Namen von Plätzen oder Straßen abgebildet werden, werden Knoten und Kanten mit diesen zusätzlichen Informationen versehen. Solche Graphen, bei denen Knoten und Kanten weitere Attribute tragen können, heißen *attributierte Graphen*.

Konzeptmodellierung mit Graphen

Zur Konzeptmodellierung mit Graphen wird ein möglichst genereller Ansatz benötigt, der die verschiedenen Graphtypen geeignet zusammenfaßt. Solche generellen Ansätze werden beispielsweise bei der Konzeptmodellierung durch konzeptuelle Graphen, durch PROGRES-Graphen oder durch *TGraphen* eingeführt.

Konzeptuelle Graphen (conceptual graphs) [Sowa, 1984, S 69ff], [Sowa, 1992], [Sowa/Zachman, 1992] werden als anschauliche und leicht lesbare Notation für getypte Prädikatenlogik erster Stufe eingeführt und gehen auf Überlegungen zur Wissensrepräsentation zurück. Die Diskurswelt wird auch hier durch Objekte (concept) und Beziehungen (conceptual relation) modelliert. Sowohl die Objekte als auch die Beziehungen werden in konzeptuellen Graphen durch Knoten repräsentiert. Konzeptuelle Graphen werden daher als bipartite Graphen definiert, d. h. als Graphen mit zwei unterschiedlichen Arten von Knoten (für Objekte und Beziehungen), bei denen ausschließlich Knoten verschiedener Art miteinander verbunden sind.

Konzepte dienen zur Repräsentation von Objekten und Attributen. Sie können durch eine Typangabe und eine Konkretisierung (referent) näher charakterisiert werden. Durch diese Konkretisierung können Attributwerte gesetzt oder Objektbezeichner definiert werden. Attribute werden in konzeptionellen Graphen durch mit Literalen konkretisierte Objektknoten modelliert und können ausschließlich Objekten zugeordnet werden. Ein eigenständiges Attributierungskonzept liegt nicht vor. Es ist ferner möglich, Objekte zu quantifizieren und Aussagen über Objektmengen zu formulieren. Dieses erfolgt jeweils durch zusätzliche Annotation der Konzeptknoten.

Beziehungsknoten beschreiben die Zusammenhänge zwischen Objektknoten. Zur Modellierung von Reihenfolgen können diese Inzidenzen angeordnet werden. Ein Beziehungsknoten muß zu mindestens einem Objektknoten adjazent sein. Wie auch die Objektknoten können die Beziehungsknoten typisiert werden. Das hierbei verwendete hierarchische Typkonzept erlaubt auch Mehrfachvererbungen. Darüber hinaus wird gefordert, daß konzeptuelle Graphen nur endlich viele Objekt- und Beziehungs-Knoten besitzen, die alle untereinander verbunden sind. Konzeptuelle Graphen können daher als *gerichtete, bipartite, zusammenhängende, endliche, angeordnete und typisierte Graphen* aufgefaßt werden.

Die bei [Sowa, 1984] verwendeten Graphen können auch als Hypergraphen (vgl. z. B. [Berge, 1976, S. 389ff]) aufgefaßt werden, bei denen die Konzeptbeziehungen als n-äre Kanten ($n \geq 1$) betrachtet werden. Wird die Arität der Beziehungskonzepte auf binäre Beziehungen eingeschränkt, können die Beziehungen auch direkt durch Kanten modelliert werden. Ein spezieller Knotentyp oder eine hypergraphenartige Betrachtung der Kanten zur Repräsentation von Beziehungen ist dann nicht nötig. Durch die Einschränkung auf ausschließlich binäre Beziehungen wird die Ausdrucksmächtigkeit der Graphen nicht eingeschränkt, da n-äre Beziehungen ($n > 2$) durch Modellierung der Beziehung als zusätzlichen Knoten und n binären Beziehungen zwischen diesem und den hiermit in Beziehung gesetzten Knoten modelliert werden kann.

In *PROGRES* (PROgrammed Graph REwriting System) [Schürr, 1991a], [Schürr, 1991b], [Zündorf, 1996] werden zur Konzeptmodellierung *endliche, typisierte, attributierte und gerichtete Graphen* verwendet. Die Typisierung bezieht sich sowohl auf Knoten als auch auf Kanten. Die Attributierung, die im Gegensatz zu [Sowa, 1984] als eigenständiges Konzept aufgefaßt wird, wird aber ausschließlich auf Knoten bezogen.

In dieser Arbeit erfolgt die Konzeptmodellierung auf Basis von *TGraphen* [Ebert / Franzke, 1995]. *TGraphen* sind *endliche, gerichtete, typisierte, attributierte und angeordnete* Graphen, die somit eine Obermenge der konzeptuellen Graphen und der PROGRES-Graphen beschreiben. Wie auch in PROGRES-Graphen werden in *TGraphen* Knoten, Kanten und Attribute als eigenständige Konzepte betrachtet. Hier unterscheiden sich die Graphansätze auch von der gängigen Auffassung der *objektorientierten Modellierung*, in der Beziehungen zwischen Objekten eher als Referenzen durch Attribute eines Objekts beschrieben werden. In graphbasierten Ansätzen können dagegen objektartige und beziehungsartige Konzepte unabhängig voneinander betrachtet werden. Ebenso erlaubt die graphbasierte Modellierung auch die Untersuchung einer Beziehung ausgehend von jedem der (beiden) durch eine Kante verbundenen Objekte, unabhängig von der Kantenrichtung.

Formalisierung von *TGraphen*[1]

TGraphen sind formal durch Z-Spezifikationen[2] eingeführt (vgl. [Dahm et al., 1998a]). Grundlegende Elemente (*Element*) zum Aufbau von *TGraphen* sind Knoten (*Vertex*) und Kanten (*Edge*), die durch die folgende freie Typdefinition eingeführt werden. Knoten und Kanten werden hierbei zur eindeutigen Identifizierung durch natürliche Zahlen modelliert.

$$Element ::= vertex\langle\langle \mathbb{N} \rangle\rangle \mid edge\langle\langle \mathbb{N} \rangle\rangle$$
$$Vertex == \operatorname{ran} vertex$$
$$Edge == \operatorname{ran} edge$$

TGraphen sind gerichtet, d. h. eine Kante kann sowohl in einen Knoten eingehen (*in*) als auch aus ihm herausgehen (*out*). Konstanten zur Beschreibung der Kantenrichtungen werden durch *Dir* eingeführt.

$$Dir ::= in \mid out$$

Knoten und Kanten können in *TGraphen* typisiert und attributiert werden. Hierzu werden Typbezeichner (*TypeId*) und Attributbezeichner (*AttrId*) benötigt, die aus einer gegebenen Menge der Bezeichner (*Id*) entnommen werden. Die Zuordnung eines Attributbezeichners zu einem Wert (aus einer gegebenen Wertemenge *Value*) erfolgt durch die Funktion *AttributeInstanceSet*.

$$[Id, Value]$$
$$TypeId == Id$$
$$AttrId == Id$$
$$AttributeInstanceSet == AttrId \twoheadrightarrow Value$$

[1] Die im folgenden dargestellte Formalisierung von *TGraphen* und des hierauf basierenden *EER/GRAL*-Ansatzes zur graphbasierten Modellierung ist das Ergebnis langer und ausführlicher Diskussionen in der Arbeitsgruppe Ebert des Instituts für Softwaretechnik der Universität Koblenz-Landau. An der Entwicklung, Formalisierung und Erprobung wesentlich beteiligt waren Martin Castensen, Peter Dahm, Jürgen Ebert, Angelika Franzke und Manfred Kamp.

[2] Zur formalen Notation der in dieser Arbeit angesprochenen Zusammenhänge wird die Spezifikationssprache Z [King et al., 1988], [Spivey, 1992] verwendet. Einführungen in Z finden sich z. B. in [Diller, 1990] oder in [Wordsworth, 1993].

Diese Grundstrukturen werden im Z-Schema *Graph* zusammengefaßt. *TGraphen* bestehen aus einer Folge von Knoten V und aus einer Folge von Kanten E. Knoten und Kanten werden hier als eigenständige und gleichberechtigte Konzepte betrachtet. Jedem Knoten ist eine evtl. leere, geordnete Liste Λ von Kanten (ohne Doppeleinträge [TGraph 1]) zugeordnet, in der für jede Kante markiert wird, ob sie in den Knoten eingeht oder aus ihm ausgeht. Jede Kante ist zu genau einem Knoten als eingehende und zu genau einem Knoten als ausgehende Kante inzident [TGraph 2]. Zur Typisierung und Attributierung ist jedem Knoten und jeder Kante des betrachteten *TGraphen* ([TGraph 3] und [TGraph 4]) ein Typbezeichner (*type*) und eine (evtl. leere) Menge von Attribut-Werte-Paaren (*value*) zugeordnet.

TGraph

V : iseq *Vertex*
E : iseq *Edge*
Λ : *Vertex* $\twoheadrightarrow$ seq$(Edge \times Dir)$
type : *Element* $\twoheadrightarrow$ *TypeId*
value : *Element* $\twoheadrightarrow$ *AttributeInstanceSet*

[TGraph 1]:
$\Lambda \in \operatorname{ran} V \twoheadrightarrow \operatorname{iseq}(E \times Dir)$

[TGraph 2]:
$\forall e : \operatorname{ran} E \bullet \exists_1 v, w : \operatorname{ran} V \bullet (e, in) \in \operatorname{ran}(\Lambda(v)) \wedge (e, out) \in \operatorname{ran}(\Lambda(w))$

[TGraph 3]:
$\operatorname{dom} type = V \cup E$

[TGraph 4]:
$\operatorname{dom} value = V \cup E$

Die hier skizzierte Definition von *TGraphen* ist sehr allgemein gehalten. Beispielsweise wird hier nicht gefordert, daß Attributstrukturen von Knoten und Kanten nach deren Typzugehörigkeit definiert sind. Auch werden dieselben Typen für Knoten und Kanten zugelassen. Einschränkungen dieser Art werden erst bei einer Konzeptmodellierung auf Ebene der Allgemeinbegriffe interessant, so daß diese in Abschnitt 5.2.2 zur Beschreibung von Klassen von *TGraphen* berücksichtigt werden.

Modellierung mit *TGraphen*

Die Darstellung von *TGraphen* erfolgt durch die visuellen Sprachen des Objekt-Instanzparadigmas. Der im *EER/GRAL*-Ansatz verwendete Darstellungsdialekt wurde in Kapitel 3.5.1 als notationelle Grundform des Objekt-Instanzparadigmas eingeführt.

Generelle Modellierungsprinzipien zur Modellierung mit *TGraphen* werden in [Ebert et al., 1996b] formuliert. Die zu beschreibenden Konzepte sind je nach Modellierungsziel auf durch Knoten repräsentierte Objekte oder auf durch Kanten repräsentierte Beziehungen abzubilden:

- Jedes für den Modellierungskontext relevante Objekt wird durch genau einen Knoten modelliert.
- Jede Beziehung zwischen solchen Objekten wird durch genau eine Kante beschrieben.

- Ähnliche Objekte bzw. Beziehungen werden durch Knoten bzw. Kanten desselben Typs modelliert.
- Weitere charakteristische Informationen zu Objekte und Beziehungen werden in Attributinstanzen zu den jeweiligen Knoten und Kanten notiert.
- Die Festlegung der Anordnung der Beziehungen zueinander erfolgt durch Definition einer Kantenreihenfolge.

Die Entscheidung, ob ein Konzept als Objekt oder als Beziehung aufgefaßt wird und dementsprechend als Knoten oder Kante modelliert wird, ist vom Modellierungsziel und von der individuellen Wahrnehmung des Modellierers abhängig. Gleiches gilt auch für die Einordnung als Konzept oder als Attribut. Als grundsätzliche Richtlinie sollten aber wesentliche und explizit benötigte Konzepte, die auch die Struktur des Modells bestimmen, durch Objekte beschrieben werden. Konzepte, die eher Zuordnungen der Objekte in bestimmten Kontexten beschreiben, sollten als Beziehungen aufgefaßt werden. Durch Attribute sollten solche Wahrnehmungen modelliert werden, denen im aktuellen Modellierungskontext keine eigenständige Identität zugestanden wird.

5.2.2 Graphklassen

Erfolgt eine Modellierung auf Ebene der Allgemeinbegriffe, steht nicht mehr die Modellierung eines einzelnen Objekts oder einer einzelnen Beziehung zwischen zwei Objekten im Vordergrund. Betrachtet werden hier Mengen (oder Klassen) von Objekten und Beziehungen, die aufgrund gleicher Merkmale oder struktureller Eigenschaften gebildet werden. Aus Sicht der graphbasierten Konzeptmodellierung auf Allgemeinbegriff-Ebene sind folglich Schemata zur Beschreibung von Mengen von Graphen mit gleichen Eigenschaften zu untersuchen. Diese Graphmengen werden auch *Graphklassen* genannt.

Konzeptmodellierung mit Graphklassen

Graphklassen können zu einen operational mit Hilfe von Graphgrammatiken oder deklarativ durch direkte Angabe der geforderten Eigenschaften definiert werden.

Die *operationale Definition* einer Graphklasse durch *Graphgrammatiken* (vgl. z. B. [Nagl, 1979], [Göttler, 1988], [Schürr / Westfechtel, 1992], [Rozenberg, 1997]) besteht im wesentlichen aus einem Startgraphen und einer Regelmenge zur Festlegung von Graphproduktionen. Durch Graphproduktionen werden — analog zu den Regeln in Grammatiken textueller Sprachen — Regeln zur Ersetzung eines (nicht-terminalen) Teilgraphen durch einen anderen Teilgraphen sowie die Einbettung dieses Teilgraphen in den Ursprungsgraphen formuliert. Die hierdurch definierte Graphklasse umfaßt alle Graphen, die ausgehend von dem Startgraphen durch Anwendung der Graphproduktionen generiert werden können und keine weiteren, nicht-terminalen Teilgraphen enthalten. Die Grapheigenschaften werden bei der Definition durch Graphgrammatiken indirekt durch einen Generierungsprozeß festgelegt.

Deklarative Ansätze zur Festlegung von Graphklassen beschreiben dagegen die Menge der gültigen Graphen direkt durch Angabe der Grapheigenschaften. Diese Eigenschaften betreffen zum

einen die Graphkomponenten, d. h. die zugelassenen Knoten- und Kantentypen, deren Attributstruktur und die Inzidenzbeziehungen zwischen Knoten- und Kantentypen. Zum anderen sind auch strukturelle Anforderungen an den gesamten Graphen wie z. B. die Forderung nach Azyklizität oder Baumartigkeit zu formulieren.

Die deklarative Beschreibung von Graphklassen erfolgt mit den Beschreibungsmitteln des Objekt-Beziehungsparadigmas ergänzt um zusätzliche Einschränkungen (vgl. z. B. [Elmasri / Wiederhold, 1983] [Carstensen et al., 1994], [Schürr, 1994], [Ebert / Franzke, 1995]). Knoten- und Kantentypen werden hierbei durch Objekt- und Beziehungstypen definiert. Durch die Attributzuordnung der Objekt- und Beziehungstypen wird die Attributstruktur der Graphen und durch die Inzidenzen der Beziehungstypen wird die Inzidenzstruktur der Graphen festgelegt. Zur Strukturierung der Graphklassen-Definitionen können auch die höheren Modellierungskonstrukte der Objekt-Beziehungsmodellierung wie Generalisierung und Aggregation verwendet werden. Die Formulierung zusätzlicher Einschränkungen, die nicht graphisch durch Objekt-Beziehungsdiagramme notiert werden, erfolgt durch passende formale graphische oder textuelle Notationen.

Das *Graph Ersetzungssystem PROGRES* (PROgrammed Graph REwriting System) [Schürr, 1991a], [Zündorf, 1996] kann als graphbasierter Ansatz zur Konzeptmodellierung aufgefaßt werden, der deklarative Beschreibungen von Graphklassen mit Produktionen der Graphgrammatiken verbindet. Deklarative Graphklassen-Spezifikationen werden hier zur konzeptionellen Beschreibung der statischen Struktur eines Systems verwendet. Produktionsregeln beschreiben erlaubte Veränderungen dieser Graphen. Diese Produktionsregeln sind im Gegensatz zu Graphgrammatiken so angelegt, daß die resultierenden Graphen ebenfalls der spezifizierten Graphklassendefinition genügen.

Zur Definition der Graphklassen kann in PROGRES alternativ eine textuelle oder eine graphische Notation [Schürr, 1994] verwendet werden. PROGRES unterstützt hierbei die Definition von Knoten- und Kantentypen. Knotentypen können sowohl in einer Typhierarchie mit Mehrfachvererbung angeordnet sein als auch mit Attributierungsschemata versehen werden. Für Kanten ist lediglich eine flache Typstruktur ohne Attributierung vorgesehen. Ein Verlust an Ausdrucksmächtigkeit entsteht hierdurch nicht, da attributierte Kanten auch durch Knoten mit entsprechenden (nicht-attributierten) Inzidenzen modelliert werden können (vgl. auch [Zinnen, 1995]). Knotenattribute können sowohl Werte der aus Programmiersprachen bekannten Standardtypen als auch Werte bereits definierter Knotentypen annehmen. Solche Knoten-wertigen Attribute stellen jedoch nur eine alternative Kantennotation dar. Darüber hinaus werden in PROGRES intrinsische Attribute, die originär Knoten zugeordnet sind, und abgeleitete Attribute unterschieden, die durch Berechnungsvorschriften aus dem vorliegenden Graphen ermittelt werden. Mit Ausnahme der Generalisierung unterstützt PROGRES keine weiteren höheren Modellierungskonstrukte. Zusätzliche Integritätsbedingungen können in PROGRES ebenfalls sowohl textuell als auch graphisch notiert werden [Münch et al., 1998]. Aufgrund der Ausrichtung von PROGRES als Graphersetzungssystem ist der auch Formalismus zur Beschreibung der Integritätsbedingungen deutlich auf die Verwendung in Graphersetzungsregeln bezogen. Integritätsbedingungen werden durch Graphmuster notiert, die in All- und Existenzaussagen eingebunden werden können.

Die Konzeptmodelle zur Beschreibung des Referenz-Metaschemas der visuellen Modellierungssprachen der Organisations- und Softwaretechnik werden deklarativ durch *EER/GRAL-Graphklassen* beschrieben. Die statische Struktur der hierin enthaltenen *TGraphen* wird mit Hilfe

von erweiterten Objekt-Beziehungsdiagrammen dargestellt. Im Gegensatz zu PROGRES werden in diesem Ansatz zur graphbasierten Konzeptmodellierung auch Kanten als Objekte erster Ordnung verstanden, d. h. Kanten können attributiert und in ein hierarchisches Typsystem eingeordnet werden (vgl. hierzu auch die auf [Frege, 1891] zurückgehende Gleichbehandlung von Begriffen und Beziehungen. Neben der Generalisierung steht dem Modellierer in *EER/GRAL* auch die Aggregation als höheres Modellierungskonstrukt zur Verfügung. Zusätzliche Integritätsbedingungen werden in der Spezifikationssprache *GRAL* (GRAph specification Language) notiert, die Prädikatenlogik erster Stufe mit mehreren, endlichen Sortenmengen unterstützt.

Zur Definition von Graphklassen sind drei Aspekte genauer zu betrachten. Im *Typsystem* wird festgelegt, welche Knoten- und Kantentypen in der Graphklasse erlaubt sind, wie diese attributiert sind, und wie diese in der Typhierarchie angeordnet sind. Das *Inzidenzsystem* bildet die Beziehungen zwischen Knoten- und Kantentypen ab. Es legt fest, welchen Knotentyp die Knoten haben, die über eine Kante des betrachteten Kantentyps zueinander adjazent sind. Im *Invariantensytem* werden zusätzlich Anforderungen an die Graphklasse definiert. Während das Typ- und das Invariantensystem vollständig mit Sprachen des Objekt-Beziehungsparadigmas (vgl. Kapitel 3.5.3) beschrieben werden können, können aus dem Invariantensystem nur Anforderungen zur Kardinalität und zur Injektivität von Beziehungen graphisch notiert werden. Weitere Invarianten sind textuell durch *GRAL* zu notieren.

Formalisierung von Graphklassen

Im folgenden werden zunächst die Schemata zur Definition von Graphklassen formal spezifiziert. Da diese Schemata originär Objekt-Beziehungsdiagramme (extended Entity-Relationship-Diagramme; *EER*) sind, die mit einer graphbasierten Semantik versehen sind, erfolgt die Spezifikation allgemein und in der Terminologie der Objekt-Beziehungsmodellierung. Anschließend wird die graphbasierte Semantik dieser Objekt-Beziehungsdiagramme durch Angabe der *TGraphen*, die zu einem solchen Schema „passen“, denotationell spezifiziert. Hierbei werden Knoten als Extensionen der Objekttypen und Kanten als Extensionen der Beziehungstypen aufgefaßt [Dahm et al., 1998a]. Eine Einführung in graphische Notation zur Formulierung von Graphklassen und in *GRAL* beschließt dieses Kapitel.

Typsystem. Das Typsystem (*TypeSystem*) einer *EER*-Schemadefinition definiert sowohl die Objekt- und Beziehungstypen als auch die Zuordnung von Attributschemata zu diesen Typen.

Zunächst wird die Menge der Typbezeichner (*types*) festgelegt. Eine Teilmenge hiervon bildet die Menge abstrakten Typbezeichner (*abstractTypes*) [TS 1][3]. Die Typbezeichner werden in Bezeichner für Objekttypen (*entityTypes*) und für Beziehungstypen (*relationshipTypes*) unterschieden [TS 2]. Zur Notation unterschiedlicher Rollen in Aggregationen werden Rollentypbezeichner (*roleTypes*) eingeführt. Diese Rollen werden als Beziehungen aufgefaßt [TS 3].

In dem hier verwendeten Entity-Relationship-Modell können sowohl Objekttypen als auch Beziehungstypen attributiert werden. Jedem Typbezeichner eines Modells wird hierzu durch die

[3] In der Semantikdefinition wird hierzu später definiert, daß in einem Graphen zu diesen abstrakten Typen keine Instanzen existieren.

Abbildung *typeDefinitionSet* ein (evtl. auch leeres) Attributschemata (*AttributsSchema*) zugeordnet [TS 4]. Attributschemata sind hierzu als endliche Abbildungen der Attributbezeichner (*AttrId*) in eine Wertebereichsmenge (*Domain*) definiert.

$$AttributeSchema == AttrId \twoheadrightarrow Domain$$

Die in *EER/GRAL* verwendete Wertebereichsmenge bezieht sich auf die Standardwertebereiche für ganze und reelle Zahlen, für Zeichenketten, für Wahrheitswerte und Aufzählungen. Desweiteren können zusammengesetze Wertebereiche durch Listen-, Tupel- und Recordbildung definiert werden (vgl. [Dahm et al., 1998a, Anhang 4.A]).

TypeSystem

$types : \mathbb{F}\ TypeId$
$abstractTypes : \mathbb{F}\ TypeId$
$entityTypes : \mathbb{F}\ TypeId$
$relationshipTypes : \mathbb{F}\ TypeId$
$roleTypes : \mathbb{F}\ TypeId$
$typeDefinitionSet : TypeId \twoheadrightarrow AttributeSchema$
$(\_isA\_) : TypeId \leftrightarrow TypeId$

[TS 1]: $abstractTypes \subseteq types$
[TS 2]: $\langle entityTypes, relationshipTypes \rangle$ partition $types$
[TS 3]: $roleTypes \subseteq relationshipTypes$
[TS 4]: $\mathrm{dom}(typeDefinitionSet) = types$
[TS 5]: $(\_isA\_) \subseteq ((entityTypes \times entityTypes) \cup (relationshipTypes \times relationshipTypes))$
[TS 6]: $\forall r_1, r_2 : relationshipTypes \mid r_1\ isA\ r_2 \bullet r_2 \in roleTypes \Rightarrow r_1 \in roleTypes$
[TS 7]: $\forall t, t_1, t_2 : types \mid t\ isA^*\ t_1 \wedge t\ isA^*\ t_2 \bullet \neg\ \exists\ a : AttrId \mid a \in \mathrm{dom}(typeDefinitionSet(t_1)) \cap \mathrm{dom}(typeDefinitionSet(t_2)) \bullet typeDefinitionSet(t_1)(a) \neq typeDefinitionSet(t_2)(a)$
[TS 8]: $Entity \in entityTypes$
[TS 9]: $Relationship \in relationshipTypes$
[TS 10]: $\forall e : entityTypes \bullet e\ isA^*\ Entity$
[TS 11]: $\forall r : relationshipTypes \bullet r\ isA^*\ Relationship$

Zur Festlegung einer Typhierarchie wird die Relation *_isA_* eingeführt. Hierbei sind zwei voneinander unabhängige Typhierarchien für Objekt- und Beziehungstypen zu fordern [TS 5]. Für rollenwertige Beziehungstypen wird darüber hinaus gefordert, daß deren Untertypen ebenfalls rollenwertig sind [TS 6]. Um Konflikte bei den Attributstrukturen der Untertypen auszuschließen, wird ferner in [TS 7] verlangt, daß gleichnamigen Attributen in Ober- und Untertyp auch die gleichen Wertebereiche zugeordnet sind.

Um einfach Aussagen über jeweils alle Objekt- oder Beziehungstypen formulieren zu können, werden generelle Obertypen *Entity* und *Relationship* eingeführt, die Bestandteil jedes Typsystems sind [TS 8], [TS 9].

$$\begin{array}{|l}
\textit{Entity} : \textit{TypeId} \\
\textit{Relationship} : \textit{TypeId} \\
\hline
\textit{Entity} \neq \textit{Relationship}
\end{array}$$

Alle Objekttypen sind dann Untertypen von *Entity* [TS 10] und alle Beziehungstypen sind Untertypen von *Relationship* [TS 11].

Inzidenzsystem. Die Zusammenhänge zwischen Objekttypen und Beziehungstypen werden im Inzidenzsystem beschrieben, das hierzu das Schema *TypeSystem* des Typsystems inkludiert. Jedem Beziehungstyp wird durch die Funktion *relates* ein Paar von Objekttypen zugeordnet [IncS 1], [IncS 2]. Hierdurch wird auch die Richtung der Beziehungstypen definiert und bestimmt, daß Beziehungstypbezeichner modellweit eindeutig sind.[4]

IncidenceSystem

$$\begin{array}{|l}
\textit{TypeSystem} \\
\textit{relates} : \textit{TypeId} \pfun (\textit{TypeId} \times \textit{TypeId}) \\
\textit{aggregatesInDir} : \mathbb{F}\, \textit{TypeId} \\
\hline
\text{[IncS 1]:} \quad \mathrm{dom}\, \textit{relates} = \textit{relationshipTypes} \\
\text{[IncS 2]:} \quad \mathrm{ran}\, \textit{relates} \subseteq \textit{entityTypes} \times \textit{entityTypes} \\
\text{[IncS 3]:} \quad \forall r_1, r_2 : \textit{relationshipTypes} \mid r_1 \ \textit{isA}^* \ r_2 \bullet \\
\qquad \textit{first}(\textit{relates}(r_1)) \ \textit{isA}^* \ \textit{first}(\textit{relates}(r_2)) \\
\qquad \land \ \textit{second}(\textit{relates}(r_1)) \ \textit{isA}^* \ \textit{second}(\textit{relates}(r_2)) \\
\text{[IncS 4]:} \quad \textit{aggregatesInDir} \subseteq \textit{roleTypes} \\
\text{[IncS 5]:} \quad \forall r_1, r_2 : \textit{roleTypes} \mid r_1 \ \textit{isA}^* \ r_2 \bullet \\
\qquad r_1 \in \textit{aggregatesInDir} \Leftrightarrow r_2 \in \textit{aggregatesInDir}
\end{array}$$

Werden zu einem Beziehungstyp Untertypen gebildet, ist sicherzustellen, daß die durch den Untertyp in Beziehung gesetzten Objekttypen auch mit den durch den Obertyp in Beziehung gesetzten Objekttypen verträglich sind [IncS 3]. Start- bzw. Ziel-Objekttyp des Unterbeziehungstyps müssen gleich oder Untertypen der entsprechenden Objekttypen zum Oberbeziehungstyp sein.

Aggregationsbeziehungen werden durch *roleTypes* modelliert. Hierdurch werden Komponente und Aggregat in Beziehung gesetzt. Die Aggregationsbeziehung kann sowohl auf die Komponente als auch auf das Aggregat gerichtet sein. Zur Unterscheidung wird die Menge *aggregatesInDir* als Teilmenge von *roleTypes* eingeführt [IncS 4]. Ist der betrachtete Rollentyp in dieser Menge enthalten, so beschreibt der Ziel-Objekttyp der Beziehung den Typ der Aggregate, andernfalls den der Komponente. Diese Ausrichtung der Beziehung wird auch auf hiervon abgeleitete Untertypen übertragen [IncS 5].

Invariantensystem. Im *EER*-Teil des Invariantensystems (*InvariantSystem*) werden schließlich Aussagen zur Kardinalität und zur Injektivität der Beziehungen formalisiert. Das entsprechende Z-Schema, inkludiert das zuvor definierte Inzidenz-System *IncidenceSystem*.

[4] In konkreten *EER/GRAL*-Modellierungen werden zur Beschreibung ähnlicher Beziehungstypen häufig dieselben Beziehungstyp-Bezeichner verwendet. Diese modellieren dann jeweils, nicht explizit ausgewiesene, eindeutige Spezialisierungen entspechender Supertypen.

Kardinalitäten sind durch ein Paar von natürlichen Zahlen zur Beschreibung der Ober- und Untergrenze formalisiert, wobei die Untergrenze kleiner oder gleich der Obergrenze sein muß.

Cardinality

$$min : \mathbb{N}$$
$$max : \mathbb{N}_1$$
$$min \leq max$$

Jedem Beziehungstyp wird in *InvariantSystem* durch die Funktion *limits* ein Paar von Kardinalitäten zugeordnet [InvS 1]. Diese definieren wie viele Beziehungen des angegebenen Typs ein Objekt des Start- bzw. Zieltyps eingehen darf. Für Beziehungstypen kann ferner noch durch die Menge *injective* gefordert werden, daß höchstens eine Beziehung diesen Typs dieselben Objekte in derselben Richtung miteinander verbinden darf [InvS 2]. Diese Eigenschaft überträgt sich auch auf abgeleitete Unterbeziehungstypen [InvS 3].

InvariantSystem

IncidenceSystem

$$limits : TypeId \nrightarrow (Cardinality \times Cardinality)$$
$$injective : \mathbb{F}\, TypeId$$

[InvS 1]: $\mathrm{dom}\, limits = relationshipTypes$

[InvS 2]: $injective \subseteq relationshipTypes$

[InvS 3]: $\forall r_1, r_2 : relationshipTypes \mid r_1 \ isA^* \ r_2 \bullet r_2 \in injective \Rightarrow r_1 \in injective$

***EER*-Schema.** Typsystem, Inzidenzsystem und *EER*-Teil des Invariantensystem werden im Schema *EERSchema* zusammengefaßt.

EERSchema

InvariantSystem

$$name : Id$$

Ein *EER*-Schema umfaßt ein Invariantensystem und somit auch ein Inzidenz- und ein Typsystem sowie einen Bezeichner zur eindeutigen Identifizierung des Schemas.

TGraphen und *EER*-Schemata

Durch ein *EER*-Schema wird eine Graphklasse definiert. Diese Graphklasse umfaßt alle *TGraphen*, die die durch das Schema formulierten Eigenschaften erfüllen. Die *TGraph*-basierte Semantik von *EER*-Schemata wird durch die semantische Menge $D_{EERSchema}$ beschrieben.

$$D_{EERSchema} : EERSchema \rightarrow \mathbb{P}\, TGraph$$

$$\forall\, eerSpec : EERSchema \bullet$$
$$D_{EERSchema}[\![eerSpec]\!] = \{g : TGraph \mid g \models_{EERSchema} eerSpec\}$$

Ein *EER*-Schema denotiert die Menge der *TGraphen*, die zu dem gegebenen Schema „passen“. Dieses wird durch die Relation $\models_{EERSchema}$ formalisiert, die bezogen auf das Typsystem, das Inzidenzsystem und den *EER*-Teil des Invariantensystem definiert wird.

Typsystem. Ein Graph paßt genau dann zu einem Typsystem, wenn die Menge der Objekttypen des Typsystems alle im Graphen verwendeten Knotentypen [matchesTS 1] und die Menge der Beziehungstypen alle im Graphen verwendeten Kantentypen [matchesTS 2] enthält. Der Graph darf keine Elemente eines abstrakten Typs enthalten [matchesTS 3]. Dieses wird durch die Relation $\models_{TypeSystem}$ formalisiert:

$$(\_ \models_{TypeSystem} \_) : TGraph \leftrightarrow TypeSystem$$

$$\forall G : TGraph;\ TypeSystem \bullet$$
$$G \models_{TypeSystem} \theta TypeSystem \Leftrightarrow$$
$$\text{[matchesTS 1]:}\quad \forall v : G.V \bullet type(v) \in entityTypes$$
$$\text{[matchesTS 2]:}\quad \forall e : G.E \bullet type(e) \in relationshipTypes$$
$$\text{[matchesTS 3]:}\quad \forall elem : G.V \cup G.E \bullet G.type(elem) \notin abstractTypes$$
$$\text{[matchesTS 4]:}\quad \forall elem : G.V \cup G.E \bullet G.value(elem) \models_{AttributeSchema} (\bigcup \{t : types \mid G.type(elem)\ isA^*\ t \bullet typeDefinitionSet(t)\})$$

Neben diesen strukturellen Eigenschaften muß auch sichergestellt sein, daß die Attributwerte aller Graphelemente den im Schema definierten Wertebereichen genügen [matchesTS 4]. Hierbei ist die Vererbung der Attributschemata der jeweiligen Obertypen zu beachten. Die Attributbelegungen (*AttributeInstanceSet*) der Graphelemente müssen der Zusammenfassung der Attributschemata dieser Obertypen genügen.

$$(\_ \models_{AttributeSchema} \_) : AttributeInstanceSet \leftrightarrow AttributeSchema$$

$$\forall inst : AttributeInstanceSet;\ def : AttributeSchema;\ db : LegalDomainBinding \bullet$$
$$inst \models_{AttributeSchema} def \Leftrightarrow$$
$$\text{[matchesAS 1]:}\quad \mathrm{dom}\, inst = \mathrm{dom}\, def$$
$$\text{[matchesAS 2]:}\quad \forall attr : \mathrm{dom}\, inst \bullet inst(attr) \in valueSetOf(def(attr))$$

Eine Attributbelegung paßt genau dann zu einem Attributschema (vgl. $\models_{AttributeSchema}$), wenn die Attributbelegung und das Attributschema dieselben Attributbezeichner verwenden [matchesAS 1] und die Attributwerte Elemente der Wertemenge des entsprechenden Wertebereichs [matchesAS 2] sind. Diese Wertemengen werden durch die Funktion $ValueSetOf : EERDomain \rightarrow \mathbb{P}\, value$ über den Basiswertebereichen und den zusammengesetzten Wertebereichen definiert. Eine ausführliche Darstellung des in *EER/GRAL* auf Basis des Graphenlabors [Dahm / Widmann, 1998] realisierten Wertebereichssystems ist in [Dahm et al., 1998a, Anhang 4.A] beschrieben.

Inzidenzsystem. Die Relation $\models_{IncidenceSystem}$ definiert, wann ein Graph zu einem Inzidenzsystem paßt. Hierzu muß er zunächst dem im Inzidenzsystem enthaltenen Typsystem genügen [matchesIncS 1]. Ebenso müssen im Graph aber auch die Inzidenzdefinitionen des Schemas berücksichtigt werden [matchesIncS 2].

Jede Kante eines dem Inzidenzsystem genügenden Graphen muß hierzu aus einem Knoten ausgehen bzw. in einen Konten eingehen, dessen Typ dem Start- bzw. Zieltyp des Beziehungstyps der Kante entspricht[5]. Hierbei sind auch die jeweiligen Untertypen zu berücksichtigen.

$$(\_ \models_{\mathit{IncidenceSystem}} \_) : \mathit{TGraph} \leftrightarrow \mathit{IncidenceSystem}$$

$$\begin{aligned}
&\forall G : \mathit{TGraph};\ \mathit{IncidenceSystem} \bullet \\
&\quad G \models_{\mathit{IncidenceSystem}} \theta\mathit{IncidenceSystem} \Leftrightarrow \\
&\qquad \text{[matchesIncS 1]:}\quad G \models_{\mathit{TypeSystem}} \theta\mathit{TypeSystem} \\
&\qquad \text{[matchesIncS 2]:}\quad \forall e : G.E \bullet \\
&\qquad\qquad G.\mathit{type}(\alpha(G,e))\ \mathit{isA}^*\ \mathit{first}(\mathit{relates}(G.\mathit{type}(e))) \wedge \\
&\qquad\qquad G.\mathit{type}(\omega(G,e))\ \mathit{isA}^*\ \mathit{second}(\mathit{relates}(G.\mathit{type}(e)))
\end{aligned}$$

Invariantensystem. Durch die Relation $\models_{\mathit{InvariantSystem}}$ wird festgelegt, wann ein Graph zu einem Invariantensystem paßt. Dieses ist genau dann der Fall, wenn der Graph auch zum enthaltenen Inzidenzsystem paßt [matchesInvS 1] und den Kardinalitäts- [matchesInvS 2] und Injektivitätsbedingungen [matchesInvS 3] genügt.

$$(\_ \models_{\mathit{InvariantSystem}} \_) : \mathit{TGraph} \leftrightarrow \mathit{InvariantSystem}$$

$$\begin{aligned}
&\forall G : \mathit{TGraph};\ \mathit{InvariantSystem} \bullet \\
&\quad G \models_{\mathit{InvariantSystem}} \theta\mathit{InvariantSystem} \Leftrightarrow \\
&\qquad \text{[matchesInvS 1]:}\quad G \models_{\mathit{IncidenceSystem}} \theta\mathit{IncidenceSystem} \\
&\qquad \text{[matchesInvS 2]:}\quad \forall r : \mathit{relationshipTypes} \bullet \\
&\qquad\qquad \forall v : G.V \mid G.\mathit{type}(v)\ \mathit{isA}^*\ \mathit{first}(\mathit{relates}(r)) \bullet \\
&\qquad\qquad\quad \mathit{first}(\mathit{limits}(r)).\mathit{min} \\
&\qquad\qquad\qquad\quad \leq \#(G.\Lambda(v) \upharpoonright \{e : G.E \mid \mathit{type}(e)\ \mathit{isA}^*\ r \bullet (e,\mathit{out})\}) \\
&\qquad\qquad\qquad\quad \leq \mathit{first}(\mathit{limits}(r)).\mathit{max} \\
&\qquad\qquad \wedge \\
&\qquad\qquad \forall v : G.V \mid G.\mathit{type}(v)\ \mathit{isA}^*\ \mathit{second}(\mathit{relates}(r)) \bullet \\
&\qquad\qquad\quad \mathit{second}(\mathit{limits}(r)).\mathit{min} \\
&\qquad\qquad\qquad\quad \leq \#(G.\Lambda(v) \upharpoonright \{e : G.E \mid \mathit{type}(e)\ \mathit{isA}^*\ r \bullet (e,\mathit{in})\}) \\
&\qquad\qquad\qquad\quad \leq \mathit{second}(\mathit{limits}(r)).\mathit{max} \\
&\qquad \text{[matchesInvS 3]:}\quad \forall e, e_1, e_2 : G.E \mid G.\mathit{type} \mid (e) \in \mathit{injective} \\
&\qquad\qquad \wedge\, G.\mathit{type}(e_1)\ \mathit{isA}^*\ G.V\mathit{type}(e)\ \mathit{isA}^{\sim *}\ G.\mathit{type}(e_2) \bullet \\
&\qquad\qquad \alpha(e_1) = \alpha(e_2) \wedge \omega(e_1) = \omega(e_2) \Leftrightarrow e_1 = e_2
\end{aligned}$$

Ein Graph erfüllt die Kardinalitätsbedingungen eines Invariantensystems, wenn jeder Knoten des Graphen zu nicht weniger und zu nicht mehr Kanten eines Typs inzident ist, als durch *limits* gefordert ist. Das erste Kardinalitätsintervall von *limits* definiert hierbei den Grad der Knoten des Start-Objekttyps und das zweite Kardinalitätsintervall den des Ziel-Objekttyps. Die Kardinalitätsangaben werden in *EER/GRAL* für den Beziehungstyp einschließlich seiner Untertypen fest-

[5] Die *GRAL*-Funktionen $\alpha, \omega : \mathit{TGraph} \times \mathit{Edge} \nrightarrow \mathit{Vertex}$ bestimmen zu einer Kante den Start bzw. Zielknoten (vgl. auch Abbildung 5.7).

gelegt, so daß der Grad jedes Knotens, bezogen auf Kanten der hierdurch definierten Beziehungstypen, zwischen der jeweiligen Untergrenze und Obergrenze liegen müssen. Die Injektivitäts-Invariante bezieht sich hierbei auf den Beziehungstyp einschließlich seiner Untertypen.

***EER*-Schema.** Die zuvor definierten Relationen werden in der Relation $\models_{EERSchema}$ zusammengefaßt. Ein Graph „paßt“ genau dann zu einem *EER*-Schema, wenn er dem Invariantensystem und damit auch zu den hierin enthaltenen Inzidenz- und Typsystemen genügt.

$$(\_ \models_{EERSchema} \_) : TGraph \leftrightarrow EERSchema$$

$$\forall G : TGraph;\ EERSchema \bullet$$
$$G \models_{EERSchema} \theta EERSchema \Leftrightarrow G \models_{InvariantSystem} \theta InvariantSystem$$

Modellierung mit Graphklassen

Zur Defintion von Graphklassen wird auf die visuellen Modellierungssprachen des Objekt-Beziehungsparadigma (vgl. Kapitel 3.5.3) zurückgegriffen. Die graphische Notation wird im folgenden entlang des Typsystems, des Inzidenzsystems und des Invariantensystems beschrieben. Da die Definition von Kantentypen eng an die Festlegung des Inzidenzsystems gebunden ist, werden die graphischen Notationen des Typ- und Inzidenzsystems gemeinsam eingeführt.

Typ- und Inzidenzsystem. Knotentypen werden durch ein Rechteck, das mit einem eindeutigen Typbezeichner[6] annotiert ist, eingeführt. Linien, die ebenfalls mit einem modellweit eindeutigen Bezeichner markiert sind, notieren Kantentypen. Zur Festlegung des Inzidenzsystems verlaufen diese Linien zwischen den Repräsentation der Knotentypen, deren Adjazenz hierdurch definiert wird. Die Richtung des Kantentyps wird durch ein Winkelsymbol in der Mitte der Linie festgelegt.

Die Definition der Attributstruktur von Knoten- und Kantentypen erfolgt in Ovalen, die Attributbezeichner und deren Wertebereiche enthalten. Attributstrukturen zu Knotentypen können alternativ auch in dem Rechteck zur Knotentypdefinition notiert werden.

Beispiel 5.1 (Notation von Knoten- und Kantentypen)

Eine einfache Graphklasse (*simpleDataflowDescription*) zur Konzeptmodellierung von Datenflußdiagrammen ist in Abbildung 5.2 dargestellt. Dieses *EER*-Diagramm definiert die Knotentypen *Dataflow* und *Process*, sowie einen Kantentyp *relates*. In hierzu passenden Graphen sind Knoten vom Typ *Dataflow* somit durch Kanten vom Typ *relates* mit Knoten vom Typ *Process* verbunden.

Sowohl *Dataflow*- wie auch *Process*-Knoten sind mit einem *string*-wertigen *name*-Attribut ausgezeichnet. Das Winkelsymbol der *relates*-Beziehung zeigt an, daß *relates*-Kanten von *Dataflow*- nach *Process*-Knoten verlaufen. □

Zur *Generalisierung* von Knotentypen werden zwei alternative Notationen verwendet. Die Hierarchie von Knotentypen kann zum einen durch die Venn-Notation ausgedrückt werden. Hierbei

[6] Knotentyp-, Kantentyp und Attributbezeicher werden in *EER/GRAL*-Modellen in englischer Sprache notiert.

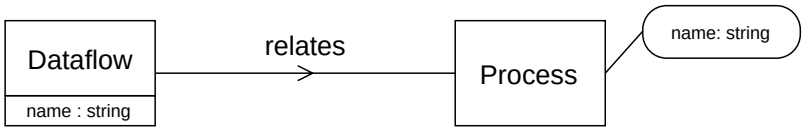


Abbildung 5.2: Notation von Knoten- und Kantentypen

werden die Untertypen in die Darstellung des Obertypen eingebettet[7]. Durch diese Notation wird symbolisiert, daß die Extensionen der Untertypen Teilmengen der Extensionen der Obertypen sind. Daneben hat diese Notationsform auch den Vorteil, daß Knotentyphierarchien auch zusammenhängend in einem Cluster (vgl. die Generalisierungs-Abstraktion als Clustering-Strategie in [Teorey et al., 1989]) dargestellt werden können. Alternativ kann auch die in [Rumbaugh et al., 1991] oder [Rumbaugh et al., 1999] verwendete „Dreiecks-Notation“ genutzt (vgl. Abbildung 5.4) werden. Diese bietet sich vor allem bei der Modellierung von Mehrfachvererbungen an. Die Generalisierung von Kantentypen erfolgt durch Annotation mit den Bezeichnern der Obertypen. Zur Unterscheidung von Ober- und Untertyp werden die Namen der Obertypen in Klammern dargestellt. Abstrakte Knotentypen, d. h. Typen, zu denen keine direkten Instanzen existieren, werden durch Schraffuren und abstrakte Kantentypen durch unterbrochene Linien beschrieben.

Beispiel 5.2 (Notation von Generalisierungen)

Ein Beispiel für Generalisierungen wird durch die Graphklasse *DataflowDescription* in Abbildung 5.3 gegeben. Die Knotentypen *Process* und *Terminator* werden hier zu *DfObject* generalisiert. Von diesem abstrakten Obertypen erben *Process* und *Terminator* das *name*-Attribut.

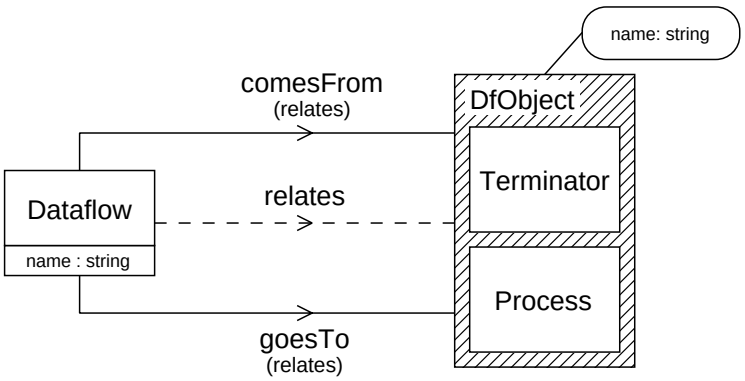


Abbildung 5.3: Notation von Generalisierungen

Zur Unterscheidung von Quelle und Ziel eines Datenflusses werden die Kantentypen *comesFrom* und *goesTo* eingeführt. Diese sind Spezialisierungen des (abstrakten) Obertyps *relates.* □

[7] Diese Darstellung darf nicht verwechselt werden mit Notation der Aggregation in FUSION [Coleman et al., 1994] oder der Komposition in UML [Rumbaugh et al., 1999, S. 230], die in diesen Ansätzen durch Ineinander-Schachtelung ausgedrückt wird.

Als eine Variante der Beziehungstypen werden *Aggregationen* verwendet. Aggregationen beschreiben solche Beziehungen zwischen Knotentypen, die anzeigen, daß der eine Knotentyp Bestandteil (Komponente) des anderen (Aggregat) ist. Aggregationen dienen auch zur Modellierung n-ärer Beziehungen. Zur Notation von Aggregationen werden auf der Spitze stehende Quadrate verwendet, die direkt an die Darstellung des Aggregat-Knotentyps angrenzen. Über mit Bezeichnern versehene Linien ist dieser mit den Komponenten-Knotentypen verbunden. Der Bezeichner beschreibt hierbei die Rolle, in der die Komponente in die Aggregation einfließt. Diese Beziehungstypen, sind analog zu Kantentypen durch Winkelsymbole zur Notation der Richtung markiert. Ferner kann ihnen in einen Oval eine Attributstruktur zugeordnet werden.

Beispiel 5.3 (Notation von Generalisierungen und Aggregationen)

Abbildung 5.4 zeigt das Klassendiagramm der Graphklasse *RegularProcessDescription* zur Beschreibung der Feinstruktur von Prozessen. Die Klasse der *Process*-Knoten wird hierbei in *AtomicProcess*, *Iteration* und *Sequence* spezialisiert. Hierzu wurde die Dreiecks-Notation verwendet.

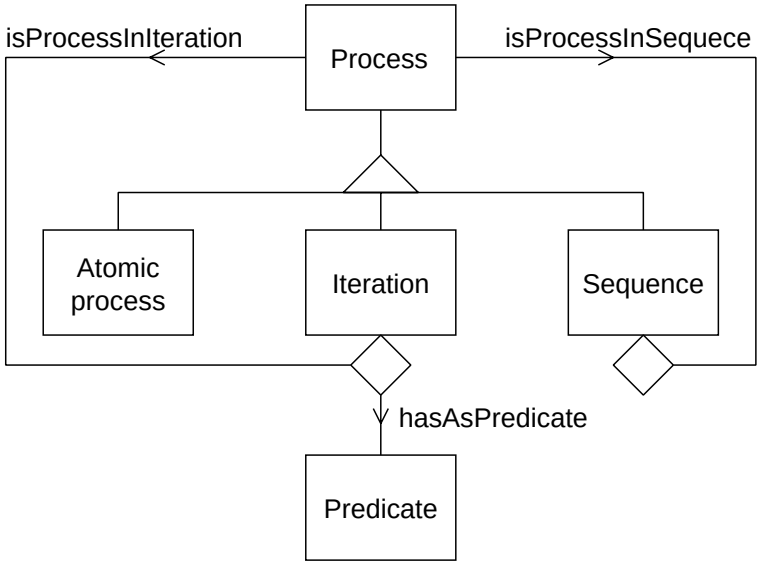


Abbildung 5.4: Notation von Generalisierungen und Aggregationen

Eine *Iteration* besteht hierbei aus einem *Predicate*, welches das Abbruchkriterium der Wiederholung angibt und aus dem iterierten *Process*. Prozesse treten hierbei in der Rolle *isProcessInIteration* und Prädikate in der Rolle *hasAsPredicate* auf. In dieser Aggregation ist die *isProcessInIteration*-Beziehung zum Aggregat hin und die *hasAsPredicate*-Beziehung zur Komponente hin ausgerichtet. Durch die Aggregation *Sequence* werden Folgen von Prozessen beschrieben. □

Invariantensystem. Der graphisch ausdrückbare Anteil des Invariantensystems umfaßt die Angaben zur Kardinalität der Kantentypen und zur Definition injektiver Beziehungen.

Kardinalitäten werden durch Annotation der Kantentypen an den jeweiligen Enden definiert. Markierungen am gegenüberliegenden Ende einer Linie machen hierbei Aussagen über den Grad

eines Knotens bezogen auf den betrachteten Kantentyp. Gehen beliebig viele Kanten in den Knoten ein bzw. aus ihm aus, wird dieses durch eine doppelte, nicht ausgefüllte Pfeilspitze (▷▷) notiert. Eine ausgefüllte Doppelspitze (▶▶), fordert das Ein- bzw. Ausgehen von mindestens einer Kante, eine nicht ausgefüllte, einfache Pfeilspitze (▷) fordert die Inzidenz von höchstens einer Kante und eine ausgefüllte, einfache Pfeilspitze (▶) fordert die Inzidenz von genau einer Kante. Werden darüber hinaus andere Einschränkungen benötigt, kann dieses durch entsprechende textuelle Markierungen erfolgen. In der graphischen Notation der *EER*-Diagramme werden Kantentypen grundsätzlich als *injektiv* angenommen. Sollen Mehrfachkanten zugelassen werden, wird dieses explizit durch einen Kreis (○) neben dem Winkelsymbol zur Beschreibung der Kanterichtung notiert.

Beispiel 5.4 (Notation von Kardinalitäten)

Das Klassendiagramm in Abbildung 5.5 faßt die Klassendiagramme aus Abbildung 5.3 und 5.4 zur Definition der Graphklasse *SimpleProcessDescription* zusammen und ergänzt noch fehlende Invarianten.

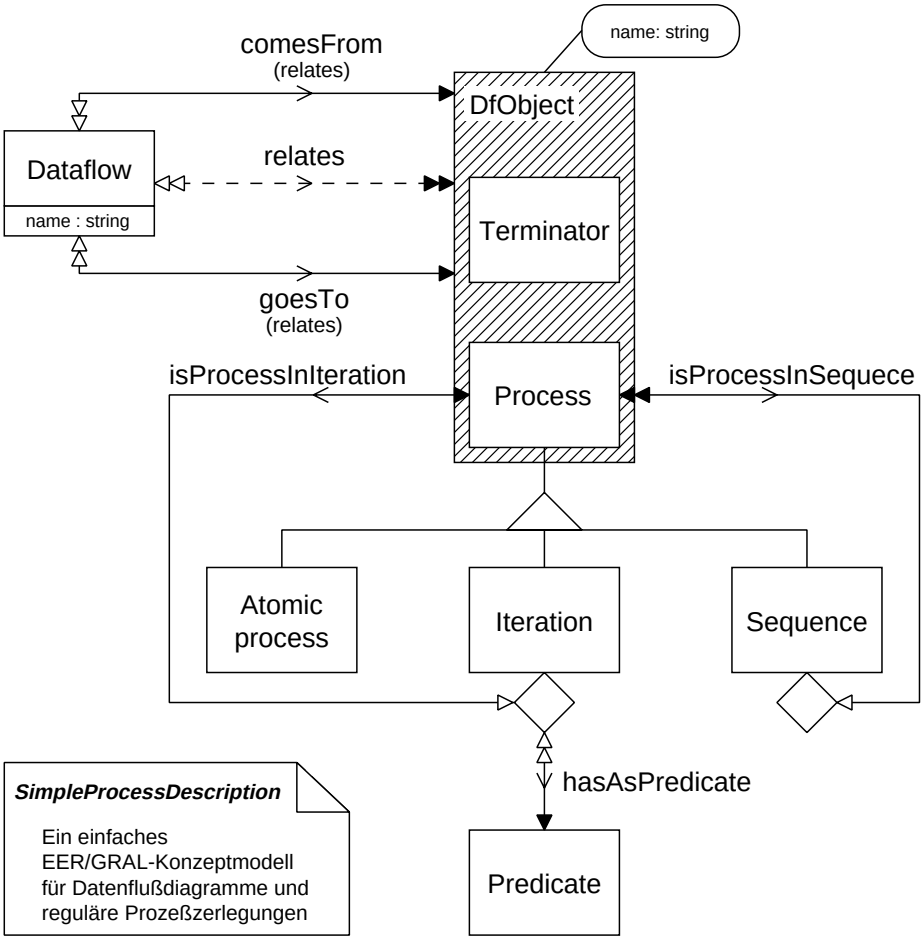


Abbildung 5.5: Notation von Kardinalitäten und Injektivitäten

Dataflow-Knoten sind über *comesFrom* bzw. über *goesTo*-Kanten zu genau einem *DfObject* adjazent. Umgekehrt können aber *DfObject*-Knoten zu beliebig vielen *Dataflow*-Knoten benachbart sein.

Mit den gleichen Mitteln werden auch die Invarianten zu Aggregationen definiert. Einer *Iteration* ist genau ein iterierter *Prozess* und genau ein *Predicate* zugeordnet. Ein *Predicate* kann aber in beliebig vielen *Iteration*en verwendet werden. Dagegen dürfen *Prozesse* Komponenten höchstens einer *Iteration* sein. Eine *Sequence* besteht aus einer nicht leeren Folge von *Prozess*en. Wie auch bei der Iteration können *Prozesse* nur Komponenten höchstens einer *Sequence* sein. □

Kommentare zu Klassendiagrammen werden in *EER/GRAL* in einem Rechteck mit „Eselsohr" notiert.

GRAL

Zur Formulierung weiterer Invarianten ist die zuvor dargestellte visuelle Notation zur Beschreibung von Graphklassen noch um eine textuelle Sprache zu ergänzen. Hierzu wird die GRAph specification Language (*GRAL*) verwendet. Die erste Fassung von *GRAL* [Capellmann / Franzke, 1991] wurde in verschiedenen Projekten (einen Überblick hierzu gibt [Ebert et al., 1996b]) eingesetzt und weiterentwickelt. Die aktuelle Version *GRAL* 2.0, die auch in dieser Arbeit eingesetzt wird, ist in [Franzke, 1997] beschrieben.

GRAL wurde als Sprache zur Formalisierung von Invarianten an *TGraph*-Klassen entwickelt. Prädikate in getypter Logik erster Stufe ergänzen die *EER*-Graphklassen-Definitionen. Eine *EER/GRAL*-Graphklassen-Definition denotiert dann die Menge aller *TGraphen*, die dem *EER*-Diagramm entsprechen (vgl. die Relation $\_ \models_{EERSchema} \_$) und zusätzlich die in *GRAL* spezifizierten weiteren Prädikate erfüllen.

***GRAL*-Prädikate und -Funktionen.** Zur Notation der *GRAL*-Prädikate wird auf die Spezifikationssprache Z [Spivey, 1992] zurückgegriffen. *GRAL* umfaßt eine Teilmenge des Sprachumfangs von Z. Diese umfaßt vollständig den Z-Sprachumfang für Mengentheorie und Prädikatenlogik erster Stufe. Zur Unterstützung der graphbasierten Modellierung wird *GRAL* ergänzt um eine umfangreiche und erweiterbare *TGraph*-Bibliothek.

Prädikat	Signatur	Beschreibung
isConnected	$\mathbb{P}$ *TGraph*	zusammenhängender *TGraph*
isDag	$\mathbb{P}$ *TGraph*	azyklischer, gerichteter *TGraph*
isTree	$\mathbb{P}$ *TGraph*	baumartiger *TGraph* [8]
isAbstract	$\mathbb{P}$ *TypeId*	abstrakter Knoten- oder Kantentyp [9]
isConcrete	$\mathbb{P}$ *TypeId*	konkreter Knoten- oder Kantentyp [9]

Abbildung 5.6: Prädikate der *GRAL*-Bibliothek (Auswahl)

Diese Bibliothek umfaßt Prädikate und Funktionen zur Beschreibung grundlegender Grapheigenschaften, zur Berechnung von Teilgraphen, zur Berechnung von Eigenschaften von Graphelementen und zur Beschreibung von Schemaeigenschaften. Eine Übersicht über die in dieser

[8] Kanten werden in Bäumen als von den Elternknoten zu den Kinderknoten gerichtet aufgefaßt.
[9] Dieses Prädikat erweitert den Sprachumfang von [Franzke, 1997].

Arbeit verwendete *GRAL*-Prädikate und -Funktionen wird in Abbildung 5.6 und 5.7 gegeben. Die formale Semantik dieser Prädikate und Funktionen wird in [Franzke, 1997, Appendix C] definiert.

GRAL unterstützt die Unterscheidung von Typen und Klassen. Während die Extension eines Typs die Graphelemente umfaßt, denen genau dieser Typ zugeordnet ist, beschreibt die Extension einer Klasse die Graphelemente diesen Typs und seiner Untertypen. Die Extensionen von Knoten- bzw. Kantentypen werden durch V^{type} bzw. E^{type} berechnet, die Extensionen der Knoten- bzw. Kantenklassen durch V bzw. E.

Die konkrete Syntax von *GRAL* erlaubt die in Z üblichen Prefix- und Infix-Notationen (z. B. $\delta(G, \rightharpoonup, EType, v)$). Die Notation von Graph-, Richtungs- und Typangaben als tiefgestellte Indizes (z. B. $\delta_{G,\rightharpoonup,EType}(v)$), die *GRAL*-Ausdrücke i. allg. lesbarer machen, ist ebenfalls zugelassen. Richtungsangaben zu Kanten werden sowohl textuell („*in*“, „*out*“) als auch symbolisch durch „$\leftharpoonup$“ und „$\rightharpoonup$“ notiert. Ist die Kantenrichtung beliebig, wird dieses durch „$\leftrightharpoons$“ beschrieben.

Zur Vereinfachung der *GRAL*-Prädikate und -Funktionen können Parameter, die (statisch) aus dem Kontext erschlossen werden können, ausgelassen werden. Insbesondere kann hier häufig die Angabe des betrachteten Graphen entfallen. Unterbleibt die Angabe von Typ- ($\langle TypeId \rangle$) oder Richtungsangaben ($\langle Dir \rangle$) wird hierfür als Defaultwert der allgemeinste Typbezeicher (*Entity* oder *Relationship*) bzw. die beliebige Richtungsangabe („$\leftrightharpoons$“) angenommen.

***GRAL*-Pfadbeschreibungen.** Ein wesentliches Modellierungsmittel in *GRAL* sind *Pfadbeschreibungen* zur Spezifikation von Aussagen über den Zusammenhang zwischen Knoten. Pfadbeschreibungen sind reguläre Ausdrücke über Knoten- und Kantensymbole:

- *Einfache Pfadbeschreibungen* bestehen aus einem Kantensymbol „$\leftharpoonup$“, „$\rightharpoonup$“ oder „$\leftrightharpoons$“ zur Beschreibung eingehender, ausgehender und beliebig gerichteter Kanten. Zur Einschränkung auf bestimmte Kantentypen kann das Kantensymbol durch einen Typbezeichner annotiert werden.

- *Zusammengesetzte Pfadbeschreibungen* ermöglichen Sequenzen, Iterationen, Alternativen, Optionen und Klammerungen von Pfadbeschreibungen und spezifizieren Einschränkungen an den Typ des Zielknotens einer Pfadbeschreibung.

  Seien p_1 und p_2 Pfadbeschreibungen und *vType* ein Knotentypbezeichner, dann ist auch
  - die Sequenz $p_1 p_2$,
  - die Iteration p_1^+ (transitiver Abschluß),
  - die Iteration p_1^* (transitiver, reflexiver Abschluß),
  - die Alternative $p_1 \mid p_2$,
  - die Option $[p_1]$,
  - die Klammerung (p_1) und
  - die Einschränkung $p_1 \,\&_{vType}$,

  eine Pfadbeschreibung.

Pfadbeschreibungen werden sowohl als Prädikate als auch als Funktionen verwendet. Infix notierte *Pfad-Prädikate* $p : Vertex \leftrightarrow Vertex$ der Form $v\ p\ w$ überprüfen ob der Knoten w über ein

[10] Diese Funktion erweitert den Sprachumfang von [Franzke, 1997].

Funktion	Signatur	Beschreibung
V	$TGraph \times TypeId \nrightarrow \mathbb{F}\, Vertex$	Menge der Knoten der Knotenklasse $\langle TypeId \rangle$
V^{type}	$TGraph \times TypeId \nrightarrow \mathbb{F}\, Vertex$	Menge der Knoten des Knotentyps $\langle TypeId \rangle$
E	$TGraph \times TypeId \nrightarrow \mathbb{F}\, Edge$	Menge der Kanten der Kantenklasse $\langle TypeId \rangle$
E^{type}	$TGraph \times TypeId \nrightarrow \mathbb{F}\, Edge$	Menge der Kanten des Kantentyps $\langle TypeId \rangle$
E^{-}	$TGraph \times TypeId \nrightarrow \mathbb{F}\, Edge$	Menge der Kanten der Kantenklasse $\langle TypeId \rangle$ in umgekehrter Kantenrichtung[10]
$E^{type^{-}}$	$TGraph \times TypeId \nrightarrow \mathbb{F}\, Edge$	Menge der Kanten des Kantentyps $\langle TypeId \rangle$ umgekehrter Kantenrichtung[10]
$\_\cdot\_$	$TGraph \times Element \times AttrId \nrightarrow Value$	Wert des durch $\langle AttrId \rangle$ angegebenen Attributs zum Graphelement $\langle Element \rangle$
α, ω	$TGraph \times Edge \nrightarrow Vertex$	Anfangs- bzw. Zielknoten der Kante $\langle Edge \rangle$
δ	$TGraph \times Dir \times TypeId \times Vertex \nrightarrow \mathbb{N}$	Anzahl der zum Knoten $\langle Vertex \rangle$ inzidenten Kanten unter Berücksichtigung der Kantenrichtung ($\langle Dir \rangle$) und des Kantentyps ($\langle TypeId \rangle$)
Γ^{set}	$TGraph \times Dir \times TypeId \times Vertex \nrightarrow Vertex$	Menge der zum Knoten $\langle Vertex \rangle$ adjazenten Knoten unter Berücksichtigung der Kantenrichtung ($\langle Dir \rangle$) und des Kantentyps ($\langle TypeId \rangle$)
Λ^{set}	$TGraph \times Dir \times TypeId \times Vertex \nrightarrow Edge$	Menge der zum Knoten $\langle Vertex \rangle$ inzidenten Kanten unter Berücksichtigung der Kantenrichtung ($\langle Dir \rangle$) und des Kantentyps ($\langle TypeId \rangle$)
vGraph	$TGraph \times \mathbb{P}\, Vertex \nrightarrow TGraph$	durch die Knotenmenge $\langle \mathbb{P}\, Vertex \rangle$ knotenerzeugter Teilgraph
eGraph	$TGraph \times \mathbb{P}\, Edge \nrightarrow TGraph$	durch die Kantenmenge $\langle \mathbb{P}\, Vertex \rangle$ kantenerzeugter Teilgraph
root	$TGraph \nrightarrow Vertex$	Wurzel $\langle Vertex \rangle$ eines baumartigen Graphen [10]
order	$TGraph \times TypeId \times Edge \times Vertex \nrightarrow \mathbb{N}$	Ordnung der Kante $\langle Edge \rangle$ in der Inzidenzliste des Knotens $\langle Vertex \rangle$ bzgl. Kanten der Klasse $\langle TypeId \rangle$[10]

Abbildung 5.7: Funktionen der *GRAL*-Bibliothek (Auswahl)

Pfad der Form p von einem Knoten v aus erreichbar ist. Als *Pfad-Funktion* $p : \mathit{Vertex} \nrightarrow \mathbb{P}\,\mathit{Vertex}$ aufgefaßt, berechnet die Pfadbeschreibung vp die Menge der Knoten, die von v aus über den angegebenen Pfad erreichbar sind und pv die Menge der Knoten, die v über den angegebenen Pfad erreichen.

***GRAL*-Zusicherungen zu Graphklassen-Definitionen.** *GRAL*-Prädikate beziehen sich immer auf eine Graphklassen-Definition. Durch eine **assert**-Klausel werden *GRAL*-Prädikate, zusammengefaßt und an eine Graphklassen-Definition gebunden. Zur eindeutigen Referenzierung erhalten diese Prädikate einen eindeutigen Bezeichner.

Beispiel 5.5 GRAL-Ergänzung zum EER-Schema SimpleProcessDescription

Die folgende Zusicherung enthält eine *GRAL*-Spezifikation zur Ergänzung der Graphklassendefintion *SimpleProcessDescription* aus Beispiel 5.4.

In Datenflußdiagrammen sind in der Regel keine Datenflüsse direkt zwischen Terminatoren zugelassen. Zwei *Terminator*-Knoten dürfen daher nicht über einen Pfad der Form $\leftharpoonup_{comesFrom} \rightharpoonup_{goesTo}$ miteinander verbunden sein [SPD 1]. Prozeßzerlegungen durch reguläre Strukturen sind baumartig. Diese Struktur wird in der Beispielmodellierung durch Kanten der Typen *isProcessInIteration* und *isProcessInSequence* gebildet. Der hierdurch kantenerzeugte Teilgraph (d. h. der Teilgraph, der ausschließlich Kanten dieser Typen enthält) muß baumartig sein [SPD 2].

Für dieses *GRAL*-Prädikat wurde die vereinfachte Notation verwendet. Alle Aussagen beziehen sich hier auf den Graph G der Graphklasse *SimpleProcessDescription*, der durch die **assert**-Klausel gebunden wurde. Der Parameter G wurde daher ausgelassen.

for G **in** *SimpleProcessDescription* **assert**

[SPD 1]:

$$\{t_1, t_2 : V_{Terminator} \mid t_1 \leftharpoonup_{comesFrom} \rightharpoonup_{goesTo} t_2\} = \varnothing\,;$$

[SPD 2]:

$$isTree(eGraph(E_{isProcessInIteration} \cup E_{isProcessInSequence}))$$

end.

□

Erweiterungen von *GRAL*. Durch die **assert**-Klausel ermöglicht *GRAL* die Ergänzung von *EER*-Graphklassendefinition durch Graphprädikate in Logik erster Stufe. Die Ableitung weiterer Graphklassen durch Spezialisierung vorhandener Graphklassen und die Erweiterung von Graphklassendefinitionen um weitere Konzepte wie beispielsweise zusätzliche Methoden über Graphelemente sieht [Franzke, 1997] nicht vor. Da solche Modellierungsmittel insbesondere für die Ableitung von speziellen Modellen aus Referenzmodellen benötigt werden, werden hierzu weitere *GRAL*-Sprachmittel eingeführt.

Spezialisierungen von *EER/GRAL*-Graphklassendefinitionen werden durch **isA**-Klauseln notiert. *Ergänzungen* um zusätzliche Konstrukte in Graphklassendefinitionen erfolgen analog zur axiomatischen Defintion in Z [Spivey, 1992, S. 48] durch **define**-Klauseln, durch die Objekte deklariert und (optional) durch *GRAL*-Prädikate konkretisiert werden können.

Beispiel 5.6 (Spezialisierung der Graphklassenspezifikation SimpleProcessDescription)

Die Graphklasse *SimpleProcessDescription* wird zur Graphklasse *SimpleProcessDescriptionWithCnt* spezialisiert [SPDcnt 1] und um eine Komponente *ProcessCnt* ergänzt, die die Knoten vom Typ *Process* zählt. Die Spezifikation von *ProcessCnt* erfolgt durch eine Deklaration [SPDcnt 2] und die Beschreibung der Eigenschaften durch ein *GRAL*-Prädikat [SPDcnt 3].

[SPDcnt 1]: *SimpleProcessDescriptionWithCnt* **isA** *SimpleProcessDescription*;

for *G* **in** *SimpleProcessDescriptionWithCnt* **define**

[SPDcnt 2]:

ProcessCnt : $\mathbb{N}$

where

[SPDcnt 3]:

$ProcessCnt = \# V_{Process}$

end .

□

Zur Ableitung eines speziellen Modelles aus einem Referenzmodel werden die *GRAL*-Prädikate der Generalisierung mit den Prädikaten der Spezialisierung konjugiert. Durch spezielle Modelle werden die Zusicherungen an Referenzmodelle daher i. allg. weiter eingeschränkt. In wenigen Ausnahmefällen sind diese Anforderungen in speziellen Modellen zu *verallgemeinern* oder zu *ändern*. Hierzu werden in **overwrites**-Klauseln zu ändernde Zusichererungen, die durch ihren Prädikatbezeichner referenziert werden, durch eine neue Zusicherung überschrieben. Querbezüge zu Zusicherungen, die in *EER*-Diagrammen graphisch formuliert sind (z. B. Kardinalitätsanforderungen), werden analog durch Prädikatbezeichner hergestellt, die in Kommentaren zu den entsprechenden graphischen Symbolen in den Diagrammen festgelegt sind.

Beispiel 5.7 (Überschreibung von Zusicherungen)

In Zusicherung [SPD 2] wurde für Prozeßzerlegungen durch reguläre Strukturen die Baumartigkeit gefordert. Um (unnötige) Duplizierungen von Prozeßzerlegungen zu vermeiden, können gleichartige Prozeßzerlegungen auch zusammengefaßt werden. Für durch Kanten der Typen *isProcessInIteration* und *isProcessInSequence* erzeugte Teilgraphen der entsprechenden Spezialisierung *SimpleProcessDescriptionDag* der Graphklasse *SimpleProcessDescriptionWithCnt* reicht dann die Forderung nach Azyklizität aus. Zusicherung [SPDDag 2] überschreibt hierzu Zusicherung [SPD 2].

[SPDDag 1]: *SimpleProcessDescriptionDag* **isA** *SimpleProcessDescriptionWithCnt*;

for *G* **in** *SimpleProcessDescriptionDag* **assert**

[SPDDag 2]:

overwrites [SPD 2] :

$isDag(eGraph(E_{isProcessInIteration} \cup E_{isProcessInSequence}))$

end .

□

Die Zusammenfassung mehrerer Teilmodelle zu einem integrierten Modell erfolgt durch die **include**-Klausel. Hierdurch werden ebenfalls alle Prädikte der inkludierten Modelle konjugiert.

Beispiel 5.8 (Zusammenfassung von Teilmodellen)

Zur Definition des *EER*-Anteils der Graphklasse *SimpleProcessDescription* in Beispiel 5.4 wurden bereits die Graphklassen *DataflowDescription* und *RegularProcessDescription* zusammengefaßt. Die weiteren Zusicherungen an die Graphklasse *SimpleProcessDescription* wurden in Beispiel 5.5 für beide Teilgraphklassen gemeinsam formuliert. Das Prädikat [SPD 1] bzw. das Prädikat [SPD 2] hätte aber auch direkt der Definition des *EER*-Anteils von *DataflowDescription* bzw. *RegularProcessDescription* zugeordnet werden können.

Die Integration dieser Teilmodelle, entspricht der folgenden **include**-Klausel, die neben der Zusammenfassung der *EER*-Teilmodelle auch die Konjunktion der zu diesen Teilmodellen formulierten Prädikate umfaßt.

for *G* **in** *SimpleProcessDescription* **include**
DataflowDescription;
RegularProcessDescription
end.

Diese Definition ist mit der Graphklassendefinition aus Beispiel 5.5 identisch. □

5.3 Zusammenfassung

Ausgehend von einer überblicksartigen Einführung in die Konzeptmodellierung wurde der graphbasierte *EER/GRAL*-Ansatz zur Konzeptmodellierung eingeführt. Diese Einführung bezog sich sowohl auf die Formalisierung des *EER/GRAL*-Ansatzes als auch auf die hier zur Konzeptmodellierung verwendeten visuellen und textuellen Sprachen.

Das Referenz-Metaschema der Beschreibungsmittel für Organisationen und Softwaresysteme wird in Kapitel 6 mit Hilfe von *EER*-Klassendiagrammen und *GRAL*-Klauseln beschrieben.

6 Referenz-Metaschema

Die Herleitung des Referenz-Metaschemas für visuelle Modellierungssprachen erfolgt ausgehend von den in Kapitel 3 vorgestellten Beschreibungsmitteln. Es umfaßt die in Abbildung 3.3 auf Seite 39 aufgeführten Beschreibungsmittel und bildet damit die wesentlichen im Requirements-Engineering der Organisations- und Softwaretechnik verwendeten semi-formalen und informalen Modellierungsmittel ab. Formale Sprachen wie z. B. *Z* [Spivey, 1992] oder VDM [Jones, 1990] und Programmiersprachen werden in diesem Metaschema *nicht* betrachtet. Analoge Konzeptmodelle für formale Sprachen werden z. B. in [Ebert et al., 1997a] für eine eine *TGraph*-basierte Erweiterung von *Z* oder in [Ebert et al., 1998] und [Kullbach et al., 1998] für Programmiersprachen vorgestellt.

6.1 Herleitung und Spezialisierung des Referenz-Metaschemas

Ausgangspunkt der Erstellung der in Kapitel 4.3 skizzierten Metaschemata zur Unternehmensmodellierung waren in der Regel theoretische Überlegungen zur abzubildenden Diskurswelt. Die hierbei als wichtig erachteten Konzepte und ihre Beziehungen wurden zunächst modelliert. Anschließend wurden Beschreibungstechniken ausgewählt bzw. festgelegt, die eine metaschemakonforme Modellierung erlauben. Die Wahl der Beschreibungstechniken war in diesen Arbeiten (vgl. hierzu insbesondere [Olle et al., 1991], [Scheer, 1992] und [Frank, 1994]) vom *Metaschema* determiniert.

Für die Herleitung des Referenz-Metaschemas der visuellen Modellierungsprachen für Organisationen und Softwaresysteme werden stattdessen die Beschreibungsmittel selbst in den Vordergrund gestellt. Die wesentlichen Konzepte der Sprachen der einzelnen Paradigmen, die in den Kapiteln 3.2 bis 3.5 herausgearbeitet wurden, werden im Referenz-Metaschema abgebildet. Dieses Referenz-Metaschema der Beschreibungsmittel für Organisationen und Softwaresysteme ist folglich durch die *Beschreibungsmittel* bestimmt.

Entlang des in Kapitel 3.1 vorgestellten Klassifikationsschemas der Beschreibungsmittel für Organisationen und Softwaresysteme werden in den Kapiteln 6.2 bis 6.5 für jedes Beschreibungsparadigma zunächst isolierte *EER/GRAL*-Konzeptmodelle erstellt. Diese Konzeptmodelle können als Referenz-Metaschemata für die Beschreibungsmittel der jeweiligen Paradigmen angesehen werden. Ihre Referenz-Eigenschaft wird durch exemplarische Anwendung auf verschiedene notationelle Varianten des betrachteten Beschreibungsparadigmas belegt.

Die Ableitung eines spezialisierten Modells aus einem Referenzmodell erfolgt durch Anpassung des Referenzmodells an diese spezialisierten Anforderungen. Hierzu sind Referenzmodelle um Konzepte *einzuschränken*, die für das spezielle Modell ohne Belang sind, oder um zusätzliche objekt- und beziehungsartige Konzepte zu *erweitern*, die im Referenzmodell nicht abgebildet sind (vgl. zur Spezialisierung von Referenzmodellen auch [Hars, 1994, S. 129ff]).

Die Anwendung der Referenz-Metaschemata zur Ableitung der Metaschemata konkreter Beschreibungsmittel erfolgt durch Spezialisierung des *EER/GRAL*-Konzeptmodells des jeweils betrachteten Beschreibungsparadigmas. Hierdurch werden die Konzepte des Referenz-Metaschemas auf das spezialisierte Metaschema übertragen. Einschränkungen und Erweiterungen des spezialisierten Metaschemas werden sowohl graphisch durch Klassendiagramme als auch durch zusätzliche *GRAL*-Prädikate und Definitionen beschrieben.

In Kapitel 6.6 erfolgt schließlich die Integration der Referenz-Metaschemata der Beschreibungsparadigmen zum integrierten Referenz-Metaschema der visuellen Modellierungssprachen für Organisationen und Softwaresysteme. Dieses Referenz-Metaschemata wird in Teil III zur Darstellung von Metaschemata der Beschreibungsmittel konkreter Modellierungsmethoden und zur Entwicklung von Modellierungswerkzeugen angewandt.

6.2 Metaschemata der Aufgabensicht

Im Metaschema der Aufgabensicht sind die visuellen Modellierungssprachen des Aufgabengliederungsparadigmas zusammengefaßt. Zentrale Beschreibungskonzepte der Aufgabensicht sind *Aufgaben* mit ihren Verrichtungs- und Objekt-Komponenten. Das Metaschema des Aufgabengliederungsparadigmas ergänzt diese um Konzepte zur Zerlegung von Aufgaben in Teilaufgaben.

6.2.1 Metaschemata des Aufgabengliederungsparadigmas

Mit den Sprachen des Aufgabengliederungsparadigmas (vgl. Kapitel 3.2.1) wird die Zerlegung von Aufgaben in Teilaufgaben dargestellt. Das Metaschema des Aufgabengliederungsparadigmas stellt hierzu Konzepte zur Abbildung der Aufgaben, zur Beschreibung ihrer charakteristischen Eigenschaften und der Gliederung der Aufgaben nach unterschiedlichen Aufgabentypen bereit.

Referenz-Metaschema des Aufgabengliederungsparadigmas

Abbildung 6.1 zeigt das Referenz-Metaschema des Aufgabengliederungsparadigmas *TDP* (vgl. auch [Scheer, 1992, S. 67] und [Winter / Ebert, 1996, S. 108]). Aufgaben (*Task*) ist genau ein Prozeß (*Process*) als Verrichtungskomponente und eine Objektklasse (*ObjectClass*) als Objektkomponente zugeordnet. Darüber hinaus sind Aufgaben durch ihre Namen (*name*), den Ort der Aufgabenerledigung (*location*), ihre Dauer (*duration*), den Zeitpunkt ihrer Erledigung (*time*) und der hierzu eingesetzten Hilfsmittel (*resources*) charakterisiert. Konkretisierungen der Konzepte *Process* und *ObjectClass* aus Ablauf- bzw. aus Objektsicht finden sich in Kapitel 6.4 und 6.5.

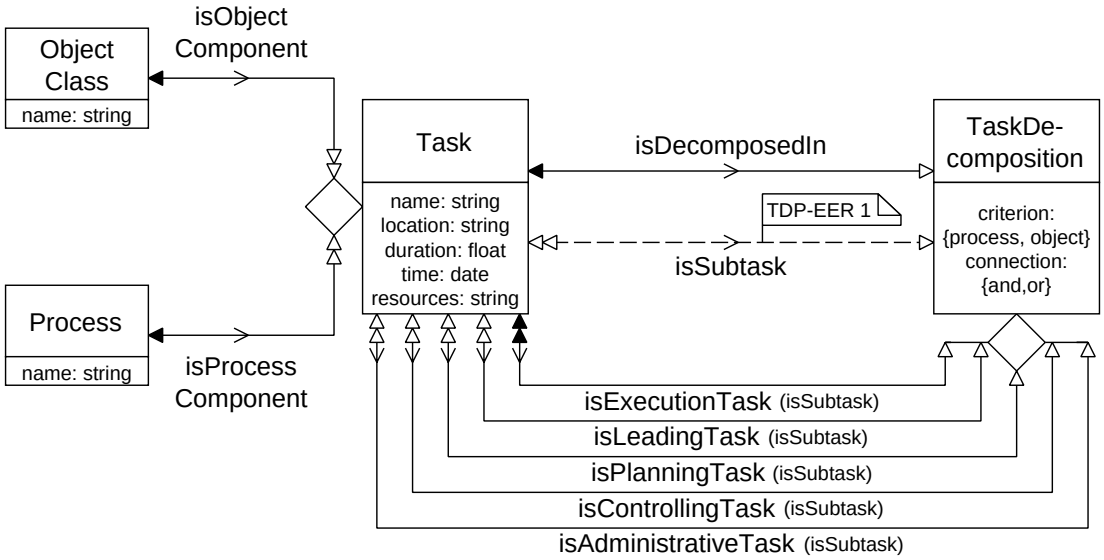


Abbildung 6.1: Referenz-Metaschema des Aufgabengliederungsparadigmas (Task Decomposition Paradigm, *TDP*)

Jeder Aufgabe kann eine Aufgabenzerlegung (*TaskDecomposition*) zugeordnet werden, die die jeweiligen Teilaufgaben durch den abstrakten Kantentyp *isSubtask* zusammenfaßt. Die Injektivität von *isSubtask* stellt sicher, daß eine Teilaufgabe nur höchstens einmal in einer Aufgabengliederung enthalten ist.

Gemäß der Gliederungsmerkmale Rang, Phase und Zweck (vgl. Kapitel 3.2.1) werden verschiedene Teilaufgabenbeziehungen unterschieden. Jede Aufgabengliederung umfaßt mindestens eine Ausführungsaufgabe (*isExecutionTask*) und beliebig viele Leitungs- (*isLeadingTask*), Planungs- (*isPlanningTask*), Kontroll- (*isControllingTask*) oder Verwaltungsaufgaben (*isAdministrativeTask*). Die Untergliederung in Ausführungsaufgaben entlang der Objekt- oder der Verrichtungskomponente sowie die Art Verknüpfung der Teilaufgaben durch Und- oder Oder-Gliederungen wird durch die Attributierung der *TaskDecomposition* modelliert. Jede Aufgabenzerlegung untergliedert eine Aufgabe nach genau einem Gliederungsprinzip und nach genau einer Verknüpfungsart.

Konkrete Notationsformen des Aufgabengliederungsparadigmas beschreiben in der Regel ausschließlich baumartige Aufgabenzerlegungen. Hierbei bezieht sich die Baumartigkeit nur auf die *isDecomposedIn*-Kanten in Kantenrichtung und die *isSubtask*-Kanten in umgekehrter Kantenrichtung [TDP 1].

for G **in** *TDP* **assert**

[TDP 1]:

$$isTree(eGraph(E_{isDecomposedIn} \cup E^{-}_{isSubtask}))$$

end .

Die Gesamtaufgabe der so modellierten Aufgabengliederung wird durch die Wurzel dieses Baumes *mainTask* abgebildet.

for G **in** *TDP* **define**

$mainTask : V_{Task}$

where

$mainTask = root(eGraph(E_{isDecomposedIn} \cup E^{-}_{isSubtask}))$

end.

Spezialisierungen des Referenz-Metaschemas

In der Literatur zur Aufgabengliederung werden je nach Anschauung weitere Anforderungen oder Einschränkungen an die Modellierung von Aufgabengliederungen gestellt. Im folgenden werden diese durch Spezialisierungen des Referenz-Metaschemas des Aufgabengliederungsparadigmas beschrieben.

Aufgabengliederungen nach Nordsieck. [Nordsieck, 1962, S. 14] betrachtet lediglich Aufgabengliederungen nach Verrichtung oder Objekt. Als ersten Gliederungsschritt fordert er die „*Ausgliederung der Verwaltungsaufgaben*“. Für die weiteren Zerlegungen sind ausschließlich Verrichtungs- und Objektgliederungen in Ausführungsaufgaben zugelassen. Somit ist für die Modellierung von Aufgabengliederungsplänen nach Nordsieck zu fordern, daß nur die Gliederung der Gesamtaufgabe *mainTask* höchstens eine Verwaltungsaufgabe enthält [TDPNordsieck 1], [TDPNordsieck 2]; alle weiteren Gliederungen umfassen ausschließlich Ausführungsaufgaben. Gliederungen in Leitungs- Planungs-, und Kontrollaufgaben werden bei Nordsieck nicht unterschieden [TDPNordsieck 3]. Das Metaschema *TDPNordsieck* spezialisiert hierzu das Referenz-Metaschema des Aufgabengliederungsparadigmas.

TDPNordsieck **isA** *TDP* ;

for G **in** *TDPNordsieck* **assert**

[TDPNordsieck 1]:

$\# E_{isAdminstrativeTask} \leq 1$;

[TDPNordsieck 2]:

$\forall e : E_{isAdminstrativeTask} \bullet \omega(e) \leftarrow_{isDecomposedIn} mainTask$;

[TDPNordsieck 3]:

$E_{isLeadingTask} = E_{isPlanningTask} = E_{isControllingTask} = \varnothing$

end.

Aufgabengliederungen nach Kosiol. [Kosiol, 1976, S. 52] lehnt eine kombinierte Anwendung der Gliederungsprinzipien Objekt und Verrichtung ab. Er befürchtet, daß durch eine Mischung bereits Entwurfsentscheidungen für die zu entwickelnde Organisationsstruktur vorweg genommen werden.

TDPKosiol **isA** *TDP* ;

for *G* **in** *TDPKosiol* **assert**

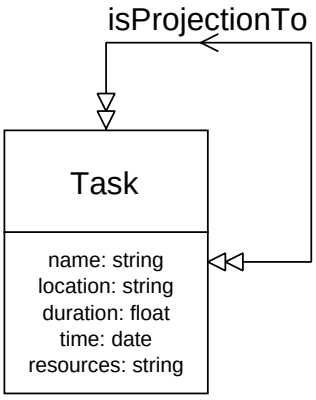
[TDPKosiol 1]:

$$(\forall\, t : V_{TaskDecomposition} \bullet t.criterion = process) \;\; xor \;\; (\forall\, t : V_{TaskDecomposition} \bullet t.criterion = object)$$

end.

Das Metaschema zur Beschreibung der Aufgabengliederungspläne nach Kosiol (*TDPKosiol*) ist ausschließlich auf Verrichtungs- oder Objektgliederungen einzuschränken [TDPKosiol 1].

Aufgabengliederungen nach Jordt/Gscheidle. Wie auch Nordsieck vertreten [Jordt / Gscheidle, (o.J.)] die Ausgrenzung der Verwaltungsaufgaben [TDPJordt/Gscheidle 1], [TDPJordt/Gscheidle 2]. Während bei Nordsieck keinerlei Bezüge zwischen den Nicht-Verwaltungsaufgaben und den Verwaltungsaufgaben beschrieben werden, führen [Jordt / Gscheidle, (o.J.), Lehrbrief 2, S. 37] das Konzept der *Projektion* ein, durch das Verwaltungsaufgaben denjenigen Ausführungsaufgaben zugeordnet werden können, deren Erledigung mittelbar von der Bearbeitung der Verwaltungsaufgaben abhängt.



TDPJordtGscheidle **isA** *TDP* ;

for *G* **in** *TDPJordtGscheidle* **assert**

[TDPJordtGscheidle 1]:

$$\#\, E_{isAdminstrativeTask} \leq 1 \;;$$

[TDPJordtGscheidle 2]:

$$\forall\, e : E_{isAdminstrativeTask} \bullet \omega(e)\; {}^{\leftarrow}_{isDecomposedIn}\; mainTask \;;$$

[TDPJordtGscheidle 3]:

$$\forall\, t, a : V_{Task} \bullet t\; {}^{\leftarrow}_{isProjectionTo}\; a \Rightarrow t\; ({}^{\rightarrow}_{isExecutionTask}{}^{\leftarrow}_{isDecompositionIn})^{+}\; mainTask\; {}^{\rightarrow}_{isDecompositionIn}{}^{\leftarrow}_{isAdminstrativeTask}\; ({}^{\rightarrow}_{isDecompositionIn}{}^{\leftarrow}_{isSubtask})^{*}\; a \;;$$

[TDPJordtGscheidle 4]:

$$\forall\, v : V_{TaskDecomposition} \bullet \delta_{{}^{\leftarrow}_{isSubtask}}(v) \leq 5$$

end.

Abbildung 6.2: Spezialisierung des Referenz-Metaschemas des Aufgabengliederungsparadigmas nach [Jordt / Gscheidle, (o.J.)] *TDPJordtGscheidle*

Das Referenz-Metaschema des Aufgabengliederungsparadigmas ist zur Abbildung der Anforderungen von [Jordt / Gscheidle, (o.J.)] um einen weiteren Kantentyp *isProjectionTo* zwischen *Tasks* zu ergänzen. Diese Kanten verlaufen ausschließlich zwischen Knoten des Teilbaums der Verwaltungsaufgaben und Knoten des Teilbaums der Nicht-Verwaltungsaufgaben zur Gesamtaufgabe *mainTask* [TDPJordtGscheidle 3]. Die entsprechende Anpassung des Referenz-Metaschemas zum Metaschema *TDPJordtGscheidle* ist in Abbildung 6.2 dargestellt.

[Jordt / Gscheidle, (o.J.), Lehrbrief 2, S. 29] fordern aus Gründen der Übersichtlichkeit weiter, daß eine Aufgabe nicht in mehr als fünf Teilaufgaben untergliedert wird. Diese Forderung [TDP-JordtGscheidle 4] ist ebenfalls im *GRAL*-Teil von Abbildung 6.2 aufgeführt.

Funktionsbäume. Zur Beschreibung von Aufgabengliederungen nach dem Ansatz der Architektur integrierter Informationssysteme (ARIS) [Scheer, 1992, S. 64ff], [Scheer, 1994, S. 20] werden Funktionsbäume verwendet. Das Metaschema zur Abbildung der ARIS-Funktionsbäume (*TDPAris*) kann ebenfalls auf das Metaschema des Aufgabengliederungsparadigmas zurückgeführt werden.

ARIS-Funktionsbäume stellen nur die Gliederung von Aufgaben in Teilaufgaben dar. Eine explizite Unterscheidung von Ausführungs-, Leitungs-, Planungs-, Kontroll- und Verwaltungsaufgaben wird hier nicht modelliert [ARIS-TDP 1]. Der Kantentyp *isSubtask* ist im Gegensatz zum Referenz-Metaschema daher als konkret zu vereinbaren [ARIS-TDP 2]. Auch werden in Funktionsbäumen weder zusätzliche Aufgabeneigenschaften beschrieben noch wird die Art der Aufgabengliederung dargestellt, so daß die Attributstrukturen der Aufgaben- und Aufgabenzerlegungskonzepte unberücksichtigt bleiben [ARIS-TDP 3].

TDPAris **isA** *TDP* ;

for G **in** *ArisTDP* **assert**

[ARIS-TDP 1]:

$$E^{type}_{isSubtask} = E_{isSubtask} ;$$

[ARIS-TDP 2]:

overwrites [TDP-EER 1] :

$isConcrete(isSubtask)$;

[ARIS-TDP 3]:

$$\forall\, t : V_{Task} \bullet t.location = t.duration = t.time = t.resource = \perp ;$$

$$\forall\, d : V_{TaskDecomposition} \bullet d.criterion = d.connection = \perp$$

end.

6.3 Metaschemata der Aufbausicht

Die visuellen Modellierungssprachen der Aufbausicht dienen zur Darstellung der Strukturkomponenten von Organisationen und deren Zusammenhängen. Mit den Sprachen des Stellengliederungsparadigmas wird die Einbettung von *Organisationseinheiten* in die Organisationsstruktur dargestellt. Die Sprachen des Kommunikationsparadigmas modellieren die Informations- und Kommunikationsbeziehungen zwischen *Organisationseinheiten*.

6.3.1 Metaschemata des Stellengliederungsparadigmas

Die Zerlegung der Strukturkomponente einer Organisation in Stellen und Abteilungen ist der zentrale Beschreibungsgegenstand der Sprachen des Stellengliederungsparadigmas (vgl. Kapi-

tel 3.3.1). Im Metaschema des Stellengliederungsparadigmas werden hierzu die Konzepte zur Modellierung von Organisationseinheiten bereitgestellt, die in Stellen, Abteilungen und Stellenmehrheiten unterschieden werden können. Ebenso werden die verschiedenen, zwischen Organisationseinheiten vorliegenden, strukturellen und leitungsbezogenen Beziehungen im Metaschema abgebildet.

Referenz-Metaschema des Stellengliederungsparadigmas

Der *EER*-Teil des Referenz-Metaschemas des Stellengliederungsparadigmas (*PDP*) ist in Abbildung 6.3 dargestellt (vgl. [Winter / Ebert, 1996, S. 108]). Ein ähnliches Referenz-Metaschema, das zur Beschreibung der Aufbausicht vor dem Hintergrund der Beurteilung der Modellierungsmöglichkeiten von Workflow-Management-Systemen zur Darstellung aufbauorganisatorischer Aspekte entwickelt wurde, findet sich in [Rosemann / zur Mühlen, 1998]. Zur Beschreibung statischer Organisationsstrukturen wird auch in [Scheer, 1992, S. 92] ein starkt vereinfachtes Metamodell vorgestellt.

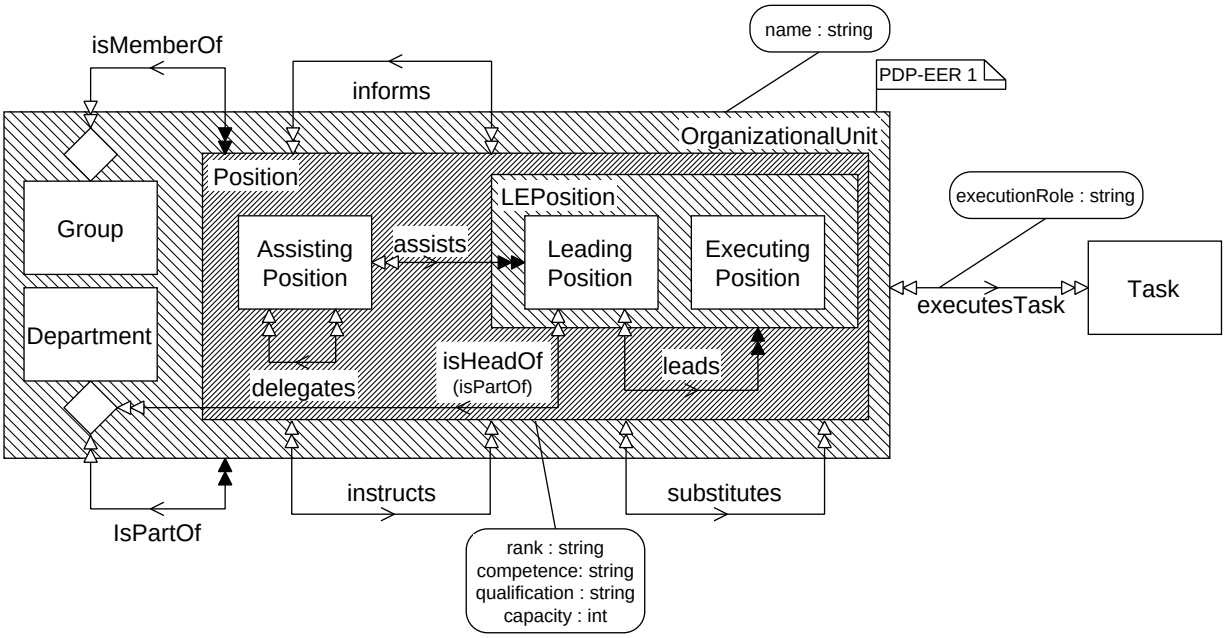


Abbildung 6.3: Referenz-Metaschema des Stellengliederungsparadigmas (Position Decomposition Paradigm, *PDP*)

Wesentliche Beschreibungskonzepte sind hier die Organisationseinheiten (*OrganizationalUnit*), die sowohl Stellen (*Position*) als auch Abteilungen (*Department*) und Stellenmehrheiten (*Group*) umfassen. Stellen sind durch ihren Rang in der Organisation, ihre Kompetenzen und die benötigte Qualifikation des Stelleninhabers charakterisiert. Das *capacity*-Attribut dient zur Modellierung mehrfach besetzter Stellen. Zur weiteren Unterscheidung werden Stellen in Leitungsstellen (*LeadingPosition*), Ausführungsstellen (*ExecutingPosition*) und Stabsstellen (*AssistingPosition*) unterschieden.

Zwischen Stellen vorliegende Beziehungen werden durch Kantentypen modelliert. Das Referenz-Schema sieht hierzu Informationsbeziehungen (*informs*), Vertretungsbeziehungen

(*substitutes*) und fachliche Weisungsbeziehungen (*instructs*) zwischen beliebigen Stellen sowie disziplinarische Weisungsbeziehungen (*leads*) von Leitungsstellen zu nachgeordneten Leitungsstellen bzw. Ausführungsstellen und Stabsbeziehungen (*assists*) zu Stabsstellen vor. Eine ebenfalls mögliche Substrukturierung der Stabsstellen erfolgt durch *delegates*-Kanten. Die disziplinarischen Weisungsbeziehungen sind zyklenfrei [PDP 1]. In Abteilungen (*Department*) sind beliebige Organisationseinheiten zusammengefaßt (*isPartOf*). Die Strukturierung der Abteilungen ist hierbei baumartig [PDP 2]. Abteilungen werden durch mindestens eine Leitungsstelle geleitet (*isHeadOf*). In Abteilungen sind nur solche Stellen zusammengefaßt, die dem Leiter dieser Abteilung (*isHeadOf*) auch disziplinarisch oder weisungsbezogen unterstellt sind [PDP 3]. Stellenmehrheiten (*Group*) fassen ausschließlich Stellen (*isMemberOf*) zusammen. Eine Leitungsstelle ist hier nicht vorgesehen.

for G **in** *PDP* **assert**

[PDP 1]:

$$isDag(eGraph(E_{leads}))\ ;$$

[PDP 2]:

$$isTree(eGraph(E^{-}_{isPartOf}))\ ;$$

[PDP 3]:

$$\forall\, d : V_{Department};\ h : V_{LeadingPosition} \mid h \rightharpoonup_{isHeadOf} d \bullet d \leftharpoonup^{+}_{isPartOf} \&_{Position} \subseteq h(\rightharpoonup_{leads} \mid \rightharpoonup_{instructs})^{*}$$

end.

Die Verbindung zwischen der Aufbausicht und der Aufgabengliederungssicht (vgl. Kapitel 6.2) wird über das Konzept der Aufgaben (*Task*) hergestellt. Jeder Organisationseinheit können hierzu die von ihr zu bearbeitenden Aufgaben zugeordnet (*executesTask*) werden. Die Modellierung der verschiedenen Arten der Aufgabenerledigung erfolgt durch entsprechende Attributierung der *executesTask*-Kanten durch die jeweiligen Funktionen.

Spezialisierungen des Referenz-Metaschemas

Konkrete Beschreibungsmittel des Stellengliederungsparadigmas stellen mit Ausnahme von textuellen Stellenbeschreibungen nur einen Ausschnitt der durch das Referenz-Metaschema abgebildeten Aspekte der Aufbauorganisation dar. Die folgenden Abschnitte skizzieren die Anwendung des Referenz-Metaschemas aus Abbildung 6.3 auf wichtige Vertreter der Sprachen des Stellengliederungsparadigmas.

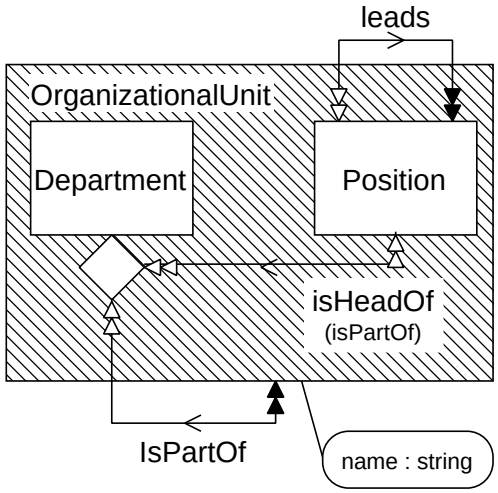
Organigramme für Einliniensysteme. In Organigrammen zu Einlinien-Systemen, die im Metaschema *DPDSingleLine* formalisiert sind, werden nur disziplinarische Weisungsbeziehungen zwischen Leitungsstellen und Ausführungsstellen modelliert. Erweiterungen zu Stab-Liniensystemen umfassen auch die Zuordnung von Stabsstellen und deren Hierarchie. Abteilungen, Gruppenzugehörigkeiten sowie die restlichen durch das Referenz-Metamodell unterstützten Beziehungstypen werden nicht abgebildet [PDPSingleLine 1]. Die in solchen Einliniensystemen

modellierten Leitungsbeziehungen folgen dem Prinzip der Einheit der Auftragserteilung und sind daher auf baumartige Strukturen [PDPSingleLine 2] eingeschränkt.

PDPSingleLine **isA** *PDP* ;
for G **in** *PDPSingleLine* **assert**
[PDPSingleLine 1]:
$V_{Group} = V_{Department} = \varnothing$;
$E_{informs} = E_{instructs} = E_{substitutes} = \varnothing$;

[PDPSingleLine 2]:
$isTree(eGraph(E_{leads} \cup E^{-}_{assists} \cup E_{delegates}))$
end.

Abteilungsorganigramme. Die Zerlegung einer Organisation in Abteilungen wird durch Abteilungsorganigramme (vgl. das Metaschema *PDPDepartment* in Abbildung 6.4) beschrieben. Mit diesen visuellen Sprachen wird die Untergliederung von Abteilungen in Unter-Abteilungen und die Zuordnung von Stellen zu einzelnen Abteilungen dargestellt. Häufig bilden diese Notationsformen auch die Weisungsbeziehungen innerhalb der Abteilungen ab, die Unterscheidung verschiedener Stellentypen und weiterer Beziehungen zwischen den Stellen wird in der Regel nicht modelliert [PDPDepartment 1]. Das Konzept *Position* ist daher als konkret zu vereinbaren [PDPDepartment 2]. Ebenso werden Leitungs-, Weisungs-, Stellvertretungs- und Informationsbeziehungen sowie Gruppenzuteilungen nicht dargestellt [PDPDepartment 3]. In Varianten von Abteilungsorganigrammen werden die Stellen der Abteilungsleiter gegenüber den nachgeordneten Stellen optisch hervorgehoben. Zur Identifizierung dieser Leitungsstellen können die *isHeadOf*-Kanten des Referenz-Metaschemas herangezogen werden.



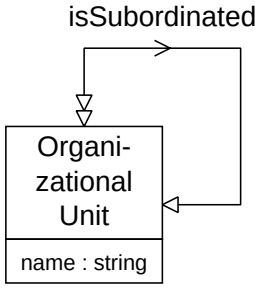
PDPDepartment **isA** *PDP* ;
for G **in** *PDPDepartment* **assert**
[PDPDepartment 1]:
$V_{Position} = V^{type}_{Position}$;
[PDPDepartment 2]:
overwrites [PDP-EER 2] :
$isConcrete(Position)$;
[PDPDepartment 3]:
$V_{Group} = \varnothing$;
$E_{assists} = E_{delegates} = E_{instructs} = E_{informs} = E_{substitutes} = \varnothing$
end.

Abbildung 6.4: Spezialisierung des Referenz-Metaschemas des Stellengliederungsparadigmas für Abteilungsorganigramme (*PDPDepartment*)

In anderen Varianten der Abteilungsorganigramme (vgl. Abbildung 3.5e) wird auch häufig auf die Modellierung der Leitungsbeziehungen komplett verzichtet. Modelliert wird dann lediglich

das hierarchische Gefüge von Stellen und Abteilungen. Gegenüber dem Metaschema *PDPDepartment* sind dann auch noch die *leads*-Kanten auszuschließen.

ARIS-Organigramme. Organigramme des Modellierungsansatzes der Architektur integrierter Informationssysteme (ARIS) [Scheer, 1994, S. 23ff] stellen beliebige Organisationseinheiten zueinander in Beziehung. Eine Unterscheidung dieser Organisationseinheiten in Abteilungen, Gruppen, Leitungsstellen, Ausführungsstellen oder Stabsstellen wird hier ebenso wenig abgebildet, wie unterschiedliche Arten struktureller Beziehungen zwischen den einzelnen Organisationseinheiten [ArisPDP 1]. Das Konzept zur Abbildung der Organisationseinheiten ist daher auch als konkret zu vereinbaren [ArisPDP 2]. ARIS-Organigramme beschreiben lediglich baumartige Unterstellungsbeziehungen zwischen einzelnen Organisationseinheiten, die im Metaschema durch den zusätzlichen Kantentyp *isSubordinated* abgebildet werden [ArisPDP 3]. Vergleiche hierzu auch den Ausschnitt des ARIS-Metaschemas des Fachkonzepts Organisation in [Scheer, 1992, S. 92].



ArisPDP **isA** *PDP* ;
for *G* **in** *ArisPDP* **assert**
[ARIS-PDP 1]:
$V_{AssistingPosition} = V_{LEPosition} = \varnothing$;
$E_{informs} = V_{instructs} = V_{substitutes} = V_{delegates} = V_{assists} = V_{leads} = \varnothing$;
[ARIS-PDP 2]:
overwrites [PDP-EER 1] :
$isConcrete(OrganizationalUnit)$;
[ARIS-PDP 3]:
$isTree(eGraph(E^{-}_{isSubordinted}))$
end.

Abbildung 6.5: Spezialisierung des Referenz-Metaschemas des Stellengliederungsparadigmas für ARIS-Organigramme (*PDPAris*)

Organigramme für Mehrliniensystem. Mehrliniensysteme folgen nicht dem Prinzip der Einheit der Auftragserteilung. Hier werden, wie für das Referenz-Metaschema des Stellengliederungsparadigmas gefordert, zyklenfreie Weisungs- und Leitungsbeziehungen dargestellt. Neben den disziplinarischen Weisungsbeziehungen sind im Gegensatz zu Einliniensystemen auch beliebige fachliche Weisungsbeziehungen zu beachten. Die Einschränkung gegenüber dem Referenz-Metaschema des Stellengliederungsparadigmas im Metaschema *PDPMultiLine* bezieht sich folglich nur auf den Ausschluß der Konzepte *Group* und *Department* sowie der Beziehungskonzepte *informs* und *substitutes* [PDPMultiLine 1].

PDPMultiLine **isA** *PDP* ;
for *G* **in** *PDPMultiLine* **assert**
[PDPMultiLine 1]:
$V_{Group} = V_{Department} = \varnothing$;
$E_{informs} = E_{substitutes} = \varnothing$
end.

Funktionendiagramme. Funktionendiagramme beschreiben die Zuordnung von Aufgaben zu Stellen. Die Darstellung der Stellengliederung erfolgt hierbei durch die Darstellungsmittel der Einliniensysteme (s.o.). Zur Darstellung der Aufgabengliederung ist in analoger Form auf das Referenz-Metaschema des Aufgabengliederungsparadigmas (vgl. Kapitel 6.2.1) zurückzugreifen. Wesentlicher Modellierungsaspekt der Funktionendiagramme ist die Rolle, in der eine Stelle eine Aufgabe bearbeitet. Diese Funktionen sind am *executionRole*-Attribut der entsprechenden *executesTask*-Kanten abzulesen.

6.3.2 Metaschemata des Kommunikationsparadigmas

Mit den Beschreibungsmitteln des Kommunikationsparadigmas (vgl. Kapitel 3.3.2) werden die Interaktionen innerhalb einer Organisation und mit externen Partnern modelliert. Das Referenz-Metaschema des Kommunikationsparadigmas bildet hierzu die Konzepte zur Modellierung der Kommunikationspartner, zur quantitativen Beschreibung der Kommunikationsbeziehungen und zur Darstellung der hierzu eingesetzten Kommunikationsmittel ab.

Referenz-Metaschema des Kommunikationsparadigmas

Wie auch das Referenz-Metaschema des Stellengliederungsparadigmas (vgl. Kapitel 6.3.1) werden im Referenz-Metaschema des Kommunikationsparadigmas (*CommP*) die Organisationseinheiten Stellen (*Position*), Abteilungen (*Department*) und Stellenmehrheiten (*Group*) im Konzept *OrganizationalUnit* zusammengefaßt. Kommunikationen (*Communication*) zwischen organisatorischen Einheiten und externen Partnern (*ExternalPartner*) werden durch den Beziehungstyp *communicates* modelliert. Die durch *communicates*-Kanten modellierte Kommunikation mehrerer Partner ist ungerichtet. An einer Kommunikation sind mindestens zwei Kommunikationspartner beteiligt. Aufgrund der Injektivität von *communicates* sind diese Kommunikationspartner verschieden.

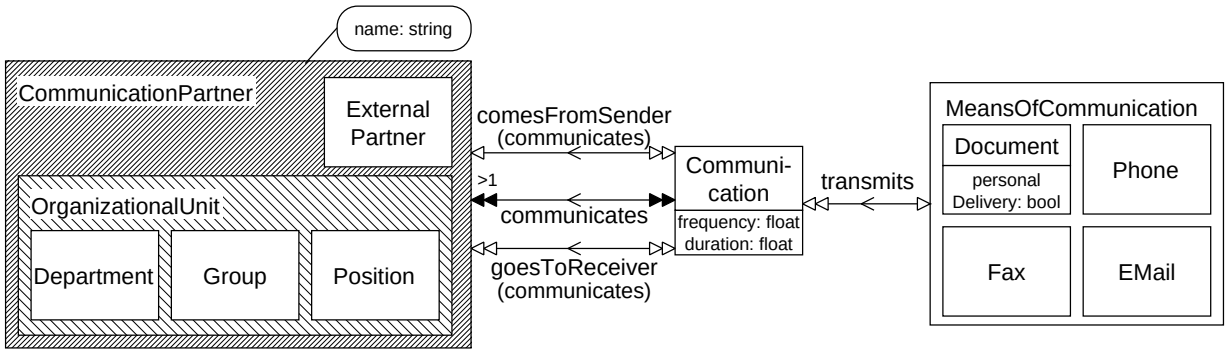


Abbildung 6.6: Referenz-Metaschema des Kommunikationsparadigmas (Communication Paradigm, *CommP*)

Zur Darstellung gerichteter Kommunikationsbeziehungen werden die Beziehungstypen *comesFromSender* und *goesToReceiver* als Spezialisierungen des Kantentyps *communicates* eingeführt, die den an den Interaktionen Beteiligten Sender- und Empfängerrollen zuweisen. Gerichte-

te Kommunikationsbeziehungen werden ausschließlich durch *comesFromSender*- und *goesToReceiver*-Kanten modelliert [CommP 1].

for G **in** *CommP* **assert**

[CommP 1]:

$$\forall\, c : V_{Communication} \mid \delta_{\rightharpoonup_{comesFromSender}}(c) = 1 \vee \delta_{\rightharpoonup_{goesToReceiver}}(c) = 1 \bullet \delta_{\rightharpoonup^{type}_{communicates}}(c) = 0$$

end.

Quantitative Kommunikationseigenschaften wie Kommunikationshäufigkeit und -dauer werden in den Attributen der *Communication*-Knoten modelliert. Jeder Kommunikation kann darüber hinaus auch das hierzu eingesetzte Kommunikationsmittel (*MeansOfCommunication*) zugeordnet werden. Hierbei werden Kommunikationen durch Telefon (*Phone*), Fax (*Fax*), E-Mail (*EMail*) und durch Dokumentenaustausch (*Document*) unterschieden.

Spezialisierungen des Referenz-Metaschemas

Auch die konkreten Darstellungsmittel des Kommunikationsparadigmas heben einzelne Modellierungsaspekte des Referenz-Metaschemas heraus und verzichten auf die Darstellung anderer.

Kommunigramme. Das Metaschema der Kommunigramme (*CommPChart*) basiert auf dem Referenz-Metaschema des Kommunikationsparadigmas. Kommunigramme beschreiben das Vorliegen von Kommunikationsbeziehungen zwischen jeweils genau zwei Partnern [CommPChart 1] sowie die Häufigkeit oder die Dauer dieser Interaktionen. Zur Beschreibung der quantitativen Kommunikationsaspekte sind die *frequency*- oder *duration*-Attributierungen der *Communication*-Knoten zu visualisieren. In Kommunigramm-Varianten wie z. B. in den Abbildungen 3.10, 3.11 und 3.12 wird höchstens eine Interaktionsbeziehung zwischen zwei Kommunikationspartnern dargestellt [CommPChart 2]. Die zur Kommunikation genutzten Hilfsmittel werden in Kommunigrammen nicht abgebildet [CommPChart 3]. Die Kommunikationsrichtung zwischen zwei Partnern kann anhand der *comesFromSender*- und *goesToReceiver*-Kanten abgelesen werden.

CommPChart **isA** *CommP* ;

for G **in** *CommPChart* **assert**

[CommPChart 1]:

$$\forall\, c : V_{Communication} \bullet \delta_{\rightharpoonup_{communicates}}(c) = 2\ ;$$

[CommPChart 2]:

$$\forall\, v, w : V_{CommunicationPartner} \bullet \#\{c : V_{Communication} \mid v \leftharpoonup_{comesFromSender} c \rightharpoonup_{goesToReceiver} w\} \leq 1\ ;$$

[CommPChart 3]:

$$V_{MeansOfCommunication} = \varnothing$$

end.

Kooperationsbilder. Wie auch in Kommunikationsdiagrammen werden durch Kooperationsbilder (vgl. das Metaschema *CommPPic*) Kommunikationsbeziehungen zwischen genau zwei Kommunikationspartnern beschrieben [CommPPic 1]. Diese Kommunikationsbeziehungen sind generell gerichtet [CommPPic 2]. Kooperationsbilder beschreiben darüber hinaus die zur Interaktion verwendeten Hilfsmittel. Im Gegensatz zu der Darstellung durch Kommunigramme, bei denen höchstens eine Kommunikationsbeziehung zwischen zwei Partnern dargestellt wird, können hier beliebig viele Interaktionen modelliert werden, die jedoch jeweils unterschiedliche Medien verwenden [CommPPic 3].

CommPPic **isA** *CommP* ;

for G **in** *CommPPic* **assert**

[CommPPic 1]:

$$\forall\, c : V_{Communication} \bullet \delta_{\rightharpoonup_{communicates}}(c) = 2 ;$$

[CommPPic 2]:

$$V^{type}_{communicates} = \varnothing ;$$

[CommPPic 3]:

$$\forall\, c_1, c_2 : V_{Communication};\ m_1, m_2 : V_{MeansOfCommunication} \mid c_1 \neq c_2 \wedge c_1 \rightharpoonup_{comesFromSender} \leftharpoonup_{comesFromSender} c_2 \wedge c_1 \rightharpoonup_{goesToReceiver} \leftharpoonup_{goesToReceiver} c_2 \wedge m_1 \rightharpoonup_{transmits} c_1 \wedge m_2 \rightharpoonup_{transmits} c_2 \bullet type(m_1) \neq type(m_2)$$

end .

6.4 Metaschemata der Prozeßsicht

In den Metaschemata der Ablaufsicht sind die visuellen Sprachen zur Darstellung *logischer und zeitlicher* Aspekte der Aufgabenbearbeitung zusammengefaßt. Die Metaschemata der vier Beschreibungsparadigmen der Ablaufsicht werden in den folgenden Kapiteln eingeführt. Das nach außen sichtbare Verhalten eines Systems wird mit den Sprachen des Datenflußparadigmas beschrieben. Die Sprachen des Zustandsübergangsparadigmas beschreiben das dynamische Systemverhalten aus Sicht der Veränderung von Systemzuständen. Im Netzparadigma sind die Beschreibungsmittel zusammengefaßt, die das Systemverhalten entlang der Folgebeziehungen zwischen Prozessen und/oder Ereignissen darstellen. Regulär strukturierte Kontrollflüsse dienen zur Darstellung von Prozeßabläufen im Kontrollflußparadigma.

6.4.1 Metaschemata des Datenflußparadigmas

Zur Beschreibung des nach außen sichtbaren Systemverhaltens mit den Sprachen des Datenflußparadigmas wird das System als ein Netz funktionaler Komponenten aufgefaßt, die durch den Austausch von Daten oder Gegenständen miteinander interagieren (vgl. Kapitel 3.4.1). Das Metaschema des Datenflußparadigmas stellt daher Konzepte zur Modellierung dieser funktionalen Komponenten (Prozesse) und deren Interaktionsbeziehungen (Datenflüsse) bereit. Datenflußmodellierungen sind ein wesentliches Hilfmittel bei der funktionalen Zerlegung eines Systems

in Teilsysteme. Hierzu bildet das Metaschema entsprechende Konzepte zur Modellierung von Prozeßverfeinerung ab. Diverse Beschreibungsmittel des Datenflußparadigmas verwenden zur Konkretisierung der Modellierungen neben den Prozessen und Datenflüssen zusätzliche Modellierungskonstrukte, die im Referenz-Metaschema verallgemeinert abgebildet sind.

Referenz-Metaschema des Datenflußparadigmas

Abbildung 6.7 enthält den *EER*-Teil des Referenz-Metaschemas des Datenflußparadigmas (*DFP*) (vgl. auch [EIA/IS-115, 1995], [Drüke, 1996], [Winter / Ebert, 1996, S. 115], [Ebert et al., 1996b]). Die funktionalen Systemkomponenten werden durch das Konzept *Process* abgebildet. Interaktionsbeziehungen werden im Konzept *Dataflow* beschrieben. Solche Datenflüsse verbinden jeweils zwei Datenflußobjekte (*DfObject*) miteinander, die entweder Prozesse (*Process*), Speicher (*Store*), Schnittstellen zur Systemumwelt (*Terminator*) oder Kontaktpunkte in Verfeinerungen (*PointOfContact*) modellieren. Nicht zugelassen sind hierbei Datenflüsse ausschließlich zwischen Speichern, zwischen Schnittstellen oder zwischen Datenflußsurrogaten in Verfeinerungen (*PointOfContact*) [DFP 1]. Datenflußbeziehungen sind binär modelliert, so daß jeweils Quelle (*dfComesFrom*) und Ziel (*dfGoesTo*) eines Datenflusses unterschieden werden kann.

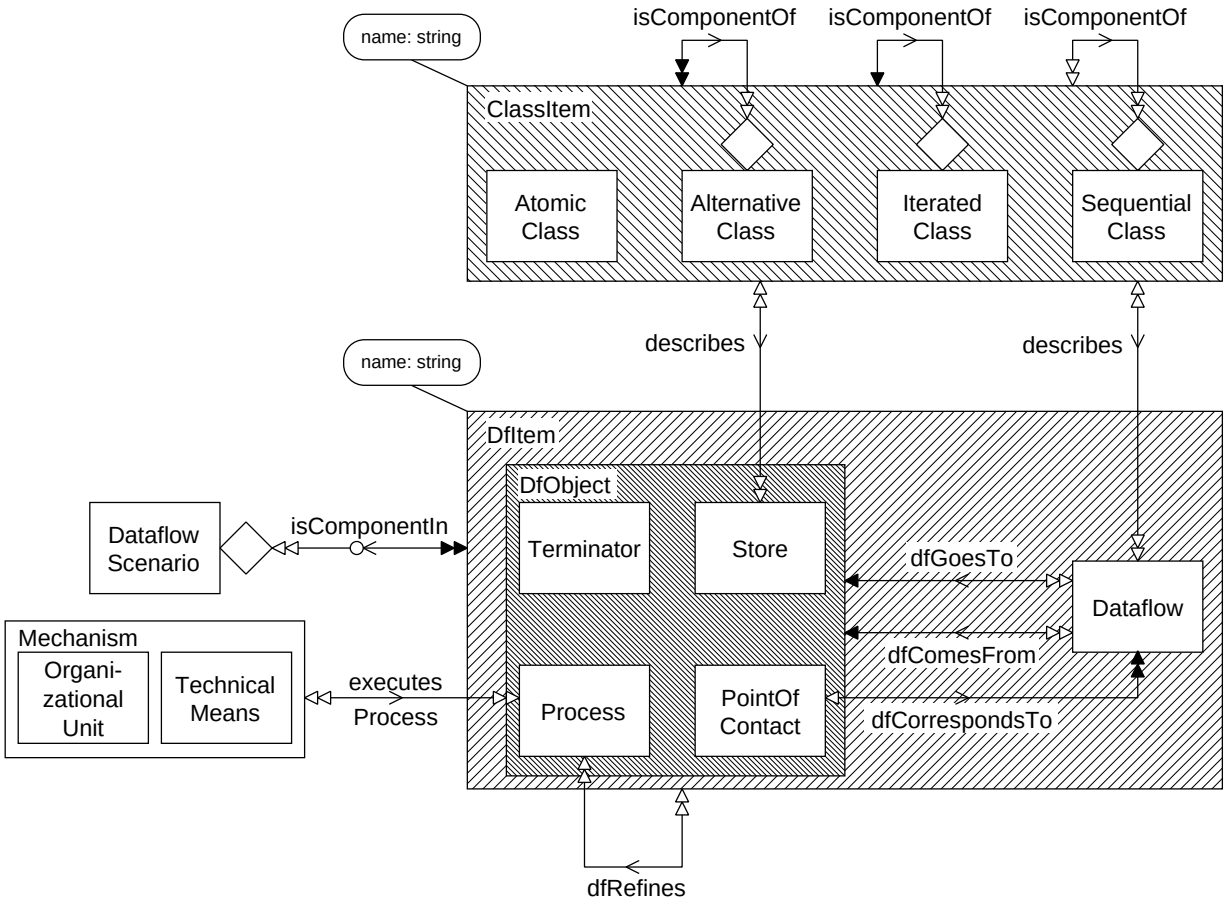


Abbildung 6.7: Referenz-Metaschema des Datenflußparadigmas (Data Flow Paradigm, *DFP*)

Sowohl Datenflüsse als auch Speicher können durch die visuelle Modellierungsprachen des Objekt-Beziehungsparadigmas (Kapitel 3.5.3) näher konkretisiert werden. Der Kantentyp *describes* stellt hierzu die Verbindung zwischen dem Metaschemata des Datenflußparadigmas und des Objekt-Beziehungsparadigmas (vgl. Kapitel 6.5.3) her. In Datenflußmodellierungen erfolgt die Konkretisierung i. allg. durch regulär strukturierte Objektgeflechte, die mittels Datenlexika (vgl. auch Kapitel 6.5.3) beschrieben werden. Diese Objektstrukturen werden im Metaschema des Datenflußparadigmas durch das Konzept *ClassItem* abgebildet, das in Unterkonzepte zur Beschreibung atomarer Objektklassen und durch Alternativen, Iterationen oder Sequenzen konstruierte Objektklassen spezialisiert ist.

In einigen Datenflußdialekten ([Ross, 1977], [Gane / Sarson, 1979]) werden Prozessen auch ihre menschlichen und technischen Handlungsträger zugeordnet. Dieses wird im Metaschema durch das Konzept *Mechanism* abgebildet, das durch seine Verfeinerung *OrganizationalUnit* auch den Querbezug zu den Metaschemata der Aufbausicht (vgl. Kapitel 6.3) herstellt. Die Metamodellierung der Sequenzialisierungen von Datenflußdiagrammen zur Beschreibung von Anwendungsszenarien erfolgt durch das Konzept *DataflowScenario*, das Konzepte der Datenflußmodellierung (*DfItem*) zu Szenarien zusammenfaßt.

Die Verfeinerung von Prozessen durch weitere Datenflußkonzepte (*DfItem*) wird durch den Kantentyp *dfRefines* modelliert. Prozesse sollten hierbei in drei bis sechs Unterprozesse untergliedert werden [DFP 2]. Das Metaschema des Datenflußparadigmas ermöglicht die mehrfache Verwendung von Prozeßverfeinerungen[1] (vgl. auch [EIA/IS-115, 1995], [Flatscher, 1998, S. 229]). Diese Verfeinerungsstruktur ist zyklenfrei [DFP 3]. Für die Mehrfachverwendung muß weiter gelten, daß die Elemente einer Datenflußbeschreibung (*DfItem*) die eine solche Verfeinerung bilden, auch in jeder Verfeinerung vollständig enthalten sind [DFP 4]. Datenflüsse sind ausschließlich zwischen solchen Datenflußobjekten zugelassen, die dieselben Prozesse verfeinern [DFP 5].

In Verfeinerungen dienen *PointOfContact*-Knoten als Surrogate der Datenflüsse, die in den verfeinerten Prozeß ein- bzw. aus ihm ausgehen. Dieser Bezug wird durch *dfCorrespondsTo*-Kanten hergestellt [DFP 6]. Verfeinerungen müssen strukturell balanciert sein, d. h. die Verfeinerung eines Prozesses besitzt nach außen dasselbe Verhalten wie der verfeinerte Prozeß. Zu jedem in einen verfeinerten Prozeß ein- oder aus ihm ausgehenden Datenfluß existiert in der Verfeinerung genau ein *PointOfContact*-Knoten als Start- bzw. Endpunkt dieser Datenflüsse in der Verfeinerung [DFP 7]. Diese Kontaktpunkte sind für alle Datenflüsse eines verfeinerten Prozesses verschieden [DFP 8]. Datenflüsse sind in Verfeinerungen nur zwischen Objekten dieser Verfeinerung zugelassen, Datenflüsse zwischen Verfeinerungen sind auszuschließen [DFP 9].

Ebenso muß sichergestellt sein, daß die Daten, die durch die Datenflüsse der Verfeinerung transportiert werden, mit den Daten der korrespondierenden Datenflüsse des verfeinerten Prozesses verträglich sind. In Zusicherung [DFP 10] wird dieses durch Rückgriff auf die Strukturierung der transportierten Daten spezifiziert. Analog ist auch zu fordern, daß zu Speichern adjazente Datenflüsse auch mit dem Speicherinhalt verträgliche Daten lesen bzw. schreiben [DFP 11].

[1] Vielfach wird in Datenflußdialekten die Mehrfachverwendung ausgeschlossen und eine baumartige Verfeinerungsstruktur gefordert. Für das Referenz-Metaschema wird jedoch nur die allgemeinere Zyklenfreiheit verlangt.

for G **in** *DFP* **assert**

[DFP 1]:

$$\{s_1, s_2 : V_{Store} \mid s_1 \leftarrow_{dfComesFrom} \rightarrow_{dfGoesTo} s_2\} = \varnothing$$
$$\{t_1, t_2 : V_{Terminator} \mid t_1 \leftarrow_{dfComesFrom} \rightarrow_{dfGoesTo} t_2\} = \varnothing$$
$$\{p_1, p_2 : V_{PointOfContact} \mid p_1 \leftarrow_{dfComesFrom} \rightarrow_{dfGoesTo} p_2\} = \varnothing ;$$

[DFP 2]:

$$\forall p : V_{Process} \mid \delta_{\leftarrow_{dfRefines}}(p) \neq 0 \bullet 3 \leq \delta_{\leftarrow_{dfRefines}}(p) \leq 6 ;$$

[DFP 3]:

$$isDag(eGraph(E_{dfRefines})) ;$$

[DFP 4]:

$$\forall a, b : V_{DfItem} \mid a \rightarrow_{dfRefines} \leftarrow_{dfRefines} b \bullet$$
$$a \rightarrow_{dfRefines} = b \rightarrow_{dfRefines} ;$$

[DFP 5]:

$$\forall a, b : V_{DfObject} \mid a \leftarrow_{dfComesFrom} \rightarrow_{dfGoesTo} b \bullet$$
$$a \rightarrow_{dfRefines} = b \rightarrow_{dfRefines} ;$$

[DFP 6]:

$$\forall c : E_{dfCorrespondsTo} \bullet \exists p : V_{Process} \mid \delta_{\leftarrow_{dfRefines}}(p) > 0 \bullet$$
$$\omega(c) \ (\rightarrow_{dfGoesTo} | \rightarrow_{dfComesFrom}) \ p ;$$

[DFP 7]:

$$\forall d : V_{Dataflow};\ p : V_{Process} \mid \delta_{\leftarrow_{dfRefines}}(p) > 0 \wedge d \rightarrow_{dfGoesTo} p \bullet$$
$$\exists_1 poc : V_{PointOfContact} \mid d \leftarrow_{dfCorresponds} poc \rightarrow_{dfRefines} p \bullet$$
$$poc \leftarrow_{dfComesFrom} \rightarrow_{dfGoesTo} \rightarrow_{dfRefines} p ;$$

$$\forall d : V_{Dataflow};\ p : V_{Process} \mid \delta_{\leftarrow_{dfRefines}}(p) > 0 \wedge d \rightarrow_{dfComesFrom} p \bullet$$
$$\exists_1 poc : V_{PointOfContact} \mid d \leftarrow_{dfCorresponds} poc \rightarrow_{dfRefines} p \bullet$$
$$poc \leftarrow_{dfGoesTo} \rightarrow_{dfComesFrom} \rightarrow_{dfRefines} p ;$$

[DFP 8]:

$$\forall p : V_{Process};\ d_1, d_2 : V_{Dataflow};\ poc_1, poc_2 : V_{PointOfContact} \mid$$
$$d_1 \neq d_2 \wedge$$
$$d_1 \ (\rightarrow_{dfGoesTo} | \rightarrow_{dfComesFrom}) \ p \leftarrow_{dfRefines} poc_1 \rightarrow_{dfCorrespondsTo} d_1 \wedge$$
$$d_2 \ (\rightarrow_{dfGoesTo} | \rightarrow_{dfComesFrom}) \ p \leftarrow_{dfRefines} poc_2 \rightarrow_{dfCorrespondsTo} d_2 \bullet$$
$$poc_1 \neq poc_2 ;$$

[DFP 9]:

$$\forall d : V_{Dataflow};\ p : V_{Process} \mid d \rightarrow_{dfRefines} p \bullet$$
$$d \ (\rightarrow_{dfGoesTo} | \rightarrow_{dfComesFrom}) \rightarrow_{dfRefines} p ;$$

[DFP 10]:

$$\forall\, poc \,:\, V_{PointOfContact} \bullet poc \rightharpoonup_{dfCorrespondsTo} \leftharpoondown_{describes} \leftharpoondown^{*}_{isComponentOf} \rightharpoonup_{describes} (\rightharpoonup_{dfGoesTo} | \rightharpoonup_{dfComesFrom})\, poc\ ;$$

[DFP 11]:

$$\forall\, d \,:\, V_{Dataflow} \bullet d \rightharpoonup_{dfGoesTo} \rightharpoonup_{dfComesFrom} \&_{Store} \leftharpoondown_{describes} \leftharpoondown^{*}_{isComponentOf} \rightharpoonup_{describes} d$$

end.

Spezialisierungen des Referenz-Metaschemas

Die Metaschemata der verschiedenen Dialekte und Darstellungsvarianten des Datenflußparadigmas können ebenfalls aus dem Referenz-Metaschema abgeleitet werden.

Kontextdiagramme. Kontextdiagramme (vgl. das Metaschema *DFPContext*) beschreiben genau *einen* Prozeß in seiner Systemumgebung [DFPContext 1]. Dieser Prozeß beschreibt in der Regel die Wurzel einer Verfeinerungshierarchie. Aus dem Prozeß eines Kontextdiagramms gehen folglich keine *dfRefines*-Kanten aus [DFPContext 2], und das Kontextdiagramm enthält keine *PointOfContact*-Knoten [DFPContext 3]. Auch werden in Kontextdiagrammen keine Speicher (*Store*), keine Szenarien (*DataflowScenario*) und keine Mechanismen (*Mechanism*) modelliert [DFPContext 3].

DFPContext **isA** *DFP* ;

for *G* **in** *DFPContext* **assert**

[DFPContext 1]:

$$\#\, V_{Process} = 1\ ;$$

[DFPContext 2]:

$$\forall\, p \,:\, V_{Process} \bullet \delta_{\rightharpoonup_{dfRefines}}(p) = 0\ ;$$

[DFPContext 3]:

$$V_{PointOfContact} = V_{Store} = V_{DataflowScenario} = V_{Mechanism} = \varnothing$$

end.

Datenflußdiagramme. Datenflußdiagramme der strukturierten Analyse [DeMarco, 1978] entsprechen weitestgehend dem Referenz-Metaschema des Datenflußparadigmas. In diesen Datenflußdiagrammen (vgl. das Metaschema *DFPdeMarco*) werden lediglich keine Schnittstellen (*Terminator*) verwendet, die den Kontextdiagrammen vorbehalten sind. Auch werden keine Szenarien (*DataflowScenario*) und Mechanismen (*Mechanism*) abgebildet [DFPdeMarco 1]. Aufgrund der für Datenflußdiagramme geforderten Nummerierung [Yourdon, 1989, S. 165ff] sind Prozeßzerlegungen baumartig [DFPdeMarco 2].

DFPdeMarco **isA** *DFP* ;

for G **in** *DFPdeMarco* **assert**

[DFPdeMarco 1]:

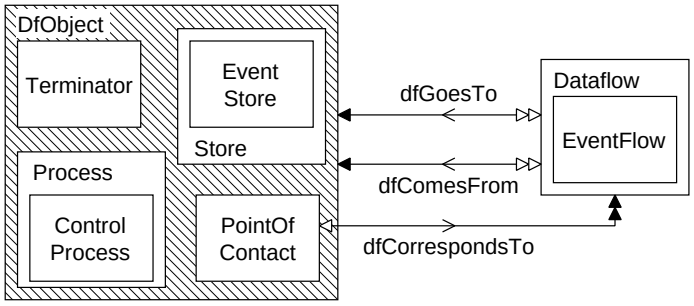
$$V_{Terminator} = V_{DataflowScenario} = V_{Mechanism} = \varnothing ;$$

[DFPdeMarco 2]:

$$isTree(eGraph(E_{dfRefines}))$$

end.

Echtzeit-Datenflußdiagramme. Die Echtzeit-Erweiterung der strukturierten Analyse [Ward / Mellor, 1985] ergänzen Datenflußdiagramme nach [DeMarco, 1978] um zusätzliche Kontrollelemente. Prozesse, Datenflüsse und Speicher sind durch Kontrollprozesse (*ControlProcess*), durch Ereignisflüsse (*EventFlow*) und durch Ereignisspeicher (*EventStore*) zu spezialisieren.



DFPRTSA **isA** *DFPdeMarco* ;

for G **in** *DFPRTSA* **assert**

[DFPRTSA 1]:

$$\forall\, d : V_{Dataflow} \mid d(\rightharpoonup_{dfGoesTo}|\rightharpoonup_{dfComesFrom}) \,\&_{ControlProcess \cup EventStore}\, \bullet\; type(d) = EventFlow ;$$

[DFPRTSA 2]:

$$\forall\, e : V_{EventFlow} \bullet \#(e(\rightharpoonup_{dfGoesTo}|\rightharpoonup_{dfComesFrom}) \,\&_{ControlProcess \cup EventStore}) > 1$$

end.

Abbildung 6.8: Spezialisierung des Referenz-Metaschemas des Datenflußparadigmas für Echtzeit-Datenflußdiagramme (*DFPRTSA*)

Kontrollprozesse und Ereignisspeicher sind ausschließlich zu Ereignisflüssen inzident [DFPRTSA 1], und Ereignisflüsse sind zu mindestens einem *ControlProcess* oder *EventStore* inzident [DFPRTSA 2]. Das Metaschema der Echtzeit-Datenflußdiagramme (*DFPRTSA*) erweitert hierzu das Metaschema der Datenflußdiagramme *DFPdeMarco*.

Datenflußpläne. Datenflußpläne beschreiben neben den Datenflußbeziehungen zwischen den Prozessen auch Trägermedien für den Datenaustausch und unterscheiden verschiedene Arten der Speicherung und der Prozeßbearbeitung. Zur Abbildung dieser Erweiterungen ist das Referenz-Metaschema zum Metaschema *DFPFlowChart* zu erweitern. In Abbildung 6.9 sind diese Spezialisierungen der Konzepte *Store* und *Process* sowie die Ergänzung der Trägermedien (*MeansOfDataflow*) dargestellt.

In Datenflußplänen werden keine Szenarien und keine menschlichen oder technischen Handlungsträger, durch die Prozesse bearbeitet werden, modelliert [DFPFlowChart 1]. Auch kennen sie keine weitere Strukturierung der ausgetauschten Daten [DFPFlowChart 2].

DFPFlowChart **isA** *DFP* ;

for G **in** *DFPFlowChart* **assert**

[DFPFlowChart 1]:

$$V_{DataflowScenario} = V_{Mechanism} = \varnothing ;$$

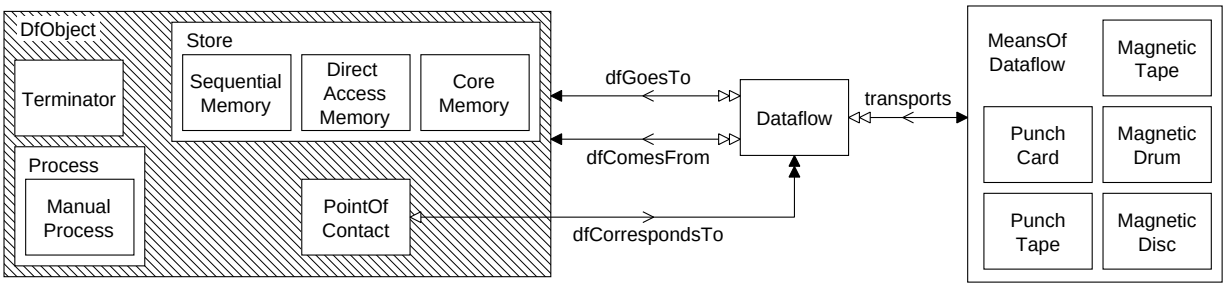
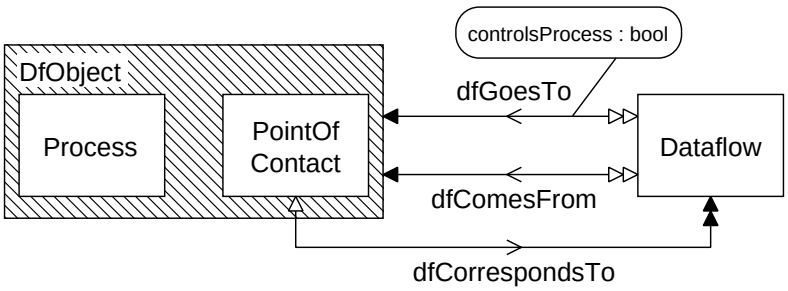


Abbildung 6.9: Spezialisierung des Referenz-Metaschemas des Datenflußparadigmas für Datenflußpläne (*DFPFlowChart*)

[DFPFlowChart 2]:
$$V_{ClassItem} = V_{AtomicClass}$$
end.

SADT-Diagramme. Auch das Metaschema der SADT-Aktivitätsdiagramme *DFPSADT* in Abbildung 6.10 erfordert nur eine geringe Anpassung des Referenz-Metaschemas des Datenflußparadigmas. SADT unterscheidet zwischen eingehenden Datenflüssen, deren Daten im Prozeß verändert werden, und solchen Datenflüssen, die die Bearbeitung im Prozeß steuern. Diese Unterscheidung wird durch ein zusätzliches Attribut (*controlsProcess*) der *dfGoesTo*-Kanten modelliert. In SADT-Aktivitätsdiagrammen werden keine Speicher (*Store*) und Schnittstellen (*Terminator*) verwendet. Das Referenz-Metaschema ist um diese Konzepte einzuschränken [DFPSADT 1]. Prozeßzerlegungen sind auch für SADT-Aktivitätsdiagramme baumartig [DFPSADT 2].

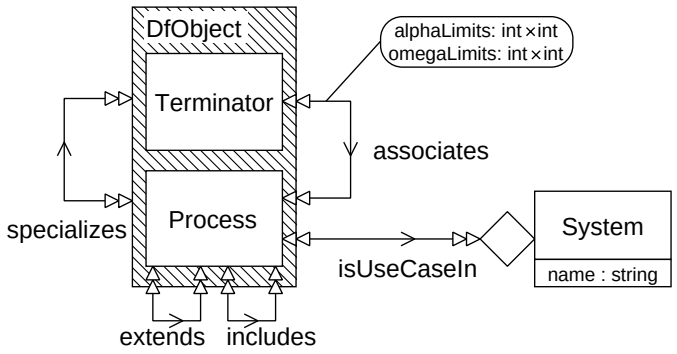


DFPSADT **isA** *DFP* ;
for *G* **in** *DFPSADT* **assert**
[DFPSADT 1]:
$$V_{Store} = V_{Terminator} = \varnothing ;$$
[DFPSADT 2]:
$$isTree(eGraph(E_{dfRefines}))$$
end.

Abbildung 6.10: Spezialisierung des Referenz-Metaschemas des Datenflußparadigmas für SADT-Aktivitätsdiagramme (*DFPSADT*)

Anwendungsfalldiagramme. Das Metaschema der Anwendungsfalldiagramme der Unified Modeling Language (UML) (*DFPUseCase*) (vgl. auch das Teil-Metaschema in [OMG, 1999, S. 2-112]) kann ebenfalls auf das Referenz-Metaschema des Datenflußparadigmas zurückgeführt werden. Durch Anwendungsfalldiagramme werden Systeme entlang ihrer Interaktionsbeziehungen zwischen Anwendungsfällen (Use Cases) und Akteuren beschrieben. Das System (*System*) selbst ist durch die hierin zusammengefaßten Anwendungsfälle bestimmt. Im Metaschema werden Anwendungsfälle durch *Process*- und Akteure durch *Terminator*-Konzepte abgebildet. Das

Metaschema der Anwendungsfalldiagramme aus Abbildung 6.11 enthält keine Konzepte zur Darstellung von Datenspeichern, Kontaktpunkten, Szenarien und Mechanismen [DFPUseCase 1]. Interaktionsdiagramme beschreiben nur das Vorliegen einer Interaktionsbeziehung. Über die ausgetauschten Daten werden keine Aussagen getroffen, so daß auch keine *dataflow*-Knoten benötigt werden [DFPUseCase 1]. Prozeßverfeinerungen werden in Anwendungfalldiagrammen nicht abgebildet [DFPUseCase 2]. Konkretisierungen von Prozessen werden in der UML z. B. durch Aktivitätsdiagramme notiert.



DFPUseCase **isA** *DFP* ;

for *G* **in** *DFDRTSA* **assert**

[DFPUseCase 1]:
$V_{Store} = V_{PointofContact} = V_{Dataflow} = V_{DataflowScenario} = V_{Mechanism} = \varnothing$;

[DFPUseCase 2]:
$E_{dfRefines} = \varnothing$;

[DFPUseCase 3]:
$\forall\, e : E_{specializes} \bullet type(\alpha(e)) = type(\omega(e))$

end.

Abbildung 6.11: Spezialisierung des Referenz-Metaschemas des Datenflußparadigmas für Anwendungsfalldiagramme (*DFPUseCase*)

Interaktionsbeziehungen (Assoziationen) werden durch *associates*-Kanten beschrieben, die auch um Kardinalitätsangaben ergänzt werden können. Die Anwendungsfälle eines Systems sind voneinander unabhängig, so daß Assoziationen nur zwischen Anwendungsfällen und Akteure zugelassen sind [Rumbaugh et al., 1999, S. 489]. Die Koordination der Anwendungsfälle eines Systems erfolgt durch die Akteure.

Sowohl Anwendungsfälle als auch Akteure können spezialisiert werden (*specializes*). Generalisierungsbeziehungen beziehen sich jeweils ausschließlich auf Anwendungsfälle (*Process*) oder Akteure (*Terminator*) [DFPUseCase 2]. Anwendungsfälle können darüber hinaus andere Anwendungsfälle erweitern (*extends*) und einbinden (*includes*).

6.4.2 Metaschemata des Zustandsübergangsparadigmas

Mit den visuellen Modellierungssprachen des Zustandsübergangsparadigmas (vgl. Kapitel 3.4.2) werden reaktive Systeme durch Zustände und Übergänge zwischen diesen modelliert. Zustandsübergänge werden durch das Eintreffen von externen oder internen Ereignissen ausgelöst. Sowohl den Zustandsübergängen als auch den Zuständen können Systemreaktionen oder Aktionen zugeordnet werden, die beim Übergang bzw. in den Zuständen bearbeitet werden. Das Referenz-Metaschema des Zustandsübergangsparadigmas bildet hierzu die Konzepte zur Beschreibung und Strukturierung von Zuständen, zur Darstellung der Zustandsübergänge und zur Modellierung der Ereignisse und Systemreaktionen ab.

Referenz-Metaschema des Zustandsübergangsparadigmas

Der *EER*-Teil des Referenz-Metaschemas des Zustandsübergangsparadigmas *STP* ist in Abbildung 6.12 dargestellt (vgl. auch [Ebert, 1993], [Winter / Ebert, 1996, S. 114]).

Zur Strukturierung des Systemzustands wird dieser in den modernen Beschreibungsmitteln des Zustandsübergangsparadigmas durch elementare und zusammengesetzte Teilzustände (*Blobs*) modelliert. *Blobs* unterscheiden sich in atomare (*AtomicBlobs*) in *XorBlobs*, zur Verfeinerung von Teilzuständen, und in *AndBlobs*, zur Beschreibung nebenläufiger Prozesse. Hierbei enthalten *XorBlobs* weitere beliebige *Blobs*, von denen einer durch eine *startsWith*-Kante als Startblob ausgezeichnet sein kann. *AndBlobs* enthalten mindestens zwei voneinander unabhängige *XorBlobs*. *XorBlobs* können zusätzlich um einen „History"-Mechanismus (Attribut *history*) ergänzt werden. Hierdurch wird angezeigt, daß der zuletzt besuchte Blob beim Wiederbetreten in den *XorBlob* als Startblob verwendet wird. Atomare Blobs sind zur Beschreibung solcher Teilzustände, aus denen keine weiteren Übergänge möglich sind, durch *EndBlobs* spezialisiert [STP 1]. Die Strukturierung der Blobs entlang der *contains*-Kanten ist baumartig [STP 2]. Die Wurzel dieses Baumes spiegelt den Zustand des gesamten Systems wider. Dieser wird durch einen *XorBlob* modelliert [STP 3], zu dem keine Transitionen inzident sind [STP 4] und der genau einen als Startblob ausgezeichneten *Blob* enthält [STP 5].

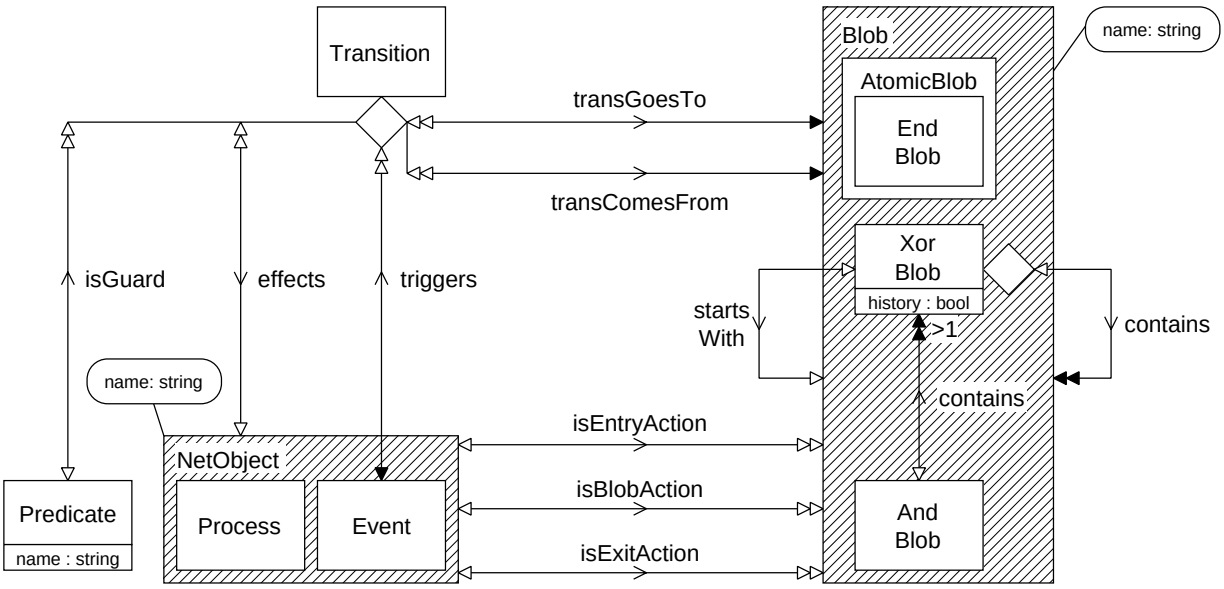


Abbildung 6.12: Referenz-Metaschema des Zustandsübergangsparadigmas, State Transition Paradigm, *STP*)

Übergänge zwischen zwei *Blobs* werden durch *Transition*-Knoten modelliert. Die Richtung dieser Übergänge ist anhand der *transComesFrom* und *transGoesTo*-Kanten ablesbar. Übergänge werden durch Ereignisse (*Event*) ausgelöst. Durch optionale Wächter (*isGuard*) können diese Übergänge auch noch von zusätzlichen Prädikaten (*Predicate*) abhängig gemacht werden.

Zustandsübergänge können Reaktionen des Systems bewirken, die im Konzept *NetObject* zusammengefaßt sind. Diese Systemreaktionen werden in Systemprozesse (*Process*) und Ereignisse (*Event*) unterschieden, die weitere Übergänge auslösen (Broadcast) können. Systemreaktionen

können aber auch den einzelnen Blobs zugeordnet werden. Diese werden bei Betreten des Blobs (*isEntryAction*), während das System in diesem Blob verweilt (*isBlobAction*) oder beim Verlassen des Blobs (*isExitAction*) ausgeführt.

Übergänge sind sowohl in die *Blobs* eines *XorBlobs* als auch in den *XorBlob* als Gesamtheit möglich. Geht mindestens ein Übergang in den *XorBlob* als Gesamtheit ein, ist festzulegen, welcher Startzustand durch den *XorBlob* eingenommen wird (vgl. auch [Ebert / Süttenbach, 1997a, DM7], [Süttenbach / Ebert, 1997, STD6]). Ebenso ist zu jedem *XorBlob* eines *AndBlobs* ein Startblob anzugeben [STP 6]. Solche Startblobs müssen jeweils im entsprechenden *XorBlob* enthalten sein [STP 7]. *AndBlobs* enthalten mindestens zwei voneinander unabhängige *XorBlobs*. Zwischen diesen verlaufen daher auch keine Übergänge [STP 8] [Ebert, 1993, constraint 1].

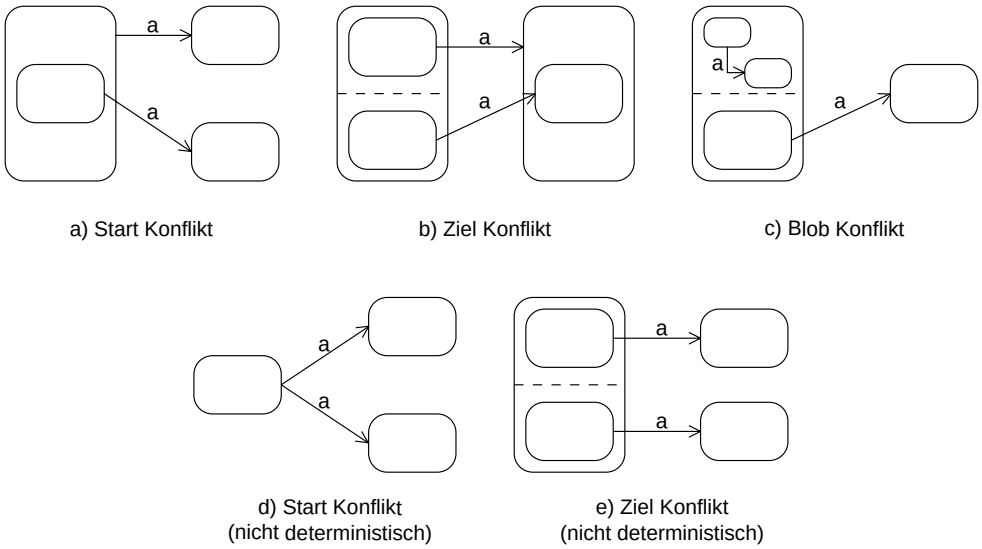


Abbildung 6.13: Konfligierende Übergänge

Ereignisse lösen Übergänge zwischen *Blobs* aus. Hierbei muß deterministisch entschieden werden können, in welchem Folgezustand sich das System anschließend befindet. Mehrere Übergänge können hierbei aufgrund gleicher Ereignisse zu Konflikten führen. Diese Konflikte werden bereits in [Harel, 1987] durch Verbot entsprechender Modellierungen ausgeschlossen. [Ebert, 1993] schlägt vor, in einigen dieser Konfliktfälle (vgl. Abbildung 6.13a–c) jeweils den innersten Übergang schalten zu lassen. Auszuschließen sind dann nur noch wenige Modellierungen wie sie in Abbildung 6.13d–e exemplarisch dargestellt sind.

In Anlehnung an [Ebert, 1993, constraint 2] wird in Zusicherung [STP 9] für das Referenz-Metaschema des Zustandsübergangsparadigmas gefordert, daß ausgehend von einem korrekten Systemzustand bei Eintreffen eines Ereignisses, das mehrere Übergänge auslöst, das System auch nur in einen korrekten Systemzustand übergeht. Übergänge aufgrund desselben Ereignisses aus demselben *Blob* oder aus verschiedenen, zueinander parallel laufenden *Blobs* enden entweder in genau einem *Blob* oder in weiteren zueinander parallelen *Blobs*[2].

for *G* **in** *STP* **assert**

[STP 1]:

$$\forall\, e \,:\, V_{EndBlob} \,\bullet\, \delta_{\leftarrow_{transComesFrom}}(e) \;=\; 0\;;$$

[STP 2]:

$$isTree(eGraph(E_{contains}))\ ;$$

[STP 3]:

$$type(root(eGraph(E_{contains}))) = XorBlob\ ;$$

[STP 4]:

$$\delta_{\leftharpoondown_{transComesFrom}}(root(eGraph(E_{contains}))) = \delta_{\leftharpoondown_{transGoesTo}}(root(eGraph(E_{contains}))) = 0\ ;$$

[STP 5]:

$$\exists_1\, b : V_{Blob} \bullet b \leftharpoondown_{contains} root(eGraph(E_{contains})) \rightharpoonup_{startsWith} b\ ;$$

[STP 6]:

$$\forall\, x : V_{XorBlob} \mid \delta_{\leftharpoondown_{transGoesTo}}(x) > 0 \vee x \leftharpoondown_{contains} \&_{AndBlob} \neq \varnothing \bullet \\ \delta_{\rightharpoonup_{startsWith}}(x) = 1\ ;$$

[STP 7]:

$$\forall\, e : E_{startsWith} \bullet \alpha(e) \rightharpoonup_{contains} \omega(e)\ ;$$

[STP 8]:

$$\forall\, x, y : V_{XorBlob} \mid x \leftharpoondown_{contains} \&_{AndBlob} \rightharpoonup_{contains} y \wedge x \neq y \bullet \\ \neg\ (x \rightharpoonup^*_{contains} \leftharpoondown_{transComesFrom} \rightharpoonup_{transGoesTo} \leftharpoondown^*_{contains} y)\ ;$$

[STP 9]:

$$\forall\, t_1, t_2 : V_{Transition};\ a, b, c, d : V_{Blob} \mid t_1 \neq t_2 \wedge \\ t_1 \leftharpoondown_{triggers} \&_{Event} \rightharpoonup_{triggers} t_2 \wedge \\ a \leftharpoondown_{transComesFrom} t_1 \rightharpoonup_{transGoesTo} b \wedge \\ c \leftharpoondown_{transComesFrom} t_2 \rightharpoonup_{transGoesTo} d \bullet \\ lca(a,b) = lca(c,d) \wedge \\ (a = c \vee \\ \quad (\neg\ (a = c \vee a \rightharpoonup^*_{contains} c \vee a \leftharpoondown^*_{contains} c) \wedge \\ \quad\quad a \leftharpoondown^*_{contains} \&_{AndBlob} \rightharpoonup^*_{contains} c)) \\ \Rightarrow \\ (b = d \vee \\ \quad (\neg\ (b = d \vee b \rightharpoonup^*_{contains} d \vee b \leftharpoondown^*_{contains} d) \wedge \\ \quad\quad b \leftharpoondown^*_{contains} \&_{AndBlob} \rightharpoonup^*_{contains} d))$$

end.

[2] Zusicherung [STP 9] verwendet die Funktion *lca*, die den kleinsten gemeinsamen Vorfahren zweier Blobs in der *contains*-Struktur bestimmt.

$$lca : V_{Blob} \times V_{Blob} \to V_{Blob}$$

$$lca = \lambda v, w : V_{Blob} \bullet \mu x : V_{Blob} \bullet v \leftharpoondown^*_{contains} x \rightharpoonup^*_{contains} w \wedge \\ \forall y : V_{Blob} \mid v \leftharpoondown^*_{contains} y \rightharpoonup^*_{contains} w \bullet x \leftharpoondown^*_{contains} y$$

Spezialisierungen des Referenz-Metaschemas

Das Referenz-Metaschema des Zustandsübergangsparadigma kann zur Metamodellierung der Zustandsübergangsbeschreibungen der Object Modeling Technique (OMT) [Rumbaugh et al., 1991, S. 89ff] und der Unified Modeling Language (UML) [Booch et al., 1999, S. 243ff] direkt übernommen werden. Zu dem hier vorgestellten Referenz-Metaschema ähnliche Metaschemata finden sich für OMT in [Ebert/Süttenbach, 1997a, S. 21ff] und für die UML in [OMG, 1999, S. 2-123]. Zur Metamodellierung der Zustandsübergangsbeschreibungen nach [Harel, 1987], nach [Booch, 1994], durch Zustandsautomaten und durch Zustandstabellen und -matrizen sind wenige Einschränkungen vorzunehmen.

Statecharts nach [Harel, 1987]. Statecharts verwenden keine *EndBlobs* [STPHarel 1] und unterstützen die Modellierung von den *Blobs* zugeordneten Systemreaktionen nicht [STPHarel 2]. Zusicherung [STP 6] des Referenz-Metaschemas ist für den Dialekt nach [Harel, 1987] zu verschäfen. *XorBlobs* enthalten hier generell einen Startblob [STPHarel 3].

STPHarel **isA** *STP* ;

for G **in** *STPHarel* **assert**

[STPHarel 1]:

$$V_{EndBlob} = \varnothing ;$$

[STPHarel 2]:

$$E_{IsEntryAction} = E_{IsStateAction} = E_{IsExitAction} = \varnothing ;$$

[STPHarel 3]:

$$\forall x : V_{XorBlob} \bullet \delta_{\rightharpoonup_{startsWith}}(x) = 1$$

end.

Zustandsübergangsdiagramme nach [Booch, 1994]. Die Zustandsübergangsbeschreibungen nach [Booch, 1994] entsprechen bis auf die Verwendung paralleler Blobs dem Referenz-Metaschema. Zur Abbildung dieser Notationsvariante enthält das Metaschema *STPBooch* keine *AndBlobs* [STPBooch 1] (vgl. auch [Booch, 1994, S. 208]).

STPBooch **isA** *STP* ;

for G **in** *STPBooch* **assert**

[STPBooch 1]:

$$V_{AndBlob} = \varnothing$$

end.

Zustandsautomaten. Die Modellierung durch Zustandsautomaten oder Zustandsübergangsdiagramme verwendet nur flache Automaten, ohne weitere Strukturierung der Blobs. Diese Automaten haben einen ausgezeichneten Anfangszustand. Aus Sicht des Referenz-Metaschemas bestehen solche Modelle aus genau einem *XorBlob*, der den gesamten Automaten beschreibt

[STPAutomaton 1]. Alle weiteren Blobs sind atomar und in diesem *XorBlob* enthalten [STPAutomaton 2]. *AndBlobs* existieren in Zustandsautomaten folglich nicht [STPAutomaton 3]. Auch werden hier keine Prädikate zur Modellierung von Guards verwendet. Zustandsübergänge sind ausschließlich von Ereignissen abhängig. In Zustandsautomaten sind auch Folgezustände von Endzuständen erlaubt. Die Zusicherung [STP 1] des Referenz-Metaschemas ist daher im Metaschema für Zustandsautomaten (*STPAutomaton*) entsprechend zu verallgemeinern [STPAutomaton 4].

STPAutomaton **isA** *STP* ;

for *G* **in** *STPAutomaton* **assert**

[STPAutomaton 1]:

$$\# V_{XorBlob} = 1 ;$$

[STPAutomaton 2]:

$$\forall a : V_{AtomicBlob} \bullet a \leftharpoondown_{contains} root(eGraph(E_{contains})) ;$$

[STPAutomaton 3]:

$$V_{AndBlob} = V_{Predicate} = \varnothing ;$$

[STPAutomaton 4]:

overwrites [STP 1] :

$$\forall e : V_{EndBlob} \bullet \delta_{\leftharpoondown_{transComesFrom}}(e) \geq 0$$

end.

Wird der Automat als Moore-Automat aufgefaßt, sind Systemreaktionen ausschließlich an die Blobs gebunden. Aktionen beim Eintritt, während des Aufenthalts und beim Verlassen des Blobs werden nicht unterschieden. Im Metaschema der Moore-Automaten (*STPMoore*), das aus *STPAutomaton* abgeleitet werden kann, existieren somit kein *effects*-, *isEntryAction*- und *isExitAction*-Kanten [STPMoore 1].

STPMoore **isA** *STPAutomaton* ;

for *G* **in** *STPMoore* **assert**

[STPMoore 1]:

$$E_{effects} = E_{isEntryAction} = E_{isExitAction} = \varnothing$$

end.

Analog kann auch das Metaschema der Mealy-Automaten (*STPMealy*) aus *STPAutomaton* abgeleitet werden. In Mealy-Automaten sind Aktionen ausschließlich an Zustandsübergänge gebunden; *isEntryAction*-, *isStateAction*- und *isExitAction*-Kanten existieren hier nicht [STPMealy 1].

STPMealy **isA** *STPAutomaton* ;

for *G* **in** *STPMealy* **assert**

[STPMealy 1]:

$$E_{isEntryAction} = E_{isStateAction} = E_{isExitAction} = \varnothing$$

end.

Zustandsübergangsmatrizen und -tabellen. Zustandsübergangsdarstellungen durch Matrizen und Tabellen beschreiben wie Automaten ausschließlich flache Zustandsübergangsstrukturen. Da Anfangs- und Endzustände in diesen Darstellungen nicht explizit ausgezeichnet werden, kann das Metaschema *STPTable* aus dem Referenz-Metaschema durch Einschränkung auf ausschließlich atomare Blobs abgeleitet werden [STPTable 1].

STPTable **isA** *STP* ;

for G **in** *STPMealy* **assert**

[STPTable 1]:

$$\forall\, b \,:\, V_{Blob} \bullet \mathit{type}(b) = \mathit{AtomicBlob}\ ;$$

end.

Je nach Modellierungsinhalt ist das Referenz-Metaschema *STPTable* um weitere Konzepte zu reduzieren. So enthält beispielsweise das Metaschema zur Darstellung der Zustandsübergangsmatrix aus Abbildung 3.20b (Seite 69) keine Aktionen, während sie in der Zustandsübergangtabelle in Abbildung 3.20c enthalten sind.

6.4.3 Metaschemata des Netzparadigmas

Mit den visuellen Modellierungssprachen des Netzparadigmas (vgl. Kapitel 3.4.3) wird die Systemdynamik entlang der Folge- und Kontrollflußbeziehungen zwischen Prozessen und/oder Ereignissen modelliert. Hierbei sind auch Verzweigungen bzw. Parallelisierungen von Kontrollflüssen und deren Zusammenführungen abzubilden. Das Referenz-Metaschema des Netzparadigmas definiert hierzu die Konzepte zur Beschreibung von Prozessen, Ereignissen und ihrer verschiedenen Folgebeziehungen.

Referenz-Metaschema des Netzparadigmas

Im Referenz-Metaschema des Netzparadigmas *NP*, dessen *EER*-Anteil in Abbildung 6.14 skizziert ist, werden Prozesse (*Process*), Ereignisse (*Event*) und Flußbeziehungen (*Flow*) unterschieden (vgl. auch die rudimentären Metaschemata in [Scheer, 1992, S. 72] und [Winter/Ebert, 1996, S. 111f]). Prozesse und/oder Ereignisse, die mit einem Namen versehen sind, werden zu Netzobjekten (*NetObject*) zusammengefaßt. Durch die Beschreibungsmittel des Netzparadigmas kann auch modelliert werden, welchen Objekten oder Organisationseinheiten ein Prozeß zugeordnet ist. Diese Anknüpfungspunkte zum Stellengliederungsparadigma (vgl. Kapitel 6.3.1) und zum Objekt-Instanz-Paradigma (vgl. Kapitel 6.5.1) werden durch die Konzepte *Object* und *OrganizationalUnit* abgebildet.

Flüsse (*Flow*) beschreiben das Aufeinanderfolgen von Prozessen und Ereignissen. Die Flußrichtung wird ähnlich zu den Datenflußbeziehungen im Referenz-Metaschema des Datenflußparadigmas (vgl. Kapitel 6.4.1) durch *cfComesFrom*- bzw. *cfGoesTo*-Kanten modelliert. Zur Modellierung der verschiedenen Kontrollflußbeziehungen wird zwischen elementaren (*ElemenaryFlow*) und Operator-Kontrollflüssen (*OperatorFlow*) unterschieden. Elementare Kontrollflüsse beschreiben direkte Kontrollflußbeziehungen zwischen zwei Netzelementen. Durch

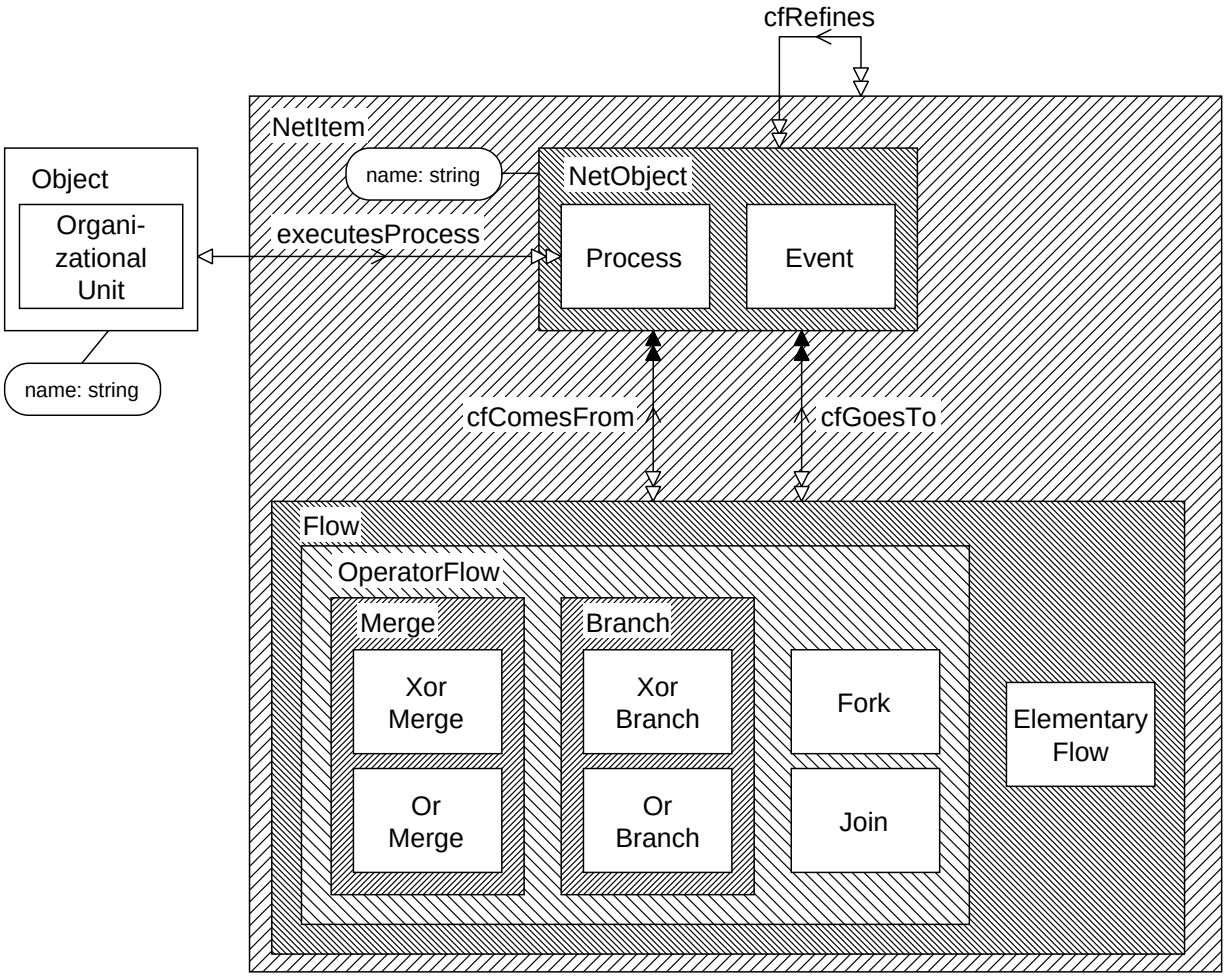


Abbildung 6.14: Referenz-Metaschema des Netzparadigmas (Net Paradigm, *NP*)

Operator-Kontrollflüsse wird die Aufspaltung bzw. Zusammenführung durch Verzweigungen oder Parallelbearbeitungen dargestellt. Die Aufteilung des Kontrollflusses in parallele Flüsse wird hierzu durch *Fork*-Knoten und die Zusammenführung durch *Join*-Knoten modelliert. Zur Darstellung von Verzweigungen und Vereinigungen werden *Branch*- und *Merge*-Knoten definiert. Je nach Art der Oder-Verzweigung oder Vereinigung werden diese in *XorBranch*- und *OrBranch*- bzw. *XorMerge*- und *OrMerge*-Knoten unterschieden.

Knoten vom Typ *ElemenaryFlow* verbinden durch *cfComesFrom*- und *cfGoesTo*-Kanten jeweils genau ein Netzelement mit genau einem Nachfolger [NP 1]. Verzweigungen (*Branch* und *Fork*-Knoten) verbinden jeweils ein Vorgängerobjekt mit beliebig vielen Nachfolgern [NP 2] und Vereinigungen (*Merge* und *Join*-Knoten) verbinden beliebig viele Vorgänger mit genau einem Nachfolger [NP 3].

Ereignisse und Prozesse können auch durch weitere Darstellungen nach dem Netzparadigma verfeinert werden. Hierzu definiert das Referenz-Metaschema des Netzparadigmas Kanten des Typs *cfRefines*, durch die Elemente (*NetItem*), die diese Verfeinerung bilden, mit dem verfeinerten Netzelement in Beziehung gesetzt werden. Wie auch die Verfeinerungsbeziehungen des Referenz-Metaschemas des Datenflußparadigmas (vgl. Kapitel 6.4.1) sind diese Verfeinerungs-

strukturen zyklenfrei [NP 4]. Verfeinerungen können in den Beschreibungsmitteln des Netzparadigmas mehrfach verwendet werden. Die Elemente, die solche Verfeinerungen bilden, sind in allen diesen Verfeinerungen enthalten [NP 5]. Kontrollflüsse sind auch hier ausschließlich zwischen Netzobjekten zugelassen, die dieselben übergeordneten Netzobjekte verfeinern [NP 6]. Vergleiche hierzu auch die Zusicherungen [DFP 4] und [DFP 5] des Referenz-Metaschemas des Datenflußparadigmas.

Weiter ist zuzusichern, daß die Objekte, die einen verfeinerten Prozeß ausführen, mit den Objekten, die die Verfeinerungen ausführen, verträglich sind. [NP 7][3].

for G **in** NP **assert**

[NP 1]: $\forall f : V_{ElementaryFlow} \bullet \delta_{\rightarrow_{cfComesFrom}}(f) = 1 \wedge \delta_{\rightarrow_{cfGoesTo}}(f) = 1$;

[NP 2]: $\forall f : V_{Branch} \cup V_{Fork} \bullet \delta_{\rightarrow_{cfComesFrom}}(f) = 1 \wedge \delta_{\rightarrow_{cfGoesTo}}(f) > 1$;

[NP 3]: $\forall f : V_{Merge} \cup V_{Join} \bullet \delta_{\rightarrow_{cfComesFrom}}(f) > 1 \wedge \delta_{\rightarrow_{cfGoesTo}}(f) = 1$;

[NP 4]: $isDag(eGraph(E_{cfRefines}))$;

[NP 5]: $\forall a, b : V_{NetItem} \mid a \rightarrow_{cfRefines} \leftarrow_{cfRefines} b \bullet a \rightarrow_{cfRefines} = b \rightarrow_{cfRefines}$;

[NP 6]: $\forall a, b : V_{NetObject} \mid a \leftarrow_{cfComesFrom} \rightarrow_{cfGoesTo} b \bullet a \rightarrow_{cfRefines} = b \rightarrow_{cfRefines}$;

[NP 7]: $\forall p_1 : V_{Process} \mid \delta_{\leftarrow_{cfRefines}} > 0 \bullet$
$p_1 \leftarrow_{executesProcess} = \bigcup \{p_2 : V_{Process} \mid p_1 \leftarrow_{cfRefines} p_2 \bullet p_2 \leftarrow_{executesProcess}\}$

end.

Die Balancierung von Kontrollflußverzweigungen und -vereinigungen und die Vermeidung von Kontrollflüssen zwischen nebenläufigen Prozeß- bzw. Ereignisfolgen wird für die Beschreibungsmittel des Netzparadigmas i. allg. *nicht* gefordert. In [Booch et al., 1999, S. 264] wird dieses z. B. für Aktivitätsdiagramme und in [Staud, 1999, S. 69] für (erweiterte) Ereignisgesteuerte Prozeßketten als besserer Modellierungsstil herausgestellt. Die Modellierungstechniken und deren Werkzeugunterstützungen schließen solche unbalancierten Kontrollflußbeziehungen jedoch nicht aus, so daß für das Referenz-Metaschema des Netzparadigmas keine Einschränkungen formuliert werden.

Spezialisierungen des Referenz-Metaschemas

Die verschiedenen Notationsvarianten des Netzparadigmas unterscheiden sich insbesondere bezüglich der Verfeinerungskonzepte, der Typen der Objekte, die miteinander in eine Folgebeziehung gebracht werden und der jeweils unterstützten Operationen über dem Kontrollfluß. In den folgenden Kapiteln werden Metaschemata für Aktivitätsdiagramme, für Vorgangskettendiagramme, für Ereignisgesteuerte Prozeßketten, für Petri-Netze und für Netzpläne aus dem Referenz-Metaschema abgeleitet.

[3] Falls die organisatorischen Einheiten gemäß des Metaschematas des Stellengliederungsparadigmas (vgl. Kapitel 6.3.1) weiter strukturiert sind, ist diese Strukturierung entsprechend zu berücksichtigen.

Aktivitätsdiagramme. In der Unified Modeling Language (UML) werden Aktivitätsdiagramme als eine Variante der Zustandsübergangsbeschreibungen [Rumbaugh et al., 1999, S. 135] eingeführt (vgl. auch das Metaschema in [OMG, 1999, S. 2-154]). Prozesse (oder Aktivitäten) werden hier als Zustände und Kontrollflußbeziehungen als Zustandsübergänge aufgefaßt.

Neben den direkten Zustandsübergängen zwischen zwei Prozessen können in Aktivitätsdiagrammen auch Verzweigungen und Vereinigungen des Kontrollflusses dargestellt werden, so daß Aktivitätsdiagramme im Gegensatz zur Einstufung in der UML im folgenden als Beschreibungsmittel des Netzparadigmas aufgefaßt werden. Abbildung 6.15 zeigt den *EER*-Teil der Spezialisierung des Referenz-Metaschemas des Netzparadigmas für Aktivitätsdiagramme (*NPActivity*). Aufgrund der Analogie zu den Zustandsbeschreibungen werden Aktivitätsdiagramme durch ein Startereignis eingeleitet und durch mindestens ein Endereignis beendet. Zur Modellierung dieser Ereignisse wird in der Spezialisierung des Referenz-Metaschemas des Netzparadigmas für Aktivitätsdiagramme (*NPActivity*) das *Event*-Konzept durch Knoten der Typen *StartEvent* und *EndEvent* spezialisiert.

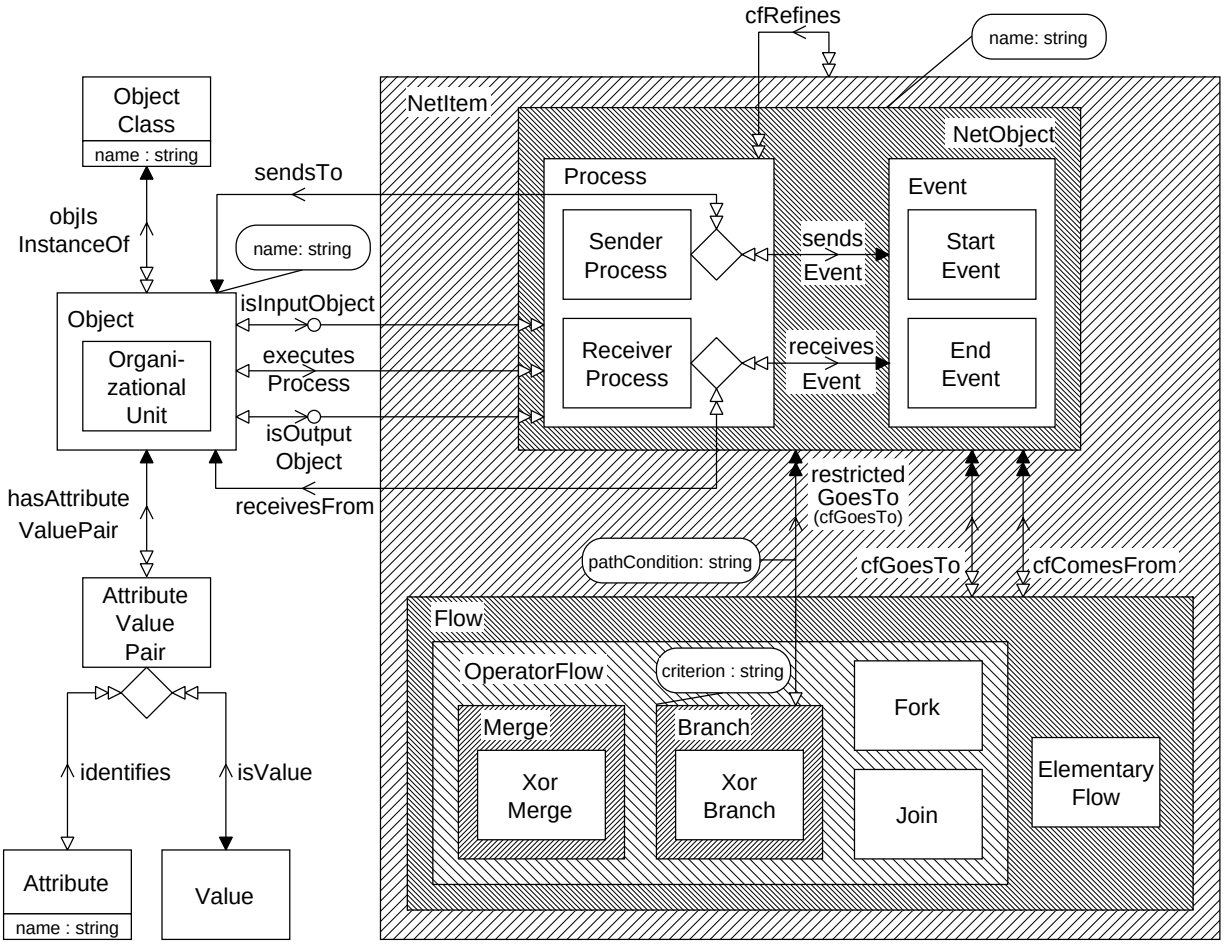


Abbildung 6.15: Spezialisierung des Referenz-Metaschemas des Netzparadigmas für Aktivitätsdiagramme (*NPActivity*)

Im Vergleich zum Referenz-Metaschema des Netzparadigmas nutzen Aktivitätsdiagramme nur exklusive Verzweigungen und Vereinigungen. Die gegenüber dem Referenz-Metaschema nicht genutzten *OrMerge*- bzw. *OrBranch*-Konzepte sind daher auszuschließen [NPActivity 1]. Kon-

trollflüsse werden nur durch *XorBranch*, *Fork*, *XorMerge* und *Join*-Operatoren verzweigt bzw. vereinigt. Xor-Verzweigungen des Kontrollflusses (*XorBranch*) können in Aktivitätsdiagrammen durch die Verzweigungskriterien annotiert werden. Hierzu wird das Attribut *criterion* der *XorBranch*-Knoten verwendet. Die korrespondierenden Bedingungen der einzelnen Kontrollflußpfade werden im *pathCondition*-Attribut der *restrictedGoesTo*-Spezialisierung der *cfGoesTo*-Kanten notiert.

Kontrollflußverzweigungen und -vereinigungen werden ausschließlich durch Kontrollflußoperatoren *OperatorFlow* beschrieben. Daher sind *Process*-Knoten auch nur zu jeweils genau einer *cfComesFrom* und genau einer *cfGoesTo*-Kante inzident [NPActivity 2]. Startereignisse sind in den modellierten Prozeß durch genau eine *cfComesFrom*-Kante eingebunden. Inzidenzen zu *cfGoesTo*-Kanten existieren nicht. Analog gehen in Endereignisse ausschließlich *cfGoesTo*-Kanten ein [NPActivity 3].

Durch Kontrollflüsse in Aktivitätsdiagrammen werden ausschließlich Flußbeziehungen zwischen (Teil-) Prozessen beschrieben. Mit Ausnahme der jeweiligen Start- und Endereignisse sind hier keine weiteren Ereignisse eingebunden. *cfComesFrom* und *cfGoesTo*-Kanten sind daher nicht zu Knoten des Typs *Event* inzident [NPActivity 4].

Die Interaktion von Prozessen mit externen Objekten wird im Metaschema für Aktivitätsdiagramme durch spezielle Prozesse beschrieben, die Ereignisse von Objekten empfangen (*ReceiverProcess*) bzw. Ereignisse an Objekte senden (*SenderProcess*). Wie auch die Start- und Endereignisse können diese Prozesse in Aktivitätsdiagrammen nicht verfeinert werden [NPActivity 5] und besitzen keine Eingabe- oder Ausgabeobjekte [NPActivity 6]. Ereignisse, die zur Interaktion mit externen Objekten genutzt werden, sind ferner keine Start- oder Endereignisse [NPActivity 7]. Neben diesen Ereignissen und den Start- und Endereignissen werden in Aktivitätsdiagrammen keine weiteren Ereignisse abgebildet [NPActivity 8].

Aus Analogie zu den Zustandsübergangsbeschreibungen der Unified Modeling Language beginnt jedes Aktivitätsdiagramm mit genau einem Startereignis und endet mit mindestens einem Endereignis. Eine Prozeßmodellierung durch Aktivitätsdiagramme besitzt daher genau ein Gesamt-Startereignis und mindestens ein Gesamt-Endereignis [NPActivity 9]. Ebenso existiert zu jedem verfeinerten Prozeß genau ein Startereignis und mindestens ein Endereignis [NPActivity 10]. Da Prozesse in Aktivitätsdiagrammen genau zu einem eingehenden und genau zu einem ausgehenden Kontrollfluß inzident sind (vgl. auch [NPActivity 2]), nehmen das Startereignis und die Endereignisse der Verfeinerung ähnliche Funktionen ein, wie die *PointOfContact*-Knoten zur Modellierung der Verfeinerung im Referenz-Metaschema des Datenflußparadigmas (vgl. Kapitel 6.4.1). Das Startereignis entspricht dem Kontaktpunkt des eingehenden Kontrollflusses und die Zusammenfassung der Endereignisse dem Kontaktpunkt des ausgehenden Kontrollflusses. Alle Prozesse und Ereignisse einer Verfeinerung bzw. des Gesamtprozesses sind von den jeweiligen Startereignissen aus erreichbar [NPActivity 11].

In Aktivitätsdiagrammen können, ähnlich zu Datenflußdiagrammen, auch die Zusammenhänge zwischen Aktivitäten und deren Ein- und Ausgabeobjekten modelliert werden [Rumbaugh et al., 1999, S. 139]. Solche Objektflüsse werden im Metaschema der Aktivitätsdiagramme durch *isInputObjekt* bzw. *isOutputObject*-Kanten abgebildet. Objekte sind hierbei durch ihren Typ (*ObjectClass*) und durch ihren inneren Zustand (*AttributValuePair*) charakterisiert. Objektflüsse zwischen Aktivitäten können dadurch modelliert werden, daß Objekte von einer Aktivität erzeugt und von mindestens einer weiteren Aktivität verbraucht werden. Es ist aber auch möglich, Ob-

jekte nur als Eingaben oder als Ausgaben einer Aktivität darzustellen. In Aktivitätsverfeinerungen sind diese Objektbezüge zu balancieren. Objekte, die Ein- bzw. Ausgaben einer verfeinerten Aktivität beschreiben, müssen sich in den Ein- bzw. Ausgaben der Verfeinerung wiederfinden und umgekehrt [NPActivity 12].

NPActivity **isA** *NP* ;

for *G* **in** *NPActivity* **assert**

[NPActivity 1]:

$$V_{OrMerge} = V_{OrBranch} = \varnothing ;$$

[NPActivity 2]:

$$\forall v : V_{Process} \bullet \delta_{\leftharpoondown_{cfComesFrom}}(v) = \delta_{\leftharpoondown_{cfGoesTo}}(v) = 1 ;$$

[NPActivity 3]:

$$\forall s : V_{StartEvent} \bullet \delta_{\leftharpoondown_{cfComesFrom}}(s) = 1 \wedge \delta_{\leftharpoondown_{cfGoesTo}}(s) = 0 ;$$
$$\forall e : V_{EndEvent} \bullet \delta_{\leftharpoondown_{cfComesFrom}}(e) = 0 \wedge \delta_{\leftharpoondown_{cfGoesTo}}(e) = 1 ;$$

[NPActivity 4]:

$$\forall e : E_{cfComesFrom} \cup E_{cfGoesTo} \bullet type(\omega(e)) \neq Event ;$$

[NPActivity 5]:

$$\forall r : E_{cfRefines} \bullet type(\omega(r)) = Process ;$$

[NPActivity 6]:

$$\forall e : E_{isInputObject} \cup E_{isOutputObject} \bullet$$
$$type(\omega(e)) = Process ;$$

[NPActivity 7]:

$$\forall e : E_{sendsEvent} \cup E_{receivesEvent} \bullet$$
$$StartEvent \neq type(\omega(e)) \neq EndEvent ;$$

[NPActivity 8]:

$$\forall e : V_{Event} \bullet type(e) = StartEvent \vee type(e) = EndEvent \vee$$
$$\delta_{\leftharpoondown_{sendsEvent,\ receivesEvent}}(e) \geq 1 ;$$

[NPActivity 9]:

$$\exists_1 s : V_{StartEvent} \bullet \delta_{\rightharpoonup_{cfRefines}}(s) = 0 ;$$
$$\exists e : V_{EndEvent} \bullet \delta_{\rightharpoonup_{cfRefines}} = 0 ;$$

[NPActivity 10]:

$$\forall p : V_{Process} \mid \delta_{\leftharpoondown_{cfRefines}}(p) > 0 \bullet$$
$$(\exists_1 s : V_{StartEvent} \bullet s \rightharpoonup_{cfRefines} p) \wedge$$
$$(\exists e : V_{EndEvent} \bullet e \rightharpoonup_{cfRefines} p) ;$$

[NPActivity 11]:

$$\forall\, s : V_{StartEvent};\ v : V_{NetItem} \mid \delta_{\leftharpoonup_{cfRefines}}(s) = 0 = \delta_{\leftharpoonup_{cfRefines}}(v) \bullet$$
$$s(\leftharpoonup_{cfComesFrom} | \rightharpoonup_{cfGoesTo})^* v\ ;$$
$$\forall\, p : V_{Process};\ v : V_{NetItem} \mid v \rightharpoonup_{cfRefines} p \bullet$$
$$p \leftharpoonup_{cfRefines} \&_{StartEvent}(\leftharpoonup_{cfComesFrom} | \rightharpoonup_{cfGoesTo})^* v\ ;$$

[NPActivity 12]:

$$\forall\, p_1 : V_{Process} \mid \delta_{\leftharpoonup_{cfRefines}}(p_1) > 0 \bullet$$
$$p_1 \leftharpoonup_{isInputObject} = \bigcup \{p_2 : V_{Process} \mid p_2 \rightharpoonup_{cfRefines} p_1 \bullet p_2 \leftharpoonup_{isInputObject}\} \ \wedge$$
$$p_1 \leftharpoonup_{isOutputObject} = \bigcup \{p_2 : V_{Process} \mid p_2 \rightharpoonup_{cfRefines} p_1 \bullet p_2 \leftharpoonup_{isOutputObject}\}$$

end.

Vorgangskettendiagramme. Vorgangskettendiagramme und erweiterte Ereignisgesteuerte Prozeßketten ergänzen (reine) Prozeßdarstellungen um Querbezüge zur Beschreibung von technischen und maschinellen Handlungsträgern und um Mittel zur Darstellung, der bei der Aufgabenerledigung benötigten bzw. erzeugten Objekte.

Das Referenz-Metaschema des Netzparadigmas ist hierzu um zusätzliche Bezüge zwischen Netzelementen und Objekten, durch die Handlungsträger und Daten modelliert werden, zu ergänzen. Der Bezug zu den Handlungsträgern wird wie im Referenz-Metaschema durch Kanten des Typs *executesProcess* hergestellt. Durch *requires*-Kanten werden den Teilprozessen die benötigten Daten und durch *delivers* die erzeugten Daten zugeordnet (vgl. [Staud, 1999, S. 51]). Querbezüge zwischen dem modellierten Gesamtprozeß und den hierin berechneten Daten werden in Vorgangskettendiagrammen auch entlang der Ereignisse beschrieben. Ereignisse symbolisieren hierbei das Vorliegen dieser Datenobjekte (vgl. [Scheer, 1992]). Dieser Zusammenhang wird durch *represents*-Kanten zwischen *Event*- und *Object*-Knoten modelliert.

Neben den aus dem Referenz-Metaschema bekannten Operatoren können in Vorgangskettendiagrammen auch kombinierte Kontrollflußoperatoren verwendet werden. Diese vereinigen eingehende Kontrollflüsse und teilen ausgehende Kontrollflüsse auf. Hierzu wird ein weiterer Kontrollflußoperator *ComposedOperatorFlow* eingeführt, dessen Attribute *joinType* und *forkType* die verschiedenen Arten der Kontrollflußvereinigung und -verzweigung abbilden.

Der *EER*-Teil der Spezialisierung des Referenz-Metaschemas zur Beschreibung von Vorgangskettendiagramme (*NPVKD*) ist in Abbildung 6.16 dargestellt.

Prozeßketten werden in Vorgangskettendiagrammen durch zusammenhängende [NPVKD 1], alternierende Folgen von Prozessen und Ereignissen modelliert. Für sämtliche Kontrollflußoperatoren (*Flow*) ist zu daher fordern, daß hierdurch nicht gleichartige Netzelemente miteinander verknüpft werden [NPVKD 2]. In Anlehnung an die Beispiel-Kontrollflüsse in [Scheer, 1994, S. 50] [IDS, 1995, S. 4-120ff] und [Staud, 1999, S. 63ff] sind hierbei diejenigen Kontrollflüsse auszuschließen, die ein Ereignis mit mehreren alternativen Prozessen verbinden [NPVKD 3]. Wie auch in Aktivitätsdiagrammen, werden Kontrollflüsse ausschließlich durch Kontrollflußoperatoren verzweigt und/oder vereinigt, so daß *Process*- und *Event*-Knoten jeweils zu höchstens einer *cfComesFrom* bzw. einer *cfGoesTo*-Kante inzident sind [NPVKD 4]. Begrenzt werden Ereignisgesteuerte Prozeßketten durch genau ein Startereignis und genau ein Endereignis [NPVKD 5].

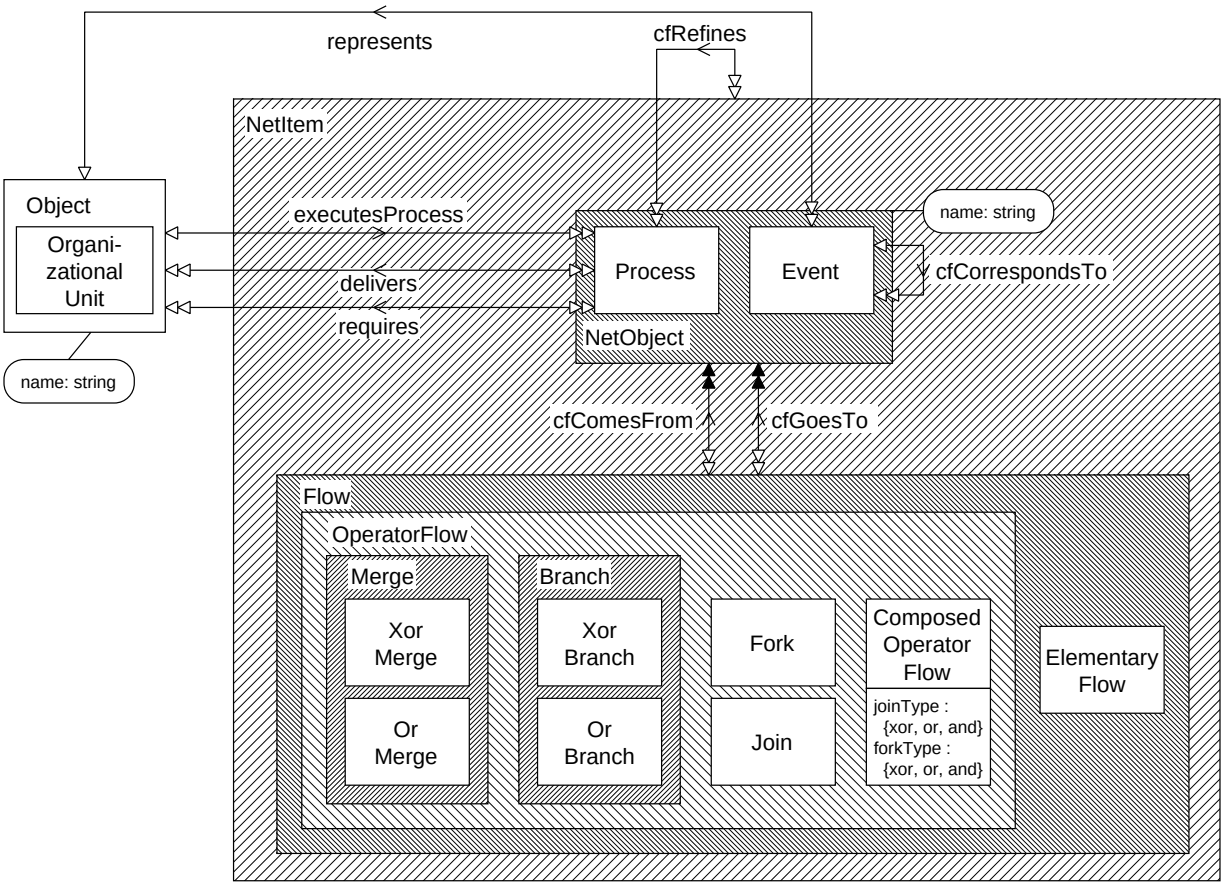


Abbildung 6.16: Spezialisierung des Referenz-Metaschemas des Netzparadigmas für Vorgangskettendiagramme (*NPVKD*)

Anforderungen an Verfeinerungen von Prozessen oder Ereignissen in Vorgangskettendiagrammen werden in [Keller et al., 1992], [Scheer, 1992] und [Scheer, 1994] nicht formuliert. Auch erlaubt das ARIS-Toolset [IDS, 1995] zur Modellierung von Prozeßfolgen nahezu beliebige Verfeinerungen von Prozessen oder Ereignissen. Rudimentäre Regeln zur Verfeinerung von Prozeßketten werden in [Staud, 1999, S. 71ff] vorgeschlagen. In Anlehnung an diese Regeln werden in Prozeßketten ausschließlich Prozesse verfeinert. Mehrfachverwendungen von Verfeinerungen sind ebenfalls möglich.

Die Balancierung der Prozeßverfeinerungen mit den verfeinerten Prozessen erfolgt entlang der Vor- und Nachereignisse der Prozesse. *cfCorrespondsTo*-Kanten verbinden hierzu die Ereignisse zu einem verfeinerten Prozeß mit ihren Surrogaten in der Verfeinerung [NPVKD 6]. Zu jedem Ereignis, das zu einem verfeinerten Prozeß adjazent ist, existiert in der Verfeinerung genau ein Surrogatereignis. Diese Surrogatereignisse beschreiben in der Verfeinerung für Vorereignisse des verfeinerten Prozesses Startereignisse und für die Nachereignisse Endereignisse [NPVKD 7]. Zu einem verfeinerten Prozeß sind diese Surrogate für alle Ereignisse eindeutig [NPVKD 8]. Schließlich ist noch zuzusichern, daß die in der Verfeinerung benötigten bzw. erzeugten Objekte auch mit den entsprechenden Objekten des verfeinerten Prozesses verträglich sind [NPVKD 9].

NPVKD **isA** *NP* ;

for *G* **in** *NPVKD* **assert**

[NPVKD 1]:

$$isConnected(eGraph(E_{cfComesFrom} \cup E_{cfComesFrom}))\ ;$$

[NPVKD 2]:

$$\forall\, v, w : V_{NetObject} \mid v \leftharpoonup_{cfComesFrom} \rightharpoonup_{cfGoesTo} w \bullet type(v) \neq type(w)\ ;$$

[NPVKD 3]:

$$\forall\, e : E_{cfComesFrom} \mid type(\omega(e)) = Event \bullet \neg\ (\ type(\alpha(e)) = XorBranch \ \vee\ type(\alpha(e)) = OrBranch \ \vee\ type(\alpha(e)) = ComposedOperatorFlow \ \wedge\ \alpha(e).forktype \neq and\ )\ ;$$

[NPVKD 4]:

$$\forall\, v : V_{NetObject} \bullet \delta_{\leftharpoonup_{cfComesFrom}}(v) \leq 1 \wedge \delta_{\leftharpoonup_{cfGoesTo}}(v) \leq 1\ ;$$

[NPVKD 5]:

$$\exists_1\, e : V_{Event} \bullet \delta_{\rightharpoonup_{cfRefines}}(e) = 0 \wedge \delta_{\leftharpoonup_{cfGoesTo}}(e) = 0\ ;$$

$$\exists_1\, e : V_{Event} \bullet \delta_{\rightharpoonup_{cfRefines}}(e) = 0 \wedge \delta_{\leftharpoonup_{cfComesFrom}}(e) = 0\ ;$$

[NPVKD 6]:

$$\forall\, c : E_{cfCorrespondsTo} \bullet \exists_1\, p : V_{Process} \mid \delta_{\leftharpoonup_{cfRefines}}(p) > 0 \wedge \alpha(c) \rightharpoonup_{cfRefines} p \bullet \omega(c)(\leftharpoonup_{cfComesFrom} \rightharpoonup_{cfGoesTo}) \mid (\leftharpoonup_{cfGoesTo} \rightharpoonup_{cfComesFrom})\, p\ ;$$

[NPVKD 7]:

$$\forall\, p : V_{Process};\ e : V_{Event} \mid \delta_{\leftharpoonup_{cfRefines}}(p) > 0 \wedge e \leftharpoonup_{cfComesFrom} \rightharpoonup_{cfGoesTo} p \bullet \exists_1\, s : V_{Event} \mid p \leftharpoonup_{cfRefines} s \rightharpoonup_{cfCorrespondsTo} e \bullet \delta_{cfGoesTo}(s) = 0\ ;$$

$$\forall\, p : V_{Process};\ e : V_{Event} \mid \delta_{\leftharpoonup_{cfRefines}}(p) > 0 \wedge e \leftharpoonup_{cfGoesTo} \rightharpoonup_{cfComesFrom} p \bullet \exists_1\, s : V_{Event} \mid p \leftharpoonup_{cfRefines} s \rightharpoonup_{cfCorrespondsTo} e \bullet \delta_{cfComesFrom}(s) = 0\ ;$$

[NPVKD 8]:

$$\forall\, p : V_{Process};\ e_1, e_2, s_1, s_2 : V_{Event} \mid e_1 \neq e_2 \wedge e_1(\leftharpoonup_{cfComesFrom} \rightharpoonup_{cfGoesTo}) \mid (\leftharpoonup_{cfGoesTo} \rightharpoonup_{cfComesFrom})\, p \wedge p \leftharpoonup_{cfRefines} s_1 \rightharpoonup_{cfCorrespondsTo} e_1 \wedge e_2(\leftharpoonup_{cfComesFrom} \rightharpoonup_{cfGoesTo}) \mid (\leftharpoonup_{cfGoesTo} \rightharpoonup_{cfComesFrom})\, p \wedge p \leftharpoonup_{cfRefines} s_2 \rightharpoonup_{cfCorrespondsTo} e_2 \bullet s_1 \neq s_2\ ;$$

[NPVKD 9]:

$$\forall\, p_1 : V_{Process} \mid \delta_{\leftharpoonup_{cfRefines}} > 0 \bullet p_1 \leftharpoonup_{requires} = \bigcup \{p_2 : V_{Process} \mid p_1 \leftharpoonup_{cfRefines} p_2 \bullet p_2 \leftharpoonup_{requires}\} \wedge p_1 \leftharpoonup_{delivers} = \bigcup \{p_2 : V_{Process} \mid p_1 \leftharpoonup_{cfRefines} p_2 \bullet p_2 \leftharpoonup_{delivers}\}$$

end.

Ereignisgesteuerte Prozeßketten. Aus dem Metaschema der Vorgangskettendiagramme (*NP-VKD*) kann das Metaschema der Ereignisgesteuerten Prozeßketten (*NPEPK*) abgeleitet werden. Gegenüber Vorgangskettendiagrammen und erweiterten Ereignisgesteuerten Prozeßketten ist in Ereignisgesteuerten Prozeßketten die Prozeßmodellierung ausschließlich auf die Folgebeziehungen von (Teil-) Prozessen und Ereignissen beschränkt. Querbezüge zu Handlungsträgern und Datenobjekten werden hier nicht modelliert. *Objekt*-Knoten sind im Metaschema der Ereignisgesteuerten Prozeßketten daher ausgeblendet [NPEPK 1].

NPEPK **isA** *NPVKN* ;
for *G* **in** *NPEPK* **assert**
[NPEPK 1]:
$$V_{Object} = \varnothing$$
end .

Petri-Netze. Prozeßmodellierungen durch Petri-Netze beschreiben ebenfalls Flüsse zwischen Prozessen (Transitionen) und Ereignissen (Stellen). Operatoren über Flüsse werden in Petri-Netzen nicht modelliert. Im Gegensatz zu Aktivitätsdiagrammen, Vorgangskettendiagrammen und Ereignisgesteuerten Prozeßketten erfolgt die Modellierung der Nebenläufigkeit in Petri-Netzen ausschließlich durch elementare Flüsse, die genau zwei Netzelemente miteinander verbinden. Auf ein Ereignis bzw. auf einen Prozeß können jedoch beliebig viele Prozesse oder Ereignisse folgen.

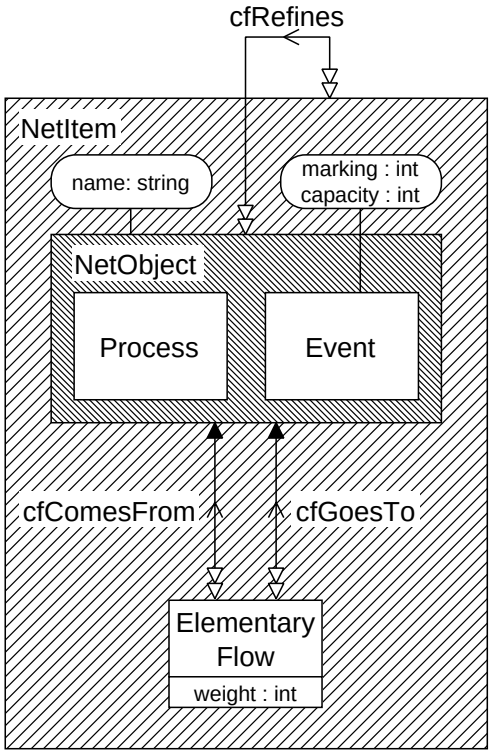


Abbildung 6.17: Spezialisierung des Referenz-Metaschemas des Netzparadigmas für Petri-Netze (*NPPetri*)

Zur Ableitung des Metaschemas für Petrinetze (*NPPetri*, vgl. Abbildung 6.17) ist das Referenz-Metaschema des Netzparadigmas daher um die Kontrollflußoperatoren einzuschränken. Ebenso werden in Petri-Netzen keine Objektbezüge modelliert. [NPPetri 1]

Die Markierung von Petri-Netzen erfordert die Ergänzung zusätzlicher Attribute. Für einfache Netzklassen wie z. B. für *Stellen-Transitions-Netze* reicht die Ergänzung der *Event*-Knoten um Attribute zur Beschreibung der aktuellen Markierung einer Stelle (*marking*) und ihrer Kapazität (*capacity*). Hierbei genügt die Markierung einer Stelle ihrer Kapazität [NPPetri 2]. Das Gewicht der Flußbeziehungen, durch das die in einem Schaltvorgang maximal übertragbaren Markierungen notiert werden, wird durch das *weight*-Attribut abgebildet.

Petri-Netze beschreiben Flußbeziehungen durch bipartite Graphen, bei denen auf Prozesse jeweils Ereignisse und auf Ereignisse jeweils Prozesse folgen. Diese Eigenschaft wird durch [NPPetri 3] zugesichert.

Petri-Netze erlauben sowohl die Verfeinerung von Ereignissen (Stellen) als auch von Prozessen (Transitionen). Hierzu werden in Anlehnung an [Baumgarten, 1996, S. 62] Stellen durch stellenberandete Netze und Transitionen durch transitionenberandete Netze verfeinert. In Verfeinerungen von Transitionen sind somit ausschließlich solche Stellen zugelassen, die Transitionen miteinander verbinden [NPPetri 4]. Eine entsprechende Zusicherung gilt für die Transitionen in Stellenverfeinerungen [NPPetri 5].

Weitere Forderungen, insbesondere hinsichtlich der Einbettung einer Verfeinerung in das übergeordnete Netz werden hier nicht erhoben, da in Petri-Netz-Verfeinerung i. allg. von diesen Aspekten abstrahiert wird (vgl. auch [Baumgarten, 1996, S. 60]). Die Umgebung, in der eine Transitions- oder Stellenverfeinerung ausgeführt wird, ist nicht Bestandteil der Verfeinerung.

NPPetri **isA** *NP* ;

for G **in** *NPPetri* **assert**

[NPPetri 1]:

$$V_{Object} = V_{OperatorFlow} = \varnothing ;$$

[NPPetri 2]:

$$\forall\, e : V_{Event} \bullet e.marking \leq e.capacity ;$$

[NPPetri 3]:

$$\forall\, v, w : V_{NetObject} \mid v \leftharpoonup_{cfComesFrom} \rightharpoonup_{cfGoesTo} w \bullet type(v) \neq type(w) ;$$

[NPPetri 4]:

$$\forall\, s : V_{Event} \mid s \rightharpoonup_{cfRefines} \&_{Process} \bullet \delta_{\leftharpoonup_{cfComesFrom}}(s) > 0 \wedge \delta_{\leftharpoonup_{cfGoesTo}}(s) > 0 ;$$

[NPPetri 5]:

$$\forall\, t : V_{Process} \mid t \rightharpoonup_{cfRefines} \&_{Event} \bullet \delta_{\leftharpoonup_{cfComesFrom}}(t) > 0 \wedge \delta_{\leftharpoonup_{cfGoesTo}}(t) > 0$$

end.

Aufbauend auf diesem einfachen Metaschema für Petri-Netze können weitere Anforderungen an Netze und (statische) Netzeigenschaften definiert werden. Zu einem gegebenen Petri-Netz inklusive seiner Markierung können beispielsweise alle aktiven Transitionen angegeben werden.

Eine Transition (*Process*) ist genau dann aktiv, wenn die Stellen (*Event*) in ihrem Vorbereich ausreichend markiert sind und die Stellen im Nachbereich ausreichende Kapazitäten haben.

for G **in** *NPPetri* **define**

$$isActive : \mathbb{P}\, V_{Process}$$

where

$$\forall t : V_{Process} \bullet isActive(t) \Leftrightarrow (\forall s : V_{Event};\ f : V_{ElementaryFlow} \mid s \leftarrowtail_{cfComesFrom} f \rightarrowtail_{cfGoesTo} t \bullet s.marking \geq f.weight) \wedge (\forall s : V_{Event};\ f : V_{ElementaryFlow} \mid t \leftarrowtail_{cfComesFrom} f \rightarrowtail_{cfGoesTo} s \bullet s.marking + f.weight \leq s.capacity)$$

end.

Durch Einschränkungen und Ergänzungen des Metaschemas für Petri-Netze können auch Metaschemata für weitere Netzklassen abgeleitet werden.

In *Bedingungs/Ereignis-Netzen* (condition event nets) [Baumgarten, 1996, S. 111] wird für alle Stellen eine Kapazität von 1 gefordert. Das Metaschema für Bedingungs/Ereignis-Netze (*NPPetriCE*) ergänzt das Metaschema der Petri-Netze um diese Forderung.

NPPetriCE **isA** *NPPetri* ;

for G **in** *NPPetriCE* **assert**

[NPPetriCE 1]:

$$\forall s : V_{Event} \bullet s.capacity = 1$$

end.

Die in Abbildung 3.24 (Seite 78) zur Modellierung der radiologischen Untersuchung verwendete Netzklasse erfordert eine Erweiterung des Metaschemas *NPPetriCE* um ein Attribut der *ElementaryFlow*-Knoten zur Modellierung der Schalt-Vorbedingungen (Guards).

In *Zustandsmaschinen* [Baumgarten, 1996, S. 72] sind alle Transitionen unverzweigt und liegen nicht am Rand des Netzes. Transitionen in Zustandsmaschinen besitzen daher jeweils genau eine Stelle in ihrem Vorbereich und genau eine Stelle in ihrem Nachbereich. Das Metaschema der Zustandsmaschinen *NPStateMachine* ergänzt das Metaschema der Petri-Netze um diese Forderung.

NPStateMachine **isA** *NPPetri* ;

for G **in** *NPStateMachine* **assert**

[NPStateMachine 1]:

$$\forall t : V_{Process} \bullet \delta_{\leftarrowtail_{cfComesFrom}}(t) = 1 = \delta_{\leftarrowtail_{cfGoesTo}}(t)$$

end.

Analog kann das Metaschema der *Synchronisationsgraphen* [Baumgarten, 1996, S. 72] aus dem Metaschema der Petri-Netze abgeleitet werden. In dieser Netzklasse sind die Stellen unverzweigt und liegen nicht am Rand des Netzes.

Für *Free-Choice-Netze* [Baumgarten, 1996, S. 74] wird gefordert, daß die Zieltransitionen vorwärtsverzweigter Stellen nicht rückwärtsverzweigt sind. Zur Beschreibung des Metaschemas

Free-Choice-Netze (*NPFreeChoice*) ist das Metaschema der Petri-Netze um die Forderung zur ergänzen, daß zu allen Transitionen, die zu einer vorwärtsverzweigten Stelle vorwärtsadjazent sind, keine weiteren Stellen vorwärtsadjazent sind.

NPFreeChoice **isA** *NPPetri* ;

for *G* **in** *NPFreeChoice* **assert**

[NPFreeChoice 1]:

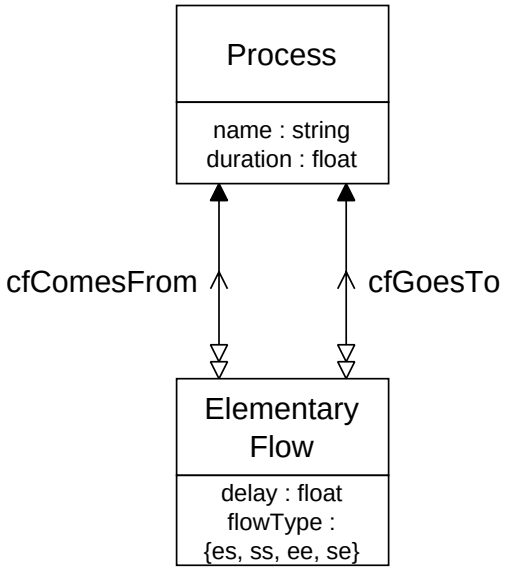
$$\forall\, s : V_{Event};\ t : V_{Process} \mid s \leftharpoonup_{cfGoesTo} \rightharpoonup_{cfComesFrom} t \ \wedge\ \delta_{\leftharpoonup_{cfGoesTo}}(s) > 0 \bullet \delta_{\leftharpoonup_{cfGoesTo}}(t) = 1$$

end.

Höhere Petri-Netze erfordern analog zu den Attributierungen im Metaschema *NPPetri* weitere Attribute zur Abbildung der in diesen Netzklassen verwendeten Markentypen und deren Kapazitäten.

Netzpläne. Ablaufbeschreibungen werden in der Netzplantechnik zur Planung, Steuerung und Überwachung von Prozeßabläufen verwendet. Die Prozeßmodellierung durch Netzpläne kann ebenfalls auf einer Spezialisierung des Referenz-Metaschemas des Netzparadigmas aufbauen. Diese Metaschemata der Netzplantechnik bilden auch die Grundlage zur Spezifikation der Methoden zur Analyse von Netzplänen.

Abbildung 6.18 zeigt die Ableitung des Metaschemas für Vorgangsknotennetzpläne aus dem Referenz-Metaschema des Netzparadigmas. Elementare Folgebeziehungen (*ElementaryFlow*) werden hier ausschließlich zwischen Prozessen (*Process*) dargestellt. Weitere Konzepte werden hier nicht verwendet [NPVKN1].



NPVKN **isA** *NP* ;

for *G* **in** *NPVKN* **assert**

[NPVKN 1]:

$V_{Object} = V_{Event} = V_{OperatorFlow} = \varnothing$;

$E_{cfRefines} = \varnothing$;

[NPVKN 2]:

isDag

end.

Abbildung 6.18: Spezialisierung des Referenz-Metaschemas des Netzparadigmas für Vorgangsknotennetzpläne (*NPVKN*)

Zur Analyse und Berechnung der zeitlichen Einordnung der Prozeßfolge sind die Prozesse mit ihrer Bearbeitungsdauer (*duration*) und die Flüsse mit evtl. zu modellierenden Wartezeiten beim

Transport oder zur Zwischenlagerung (*delay*) attributiert. Die verschiedenen Folgebeziehungen zwischen Prozessen werden durch das Attribut *flowType* unterschieden. Normalfolgen werden durch *es*, Anfangsfolgen durch *ss*, Endfolgen durch *ee* und Sprungfolgen durch *se* ausgewiesen.

Die Zeitanalyse der Netzplantechnik basiert auf zyklenfreien Prozeßbeschreibungen [NPVKN 2]. Hierzu werden in Vorgangsknotennetzplänen für Prozesse u. a. früheste Anfangs- und Endzeitpunkte, späteste Anfangs- und Endzeitpunkte und Pufferzeiten berechnet. Diese Methoden können auf Basis des Metaschemas der Vorgangsknotennetzpläne spezifiziert werden.

Der früheste Anfangszeitpunkt (earliest start time *ES*) eines Prozesses bestimmt sich aus den frühesten Zeitpunkten der Vorgängerprozesse. Ein Prozeß kann frühestens dann beginnen, wenn der letzte Vorgängerprozeß beendet ist und evtl. eingeplante Wartezeiten verstrichen sind. Für die Anfangsprozesse wird der Anfangszeitpunkt 0 angenommen. Bei der Berechnung dieser Zeitpunkte sind auch die verschiedenen Folgebeziehungen zu berücksichtigen (vgl. auch [Schwarze, 1994, S. 154]). Der früheste Endzeitpunkt (earliest end time *EF*) eines Prozesses errechnet sich aus dem frühesten Anfangszeitpunkt und der Prozeßdauer.

for G **in** *NPVKN* **define**

$$ES : V_{Process} \rightarrow float;$$
$$EF : V_{Process} \rightarrow float$$

where

$$ES = \lambda p : V_{Process} \bullet$$
$$max(\{0\} \cup$$
$$\{q : Process;\ f : ElementaryFlow \mid q \leftarrow_{cfComesFrom} f \rightarrow_{cfGoesTo} p \wedge f.flowType = es \bullet ES(q) + q.duration + f.delay\} \cup$$
$$\{q : Process;\ f : ElementaryFlow \mid q \leftarrow_{cfComesFrom} f \rightarrow_{cfGoesTo} p \wedge f.flowType = ss \bullet ES(q) + f.delay\} \cup$$
$$\{q : Process;\ f : ElementaryFlow \mid q \leftarrow_{cfComesFrom} f \rightarrow_{cfGoesTo} p \wedge f.flowType = ee \bullet ES(q) + q.duration + f.delay - p.duration\} \cup$$
$$\{q : Process;\ f : ElementaryFlow \mid q \leftarrow_{cfComesFrom} f \rightarrow_{cfGoesTo} p \wedge f.flowType = se \bullet ES(q) + f.delay - p.duration\});$$

$$EF = \lambda p : V_{Process} \bullet ES(p) + p.duration$$

end.

In ähnlicher Weise sind auch die spätesten Zeitpunkte definiert. Ein Prozeß muß spätestens zu dem Zeitpunkt beendet sein (latest finish time, *LF*), wenn unter Berücksichtigung der Wartezeiten der früheste seiner Nachfolgerprozesse spätestens beginnen muß, um den vorgegebenen Endzeitpunkt des gesamten Prozesses einzuhalten. Dieser Endzeitpunkt des gesamten Prozesses wird hierzu auf den frühesten Endzeitpunkt des letzten Prozesses gesetzt. Der späteste Anfangszeitpunkt eines Prozesses (latest start time *LS*) entspricht dem spätesten Endzeitpunkt abzüglich der Prozeßdauer.

for G **in** *NPVKN* **define**

$LF : V_{Process} \rightarrow float;$
$LS : V_{Process} \rightarrow float$

where

$$LF = \lambda p : V_{Process} \bullet$$
$$min(\{max\{q : Process \mid \delta_{\leftharpoondown_{cfComesFrom}}(q) = 0 \bullet EF(q)\}\} \cup$$
$$\{q : Process;\ f : ElementaryFlow \mid p \leftharpoondown_{cfComesFrom} f \rightharpoondown_{cfGoesTo} q \wedge f.flowType = es \bullet LF(q) - q.duration - f.delay\} \cup$$
$$\{q : Process;\ f : ElementaryFlow \mid p \leftharpoondown_{cfComesFrom} f \rightharpoondown_{cfGoesTo} q \wedge f.flowType = ss \bullet LF(q) - q.duration - f.delay + p.duration\} \cup$$
$$\{q : Process;\ f : ElementaryFlow \mid p \leftharpoondown_{cfComesFrom} f \rightharpoondown_{cfGoesTo} q \wedge f.flowType = ee \bullet LF(q) - f.delay - p.duration\} \cup$$
$$\{q : Process;\ f : ElementaryFlow \mid p \leftharpoondown_{cfComesFrom} f \rightharpoondown_{cfGoesTo} q \wedge f.flowType = se \bullet LF(q) - f.delay + p.duration\}) ;$$

$$LS = \lambda p : V_{Process} \bullet LF(p) - p.duration$$

end.

Die Pufferzeit (total float *TF*), um die die Bearbeitung eines Prozesses verzögert werden kann, ohne die Einhaltung des Endtermins des Gesamtprozesses zu gefährden, bestimmt sich aus der Differenz der spätesten und frühesten Termine.

for G **in** *NPVKN* **define**

$TF : V_{Process} \rightarrow float$

where

$$TF = \lambda p : V_{Process} \bullet LS(p) - ES(p)$$

end.

Die kritischen Prozesse, deren Zeitplanung genau eingehalten werden muß um das angestrebte Prozeßende zu erreichen, sind dann diejenigen Prozesse mit einer Pufferzeit von 0.

for G **in** *NPVKN* **define**

$isCriticalProcess : \mathbb{P}\, V_{Process}$

where

$$isCriticalProcess = \{p : V_{Process} \mid TF(p) = 0\}$$

end.

Für die weiteren Netzplanklassen können analog Spezialisierungen des Referenz-Metaschemas des Netzparadigmas abgeleitet und hierauf aufbauend diverse Analysemethoden spezifiziert werden. Für Ereignisknotennetzpläne entspricht das Metaschema nahezu dem Metaschema der Vorgangsknotennetze (*NPVKN*), bei dem die Flußbeziehungen jedoch ausschließlich zwischen Ereignissen verlaufen.

6.4.4 Metaschemata des Kontrollflußparadigmas

Durch die visuellen Sprachen des Kontrollflußparadigmas (vgl. Kapitel 3.4.4) werden Prozesse ebenfalls durch Folgen von Teilprozessen beschrieben. Die Beschreibungsmittel des Kontrollflußparadigmas verwenden im Gegensatz zu den Beschreibungsmittel des Netzparadigmas jedoch ausschließlich reguläre Folgestrukturen. Konzepte zur Abbildung von sequenziellen, parallelen, iterierten und alternativen Prozeßfolgen werden durch das Referenz-Metaschema des Kontrollflußparadigmas bereitgestellt.

Referenz-Metaschema des Kontrollflußparadigmas

Zur Modellierung dieser regulären Folgestrukturen wird im Referenz-Metaschema des Kontrollflußparadigmas *CP* (vgl. auch [Winter / Ebert, 1996, S.112f]) zwischen atomaren Prozessen (*AtomicProcess*) und zusammengesetzten Prozessen unterschieden.

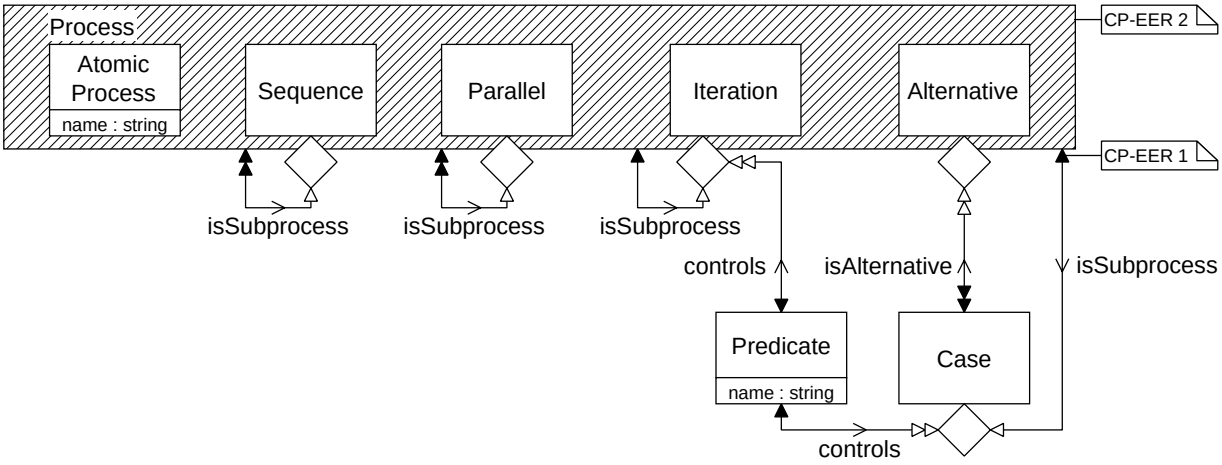


Abbildung 6.19: Referenz-Metaschema des Kontrollflußparadigmas (Control Flow Paradigm,*CP*)

Direkt aufeinander folgende Prozesse werden in Sequenzen (*Sequence*) zusammengefaßt. *Parallel*-Knoten bündeln nebenläufige Prozesse. Zu wiederholende Prozesse werden gemeinsam mit dem Iterationskriterium, das durch ein Prädikat (*Predicate*) abgebildet wird, in *Iteration*-Knoten zusammenfaßt. Die Metamodellierung von Alternativen erfolgt durch *Alternative*-Knoten, in denen die alternativen Fälle (*Case*) gruppiert werden. Diese Fälle fassen je ein Prädikat (*Predicate*) zur Festlegung der jeweiligen Alternative und einen auszuführenden Prozeß zusammen. Die Komposition von Teilprozessen, die selbst durch reguläre Strukturen untergliedert sein können,

erfolgt durch *isSubprocess*-Kanten. Diese Zerlegungen in Teilprozesse entlang der *isSubprocess*- und die *isAlternative*-Kanten sind zyklenfrei[4] [CP 1].

for G **in** CP **assert**
[CP 1]: $isDag(eGraph(E_{isSubprocess} \cup E_{isAlternative}))$
end.

Spezialisierungen des Referenz-Metaschemas

Die in Kapitel 3.4.4 aufgeführten Beschreibungsmittel des Kontrollflußparadigmas können durch marginal angepaßte Spezialisierungen des Referenz-Metaschemas des Kontrollflußparadigmas abgebildet werden. Das Referenz-Metaschema kann unverändert zur Metamodellierung von Nassi-Shneiderman-Diagrammen und von Darstellungen in Pseudo-Code, wie er beispielsweise in Abbildung 3.29a (Seite 85) genutzt wurde, verwendet werden. Diese Darstellungsmittel unterstützen die reguläre Prozeßzerlegung durch Sequenzen, Parallelbearbeitungen, Iterationen und Alternativen. Pseudo-Code Varianten sind gelegentlich aber auch nur auf einzelne dieser Strukturierungsmittel begrenzt, so daß in diesen Fällen Einschränkungen an das Referenz-Metaschema erforderlich werden.

Spezialisierungen des Referenz-Metaschemas des Kontrollflußparadigmas zur Abbildung von Jackson-Diagramme, Warnier-Orr-Diagramme und Entscheidungstabellen werden in den folgenden Abschnitten skizziert.

Jackson-Diagramme und Warnier-Orr-Diagramme. In Jackson-Diagrammen und Warnier-Orr-Diagrammen werden Prozesse ausschließlich durch Sequenzen, Iterationen und Alternativen strukturiert. Zur Darstellung von nebenläufigen Prozessen existieren hier keine Darstellungsmittel. *Parallel*-Knoten sind daher in der Spezialisierung des Referenz-Metaschemas für Jackson-Diagramme und Warnier-Orr-Diagramme (*CPJWO*) auszuschließen [CPJWO 1].

CPJWO **isA** *CP* ;
for G **in** *CPJWO* **assert**
[CPJWO 1]:
$$V_{Parallel} = \varnothing$$
end.

Entscheidungstabellen. Entscheidungstabellen beschreiben ausschließlich alternative Prozeßfolgen. Strukturierungen durch Sequenzen, Parallelbearbeitungen und Iterationen werden in Entscheidungstabellen nicht abgebildet.

Die Spezialisierung des Referenz-Metaschemas des Kontrollflußparadigmas zur Abbildung von Entscheidungstabellen (*CPTable*) schließt daher die Konzepte *Sequence*, *Parallel* und *Iteration* aus [CPTable 1]. Eine Entscheidungstabelle besteht aus genau einer Alternativen [CPTable 2],

[4] In den konkreten Notationen des Kontrollflußparadigmas werden diese zyklenfreien Strukturen i. allg. durch Duplizieren mehrfach genutzter Prozesse auf baumartige Strukturen abgebildet.

die alle Prozesse in alternativen Fällen zusammenfaßt [CPTable 3]. Im Gegensatz zum Referenz-Metaschema des Kontrollflußparadigmas können die einzelnen Fälle hierbei beliebig viele Prozesse enthalten. Die Kardinalitätsanforderung des *EER*-Teils an *isSubprocess*-Kanten ist daher durch [CPTable 4] zu überschreiben.

CPTable **isA** *CP* ;
for *G* **in** *CPTable* **assert**
[CPTable 1]:
$$V_{Sequence} = V_{Parallel} = V_{Iteration} = \varnothing ;$$
[CPTable 2]:
$$\# V_{Alternative} = 1 ;$$
[CPTable 3]:
$$\exists_1 a : V_{Alternative} \bullet (\forall p : V_{AtomicProcess} \bullet p \rightarrow_{isSubprocess} \rightarrow_{isAlternative} a) ;$$
[CPTable 4]:
overwrites [CP-EER 1]:
$$\forall c : V_{Case} \bullet 1 \leq \delta_{\rightarrow_{isSubprocess}}(c)$$
end .

6.5 Metaschemata der Objektsicht

Visuelle Modellierungssprachen zur Darstellung von Systemstrukturen durch *Objekte* und deren *Beziehungen* sind in den Metaschemata der Objektsicht zusammengefaßt. Hierbei werden Sprachen zur Modellierung dieser Objekt-Beziehungszusammenhänge auf Instanz- und auf Schema-Ebene unterschieden. Die Darstellung exemplarischer Systemstrukturen entlang konkreter oder abstrakter Objekte erfolgt durch die Sprachen des Objekt-Instanzparadigmas. Interaktionsbeziehungen zwischen Objekten zur Darstellung dynamischer Aspekte des Objektverhaltens werden durch die Beschreibungsmittel des Objekt-Interaktionsparadigmas beschrieben. Mit den Sprachen des Objekt-Beziehungsparadigmas werden Objekt-Beziehungsgeflechte schematisch dargestellt.

6.5.1 Metaschemata des Objekt-Instanzparadigmas

Die visuellen Modellierungssprachen des Objekt-Instanzparadigmas stellen statische Systemzusammenhänge exemplarisch dar. Hierzu werden mögliche strukturelle Beziehungsgeflechte zwischen Objekten beschrieben. Das Referenz-Metaschemata stellt Konzepte zur Darstellung dieser Objekte und den hierzwischen vorliegenden Beziehungen bereit. Ebenso beschreibt es Attributstrukturen zur Konkretisierung von Objekten und Beziehungen.

Referenz-Metaschema des Objekt-Instanzparadigmas

Wesentliche Konzepte des Referenz-Metaschemas des Objekt-Instanzparadigmas sind Objekte (*Object*) und hierzwischen vorliegenden Beziehungen (*Relationship*), die zu Instanzobjekten (InstanceItem) zusammengefaßt sind. Im Referenz-Metaschema werden Beziehungen zwischen Objekten als binär aufgefaßt. Ein *Relationship*-Knoten ist hierzu über abstrakte *iRelates*-Kanten mit genau zwei Objekten verbunden. Zur Modellierung von reflexiven Beziehungen (Schlingen) sind diese Kanten nicht injektiv. Beziehungen werden auch hier als gerichtet aufgefaßt. Die Ausrichtung der Beziehungen wird durch *iComesFrom*- und *iGoesTo*-Kanten abgebildet.

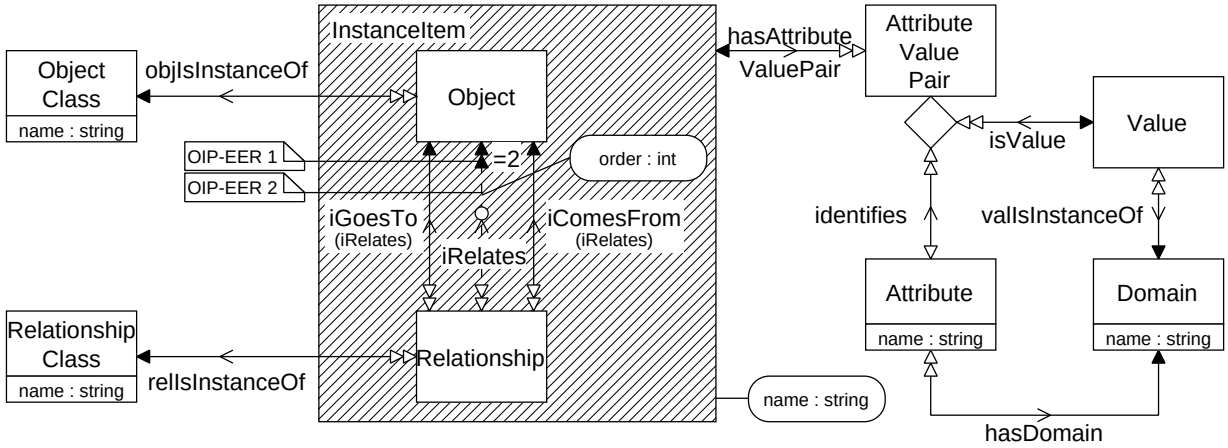


Abbildung 6.20: Referenz-Metaschema des Objekt-Instanzparadigmas (Object Instance Paradigm, *OIP*)

Zusätzlich können sowohl Objekte als auch Beziehungen attributiert werden. Hierzu werden ihnen Attribut-Wert-Paare (*AttributeValuePair*) zugeordnet, die Attribute (*Attribute*) und ihre Werte (*Value*) zusammenfassen. Den Attributen wird durch *vaiIsInstanceOf*-Kanten ihr Wertebereich *Domain* zugeordnet. Die Typisierung von Objekten (*Object*) und Beziehungen (*Relationship*) erfolgt durch *objIsInstanceOf* bzw. *relIsInstanceOf*-Kanten zu Objekt- und Beziehungsklassen (*ObjectClass* bzw. *RelationshipClass*).

Der *EER*-Teil des Referenz-Metaschemas des Objekt-Instanzparadigmas (*OIP*) wird in Abbildung 6.20 definiert. Zusätzliche textuelle Zusicherungen werden zur Abbildung der Anordnung der modellierten Beziehungen nötig. Die Anordnung einer Beziehung relativ zu einem Objekt, kann am *order*-Attribut der *iRelates*-Kanten abgelesen werden. Hierbei ist zuzusichern, daß diese Ordnung für alle *iRelates*-Kanten, die zu einem Objekt inzident sind, einen Wert zwischen 1 und dem Eingangsgrad dieses Objekts annimmt [OIP 1], und daß dieser Wert für alle zu diesem Objekt inzidenten *iRelates*-Kanten identifizierend ist [OIP 2]. Für die Attributwerte ist zuzusichern, daß diese dem Wertebereich des Attributs entsprechen [OIP 3].

for *G* **in** *OIP* **assert**

[OIP 1]:

$$\forall\, e : E_{iRelates} \bullet 1 \leq e.order \leq \delta_{\angle\_{iRelates}}(\omega(e)) ;$$

[OIP 2]:

$$\forall\, e_1, e_2 : E_{iRelates} \mid e_1 \neq e_2 \wedge \omega(e_1) = \omega(e_2) \bullet e_1.order \neq e_2.order\ ;$$

[OIP 3]:

$$\forall\, a : V_{Attribute};\ v : V_{Value} \mid a \rightharpoonup_{identifies} \leftharpoonup_{isValue} v \bullet a \rightharpoonup_{hasDomain} \leftharpoonup_{valIsInstanceOf} v$$

end .

Spezialisierungen des Referenz-Metaschemas

Die Objektdiagramme des *EER/GRAL*-Ansatzes (vgl. auch Abbildung 3.30, S. 87) folgen dem Referenz-Metaschema des Objekt-Instanzparadigmas. Diese graphbasierten Darstellungsmittel verwenden, wie im Referenz-Metaschema definiert, endliche, gerichtete, typisierte, attributierte, angeordnete Graphen. Relationen sind hier ausschließlich binär.

Für die Metaschemata der Instanzdarstellungen der Object Modeling Technique und der Unified Modeling Language ist das Referenz-Metaschema des Objekt-Instanzparadigmas leicht zu erweitern.

Objektdiagramme der Object Modeling Technique und der Unified Modeling Language. Objektdiagramme der Object Modeling Technique [Rumbaugh et al., 1991] und der Unified Modeling Language [Booch et al., 1999] erlauben auch Beziehungen beliebiger Arität. Zur Abbildung dieser Anforderungen ist die Kardinalität der *iRelates*-Kanten nach oben unbeschränkt. Die Kardinalitätsanforderungen des *EER*-Teils des Referenz-Metaschemas für *iRelates*-Kanten ist daher zu überschreiben [OIPOO 1]. Da Beziehungen in diesen Dialekten generell als ungerichtet aufgefaßt werden, kann hier auch auf die spezialisierten *iComesFrom*- und *iGoesTo*-Kanten verzichtet werden [OIPOO 2]. Die im Referenz-Metaschema als abstrakt formalisierte *iRelates*-Kanten sind daher auch als konkret zu vereinbaren [OIPOO 3]. Die Spezialisierung des Referenz-Metaschemas des Objekt-Instanzparadigmas zur Modellierung dieser Beschreibungsmittel ist im Metaschema *OIPOO* zusammengefaßt. Neben diesen Anforderungen unterstützen die Objektdiagramme der Object Modeling Technique und der Unified Modeling Language die Anordnung der Beziehungen nicht. Das Attribut *order* der *iRelates*-Kanten bleibt folglich unberücksichtigt [OIPOO 4].

OIPOO **isA** *OIP* ;

for *G* **in** *OIPOO* **assert**

[OIPOO 1]:

overwrites [OIP-EER 1]:

$$\forall\, r : Relationship \bullet 2 \leq \delta_{\rightharpoonup_{iRelates}}(r)\ ;$$

[OIPOO 2]:

$$E_{iComesFrom} = E_{iGoesTo} = \varnothing\ ;$$

[OIPOO 3]:

overwrites [OIP-EER 2]:

$isConcrete(iRelates)$;

[OIPOO 4]:

$$\forall\, i : E_{iRelates} \bullet i.order = \perp$$

end.

6.5.2 Metaschemata des Objekt-Interaktionsparadigmas

Zur Darstellung des inneren Verhaltens von Systemen werden die visuellen Sprachen des Objekt-Interaktionsparadigmas (vgl. Kapitel 3.5.2) verwendet. Dieses dynamische Verhalten von Objektgeflechten wird exemplarisch entlang des Nachrichtenaustauschs zwischen den Objekten des Systems notiert. Im Metaschema des Objekt-Interaktionsparadigmas sind daher die Konzepte zur Modellierung von Objekten, Nachrichten und deren Querbezüge abzubilden.

Referenz-Metaschema des Objekt-Interaktionsparadigmas

Das Referenz-Metaschema des Objekt-Interaktionsparadigmas *IAP* ist in seinem *EER*-Teil in Abbildung 6.21 dargestellt. Zentrale Konzepte des Objekt-Interaktionsparadigmas sind Objekte (*Object*) und hierzwischen ausgetauschte Nachrichten (*Message*). Objekte werden hierbei als Instanzen einer Objektklasse (*ObjectClass*) aufgefaßt. Die Beschreibungsmittel des Objekt-Interaktionsparadigmas beschreiben exemplarische Szenarien der Objektinteraktion. Die in einem solchen Szenario verschickten Nachrichten werden in einem Interaktions-Szenario (*InteractionScenario*) zusammengefaßt.

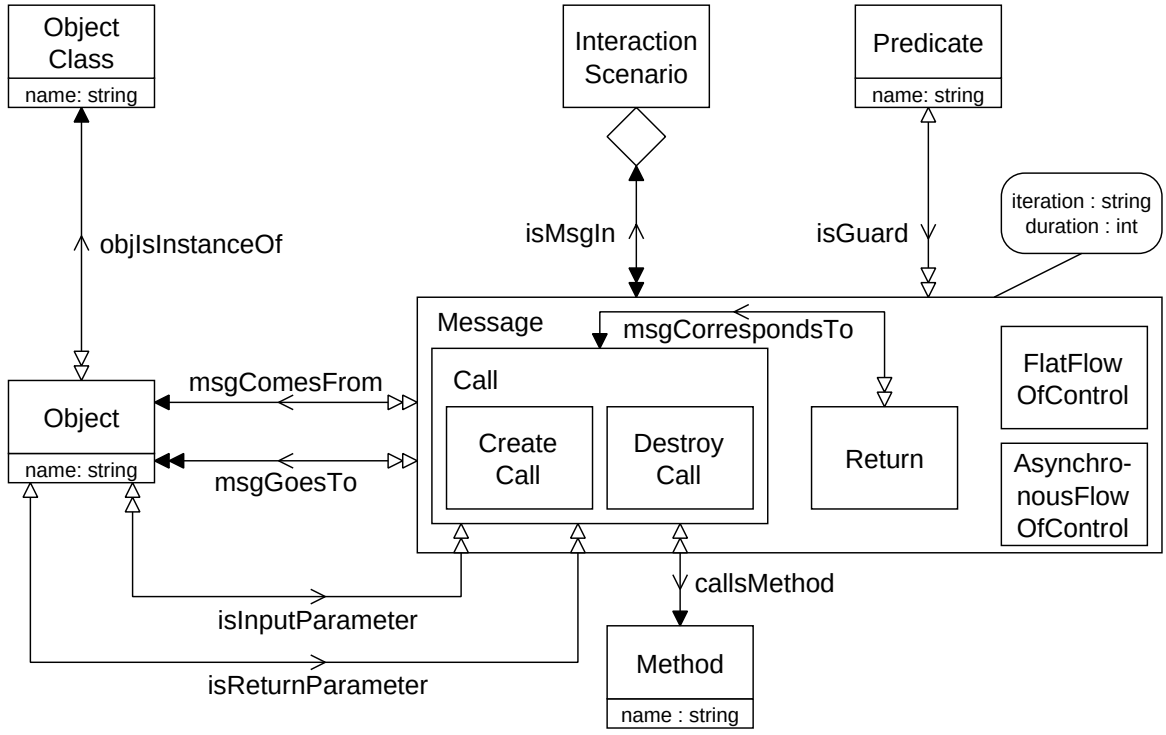


Abbildung 6.21: Referenz-Metaschema des Objekt-Interaktionsparadigmas (Interaction Paradigm, *IAP*)

Im Referenz-Metaschema werden Nachrichtentypen zur Modellierung von Methodenaufrufen (*Call*), von Rückgabenachrichten (*Return*), von synchronen und von asynchronen Nachrichten zur Weitergabe der Systemkontrolle an andere Objekte (*FlatFlowOfControl*, *Asynchronous-FlowOfControl*) unterschieden (vgl. auch [Rumbaugh et al., 1999, S. 336]). Gegenüber dem Teil-Metaschema des Kollaborationspakets der Unified Modeling Language [OMG, 1999, S. 100] bildet das Referenz-Metaschema des Objekt-Interaktionsparadigmas somit auch die in der UML üblichen Nachrichtentypen ab. Rückgabe-Nachrichten beziehen sich hierbei jeweils auf einen (zuvor erfolgen) Methodenaufruf (*msgCorrespondsTo*). Korrespondierende Methodenaufrufe und Rückgaben werden zwischen den selben Objekten ausgetauscht [IAP 1].

Aufgerufene Methoden (*Method*) werden durch Kanten des Typs *callsMethod* der Nachricht zugeordnet. Spezielle Methodenaufrufe zur Objektkonstruktion (*CreateCall*) und zur Objektzerstörung (*DestroyCall*) und zur Wertrückgabe mit anschließender Objektzerstörung sind als Spezialisierungen der *Call*-Nachrichten modelliert. Beim Austausch von Nachrichten zum Methodenaufruf können Parameter übergeben werden (*isInputParameter* und *isReturnParameter*), die wieder durch Objekte modelliert sind. Nachrichten werden nach dem Referenz-Metaschema gerichtet zwischen Objekten ausgetauscht. Sender und Empfänger sind anhand der *msgComesFrom*- und *msgGoesTo*-Kanten erkennbar. Die Reihenfolge der ausgetauschten Nachrichten ist durch die Anordnung der Inzidenzen der *isMsgIn*-Kanten zu *InteractionScenario*-Knoten modelliert. Hierbei ist sicherzustellen, daß Nachrichten zur Wertrückgabe zeitlich nach dem Methodenaufruf erfolgen [IAP 2], und daß von dem Aufruferobjekt erst nach Erhalt der Rückgabenachricht weitere Nachrichten versendet oder empfangen werden [IAP 3] [Rumbaugh et al., 1999, S. 336].

for G **in** *IAP* **assert**

[IAP 1]:

$$\forall\, c : V_{Call};\ r : V_{Return} \mid r \rightharpoonup_{msgCorrespondsTo} c \bullet$$
$$c \rightharpoonup_{msgComesFrom} = r \rightharpoonup_{msgGoesTo} \wedge\ c \rightharpoonup_{msgGoesTo} \supseteq r \rightharpoonup_{msgComesFrom};$$

[IAP 2]:

$$\forall\, e_1, e_2 : E_{isMsgIn} \mid \omega(e_1) = \omega(e_2) \wedge \alpha(e_1) \leftharpoonup_{msgCorrespondsTo} \alpha(e_2) \bullet$$
$$order_{isMsgIn}(e_1, \omega(e_1)) < order_{isMsgIn}(e_2, \omega(e_2));$$

[IAP 3]:

$$\forall\, e_1, e_2 : E_{isMsgIn} \mid \omega(e_1) = \omega(e_2) \wedge \alpha(e_1) \leftharpoonup_{msgCorrespondsTo} \alpha(e_2) \bullet$$
$$\neg\ \exists\, e_3 : E_{isMsgIn} \mid \omega(e_1) = \omega(e_2) = \omega(e_3) \wedge$$
$$order_{isMsgIn}(e_1, \omega(e_1)) < order_{isMsgIn}(e_3, \omega(e_3)) <$$
$$order_{isMsgIn}(e_2, \omega(e_2)) \bullet$$
$$\alpha(e_1) \rightharpoonup_{msgComesFrom} (\leftharpoonup_{msgComesFrom} | \leftharpoonup_{msgGoesTo}) \alpha(e_3)$$

end.

Der Nachrichtenaustausch zwischen Objekten kann durch Prädikate (*Predicate*) überwacht werden. Nachrichten werden nur dann verschickt, wenn diese Wächter erfüllt sind. Auch können Nachrichten iteriert verschickt werden. Das Iterationskriterium ist im Metamodell als Attribut *iteration* der Nachrichten modelliert. Der Zeitbedarf zum Nachrichtenaustausch zwischen Objekten wird durch das *duration*-Attribut modelliert.

Spezialisierungen des Referenz-Metaschemas

Sequenz- oder Ereignispfaddiagramme und Kollaborations- oder Ereignisflußdiagramme, wie sie in der Unified Modeling Language (UML) [Rumbaugh et al., 1999], bei [Booch, 1994] oder bei [Rumbaugh et al., 1991] verwendet werden, entsprechen weitestgehend dem Referenz-Metaschema *IAP*. Das Sequenz- und Kollaborationsdiagramme semantisch äquivalente Informationen darstellen, basieren sie auf dem gleichen Metaschema. Für die Diagramme der UML kann das Metaschema unverändert übernommen werden, die Notationen von [Booch, 1994] oder [Rumbaugh et al., 1991] erfordern minimale Ergänzungungen bzw. Einschränkungen.

Objekt- und Interaktionsdiagramme. Neben den im Referenz-Metaschema des Objekt-Interaktionsparadigmas vorgesehenen Nachrichtenarten unterscheidet [Booch, 1994] weitere Typen. Zusätzlich sieht er noch Nachrichten vor, die durch das Empfängerobjekt zurückgewiesen werden können (balking) oder die mit einer Zeitschranke (timeout) versehen sind. Zur Beschreibung des Metaschemas sind diese Nachrichtentypen als weitere Spezialisierung der *calls* zu ergänzen. Ein entsprechendes Metaschema der Objekt- und Interaktionsdiagramme nach [Booch, 1994] wird in [Süttenbach / Ebert, 1997, S. 32] dargestellt.

Ereignispfad- und Ereignisflußdiagramme. Der in der Objekt Modeling Technique (OMT) [Rumbaugh et al., 1991] verwendete Dialekt der Ereignispfad- und Ereignisflußdiagramme beschreibt nur die Interaktionen zwischen Objekten durch einfachen Nachrichtenaustausch. Verschiedene Nachrichtentypen werden hier nicht unterschieden [IAPOMT 1].

IAPRumbaugh **isA** *IAP* ;

for *G* **in** *DFPContext* **assert**

[IAPOMT 1]:

$$V_{Call} = V_{Return} = V_{FlatFlowOfControl} = V_{AsynchronousFlowOfControl} = \varnothing$$

end .

6.5.3 Metaschemata des Objekt-Beziehungsparadigmas

Im Gegensatz zu dem Beschreibungsmitteln des Objekt-Instanzparadigmas, durch die Objektzusammenhänge aus Sicht konkreter Instanzen beschrieben werden, werden durch die visuellen Modellierunssprachen des Objekt-Beziehungsparadigmas Objekt- und Datenstrukturen dargestellt. Objektklassen beschreiben Mengen oder Klassen gleichartiger Objekte, die auf Schemaebene zueinander in Beziehung gesetzt werden. Mengen gleichartiger Beziehungen, die zwischen Objekten konkreter Objektklassen vorliegen, werden durch Beziehungsklassen dargestellt. Zur Abbildung der Sprachen des Objekt-Beziehungsparadigmas werden im Referenz-Metaschema des Objekt-Beziehungsparadigmas Konzepte zur Beschreibung von Objektklassen und Beziehungsklassen, deren Attributstrukturen und deren Querbezüge definiert. Zusätzlich bieten die moderneren Ansätze zur Objekt-Beziehungsmodellierung Mittel zur Strukturierung von Objektzusammenhängen durch Generalisierung, Aggregation und Gruppierung an. Diese Strukturierungsmittel werden im Referenz-Metaschema ebenfalls abgebildet.

Referenz-Metaschema des Objekt-Beziehungsparadigmas

Der *EER*-Teil des Referenz-Metaschemas des Objekt-Beziehungsparadigmas (*ORP*) ist in Abbildung 6.22 dargestellt (vgl. hierzu auch die Metaschemata in [Winter / Ebert, 1996, S. 116ff], [Dahm et al., 1998b] und [Ebert et al., 1999a]).

Objekt- und Beziehungsklassen werden durch die Konzepte *ObjectClass* und *RelationshipClass*, die im Konzept *ClassItem* zusammengefaßt sind, modelliert. Passend zu den Beziehungen des Referenz-Metaschemas des Objekt-Instanzparadigmas modellieren die Beziehungsklassen des Objekt-Beziehungsparadigmas Klassen gerichteter, binärer Beziehungen. Der Bezug zwischen der Beziehungsklasse und den beiden in Beziehung gesetzten Objektklassen wird hierzu durch (abstrakte) *cRelates*-Kanten hergestellt, die zur Modellierung der Ausrichtung durch *cComes-From-* und *cGoesTo*-Kanten spezialisiert sind. Da Objektklassen auch zu sich selbst in Beziehung gesetzt werden können, sind die *cRelates*-Kanten als nicht injektiv zu vereinbaren.

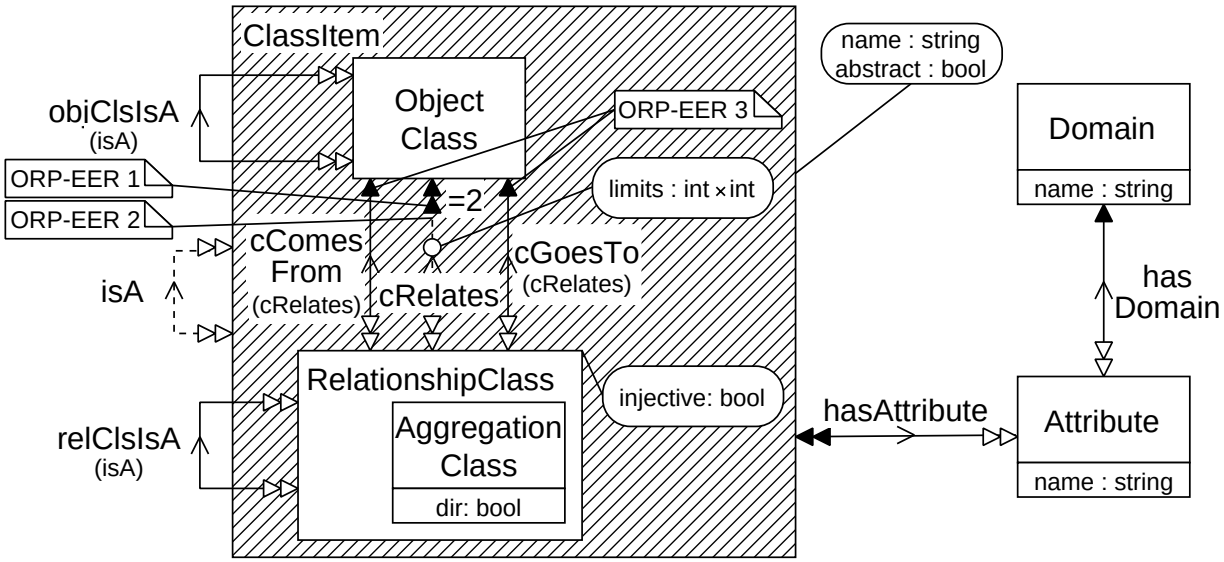


Abbildung 6.22: Referenz-Metaschema des Objekt-Beziehungsparadigmas (Object-Relationship Paradigm, *ORP*)

Die Kardinalitäten der Beziehungsklassen werden durch die *limits*-Attribute der *cRelates*-Kanten festgelegt. In *limits* können in Anlehnung an die Min/Max-Notation, die Mindest- und Maximalanzahl der Beziehungen notiert werden, die Instanzen der beteiligten Objektklassen eingehen können. Klassen injektiver Beziehungen werden durch das Attribut *injective* der entsprechenden *RelationshipClass*-Knoten ausgezeichnet. Durch das Attribut *abstract* können sowohl Objekt- als auch Beziehungsklassen als abstrakte Klassen markiert werden.

Objekte und Beziehungen können durch Attribut-Werte-Paare näher konkretisiert werden. Zur Festlegung dieser Attributstrukturen werden Objekt- und Beziehungsklassen um Attribute (*Attribute*) einschließlich deren (optionalen) Wertebereiche (*Domain*) ergänzt.

Das Referenz-Metaschema des Objekt-Beziehungsparadigmas unterstützt neben diesen Grundformen der Objekt-Beziehungsmodellierung auch die Modellierungskonstrukte der erweiterten Objekt-Beziehungsmodellierung. Generalisierungen sowohl von Objekt- als auch von Beziehungsklassen werden durch (abstrakte) *isA*-Kanten bzw. durch deren Spezialisierungen model-

liert. Durch *objClsIsA*- bzw. *relClsIsA*-Kanten werden jeweils ausschließlich Objekt- oder Beziehungsklassen generalisiert.

Für die Spezialisierung von Beziehungsklassen ist darüber hinaus zu fordern, daß die Ziel- und Endklassen der Spezialisierungen auch mit den Ziel- und Endklassen der Generalisierung verträglich sind [ORP 1]. Kardinalitätsangaben dürfen durch Spezialisierungen von Beziehungsklassen nur eingeschränkt werden [ORP 2] um den in der Generalisierung definierten Kardinalitäten zu genügen. Durch [ORP 3] wird schließlich gefordert, daß es zu abstrakten Objekt- oder Beziehungsklassen in der Vererbungshierarchie auch konkrete Spezialisierungen geben muß.

for G **in** *ORP* **assert**

[ORP 1]:

$$\forall e, f : V_{RelationshipClass};\ a, b, c, d : V_{ObjectClass} \mid e \rightharpoonup_{relClsIsA} f \wedge a \leftharpoonup_{cComesFrom} e \rightharpoonup_{cGoesTo} b \wedge c \leftharpoonup_{cComesFrom} f \rightharpoonup_{cGoesTo} d \bullet a \rightharpoonup^{*}_{objClsIsA} c \wedge b \rightharpoonup^{*}_{objClsIsA} d ;$$

[ORP 2]:

$$\forall e, f : E_{cRelates} \mid \alpha(e) \rightharpoonup_{relClsIsA} \alpha(f) \wedge \omega(e) \rightharpoonup^{*}_{objClsIsA} \omega(f) \bullet first(e.limits) \geq first(f.limits) \wedge second(e.limits) \leq second(f.limits) ;$$

[ORP 3]:

$$\forall t_1 : V_{ClassItem} \mid t_1.abstract = \mathsf{true} \bullet \exists t_2 : V_{ClassItem} \mid t_2 \rightharpoonup^{+}_{isA} t_1 \bullet t_2.abstract = \mathsf{false}$$

end.

Gruppierungen und Aggregationen werden als spezielle Beziehungen (*AggregationClass*) aufgefaßt. Für diese Beziehungen werden Kardinalitätsinformationen analog zu den *RelationshipClass*-Kanten abgebildet, so daß im Metaschema eine Sonderbehandlung für Gruppierungen entfallen kann. Durch *cComesFrom*- und *cGoesTo*-Kanten wird für *AggregationClass*-Beziehungen Aggregat und Aggregation unterschieden. Die *cComesFrom*-Kanten zeigen jeweils zum Aggregat und die *cGoesTo*-Kante zur Aggregation. Die Ausrichtung der eigentlichen vom Aggregat zur Aggregation bzw. von der Aggregation zum Aggregat wird durch das *dir*-Attribut[5] modelliert.

Objekt-Beziehungsmodellierungen durch *EER/GRAL*-Klassendiagramme genügen dem zuvor vorgestellten Referenz-Metaschema. Das Referenz-Metaschema des Objekt-Beziehungsparadigmas definiert somit auch das Metaschema aller in dieser Arbeit notierten Metaschemata und kann somit als *Meta-Metaschema* zur Modellierung der Beschreibungsmittel für Organisationen und Softwaresysteme eingeordnet werden. Die Z-Spezifikation des *EER/GRAL*-Ansatzes in Kapitel 5.2 und das Referenz-Metaschema des Objekt-Beziehungsparadigmas lassen sich ineinander überführen. Die Menge der Typbezeichner des *EER*-Schemas (*types*) ist hierzu auf die Extension des Konzepts *ClassItem* abzubilden. Ebenso entsprechen die Mengen des *EER*-Schemas (*EERSchema*) zur Formalisierung der Knotentypbezeichner (*entityTypes*), der Beziehungstypbezeichner (*relationshipTypes*), der Rollentypbezeichner (*roleTypes*) und der Attributbezeichner (*AttrId*) den Extensionen von *ObjectClass*, *RelationshipClass*, *AggregationClass* und *Attribute*. Die Relationen und Mengen des *EERSchema* zur Spezifikation des Typsystems (*TypeSystem*),

[5] Für $dir = \mathsf{true}$ zeigt die Aggregationskante parallel zur Ausrichtung der *cComesFrom*- und *cGoesTo*-Kanten zur Aggregation, andernfalls zum Aggregat.

des Inzidenzsystems (*IncidenceSystem*) und des Invariantensystems (*InvariantSystem*) finden ihre Entsprechungen in den Beziehungstypen und Attributstrukturen des Referenz-Metaschemas des Objekt-Beziehungsparadigmas.

Spezialisierungen des Referenz-Metaschemas

Metaschemata weiterer Beschreibungsmittel des Objekt-Beziehungsparadigmas lassen sich ebenfalls durch kleinere Anpassungen aus diesem Referenz-Metaschema ableiten.

Datenlexika. Durch Datenlexika werden Objektstrukturen entlang regulärer Strukturen durch Objektsequenzen, -alternativen und -iterationen beschrieben (vgl. auch die Metamodellierung des Konzepts *ObjectClass* im Referenz-Metaschema des Datenflußparadigmas in Abbildung 6.7; Seite 180). Dieses Metaschema kann auf das Referenz-Metaschema des Objekt-Beziehungsparadigmas zurückgeführt werden, indem Sequenzen als Aggregationen, Alternativen als Generalisierungen und Iterationen als Gruppierungen aufgefaßt werden.

Abbildung 6.23 skizziert das Metaschema für Datenlexika (*ORPDD*). Aggregationen und Gruppierungen werden konzeptionell unterschieden. Das Metaschema für Datenlexika sieht daher für Gruppierungen (*GroupingClass*) und Aggregationen (*AggregationClass*) eigene Konzepte vor. Die Kardinalitäten dieser Gruppierungs- und Aggregationsbeziehungen sind in Datenlexika auf der Instanzebene festgelegt. Gruppierungen sind beliebig viele Komponenten-Objekte zugeordnet [ORPDD 1] und Aggregationen in jeder Rolle genau ein Aggregat [ORPDD 2]. Komponenten bzw. Aggregate können jedoch an beliebig vielen Gruppierungen oder Aggregationen beteiligt sein.

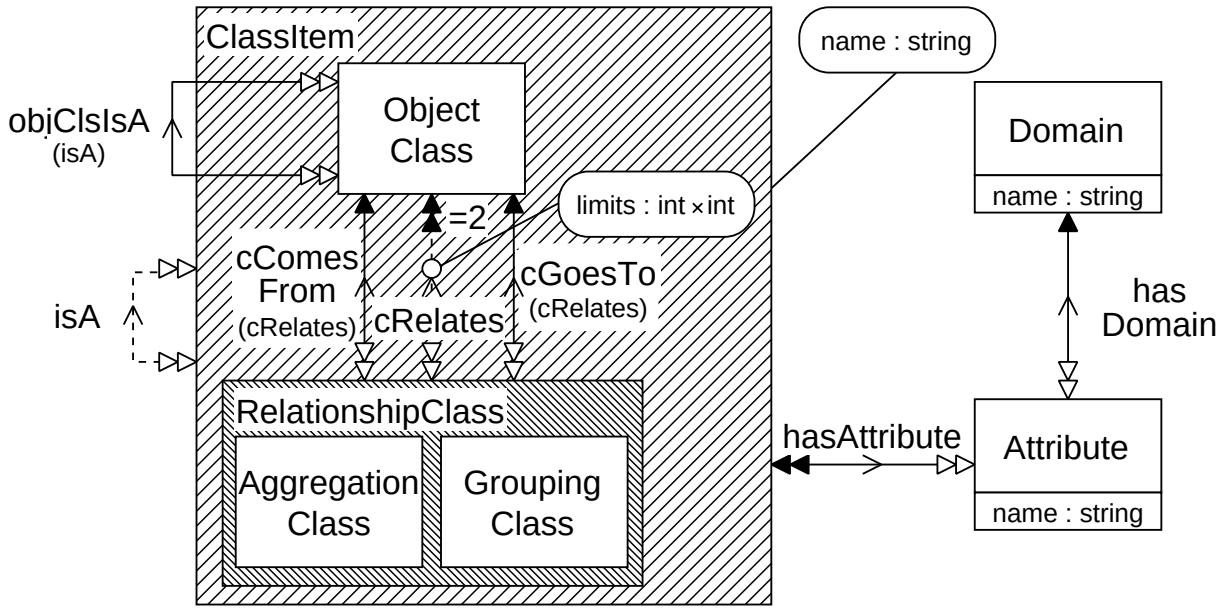


Abbildung 6.23: Spezialisierung des Referenz-Metaschemas des Objekt-Beziehungsparadigmas für Datenlexika (*ORPDD*)

Für Gruppierungen sind auch Kardinalitätseinschränkungen auf der Metaebene zu fordern. Einer Objektklasse, die eine Gruppierung beschreibt, ist genau eine Objektklasse zur Abbildung der Komponenten zugeordnet. Weitere gruppierte oder aggregierte Komponenten sind für Gruppierungen auszuschließen [ORPDD 3].

Beziehungsklassen werden in Datenlexika nicht dargestellt, so daß im Metaschema die *RelationshipClass*-Knoten als abstrakt vereinbart werden. Auch können Beziehungsklassen nicht generalisiert werden [ORPDD 4]. Ebenso werden die in Datenlexika definierten Objekt- und Beziehungsklassen generell als konkret und Beziehungsklassen als injektiv angenommen [ORPDD 5].

ORPDD **isA** *ORP* ;

for G **in** *ORPDD* **assert**

[ORPDD 1]:

$$\forall\, c : E_{cComesFrom} \mid type(\alpha(c)) = GroupingClass \bullet first(c.limits) = 0 \wedge second(c.limits) = \infty ;$$

$$\forall\, g : E_{cGoesTo} \mid type(\alpha(g)) = GroupingClass \bullet first(g.limits) = 0 \wedge second(g.limits) = \infty ;$$

[ORPDD 2]:

$$\forall\, c : E_{cComesFrom} \mid type(\alpha(c)) = AggregationClass \bullet first(c.limits) = 1 \wedge second(c.limits) = 1 ;$$

$$\forall\, g : E_{cGoesTo} \mid type(\alpha(g)) = AggregationClass \bullet first(g.limits) = 0 \wedge second(g.limits) = \infty ;$$

[ORPDD 3]:

$$\forall\, o : V_{ObjectClass} \mid o \leftharpoondown_{cGoesTo} \&_{GroupingClass} \bullet \delta_{\leftharpoondown_{cGoesTo}} = 1 ;$$

[ORPDD 4]:

$$E_{relClsIsA} = \varnothing ;$$

[ORPDD 5]:

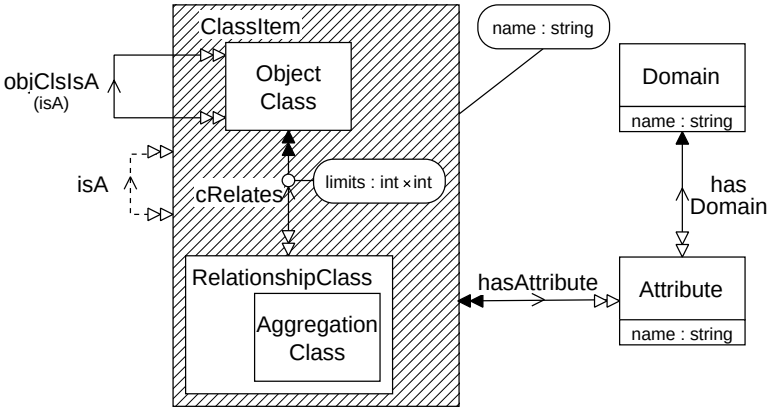
$$\forall\, i : V_{ClassItem} \bullet i.abstract = \mathsf{false} ;$$

$$\forall\, r : V_{RelationshipClass} \bullet r.injective = \mathsf{true}$$

end.

Diesem Metaschema für Datenlexika *ORPDD* genügen auch Beschreibungen zur Objektmodellierung durch Jackson-Bäume und Warnier-Orr-Diagramme, die ebenfalls auf der Modellierung durch reguläre Strukturen basieren.

Entity-Relationshipdiagramme. Entity-Relationshipdiagramme sind die klassische Form der Objekt-Beziehungsmodellierung. Die Spezialisierung des Referenz-Metaschemas des Objekt-Beziehungsparadigmas für Entity-Relationship-Diagramme (*ORPER*) in einer heute gebräuchlichen Form (vgl. z. B. [Vossen, 1994, S. 70ff], [Scheer, 1992], [Elmasri / Navathe, 2000]) ist in Abbildung 6.24 dargestellt.

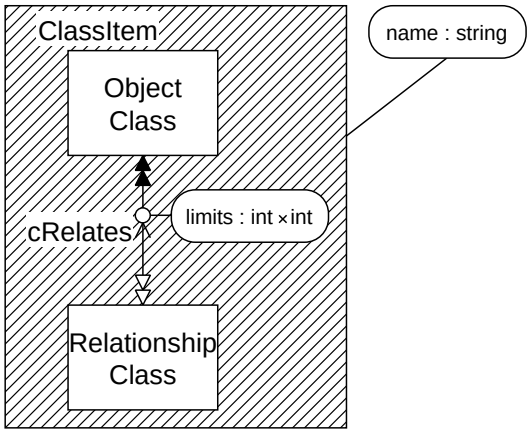


ORPER **isA** *ORP* ;
for *G* **in** *ORPER* **assert**
[ORPER 1]:
$E_{relClsIsA} = \varnothing$;
[ORPER 2]:
overwrites [ORP-EER 1]:
$\forall\, r : V_{Relationship} \bullet 2 \leq \delta_{\rightarrow cRelates}(r)$;
[ORPER 3]:
$E_{cComesFrom} = E_{cGoesTo} = \varnothing$;
[ORPER 4]:
overwrites [ORP-EER 2]:
$isConcrete(cRelates)$;
[ORPER 5]:
$\forall\, c : V_{ClassItem} \bullet c.abstract = \text{false}$;
$\forall\, r : V_{Relationship} \bullet r.injective = \text{true}$
end.

Abbildung 6.24: Spezialisierung des Referenz-Metaschemas des Objekt-Beziehungsparadigmas für Entity-Relationship-Diagramme (*ORPER*)

In diesen Dialekten wird die Generalisierung von Beziehungstypen nicht unterstützt [ORPER 1]. Entity-Relationshipdiagramme erlauben gegenüber dem Referenz-Metaschema auch Beziehungen beliebiger Arität. Die Begrenzung der Kardinalitäten der *cRelates*-Kanten nach oben ist daher aufgehoben [ORPER 2]. Da Beziehungen in diesen Dialekten als ungerichtet aufgefaßt werden, sind die *cComesFrom* und *cGoesTo*-Kanten überflüssig [ORPER 3], und die *cRelates*-Kanten sind als konkret zu vereinbaren [ORPER 4]. Graphischen Beschreibungsmittel zur Unterscheidung injektiver bzw. nicht injektiver Beziehungsklassen und abstrakter bzw. konkreter Objekt- und Beziehungsklassen werden in diesen Dialekten nicht graphisch unterschieden. Auch hier wird generell von konkreten Klassen und injektiven Beziehungsklassen ausgegangen [ORPER 5].

Der in [Chen, 1976] eingeführte ursprüngliche Dialekt zur Entity-Relationship-Modellierung kann auf das Metaschema *ORPER* zurückgeführt werden. In dieser Notation konnten noch keine Generalisierungen, Aggregationen und Attributstrukturen [ERChen 1] modelliert werden.

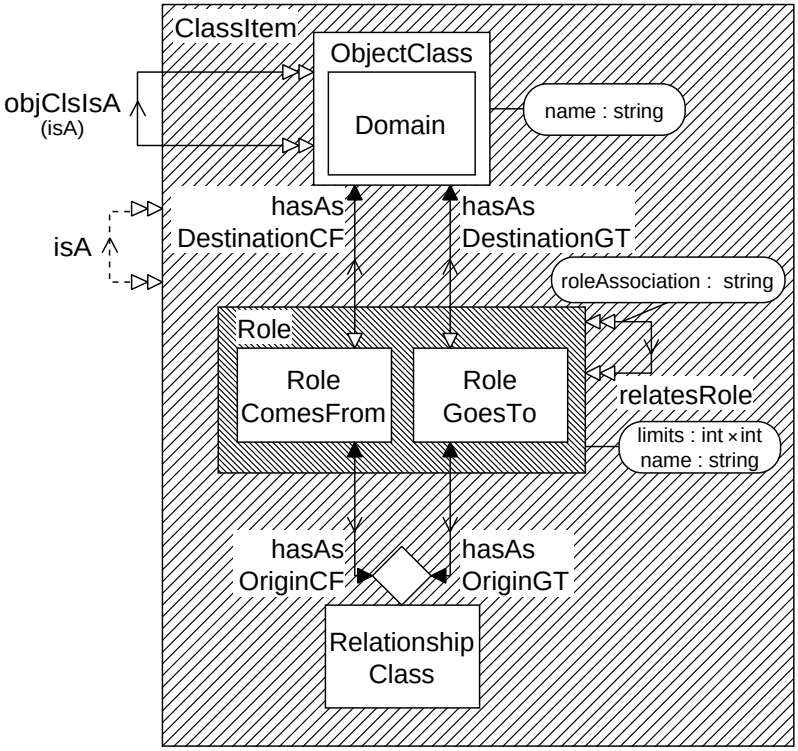


ORPChen **isA** *ORPER* ;
for *G* **in** *ORPChen* **assert**
[ORPChen 1]:
$V_{AggregationClass} = \varnothing$;
$V_{Attribute} = V_{Domain} = \varnothing$;
$E_{isA} = \varnothing$;
[ORPChen 2]:
$\forall\, c : E_{cRelates} \bullet$
$first(c.limits) = 0 \wedge$
$(\, second(c.limits) = 1 \vee$
$second(c.limits) = \infty)$
end.

Abbildung 6.25: Spezialisierung des Referenz-Metaschemas des Objekt-Beziehungsparadigmas für Entity-Relationshipdiagramme nach [Chen, 1976] (*ORPChen*)

Die Kardinalitäten der Beziehungen werden in Entity-Relationshipdiagrammen i. allg. in der M : N-Notation angegeben. Hierzu werden nur $limits$-Paare der Form $(0,1)$ (entspricht 1) und $(0,\infty)$ (entspricht M oder N) zugelassen [ERChen 2].

NIAM-Informationsstruktur-Diagramme. Eine Variante der Entity-Relationshipdiagramme, die ausschließlich binäre Beziehungen zwischen Objekten nutzen, sind NIAM-Informationsstruktur-Diagramme. Beziehungen werden in NIAM ausgehend von den Rollen, die die Objekte in der Beziehung einnehmen, modelliert. Die Bezeichner dieser beiden Rollen benennen die gesamte Beziehung. In NIAM können mit graphischen Mitteln Zusicherungen über den Objektmengen, die an einer Beziehung beteiligt sind, dargestellt werden. In der Spezialisierung des Referenz-Metaschemas des Objekt-Beziehungsparadigmas für NIAM-Informationsstruktur-Diagramme *ORPNiam* (vgl. Abbildung 6.26) werden diese Rollen daher durch die Knotentypen *RoleComesFrom* und *RoleGoesTo* modelliert. Diese ersetzen die *cRelates*-Kanten des Referenz-Metaschemas [ORPNiam 1].



ORPNiam **isA** *ORP* ;

for G **in** *ORPNiam* **assert**

[ORPNiam 1]:
$E_{cRelates} = \varnothing$;

[ORPNiam 2]:
$V_{AggregationClass} = \varnothing$;

[ORPNiam 3]:
$\forall\, c : E_{cRelates} \bullet$
$first(c.limits) = 0 \wedge$
$(\, second(c.limits) = 1 \vee$
$second(c.limits) = \infty)$;

[ORPNiam 4]:
$V_{Attributes} = \varnothing$;

[ORPNiam 5]:
$E_{relClsIsA} = \varnothing$;

[ORPNiam 6]:
$\forall\, c : V_{ClassItem} \bullet$
$c.abstract =$ true ;
$\forall\, r : V_{Relationsship} \bullet$
$r.injective =$ true

end.

Abbildung 6.26: Spezialisierung des Referenz-Metaschemas des Objekt-Beziehungsparadigmas für NIAM-Informationsstruktur-Diagramme (*ORPNiam*)

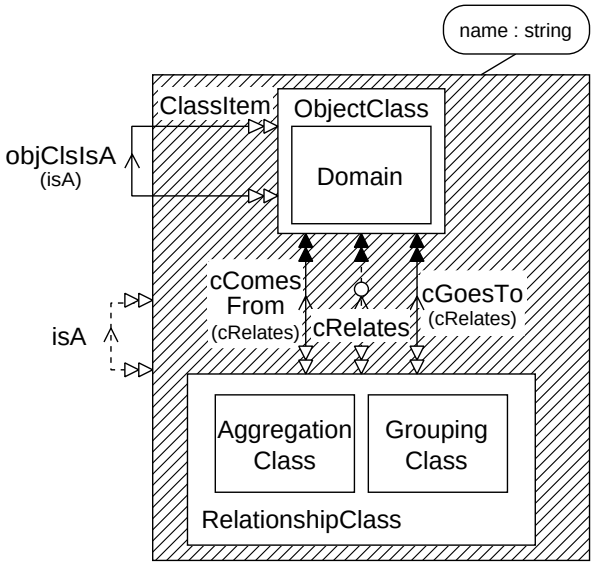
Eine Beziehung entspricht dann der Aggregation einer *RoleComesFrom*- und einer *RoleGoesTo*-Rolle, denen jeweils die in Beziehung gesetzten Objektklassen (*ObjectClass*) zugeordnet sind. Die Beziehungen zwischen den Rollen werden durch *relatesRole*-Kanten beschrieben, die mit der Art der jeweiligen Beziehung attributiert sind. Durch diese Beziehungen kann auch die Aggregation modelliert werden, so daß auch die *AggregationClass*-Spezialisierung der Beziehungsklassen im Metaschema entfallen kann [ORPNiam 2]. Kardinalitäten der Beziehungen werden in NIAM analog zu den Entity-Relationshipdiagrammen in einer graphischen Form der M : N-

Notation dargestellt [ORPNiam 3]. Diese Kardinalitätsangaben werden im Metaschema ebenfalls an den *Role*-Knoten abgetragen.

Wertebereiche von Attributen und Objektklassen werden in NIAM nicht unterschieden. Sowohl Attributzuordnungen als auch Beziehungen werden entlang des *RelationshipClass*-Konzepts modelliert. Jede Beziehung kann als Attributzuordnung aufgefaßt werden, so daß im NIAM-Metaschema gegenüber dem Referenz-Metaschema das Attributkonzept entfällt [ORPNiam 4]. Die Wertebereiche der Attribute (*domain*) des Referenz-Metaschemas werden im NIAM-Metaschema als Spezialisierung der Objektklassen *ObjectClass* aufgefaßt. Diese Attributwertebereiche umfassen ausschließlich druckbare Standardtypen.

Generalisierungen werden in NIAM nur zwischen Objektklassen zugelassen. Das Metaschema der NIAM-Informationsstrukturdiagramme enthält daher auch keine *relClsIsA*-Kanten [ORPNiam 5]. Abstrakte Klassen und nicht-injektive Beziehungsklassen werden auch in NIAM nicht berücksichtigt [ORPNiam 6].

Generische Semantische Modelle. In Dialekt der Generischen Semantischen Modelle werden gerichtete Beziehungen beliebiger Arität unterstützt. Die Richtung der Beziehungen wird auch hier durch die *cComesFrom*- und *cGoesTo*-Kanten abgebildet. Zur Modellierung beliebiger Aritäten sind in der Spezialisierung des Referenz-Metaschemas für Generische Semantische Modelle (*ORPGSM*) in Abbildung 6.27 die Kardinalitäten der *cRelates*-Kanten anzupassen.



ORPGSM **isA** *ORP* ;

for *G* **in** *ORPDD* **assert**

[ORPGSM 1]:
overwrites [ORP-EER 1]:
$\forall\ r : V_{Relationship} \bullet 2 \leq \delta_{\rightarrow cRelates}(r)$;
overwrites [ORP-EER 3]:
$\forall\ r : V_{Relationship} \bullet 1 \leq \delta_{\rightarrow cComesFrom}(r) \wedge 1 \leq \delta_{\rightarrow cGoesTo}(r)$;

[ORPGSM 2]:
$E_{relClsIsA} = \varnothing$;

[ORPGSM 3]:
$\forall\ c : E_{cComesFrom} \mid type(\alpha(c)) = GroupingClass \bullet first(c.limits) = 0 \wedge second(c.limits) = \infty$;
$\forall\ g : E_{cGoesTo} \mid type(\alpha(g)) = GroupingClass \bullet first(g.limits) = 0 \wedge second(g.limits) = \infty$;

[ORPGSM 4]:
$\forall\ c : E_{cComesFrom} \mid type(\alpha(c)) = AggregationClass \bullet first(c.limits) = 1 \wedge second(c.limits) = 1$;
$\forall\ g : E_{cGoesTo} \mid type(\alpha(g)) = AggregationClass \bullet first(g.limits) = 0 \wedge second(g.limits) = \infty$;

[ORPGSM 5]:
$\forall\ o : V_{ObjectClass} \mid o \leftarrow_{cGoesTo} \&_{GroupingClass} \bullet \delta_{\leftarrow cGoesTo}(o) = 1$;

[ORPGSM 6]:
$V_{Attribute} = \varnothing$;

[ORPGSM 7]:
$\forall\ c : V_{ClassItem} \bullet c.abstract =$ false ;
$\forall\ r : V_{Relationsship} \bullet r.injective =$ true

end .

Abbildung 6.27: Spezialisierung des Referenz-Metaschemas des Objekt-Beziehungsparadigmas für Generische Semantische Modelle (*ORPGSM*)

Jeder Beziehungsklasse (*RelationshipClass*) ist mindestens eine *cComesFrom*- und *cGoesTo*-Kante zugeordnet. Die Anzahl der Objektklassen, die über diese Kanten zu einer Beziehungsklasse adjazent sind, ist jedoch nach oben unbeschränkt [ORPGSM 1]. Generalisierungen [ORPGSM 2] und Kardinalitäten der Beziehungsklassen werden in Generischen Semantischen Modellen nicht unterstützt.

Wie auch in Datenlexika wird in Generischen Semantischen Modellen deutlich zwischen der Aggregation und der Gruppierung unterschieden. Zur Metamodellierung von Aggregationen und Gruppierungen wird auch hier das Konzept der *RelationshipClass* durch *AggregationClass* und *GroupingClass* spezialisiert. Die Kardinalitäten dieser speziellen Beziehungen sind durch die Zusicherungen [ORPGSM 3] und [ORPGSM 4] festgelegt. Für Gruppierungen ist darüber hinaus noch zu fordern, daß hierdurch genau eine Objektklasse gruppiert wird [ORPGSM 5].

Generische Semantische Modelle bilden Attributzuordnungen analog zu NIAM-Informationsstruktur-Diagrammen durch Beziehungstypen ab. Das Konzept *Domain* faßt auch hier die druckbaren Objektklassen in einer Spezialisierung von *ObjectClass* zusammen. Gegenüber dem Referenz-Metaschema des Objekt-Beziehungsparadigmas sind im Metaschema für Generische Semantische Modelle ebenfalls die *Attribut*-Knoten auszublenden [ORPGSM 6]. Abstrakte Objekt- und Beziehungsklassen bzw. nicht-injektive Beziehungsklassen werden auch hier nicht herausgestellt [ORPGSM 7].

UML-Klassendiagramme. Auch die Klassendiagramme der Unified Modeling Language (UML) können auf eine Spezialisierung des Referenz-Metaschemas des Objekt-Beziehungsparadigmas zurückgeführt werden. Gegenüber dem Referenz-Metaschema ist das Metaschema der UML-Klassendiagramme (*ORPUML*) um Konzepte zur Modellierung des nach außen sichtbaren Verhaltens der Objektklassen zu ergänzen (vgl. Abbildung 6.28). Objektklassen werden hierzu Methoden (*Method*) zugeordnet, die durch ihre Signaturen (*Signature*) konkretisiert werden können. Signaturen spezifizieren hierbei die Sorten (*Sort*) des Vorbereichs (*isInputSort*) und des Nachbereichs (*isReturnSort*), die sowohl durch Objektklassen (*ObjectClass*) als auch durch die Attributwertebereiche[6](*Domain*) gebildet werden können. Die Implementation dieser Methoden durch Prozesse (*Process*) kann mit den Beschreibungsmitteln der Prozeßsicht (vgl. Kapitel 6.4) dargestellt werden.

In den Klassendiagrammen der UML werden verschiedene Beziehungsarten unterschieden, die im Metaschema als Spezialisierungen der *RelationshipClass* abgebildet werden. Klassen strukturierter Beziehungen zwischen Objekten werden durch Assoziationen (*Association*) modelliert. In UML-Klassendiagrammen können sowohl binäre, gerichtete Beziehungen als auch n-äre, ungerichtete Beziehungen notiert werden. Die Metamodellierung gerichteter, binärer Beziehungsklassen erfolgt entlang der *cComesFrom*- und *cGoesTo*-Kanten. Zur Beschreibung der n-ären, ungerichteten Beziehungen dienen die *cRelates*-Kanten, deren Kardinalität nach oben unbeschränkt ist [ORPUML 1], und die gegenüber den Referenz-Metaschema konkret sind [ORPUML 2]. Da Beziehungen entweder gerichtet oder ungerichtet sind, werden *cRelates* und *cComesFrom*- bzw. *cGoesTo*-Kanten nicht kombiniert [ORPUML 3].

[6] Je nach Implementationsunterstützung können hier unterschiedliche Standarddatentypen (*bool*, *int*, *float*, *string*) und Typkonstruktoren durch Mengen-, Multimengen oder Tupelbildung unterschieden werden (vgl. hierzu auch [Dahm et al., 1998a, Anhang A].

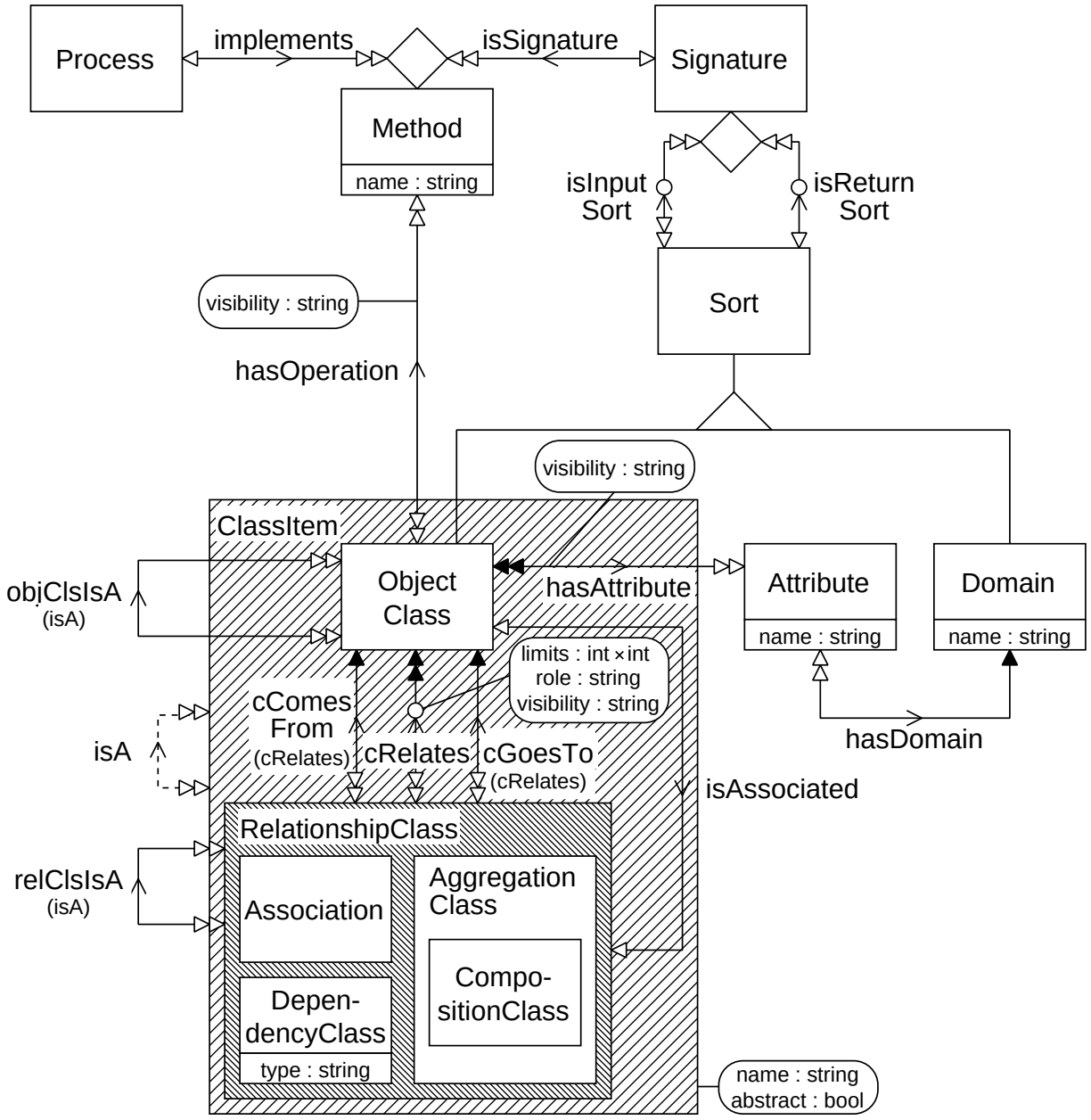


Abbildung 6.28: Spezialisierung des Referenz-Metaschemas des Objekt-Beziehungsparadigmas für UML-Klassendiagramme (*ORPUML*)

Abhängigkeitsbeziehungen (*DependencyClass*) beschreiben Benutzt-Beziehungen zwischen zwei Objektklassen. Diese Beziehungen sind generell binär und gerichtet [ORPUML 4]. Die verschiedenen Arten der Abhängigkeiten[7] werden durch das *type*-Attribut modelliert.

Aggregationen und Gruppierungen werden auch in UML-Klassendiagrammen durch ein gemeinsames Konzept beschrieben. Durch entsprechende Kardinalitätsangaben werden die Gruppierungen, wie im Referenz-Metaschema abgebildet, auf Aggregationen (*AggregationClass*) zurückge-

[7] [Booch et al., 1999, S. 137] unterscheidet die Stereotypen bind, derive, friend, instanceOf, instanciate, powertype, refine, use, access, import, extend, include, become, call, copy, send und trace.

führt. Die gerichteten Beziehungen zwischen Aggregaten und Aggregationen werden auch hier durch die *cComesFrom*- bzw. *cGoesTo*-Kanten dargestellt, *cRelates*-Kanten sind an Aggregationsbeziehungen nicht beteiligt [ORPUML 4]. In UML-Klassendiagrammen werden zwei Varianten der Aggregation unterstützt. Während die (generelle) Aggregation (*AggregationClass*) nur Teil-Ganzes-Beziehungen abbildet, beschreiben Kompositionen (*CompositionClass*) solche Aggregationen, bei denen die Existenz der Aggregate an die Existenz der Aggregation gebunden ist. Für die Komponenten einer Komposition ist daher zu fordern, daß sie genau einmal zu genau einer Komposition gehören [ORPUML 5].

Beziehungsklassen können in UML-Klassendiagrammen nicht direkt attributiert werden. Die Modellierung von Attributen und Methoden zu Assoziationen und Aggregationen erfolgt in UML durch eine assoziierte Objektklasse (*isAssociated*). Abhängigkeitsbeziehungen werden in UML auch nicht durch eine solche Uminterpretation von Objektklassen attributiert [ORPUML 6].

Kardinalitätsangaben zu Assoziationen und Aggregationen werden durch die Min/Max-Notation dargestellt. Im Metaschema wird dieses durch die *limits*-Attribute der *cRelates*-Kanten abgebildet. Rollen, die Objekte in Beziehungen einnehmen, können in UML-Klassendiagrammen ebenfalls modelliert werden. Die *cRelates*-Kanten werden hierzu um das Attribut *role* erweitert. Für Abhängigkeitsbeziehungen (*DependencyClass*) werden diese Attribute nicht unterstützt [ORPUML 7].

Wie im Referenz-Metaschema des Objekt-Beziehungsparadigmas vorgesehen, können Generalisierungen in UML-Klassendiagrammen sowohl zwischen Objektklassen als auch zwischen Beziehungsklassen modelliert werden. Abstrakte und konkrete Objekt- und Beziehungsklassen werden in UML-Klassendiagrammen unterschieden. Beziehungen werden hierbei aber auch generell als injektiv aufgefaßt [ORPUML 8].

UML-Klassendiagramme erlauben zusätzlich Angaben über die Sichtbarkeit von Objekten auf Beziehungen, Attribute und Methoden. Im Metaschema der UML-Klassendiagramme erhalten hierzu die *cRelates*- die *hasAttributes*- und die *hasOperation*-Kanten das *visibility*-Attribut, das u. a. die Werte public, protected und private annehmen kann.

ORPUML **isA** *ORP* ;

for *G* **in** *ORPUML* **assert**

[ORPUML 1]:

overwrites [ORP-EER 1]:

$\forall\, r : V_{Relationship} \bullet 2 \leq \delta_{\rightarrow_{cRelates}}(r)$;

[ORPUML 2]:

overwrites [ORP-EER 2]:

$isConcrete(cRelates)$;

[ORPUML 3]:

$\forall\, r : V_{RelationshipClass} \mid \delta_{\rightarrow_{cRelates}}(r) > 0 \bullet \delta_{\rightarrow_{cComesFrom}}(r) = 0 = \delta_{\rightarrow_{cGoesTo}}(r)$;

$\forall\, r : V_{RelationshipClass} \mid \delta_{\rightarrow_{cComesFrom}}(r) = 1 \wedge \delta_{\rightarrow_{cGoesTo}}(r) = 1 \bullet \delta^{type}_{\rightarrow_{cRelates}}(r) = 0$;

[ORPUML 4]:

$\forall\, a : V_{AggregationClass} \cup V_{DependencyClass} \bullet \delta^{type}_{\rightarrow_{cRelates}}(a) = 0$;

[ORPUML 5]:

$$\forall\, g : E_{cGoesTo} \mid type(\omega(g)) = CompositionClass \bullet first(g.limits) = 1 \wedge second(g.limits) = 1\,;$$

[ORPUML 6]:

$$\forall\, a : E_{isAssociated} \bullet type(\omega(a)) \neq DependencyClass\,;$$

[ORPUML 7]:

$$\forall\, r : V_{cRelates} \mid type(\omega(r)) = DependencyClass \bullet g.limits = \bot \wedge g.role = \bot\,;$$

[ORPUML 8]:

$$\forall\, r : V_{Relationsship} \bullet r.injective$$

end.

Vergleichbare Metaschemata für Klassendiagramme der Object Modeling Method [Rumbaugh et al., 1991] oder der Booch-Methode [Booch, 1994] werden u. a. in [Ebert / Süttenbach, 1997a, S. 6ff] und [Süttenbach / Ebert, 1997, S. 8ff] vorgestellt.

6.6 Integriertes Referenz-Metaschema

Die in den Kapiteln 6.2.1 bis 6.5.3 entwickelten Referenz-Metaschemata der zehn grundlegenden Beschreibungsparadigmen werden im folgenden zum *integrierten* Referenz-Metaschema der Beschreibungsmittel für Organisationen und Softwaresysteme zusammengefaßt.

Hierzu werden zunächst in Kapitel 6.6.1 die Referenz-Metaschemata der Beschreibungsparadigmen integriert und an das integrierte Referenz-Metaschema angepaßt. Zusätzliche *GRAL* -Zusicherungen, die die paradigmenübergreifende Konsistenz des integrierten Referenz-Metaschemas sicherstellen, werden in Kapitel 6.6.2 formuliert.

6.6.1 Integration der Referenz-Metaschemata

Das integrierte Referenz-Metaschema der Beschreibungsmittel für Organisationen und Softwaresysteme faßt die wesentlichen Konzepte der zur Modellierung eingesetzten Beschreibungsmittel in einem gemeinsamen *EER/GRAL*-Konzeptmodell zusammen. Dieses Modell inkludiert hierzu die Metaschemata des Aufgabengliederungspardigmas *TDP* (vgl. Kapitel 6.2.1), des Stellengliederungsparadigmas *PDP* (vgl. Kapitel 6.3.1), des Kommunikationsparadigmas *PDP* (vgl. Kapitel 6.3.2), des Datenflußparadigmas *DFD* (vgl. Kapitel 6.4.1), des Zustandsübergangsparadigmas *STP* (vgl. Kapitel 6.4.2), des Netzparadigmas *NP* (vgl. Kapitel 6.4.3), des Kontrollflußparadigmas *CP* (vgl. Kapitel 6.4.4), des Objekt-Instanzparadigmas *OIP* (vgl. Kapitel 6.5.1), des Objekt-Interaktionsparadigmas *IAP* (vgl. Kapitel 6.5.2) und des Objekt-Beziehungsparadigmas *ORP* (vgl. Kapitel 6.5.3).

for *G* **in** *RMS* **include**

TDP;	[Aufgabengliederungsparadigma]
PDP;	[Stellengliederungsparadigma]
CommP;	[Kommunikationsparadigma]
DFP;	[Datenflußparadigma]
STP;	[Zustandsübergangsparadigma]
NP;	[Netzparadigma]
CP;	[Kontrollflußparadigma]
OIP;	[Objekt-Instanzparadigma]
IAP;	[Objekt-Interaktionsparadigma]
ORP	[Objekt-Beziehungsparadigma]

end.

Die Integration der einzelnen Metaschemata erfolgt entlang der Konzepte, die in mehreren Paradigmen dargestellt werden. Einen Überblick über diese gemeinsamen Konzepte und die Beschreibungsparadigmen, in den sie dargestellt werden, gibt Abbildung 6.29.

In den Referenz-Metaschemata der einzelnen Beschreibungsmittel wurden bereits die Schnittstellen zu den angrenzenden Paradigmen berücksichtigt. Durch Zusammenfassen der Eigenschaften dieser gemeinsamen Konzepte, die durch ihre Attributstrukturen und ihre inzidenten Beziehungstypen charakterisiert sind, werden die Teilschemata integriert. Das resultierende Referenz-Metaschema der Beschreibungsmittel für Organisationen und Softwaresysteme (*RMS*) ist in seinem *EER*-Teil in Abbildung 6.30 dargestellt.

Konzept / Paradigma	Task	Organizational Unit	Process	Event	Domain	Object	Object Class	Relationship Class	Attribute
Aufgabengliederungsparadigma	●	●	●				●		
Stellengliederungsparadigma	●	●							
Kommunikationsparadigma		●							
Datenflußparadigma		●	●				●	●	
Zustandübergangsparadigma			●	●					
Netzparadigma		●	●	●		●			
Kontrollflußparadigma			●						
Objekt-Instanzparadigma					●	●	●	●	●
Objekt-Interaktionsparadigma						●	●		
Objekt-Beziehungsparadigma					●		●	●	●

Abbildung 6.29: Gemeinsame Konzepte der Paradigmen

● Paradigma enthält Konzept

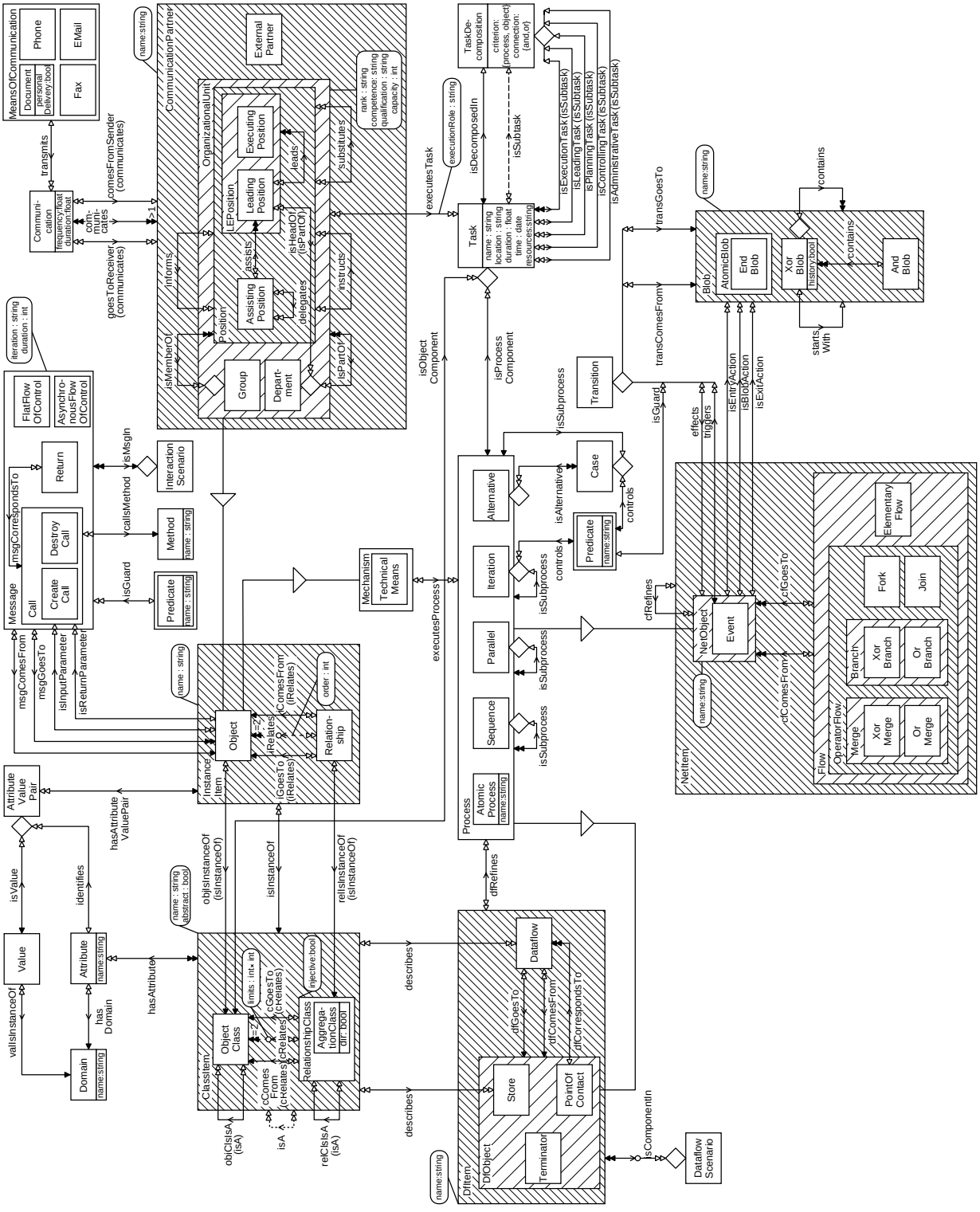


Abbildung 6.30: Integriertes Referenz-Metaschema der Beschreibungsmittel für Organisationen und Softwaresysteme (*RMS*)

Das Referenz-Metaschema der Beschreibungsmittel der Aufgabensicht (Aufgabengliederungsparadigma) findet sich rechts in der Mitte. Rechts oben sind die Referenz-Metaschemata der Aufbausicht (Stellengliederungsparadigma und Kommunikatationsparadigma) dargestellt. Die Referenz-Metaschemata der Prozeßsicht (Datenflußparadigma, Zustandsübergangsparadigma, Netzparadigma und Kontrollflußparadigma) sind im unteren Teil von Abbildung 6.30 aufgeführt. Im linken, oberen Teil sind die Teil-Metaschemata der Objektsicht (Objekt-Instanzparadigma, Objekt-Interaktionsparadigma und Objekt-Beziehungsparadigma) dargestellt.

Die Integration zum Referenz-Metaschema der Beschreibungsmittel für Organisationen und Softwaresysteme erfordert kleinere Anpassungen bei der Zusammenfassung der gemeinsamen Konzepte der Referenz-Metaschemata der einzelnen Beschreibungsparadigmen. Diese werden im folgenden entlang der zentralen Konzepte der Modellierungssichten kurz skizziert.

Aufgabensicht:

Die Einbettung des Referenz-Metaschemas des *Aufgabengliederungsparadigmas* aus der Aufgabensicht in das integrierte Referenz-Metaschema erfolgt unverändert.

Über das zentrale Konzept der Aufgaben (*Task*) werden die Beziehungen zu den restlichen Sichten hergestellt. Die zentralen Konzepte der Prozeßsicht (*Process*) und der Objektsicht (*Object*) werden in der Aufgabensicht als essenzielle Komponenten der Aufgaben abgebildet. Organisationseinheiten (*OrganizationalUnit*) der Aufbausicht werden den Aufgaben als Aufgabenträger zugeordnet.

Aufbausicht:

Entlang der organisatorischen Einheiten (*OrganizationalUnit*) erfolgt auch die Integration der Referenz-Metaschemata der Aufbausicht. Die Konzepte der *OrganizationalUnit* des *Stellengliederungsparadigmas* und des *Kommunikationsparadigmas* beschreiben denselben Sachverhalt, so daß diese miteinander verschmolzen werden können.

Der Querbezug zu dem Referenzmetaschema der Aufgabensicht erfolgt über den Beziehungstyp *executesTask,* durch Organisationseinheiten mit den von ihnen auszuführenden Aufgaben verbinden. Organisationseinheiten spezialisieren darüber hinaus auch das Objekt-Konzept (*Object*) der Objektsicht (vgl. das Referenz-Metaschema des Netzparadigmas in Kapitel 6.4.3), das wiederum das Mechanismus-Konzept (*Mechanism*) der Prozeßsicht (vgl. das Referenz-Metaschema des Datenflußparadigmas in Kapitel 6.4.1) spezialisiert. Hierdurch können organisatorische Einheiten einerseits als Objekte betrachtet werden und andererseits über das Mechanismus-Konzept den zu bearbeitenden Prozessen zugeordnet werden.

Prozeßsicht:

Das zentrale Konzept der Prozeßsicht ist der Prozeß (*Process*), das in den Metaschemata des *Datenflußparadigmas*, des *Zustandsübergangsparadigmas*, des *Netzparadigmas* und des *Kontrollflußparadigmas* (vgl. Kapitel 6.4.4) abgebildet wird. Neben der Prozeßspezialisierung durch regulär strukturierte Prozesse des Kontrollflußparadigmas sind für die anderen Paradigmen der Prozeßsicht solche Prozesse abzubilden, die nicht notwendig regulär strukturiert sind. Gegenüber dem Prozeßkonzept des Kontrollflußparadigmas ist das Prozeßkonzept des integrierten Referenz-Metaschemas daher konkret. Zur Abbildung der Be-

schreibungsmittel des Zustandsübergangsparadigmas, des Netzparadigmas und des Kontrollflußparadigmas spezialisieren diese Prozesse auch die Konzepte der Datenflußobjekte (*DfObject*) und der Netzobjekte (*NetObject*).

Im Referenz-Metaschema des Datenflußparadigmas werden die Daten, die in Speichern (*Store*) gehalten bzw. durch Datenflüsse (*Dataflow*) ausgetauscht werden, durch regulär strukturierte Datenstrukturen (*ClassItem*) modelliert. Da die Beschreibungsmittel des Objekt-Beziehungsparadigmas durch Generalisierung und Aggregation ebenfalls regulär strukturierte Daten darstellen können (vgl. Kapitel 6.5.3), werden die Daten des Datenflußparadigmas durch das *ClassItem*-Konzept des Objekt-Beziehungsparadigmas beschrieben.

Objektsicht:

Die Integration der Referenz-Metaschemata der Objektsicht erfolgt entlang des hier zentralen Konzepts der Objekte (*Object*). Neben der Darstellung der Objekte und Beziehungen auf Instanzebene durch die Beschreibungsmittel des Objekt-Instanzparadigmas und des Objekt-Interaktionsparadigmas werden Objekt- und Beziehungsinstanzen durch die Beschreibungsmittel des Objekt-Beziehungsparadigmas zu Objektklassen (*ObjectClass*) und zu Beziehungsklassen (*RelationshipClass*) zusammengefaßt.

Die Integration der einzelnen Referenz-Metaschemata der Beschreibungsparadigmen erfordert auch kleinere Anpassungen der *GRAL*-Zusicherungen. Datenbezüge des Datenflußparadigmas werden im integrierten Referenz-Metaschema durch Rückgriff auf das Referenz-Metaschema des Objekt-Beziehungsparadigmas beschrieben. Die Zusicherungen an das Referenz-Metaschema des Datenflußparadigmas (vgl. Kapitel 6.4.1) [DFP 10] und [DFP 11] sind daher an die Strukturierung des Konzepts *ClassItem* des Objekt-Beziehungsparadigmas anzupassen. Ebenso werden die Konzepte *AtomicClass*, *AlternativeClass*, *IteratedClass* und *SequentialClass* des Metaschemas des Datenflußparadigmas zur Beschreibung regulär strukturierter Datenzusammenhänge nicht verwendet [RMS-DFP 12].

for G **in** *RMS* **assert**

[RMS-DFP 10]:

overwrites [DFP 10] :

$$\forall\, poc : V_{PointOfContact} \bullet$$
$$poc \rightharpoonup_{dfCorrespondsTo} \leftharpoonup_{describes}$$
$$(\leftharpoonup_{isA} \mid (\leftharpoonup_{cGoesTo} \&_{AggregationClass} \rightharpoonup_{cComesFrom}))^*$$
$$\rightharpoonup_{describes} (\rightharpoonup_{dfGoesTo} \mid \rightharpoonup_{dfComesFrom})\ poc\ ;$$

[RMS-DFP 11]:

overwrites [DFP 11] :

$$\forall\, d : V_{Dataflow} \bullet$$
$$d \rightharpoonup_{dfGoesTo} \rightharpoonup_{dfComesFrom} \&_{Store} \leftharpoonup_{describes}$$
$$(\leftharpoonup_{isA} \mid (\leftharpoonup_{cGoesTo} \&_{AggregationClass} \rightharpoonup_{cComesFrom}))^* \rightharpoonup_{describes} d\ ;$$

[RMS-DFP 12]:

$$V_{AtomicClass} = V_{AlternativeClass} = V_{IteratedClass} = V_{SequentialClass} = \varnothing$$

end .

Im Referenz-Metaschema des Kontrollflußparadigmas ist das *Process*-Konzept als abstrakt vereinbart. Aufgrund der Zusammenfassung der *Process*-Konzepte des Datenflußparadigmas, des Zustandsübergangsparadigmas und des Netzparadigmas ist diese Anforderung im integrierten Referenz-Metaschema aufzugeben [RMS-CP 2].

for *G* **in** *RMS* **assert**
[RMS-CP 2]:
overwrites [CP-EER 2] :
isConcrete(*Process*)
end.

Die Zusicherungen des Aufgabengliederungsparadigmas, des Stellengliederungsparadigmas, des Kommunikationsparadigmas, des Zustandsübergangsparadigmas, des Netzparadigmas, des Objekt-Instanzparadigmas, des Objekt-Interaktionsparadigmas und des Objekt-Beziehungsparadigmas werden hierbei unverändert übernommen. In Anhang A sind diese Zusicherungen für die einzelnen Paradigmen noch einmal zusammenfassend dargestellt.

6.6.2 Paradigmenübergeifende *GRAL*-Zusicherungen

In multiperspektivischen Modellierungen werden Organisationen bzw. Softwaresysteme durch mehrere Teilmodelle dargestellt. Diese Teilmodelle, die in unterschiedlichen Beschreibungssprachen notiert werden, müssen zueinander konsistent sein.

Die Formalisierung dieser paradigmenübergreifenden Konzistenzbedingungen erfolgt durch weitere *GRAL*-Zusicherungen. Hierbei werden zunächst die Konsistenzbedingungen zwischen den verschiedenen Paradigmen einzelner Sichten betrachtet. Anschließend werden die sichtenübergreifenden Zusicherungen formuliert.

Die Konsistenz der Darstellungsmittel der Aufgabensicht ist bereits durch das Referenz-Metaschema des Aufgabengliederungsparadigmas definiert. Im Metaschema des Stellengliederungsparadigmas und im Metaschema des Kommunikationsparadigmas ist das Konzept der Organisationseinheiten vollständig enthalten, so daß keine weiteren Zusicherungen zur paradigmenübergreifenden Konsistenzsicherung der Beschreibungsmittel der Aufbausicht nötig werden. Der Zusammenhang zwischen den Beschreibungen der Aufgaben- und Aufbausicht bzw. zwischen der Aufgaben- und der Objektsicht wird durch die Darstellungsmittel des Aufgabengliederungsparadigmas hergestellt. Das Metaschema des Aufgabengliederungsparadigmas umfaßt daher auch die hier nötigen Zusicherungen. In ähnlicher Form ist auch die Konsistenz der Darstellungen der Prozeß- und Objektsicht in den Metaschemata des Datenflußparadigmas und des Netzparadigmas formalisiert.

Die noch nötigen paradigmen- und sichtenübergreifenden Zusicherungen beziehen sich folglich auf die Betrachtung der verschiedenen Paradigmen der Prozeß- und Objektsicht und auf die Querbezüge zwischen Aufgaben- und Prozeßsicht, Aufbau- und Prozeßsicht und Aufbau- und Objektsicht.

Paradigmenübergeifende Zusicherungen der Prozeßsicht

Prozeßzerlegungen werden sowohl mit den Beschreibungsmitteln des Datenflußparadigmas, des Netzparadigmas und des Kontrollflußparadigmas beschrieben. Wird die Zerlegung desselben Prozesses entlang unterschiedlicher Beschreibungsparadigmen dargestellt, müssen diese Zerlegungen miteinander verträglich sein. Jede dieser Prozeßzerlegungen untergliedert den betrachteten Prozeß in dieselben Teilprozesse [RMS-Process 1].

for G **in** *RMS* **assert**

[RMS-Process 1]:

$\forall\, p : V_{Process} \bullet$

[Datenfluß- und Netzparadigma]

$(\delta_{\leftharpoondown_{dfRefines}}(p) > 0 < \delta_{\leftharpoondown_{cfRefines}}(p) \Rightarrow$

$p \leftharpoondown_{dfRefines} \&_{Process} = p \leftharpoondown_{cfRefines} \&_{Process}) \wedge$

[Datenfluß- und Kontrollflußparadigma]

$(\delta_{\leftharpoondown_{dfRefines}}(p) > 0 \wedge type(p) \in \{Sequence, Parallel, Iteration, Alternative\} \Rightarrow$

$p \leftharpoondown_{dfRefines} \&_{Process} = p\,[\leftharpoondown_{isAlternative}]\ \leftharpoondown_{isSubprocess}) \wedge$

[Netz- und Kontrollflußparadigma]

$(\delta_{\leftharpoondown_{cfRefines}}(p) > 0 \wedge type(p) \in \{Sequence, Parallel, Iteration, Alternative\} \Rightarrow$

$p \leftharpoondown_{cfRefines} \&_{Process} = p\,[\leftharpoondown_{isAlternative}]\ \leftharpoondown_{isSubprocess})$

end.

Im Zusammenspiel mit Zusicherung [DFP 2] des Datenflußparadigmas impliziert [RMS-Process 1], daß Prozesse in Darstellungen des Netzparadigmas oder in Darstellungen des Kontrollflußparadigmas, die auch in einer Datenflußbeschreibung verfeinert wurden, zwischen drei und sechs Teilprozesse besitzen.

Paradigmenübergeifende Zusicherungen der Objektsicht

Die paradigmenübergreifenden Zusicherungen an die Darstellungsmittel der Objektsicht beziehen sich auf die Konsistenzsicherung zwischen Darstellungen schematischer Objektzusammenhänge des Objekt-Beziehungsparadigmas und den Darstellungen auf Instanzebene durch die Beschreibungsmittel des Objekt-Instanzparadigmas (vgl. hierzu auch die Definition der Relation $\models_{EERSchema}$ in Kapitel 5.2.2).

In Instanzdarstellungen müssen die Typen der zu einem Knoten inzidenten Kanten mit den Inzidenz-Definitionen auf Scheme-Ebene übereinstimmen [RMS-Object 1]. Die Inzidenzen in Instanzbeschreibungen müssen ebenfalls den Kardinalitäten der Schemadefinition genügen [RMS-Object 2]. In beiden Zusicherungen sind auch die in der Schemadefinition festgelegten Kantenrichtungen zu berücksichtigen.

Die Attributstruktur von Objekten (*Object*) und Beziehungen (*Relationship*) wird durch die Definition der Objekt- (*ObjectClass*) und Beziehungsklassen (*RelationshipClass*) definiert. Objekte und Beziehungen können nur solche Attribute besitzen, die für ihren Typ bzw. deren Supertypen erlaubt sind [RMS-Object 3].

Schließlich müssen die Instanzdarstellungen den in Schemadefinitionen festgelegten Forderungen nach abstrakten Objekt- und Beziehungsklassen [RMS-Object 4] und nach injektiven Beziehungsklassen [RMS-Object 5] genügen.

for G **in** *RMS* **assert**

[RMS-Object 1]:

$$\forall\, v : V_{Object};\ e : V_{Relationship} \mid v \leftharpoonup_{iComesFrom} e \bullet$$
$$v \rightharpoonup_{objIsInstanceOf} \rightharpoonup^{*}_{objClsIsA} \leftharpoonup_{cComesFrom} \leftharpoonup^{*}_{relClsIsA} \leftharpoonup_{relIsInstanceOf} e\ ;$$

$$\forall\, v : V_{Object};\ e : V_{Relationship} \mid v \leftharpoonup_{iGoesTo} e \bullet$$
$$v \rightharpoonup_{objIsInstanceOf} \rightharpoonup^{*}_{objClsIsA} \leftharpoonup_{cGoesTo} \leftharpoonup^{*}_{relClsIsA} \leftharpoonup_{relIsInstanceOf} e\ ;$$

[RMS-Object 2]:

$$\forall\, v : V_{Object};\ e : E_{cComesFrom} \mid v \rightharpoonup_{objIsInstanceOf} \rightharpoonup^{*}_{objClsIsA} \omega(e) \bullet$$
$$first(e.limits) \leq \#\{\, r : E_{iComesFrom} \mid \omega(r) = v \wedge \alpha(r) \rightharpoonup_{relIsInstanceOf} \rightharpoonup^{*}_{relClsIsA} \alpha(e)\} \leq second(e.limits)\ ;$$

$$\forall\, v : V_{Object};\ e : E_{cGoesTo} \mid v \rightharpoonup_{objIsInstanceOf} \rightharpoonup^{*}_{objClsIsA} \omega(e) \bullet$$
$$first(e.limits) \leq \#\{\, r : E_{iGoesTo} \mid \omega(r) = v \wedge \alpha(r) \rightharpoonup_{relIsInstanceOf} \rightharpoonup^{*}_{relClsIsA} \alpha(e)\} \leq second(e.limits)\ ;$$

[RMS-Object 3]:

$$\forall\, i : V_{InstanceItem};\ c : V_{ClassItem} \mid i \rightharpoonup_{IsInstanceOf} c \bullet$$
$$i \rightharpoonup_{hasAttributeValuePair} \leftharpoonup_{identifies} = c \rightharpoonup^{*}_{IsA} \rightharpoonup_{hasAttribute}\ ;$$

[RMS-Object 4]:

$$\forall\, c : V_{ClassItem} \mid c.abstract \bullet c \leftharpoonup_{IsInstanceOf} = \varnothing\ ;$$

[RMS-Object 5]:

$$\forall\, rc : V_{RelationshipClass};\ r_1, r_2 : V_{Relationship} \mid$$
$$rc.injective \wedge r_1 \rightharpoonup_{relIsInstanceOf} rc \leftharpoonup_{relIsInstanceOf} r_2 \wedge r_1 \neq r_2 \bullet$$
$$r_1 \rightharpoonup_{iGoesTo} \neq r_2 \rightharpoonup_{iGoesTo} \vee r_1 \rightharpoonup_{iComesFrom} \neq r_2 \rightharpoonup_{iComesFrom}$$

end.

Paradigmenübergeifende Zusicherungen der Aufgaben- und Prozeßsicht

Zwischen Prozeßdarstellungen und Aufgabengliederungen sind die Aufgaben- bzw. Prozeßgliederungen, die in Aufgaben bzw. in Prozessen betrachteten Objekte und die mit der Ausführung betrauten Prozeß- bzw. Aufgabenträger zu synchronisieren.

Die Zerlegung von Aufgaben in Teilaufgaben muß sich hierbei in der Zerlegung der Prozeßkomponenten der Aufgabe, die mit den Beschreibungsmitteln des Datenflußparadigmas, des Netzparadigmas oder des Kontrollflußparadigmas modelliert wurden, widerspiegeln [RMS-TaskProcess 1].

Bezüge zwischen Objekten und Aufgaben bzw. zwischen Objekten und Prozessen werden im Aufgabengliederungsparadigma und im Datenflußparadigma beschrieben. In den durch Daten-

flußbeschreibungen modellierten Prozeßkomponenten einer Aufgabe wird hierbei die Objektkomponente bearbeitet. Je nach Festlegung der Objektkomponente, ausgehend von den in der Aufgabe bearbeiteten Objekten oder ausgehend von den produzierten Objekten, findet sich die Objekt-Komponente in den eingehenden oder ausgehenden Datenflüssen zur jeweiligen Prozeßkomponente [RMS-TaskProcess 2].

Organisationseinheiten, die Aufgaben bzw. Prozesse ausführen, werden durch die Beschreibungsmittel der Aufgabensicht Aufgaben und durch die Beschreibungsmittel der Prozeßsicht Prozessen zugeordnet. Diese Organisationseinheiten müssen für Aufgaben und ihre Prozeßkomponenten übereinstimmen [RMS-TaskProcess 3].

for G **in** *RMS* **assert**

[RMS-TaskProcess 1]:

$$\forall\, p : V_{Process};\ t : V_{Task} \mid p \rightharpoonup_{isProcessComponent} t \bullet$$

[Datenflußparadigma]

$$(\delta_{\leftharpoonup_{dfRefines}}(p) > 0 \wedge \delta_{\rightharpoonup_{isDecomposedIn}}(t) > 0 \Rightarrow p \leftharpoonup_{dfRefines} \&_{Process} = t \rightharpoonup_{isDecomposedIn} \leftharpoonup_{isSubtask} \leftharpoonup_{isProcessComponent}) \wedge$$

[Netzparadigma]

$$(\delta_{\leftharpoonup_{cfRefines}}(p) > 0 \wedge \delta_{\rightharpoonup_{isDecomposedIn}}(t) > 0 \Rightarrow p \leftharpoonup_{cfRefines} \&_{Process} = t \rightharpoonup_{isDecomposedIn} \leftharpoonup_{isSubtask} \leftharpoonup_{isProcessComponent}) \wedge$$

[Kontrollflußparadigma]

$$(type(p) \in \{Sequence, Parallel, Iteration, Alternative\} \wedge \delta_{\rightharpoonup_{isDecomposedIn}}(t) > 0 \Rightarrow p\,[\leftharpoonup_{isAlternative}] \leftharpoonup_{isSubprocess} = t \rightharpoonup_{isDecomposedIn} \leftharpoonup_{isSubtask} \leftharpoonup_{isProcessComponent});$$

[RMS-TaskProcess 2]:

$$\forall\, p : V_{Process};\ t : V_{Task} \mid p \rightharpoonup_{isProcessComponent} t \wedge p(\leftharpoonup_{dfComesFrom} \mid \leftharpoonup_{dfGoesTo}) \leftharpoonup_{describes} \neq \varnothing \bullet t \leftharpoonup_{isObjectComponent} \subseteq p(\leftharpoonup_{dfComesFrom} \mid \leftharpoonup_{dfGoesTo}) \leftharpoonup_{describes};$$

[RMS-TaskProcess 3]:

$$\forall\, p : V_{Process};\ t : V_{Task} \mid p \rightharpoonup_{isProcessComponent} t \bullet t \leftharpoonup_{executesTask} = p \leftharpoonup_{executesProcess} \&_{OrganizationalUnit}$$

end.

Paradigmenübergeifende Zusicherungen der Aufbau- und Prozeßsicht

Modellierungen nach dem Datenflußparadigma beschreiben indirekt auch Kommunikationsbeziehungen zwischen Organisationseinheiten. Besteht eine Datenflußbeziehung zwischen zwei Prozessen, die durch verschiedene Organisationseinheiten ausgeführt werden, impliziert dieses auch eine Kommunikationsbeziehung zwischen den beteiligten Organisationseinheiten. Dieser Querbezug wird in [RMS-PositionProcess 1] zugesichert.

for G **in** *RMS* **assert**

[RMS-PositionProcess 1]:

$$\forall\, p_1, p_2 : V_{Process};\ s_1, s_2 : V_{OrganizationalUnit} \mid s_1 \neq s_2 \ \wedge\ s_1 \rightharpoonup_{executesProcess} p_1 \ \wedge\ s_2 \rightharpoonup_{executesProcess} p_2 \ \bullet\ p_1 \leftharpoonup_{dfComesFrom} \rightharpoonup_{dfGoesTo} p_2 \Rightarrow s_1 \leftharpoonup_{communicates} \rightharpoonup_{communicates} s_2$$

end.

Paradigmenübergeifende Zusicherungen der Aufbau- und Objektsicht

Mit den Beschreibungsmitteln des Objekt-Interaktionsparadigmas wird die Interaktion zwischen Objekten modelliert. Beschreiben solche, über Nachrichten interagierende, Objekte organisatorische Einheiten, liegt zwischen diesen ebenfalls eine Kommunikationsbeziehung vor [RMS-PositionObject 1].

for G **in** *RMS* **assert**

[RMS-PositionObject 1]:

$$\forall\, s_1, s_2 : V_{OrganizationalUnit} \ \bullet\ s_1 \leftharpoonup_{msgComesFrom} \rightharpoonup_{msgGoesTo} s_2 \Rightarrow s_1 \leftharpoonup_{communicates} \rightharpoonup_{communicates} s_2$$

end.

6.7 Zusammenfassung

Entlang der Klassifikation der Beschreibungsmittel für Organisationen und Softwaresysteme aus Kapitel 3.1 wurden für die grundlegenden Beschreibungsparadigen *EER/GRAL*-Konzeptmodelle entwickelt. Diese Konzeptmodelle beschreiben *Referenz-Metaschemata* der einzelnen Beschreibungsparadigmen.

Aus diesen Referenz-Metaschemata wurden Metaschemata der wesentlichen Notationsformen der jeweils betrachteten Paradigmen durch marginale Einschränkungen und Ergänzungen abgeleitet. Hierdurch konnte die *Referenz-Eigenschaft* dieser paradigmenbezogenen Referenz-Metaschemata nachgewiesen werden.

Die paradigmenbezogenen Referenz-Metaschemata wurden entlang der gemeinsamen Konzepte der einzelnen Pardigmen zum *integrierten Referenz-Metaschema der Beschreibungsmittel für Organisationen und Softwaresysteme* zusammengefaßt. Einen Überblick über die vollständige Formalisierung des Referenz-Metaschemas gibt Anhang A.

In Teil III wird dieses integrierte Referenz-Metaschema zur Betrachtung der Beschreibungsmittel konkreter Modellierungsmethoden und zur Entwicklung eines Softwarewerkzeugs zur Modellierungsunterstützung angewandt.

Teil III

Anwendung des Referenz-Metaschemas

Das in Kapitel 6 vorgestellte Referenz-Metaschema zielt auf eine möglichst umfassende konzeptionelle Abbildung aller wesentlichen visuellen Modellierungssprachen für Organisationen und Softwaresysteme. Im Vergleich zu Metaschemata wesentlicher Ansätze zur Organisations- und Softwaremodellierung bezieht sich dieses Referenz-Metaschema nicht auf wenige konkrete Sprachen, sondern es berücksichtigt auch die konzeptionellen Unterschiede verschiedener Dialekte einzelner Modellierungstechniken.

Die Metaschemata der Beschreibungsmittel der Architektur integrierter Informationssysteme (ARIS) [Scheer, 1992] oder der Unified Modeling Language (UML) [OMG, 1999] können als mögliche Spezialisierungen des hier eingeführten integrierten Referenz-Metaschemas aufgefaßt werden. Metaschemata zu ARIS (vgl. Kapitel 7) und UML (vgl. Kapitel 8) werden im folgenden fallstudienartig vom Referenz-Metaschema der visuellen Modellierungssprachen abgeleitet.

Aufgrund des durchgängigen *EER/GRAL*-Ansatzes zur graphbasierten Modellierung und Implementierung beschreibt das in Kapitel 6 eingeführte Referenz-Metaschemata auch Repository-Strukturen für Modellierungswerkzeuge. Repository-Strukturen für Werkzeuge zur Unterstützung konkreter Modellierungsmethoden können daher auch durch Spezialisierungen des integrierten Referenz-Metaschemas definiert werden. In Kapitel 9 wird die Anwendung des Referenz-Metaschemas bei der Entwicklung von Modellierungswerkzeugen am Beispiel eines Werkzeugs zur Software-Evaluation skizziert.

7 Architektur integrierter Informationssysteme

Die *Architektur integrierter Informationssysteme (ARIS)* [Scheer, 1992], [Scheer, 1994, Teil A], [IDS, 1995] stellt ein Rahmenwerk zur Modellierung und Entwicklung betrieblicher Informationssysteme bereit. Die Organisationsmodellierung erfolgt in ARIS entlang der Geschäftsprozesse der betrachteten Organisation. Ausgehend von diesen Prozessen wird die Organisation aus Datensicht, zur Beschreibung der benötigten und erzeugten Daten, aus Funktionssicht, zur Beschreibung der auszuführenden Funktionen und aus Organisationssicht, zur Beschreibung der Aufbauorganisation, betrachtet. Teilmodelle dieser drei Sichten werden in der Steuerungssicht miteinander integriert. Die Darstellungsmittel der Steuerungssicht stellen hierbei die Modellierungsinhalte der Datensicht, der Funktionssicht und der Organisationssicht zueinander in Beziehung (vgl. auch die Einordnung des ARIS-Ansatzes als Metamodell in Kapitel 4.3.1, S. 121).

Die Organisationsmodellierung zur Entwicklung von Informationssystemen erfolgt mit ARIS auf der Ebene des Fachkonzepts, des DV-Konzepts und des Implementationskonzepts. Diese Konzepte werden jeweils aus Datensicht, aus Funktionssicht, aus Organisationssicht und aus Steuerungssicht beschrieben. Der Schwerpunkt der Organisationsmodellierung mit dem ARIS-Ansatz liegt jedoch auf der Erstellung des Fachkonzepts (vgl. auch [Scheer, 1994, S. 16]).

Das zentrale Beschreibungsmittel des ARIS-Ansatzes sind Ereignisgesteuerte Prozeßketten zur Modellierung der Geschäftsprozesse der betrachteten Organisation aus Funktionssicht. Ergänzt wird die Darstellung um Funktionsbäume zur Beschreibung von Aufgabengliederungen und um Nassi-Shneiderman-Diagramme zur Modellierung von Prozeßausführungen. Die Beschreibung der Fachkonzepte aus Datensicht erfolgt durch Objekt-Beziehungsdiagramme. Aufbauorganisatorische Aspekte der Organisationssicht werden durch einfache Organigramme dargestellt. Zur Integration dieser Teilmodelle in der Steuerungssicht finden erweiterte Ereignisgesteuerte Prozeßketten und Vorgangsketten-Diagramme Anwendung.

[Scheer, 1992] und [Scheer, 1994, Teil A] diskutieren weitere visuelle Modellierungssprachen der einzelnen Sichten, wie Stellenbeschreibungen, Netzpläne, Datenflußdiagramme und Entscheidungstabellen, die neben weiteren Beschreibungsmitteln auch durch das ARIS-Toolset unterstützt werden. In Anwendungen des ARIS-Ansatzes zur Darstellung von Fachkonzepten für industrielle Geschäftsprozesse (vgl. z. B. [Scheer, 1994]) zur Darstellung von Krankenhausinformationssystemen (vgl. z. B. [Imhoff et al., 1996]) oder zur Darstellung der Fachkonzepte betriebswirtschaftlicher Standardsoftware (vgl. z. B. [Staud, 1999]) werden i. allg. ausschließlich (erweiterte) Ereignisgesteuerte Prozeßketten, Vorgangskettendiagramme, Funktionsbäume, Objekt-Beziehungsdiagramme und Organigramme verwendet. Die Konzepte dieser Beschrei-

bungsmittel werden im Metaschema des ARIS-Ansatzes abgebildet, das im folgenden aus dem Referenz-Metaschema der Beschreibungsmittel für Organisationen und Softwaresysteme abgeleitet wird.

7.1 Integration der Metaschemata der ARIS-Beschreibungsmittel

Zur Modellierung nach dem ARIS-Ansatz werden Sprachen des Aufgabengliederungsparadigmas, des Stellengliederungsparadigmas, des Netzparadigmas, des Kontrollflußparadigmas und des Objekt-Beziehungspardigmas verwendet. Die Referenz-Metaschemata dieser Paradigmen sind jeweils an die konkreten Anforderungen der ARIS-Sprachmittel anzupassen und zum integrierten Metaschema des ARIS-Ansatzes zusammenzufassen.

Die Struktur der ARIS-Funktionsbäume wird durch die Spezialisierung *TDPAris* des Referenz-Metaschemas des Aufgabengliederungsparadigmas modelliert (vgl. Kapitel 6.2.1). Im ARIS-Ansatz werden nur sehr einfache Organigramme zur Beschreibung der Aufbaustrukturen der betrachteten Organisation verwendet. Diese Organigramme folgen der Spezialisierung *PDPAris* des Referenz-Metaschemas des Stellengliederungsparadigmas *PDP* (vgl. Kapitel 6.3.1). Zur Darstellung der Geschäftsprozesse einer Organisation werden (erweiterte) Ereignisgesteuerte Prozeßketten und Vorgangskettendiagramme verwendet. Diese Beschreibungsmittel genügen der in Kapitel 6.4.3 vorgestellten Spezialisierung des Referenz-Metaschemas des Netzparadigmas für Vorgangskettendiagramme (*NPVKD*) bzw. dem hieraus abgeleiteten Metaschema der Ereignisgesteuerten Prozeßkette (*NPEPK*). Die in ARIS zur Beschreibung regulär strukturierter Prozeßfolgen verwendeten Nassi-Shneiderman-Diagramme entsprechen dem in Kapitel 6.4.4 eingeführten Referenz-Metaschema des Kontrollflußparadigmas (*CP*). Aus dem Referenz-Metaschema des Objekt-Beziehungsparadigmas wurde in Kapitel 6.5.3 auch eine Spezialisierung für einen heute üblichen Dialekt der Entity-Relationshipdiagramme abgeleitet. Die im ARIS-Ansatz verwendeten Beschreibungsmittel zur Modellierung des Fachkonzepts der Datensicht entsprechen diesem Metaschema *ORPER*.

Diese fünf Metaschemata bilden das Metaschema der Beschreibungsmittel zur Organisationsmodellierung entlang des ARIS-Ansatzes (*ARIS*). Abbildung 7.1 stellt den *EER*-Teil des resultierenden ARIS-Metaschemas dar.

for *G* **in** *ARIS* **include**

TDPAris;	[Spezialisierung des Referenz-Metaschemas des Aufgabengliederungsparadigmas für Funktionsbäume]
PDPAris;	[Spezialisierung des Referenz-Metaschemas des Stellengliederungsparadigmas für ARIS-Organigramme]
NPVKD;	[Spezialisierung des Referenz-Metaschemas des Netzparadigmas für Vorgangskettendiagramme]
CP;	[Referenz-Metaschemas des Kontrollflußparadigmas]
ORPER;	[Spezialisierung des Referenz-Metaschemas des Objekt-Beziehungsparadigmas für Entity-Relationshipdiagramme]

end .

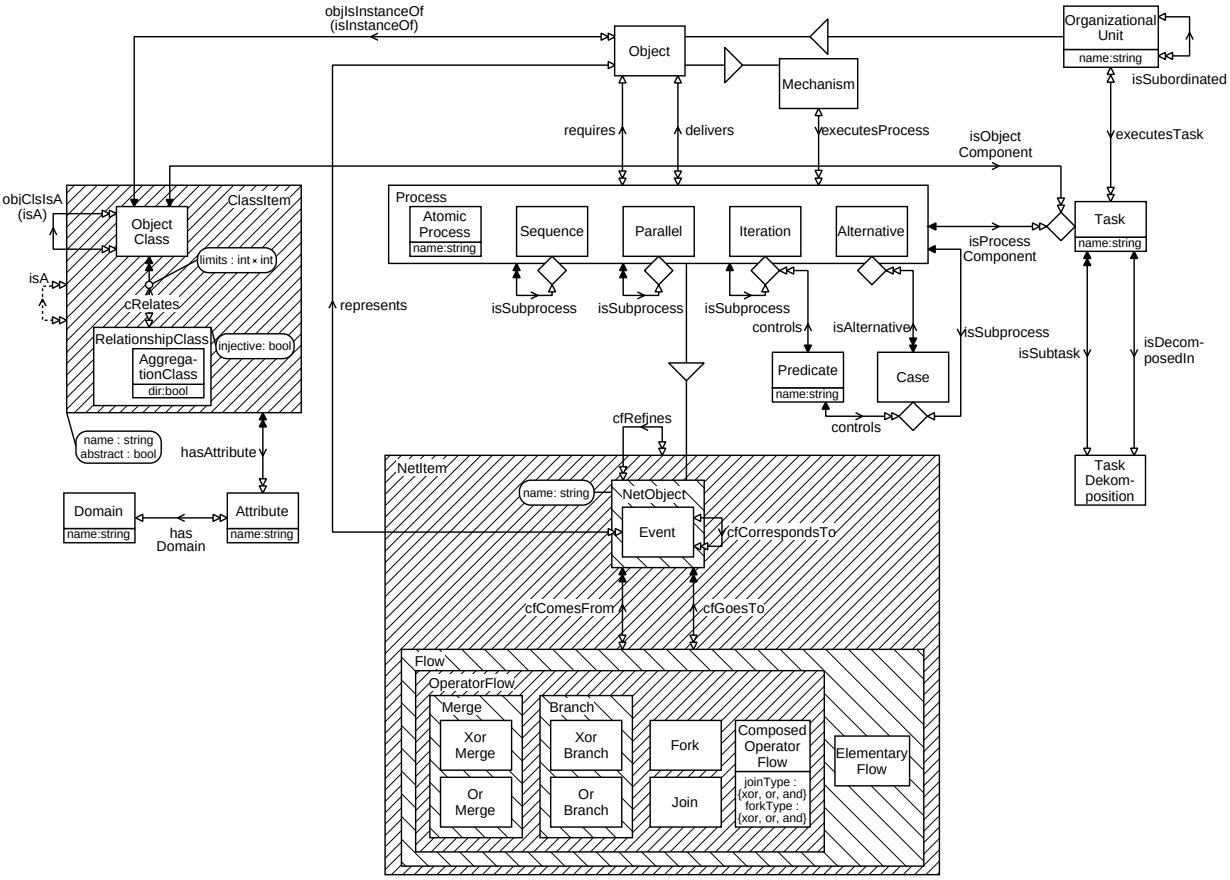


Abbildung 7.1: Spezialisierung des integrierten Referenz-Metaschemas für die Beschreibungsmittel des ARIS-Ansatzes (*ARIS*)

Aufgrund der Verschmelzung der *Process*-Konzepte der Metaschemata der Prozeßsicht durch Vorgangskettendiagramme und Nassi-Shneiderman-Diagramme ist analog zum integrierten Referenz-Metaschema das *Process*-Konzept auch im ARIS-Metaschema als konkret zu vereinbaren [ARIS-CP 1].

for *G* **in** *ARIS* **assert**

[ARIS-CP 1]:

overwrites [CP-EER 2] :

$isConcrete(Process)$

end.

7.2 Paradigmenübergreifende *GRAL*-Zusicherungen

Auch für das ARIS-Metaschema sind paradigmenübergreifende Zusicherungen zu fordern. Wie für das integrierte Referenz-Metaschema (vgl. Kapitel 6.6) werden diese entlang der Beschreibungssichten eingeführt.

Da das ARIS-Metaschema nur jeweils ein Beschreibungsparadigma der Aufgabensicht, der Aufbausicht und der Objektsicht verwendet, sind innerhalb dieser Sichten keine paradigmen-

übergreifenden Zusicherungen nötig. Die Zusammenhänge zwischen Beschreibungsmitteln der Aufgaben- und Aufbausicht bzw. der Aufgaben- und Objektsicht sind analog zum integrierten Referenz-Metaschema vollständig in den jeweiligen Metaschemata berücksichtigt. Zusätzliche Anforderungen an die Querbezüge zwischen Aufbau- und Prozeßsicht bzw. zwischen Aufbau- und Objektsicht sind für das ARIS-Metaschema ebenfalls nicht zu fordern. Kommunikationsbeziehungen, die diese Zusammenhänge herstellen, werden mit den betrachteten ARIS-Mitteln nicht dargestellt.

Paradigmenübergreifende Zusicherungen für das ARIS-Metaschema beziehen sich daher ausschließlich auf Zusicherungen innerhalb der Prozeßsicht und auf Zusicherungen zwischen der Aufgaben- und der Prozeßsicht.

7.2.1 Paradigmenübergreifende Zusicherungen der Prozeßsicht

Prozesse werden in ARIS sowohl durch Vorgangskettendiagramme und (erweiterte) Ereignisgesteuerte Prozeßketten des Netzparadigmas als auch durch Nassi-Shneiderman-Diagramme des Kontrollflußparadigmas beschrieben. Beide Beschreibungsformen unterstützen die Verfeinerung von Prozessen in Teilprozesse. Werden diese Beschreibungsmittel zur Darstellung derselben Prozesse verwendet, müssen diese Prozeßzerlegungen miteinander verträglich sein [ARIS-Process 1] (vgl. auch Zusicherung [RMS-Process 1] des Referenz-Metaschemas).

for *G* **in** *ARIS* **assert**

[ARIS-Process 1]:

$$\forall\, p : V_{Process} \bullet \delta_{\leftharpoonup_{cfRefines}}(p) > 0 \wedge type(p) \in \{Sequence, Parallel, Iteration, Alternative\} \Rightarrow p \leftharpoonup_{cfRefines} \&_{Process} = p\,[\leftharpoonup_{isAlternative}] \leftharpoonup_{isSubprocess}$$

end.

7.2.2 Paradigmenübergreifende Zusicherungen der Aufgaben- und Prozeßsicht

Durch Darstellungen der Aufgaben- und der Prozeßsicht werden in ARIS u. a. auch Aufgaben- bzw. Prozeßgliederungen, die jeweils bearbeiteten Objekte und die mit der Aufgabe bzw. Prozeßbearbeitung betrauten Organisationseinheiten betrachtet. Diese Bezüge sind in den entsprechenden ARIS-Teilmodellen zueinander verträglich darzustellen.

Die Gliederung der Organisationsaufgaben in Funktionsbäumen ist hierzu mit der Prozeßverfeinerung in Prozeßkettendarstellungen und in Kontrollflußdarstellungen zu synchonisieren [ARIS-TaskProcess 1] (vgl. auch [RMS-TaskProcess 1]). Die Beschreibungsmittel des Netzparadigmas, wie sie im ARIS-Ansatz verwendet werden, ermöglichen auch die Modellierung der Objekte, die durch Prozesse benötigt (*requires*) bzw. erzeugt (*delivers*) werden. Die Strukturen dieser Objekte müssen mit der Objektkomponente der dem Prozeß entsprechenden Aufgabe verträglich sein [ARIS-TaskProcess 2]. Wie auch für das intergierte Referenz-Metaschema sind Aufgaben- und Prozeßdarstellungen über die Organisationseinheiten zu synchronisieren, die die Aufgaben oder Prozesse ausführen [ARIS-TaskProcess 3] (vgl. auch [RMS-TaskProcess 3]).

for G **in** *ARIS* **assert**

[ARIS-TaskProcess 1]:

$$\forall\, p : V_{Process};\ t : V_{Task} \mid p \rightharpoonup_{isProcessComponent} t \bullet$$

[Netzparadigma]

$$(\delta_{\leftharpoonup_{cfRefines}}(p) > 0 \wedge \delta_{\rightharpoonup_{isDecomposedIn}}(t) > 0 \Rightarrow$$

$$p \leftharpoonup_{cfRefines} \&_{Process} = t \rightharpoonup_{isDecomposed} \leftharpoonup_{isSubtask} \leftharpoonup_{isProcessComponent}) \wedge$$

[Kontrollflußparadigma]

$$(type(p) \in \{Sequence, Parallel, Iteration, Alternative\} \wedge$$

$$\delta_{\rightharpoonup_{isDecomposedIn}}(t) > 0 \Rightarrow$$

$$p\,[\leftharpoonup_{isAlternative}]\ \leftharpoonup_{isSubprocess} = t \rightharpoonup_{isDecomposed} \leftharpoonup_{isSubtask} \leftharpoonup_{isProcessComponent});$$

[ARIS-TaskProcess 2]:

$$\forall\, p : V_{Process};\ t : V_{Task} \mid p \rightharpoonup_{isProcessComponent} t \bullet$$

$$t \leftharpoonup_{isObjectComponent} \subseteq p(\rightharpoonup_{requires} \mid \rightharpoonup_{delivers}) \rightharpoonup_{objIsInstanceOf};$$

[ARIS-TaskProcess 3]:

$$\forall\, p : V_{Process};\ t : V_{Task} \mid p \rightharpoonup_{isProcessComponent} t \bullet$$

$$t \leftharpoonup_{executesTask} = p \leftharpoonup_{executesProcess} \&_{OrganizationalUnit}$$

end.

7.3 Anwendung des ARIS-Metaschemas

Das hier vorgestellte ARIS-Metaschema bildet die Konzepte der Modellierungsmittel des ARIS-Ansatzes ab. Gegenüber dem Metaschema des ARIS-Ansatzes in [Scheer, 1992] ist das Metaschema *ARIS* auf die wesentlichen Sprachmittel zur Organisationsmodellierung auf Fachkonzeptebene eingeschränkt.

Eingesetzt werden kann dieses Metaschema u. a. zur Einführung und Schulung der ARIS-Modellierungskonzepte, zur Entwicklung von Werkzeugen zur Modellierung nach dem ARIS-Ansatz (vgl. zur Entwicklung von Werkzeugen auf Basis von Metaschemata auch Kapitel 9) oder zum Vergleich mit anderen Modellierungsansätzen z. B. mit den Sprachmitteln der Unified Modeling Language (vgl. das UML-Metaschema in Kapitel 8).

8 Unified Modeling Language

In den späten 80er und frühen 90er Jahren wurden mehr als 60 objektorientierte Modellierungsmethoden entwickelt (vgl. z. B. [Partsch, 1998], [Süttenbach, 2000]), die sich in Modellierungstechniken und Vorgehensweisen unterscheiden. Übersichten zu objektorientierten Modellierungsmethoden finden sich z. B. in [Graham, 1994], [Frank, 1997b] und [Wieringa, 1998]. Mit der *Unified Modeling Language (UML)* [Booch et al., 1999], [Rumbaugh et al., 1999] wurde, ausgehend von den Methoden Object Modeling Technique (OMT) [Rumbaugh et al., 1991], Object Oriented Design (OOD) [Booch, 1994] und Object Oriented Software Engineering (OOSE) [Jacobson et al., 1993], ein Notationsstandard für objektorientierte Modellierungsmittel geschaffen, der im November 1997 durch die Object Management Group akzeptiert wurde (vgl. auch die Einordnung des UML-Metaschemas in Kapitel 4.3.1, S. 129).

UML wird in [Booch et al., 1999, S. xv], [Rumbaugh et al., 1999, S. 3] als allgemeine Modellierungssprache zur Beschreibung, Visualisierung, Konstruktion und Dokumentation von Softwaresystemen eingeführt. Neben der Beschreibung von Softwaresystemen wird die UML aber auch zur Beschreibung von Organisations- und Geschäftsprozeßmodellen eingesetzt (vgl. z. B. [Oestereich, 1998], [Versteegen, 1998]). Die UML stellt somit einen Satz standardisierter Beschreibungsmittel zur Modellierung von Organisationen und Softwaresystemen bereit.

Zur Einordnung der Modellierungsmittel unterscheidet die UML Beschreibungsmittel zur Darstellung struktureller und dynamischer Systemaspekte, die zu modellierende Systeme aus verschiedenen Sichten beschreiben (vgl. [Rumbaugh et al., 1999, S. 23ff]). Strukturelle Systemeigenschaften werden aus statischer Sicht und aus Anwendungsfall-Sicht notiert. Wie generell in objektorientierten Modellierungsansätzen, sind auch in der Unified Modeling Language Klassendiagramme das zentrale Beschreibungsmittel der statischen Sicht. Hierdurch werden die wesentlichen Konzepte des zu modellierenden Systems und deren Beziehungen auf Schemaebene modelliert. Zur exemplarischen Darstellung dieser Zusammenhänge werden darüber hinaus Objektdiagramme angeboten. Die Funktionalität des Systems aus Sicht der Systemverwendung wird durch Anwendungsfalldiagramme modelliert. Anwendungsfalldiagramme dienen zur Identifizierung der Anwendungsfälle, die das System unterstützt, und zur Darstellung der Akteure, die mit dem System interagieren. Die Modellierung dynamischer Systemaspekte hat die Darstellung des Systemverhaltens in der Zeit zum Ziel. Hierzu bietet die UML Notationen zur Modellierung des Systems aus Zustandssicht, aus Aktivitätssicht und aus Interaktionssicht. Zur Beschreibung aus Zustandssicht werden Statecharts verwendet, die die Zustände der Systemobjekte und deren Veränderungen abbilden. Dynamische Ablauffolgen in Software- und Geschäftsprozessen werden aus Aktivitätssicht mit Hilfe von Aktivitätsdiagrammen dargestellt und zur Beschreibung der Interaktion einzelner Systemkomponenten aus Interaktionssicht werden Sequenzdiagramme und Kollaborationsdiagramme eingesetzt. Das aus dem Referenz-Metaschema der Beschreibungs-

mittel für Organisationen und Softwaresysteme hergeleitete Metaschema der Unified Modeling Language, integriert die Konzepte dieser Beschreibungsmittel.

8.1 Integration der Metaschemata der UML-Beschreibungsmittel

Die Beschreibungsmittel der UML umfassen Notationen des Datenflußparadigmas, des Zustandsübergangsparadigmas, des Netzparadigmas, des Objekt-Instanzparadigmas, des Objekt-Interaktionsparadigmas und des Objekt-Beziehungsparadigmas. Das Metaschema der UML integriert daher die Referenz-Metaschemata dieser Paradigmen bzw. deren Spezialisierungen für die konkreten Notationen der UML.

Die Konzepte der Anwendungsfalldiagramme der UML werden durch die Spezialisierung *DFD-UseCase* des Referenz-Metaschemas des Datenflußparadigmas abgebildet (vgl. Kapitel 6.4.1). UML verwendet zur Darstellung von Zustandsübergangsbeschreibungen Statecharts, deren Struktur bereits durch das Referenz-Metaschema des Zustandsübergangsparadigmas *STP* formalisiert ist (vgl. Kapitel 6.4.2). Im Gegensatz zum Metamodell der UML in [OMG, 1999] werden Aktivitätsdiagramme im Referenz-Metaschema nicht als Darstellungsmittel des Zustandsübergangsparadigmas sondern als Beschreibungsmittel des Netzparadigmas aufgefaßt (vgl. hierzu auch Kapitel 6.4.3, Seite 195). Im hier vorgestellten Metaschema der UML werden Aktivitätsdiagramme daher durch die Spezialisierung *NPActivty* des Netzparadigmas modelliert. Die in der UML verwendete Notation für Objektdiagramme entspricht der Spezialisierung des Referenz-Metaschemas des Objekt-Instanzparadigmas *OIPOO* (vgl. Kapitel 6.5.1) für Objektdiagramme objektorientierter Methoden. Sequenzdiagramme und Kollaborationsdiagramme zur Modellierung der Objektinteraktion werden durch das Referenz-Metaschema des Objekt-Interaktionsparadigmas *IAP* direkt abgebildet (vgl. Kapitel 6.5.2). Zur Abbildung des Metaschemas der UML-Klassendiagramme ist auf die UML-Spezialisierung *ORPUML* des Referenz-Metaschemas des Objekt-Beziehungsparadigmas aus Kapitel 6.5.3 zurückzugreifen.

Das resultierende Metaschema der UML inkludiert diese sechs Teil-Metaschemata. Der *EER*-Teil des UML-Metaschemas *UML* ist in Abbildung 8.1 skizziert.

for *G* **in** *UML* **include**

DFPUseCase;	[Spezialisierung des Referenz-Metaschemas des Datenflußparadigmas für Anwendungsfalldiagramme]
STP;	[Referenz-Metaschema des Zustandsübergangsparadigma]
NPActivty;	[Spezialisierung des Referenz-Metaschemas des Netzparadigmas für Aktivitätsdiagramme]
OIPOO;	[Spezialisierung des Referenz-Metaschemas des Objekt-Instanzparadigmas für Objektdiagramme objektorientierter Methoden]
IAP;	[Referenz-Metaschema des Objekt-Interaktionsparadigmas]
ORPUML;	[Spezialisierung des Referenz-Metaschemas des Objekt-Beziehungsparadigmas für UML-Klassendiagramme]

end .

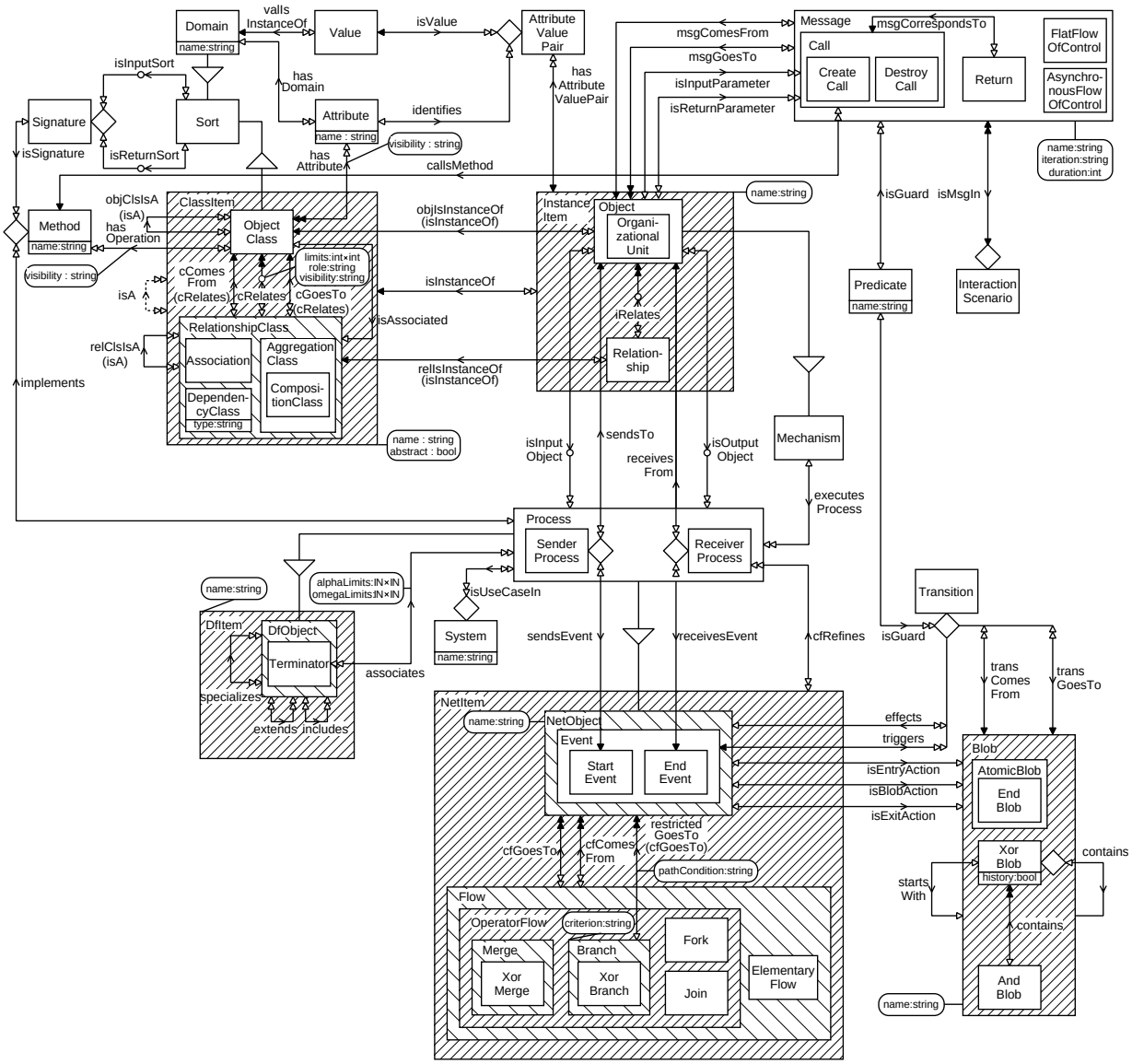


Abbildung 8.1: Spezialisierung des integrierten Referenz-Metaschemas für die Beschreibungsmittel der Unified Modeling Language (*UML*)

Da in den Anwendungsfalldiagrammen des Datenflußparadigmas keine Datenbezüge modelliert werden und in allen Metaschemata der in UML berücksichtigten Beschreibungsmitteln der Prozeßsicht (Anwendungsfalldiagramme, Aktivitätdiagramme und Statecharts) Prozesse bereits als konkret vereinbart sind, sind für das UML-Metaschema keine Anpassungen der *GRAL*-Zusicherungen der inkludierten Metaschemata erforderlich.

8.2 Paradigmenübergreifende *GRAL*-Zusicherungen

Mit den im UML-Metaschema zusammengefaßten Beschreibungsmitteln werden zu modellierende Systeme aus Prozeß- und aus Objektsicht modelliert. Die Konsistenz dieser Teilmodelle zueinander muß auch hier durch weitere *GRAL*-Zusicherungen formalisiert werden.

Zusicherungen für die Beschreibungsmittel der Prozeßsicht beziehen sich für das integrierte Referenz-Metaschema (vgl. Kapitel 6.6) und für das ARIS-Metaschema (vgl. Kapitel 7) auf die Balancierung der verschiedenen darstellbaren Prozeßzerlegungen. Prozeßzerlegungen werden mit den Sprachen der UML nur durch Aktivitätsdiagramme modelliert. Anwendungsfalldiagramme und Statecharts sehen hierfür keine Konzepte vor, so daß für die Integration der UML-Beschreibungsmittel der Prozeßsicht keine zusätzlichen *GRAL*-Prädikate nötig werden.

Sichten- und paradigmenübergreifende Zusicherungen des UML-Metaschemas beziehen sich auf die Abstimmung der Darstellungen von instanz- und schemaartigen Objekt-Beschreibungen und auf die Objektverwendung in Prozeßdarstellungen.

8.2.1 Paradigmenübergreifende Zusicherungen der Objektsicht

Objektzusammenhänge auf Instanzebene werden durch die Beschreibungsmittel des Objekt-Instanzparadigmas und des Objekt-Interaktionsparadigmas dargestellt. Diese strukturellen bzw. interaktionsbezogenen Objektdarstellungen sind mit den in Klassendiagrammen festgelegten Definitionen der entsprechenden Objekt- und Beziehungsklassen abzustimmen.

Die Inzidenz- und Attribut-Eigenschaften der Objekte in Objekt-, Sequenz- und Kollaborationsdiagrammen können analog zum Referenz-Metaschema (vgl. S. 231) mit den Definitionen aus Klassendiagrammen abgestimmt werden. Da UML-Modellierungen Beziehungen beliebiger Arität zulassen, sind hierbei kleinere Anpassungen erforderlich.

Die Knotentypen der Inzidenzen in Objektdiagrammen müssen mit den Definitionen der Beziehungsklassen übereinstimmen [UML-Object 1]. Ebenso müssen diese Inzidenzen den Kardinalitätsanforderungen des zugehörigen Klassendiagramms entsprechen [UML-Object 2]. Gegenüber den Zusicherung [RMS-Object 1] und [RMS-Object 2] des Referenz-Metaschemas ist für das UML-Metaschema die Ausrichtung der Kanten unerheblich, da Objektdiagramme Objektzusammenhänge ausschließlich durch ungerichtete Graphen darstellen. Die Zusicherungen zu verträglichen Attributstrukturen [UML-Object 3] und zur Berücksichtigung abstrakter Klassen [UML-Object 4] entsprechen den Zusicherungen des Referenz-Metaschemas. Da in UML-Klassendiagrammen Beziehungen generell als injektiv angenommen werden (vgl. Zusicherung [ORPUML 8]), sind in UML-Objektdiagrammen keine Mehrfachkanten gleichen Typs zugelassen [UML-Object 5].

Durch den Austausch von Nachrichten wird in Sequenz- und Kollaborationsdiagrammen das Interaktionsverhalten der Objekte modelliert. Durch Nachrichten können hierbei Methoden der Empfängerobjekte aufgerufen werden. Zur Konsistenzsicherung zwischen den Darstellungen des Objekt-Interaktionsparadigmas und den Darstellungen des Objekt-Beziehungsparadigmas ist sicherzustellen, daß nur solche Methoden aufgerufen werden, die der Klassendefinition des Empfängerobjekts entsprechen [UML-Object 6]. Für diese Nachrichten ist ebenfalls zu fordern, daß ihre Ein- und Ausgabeparameter der entsprechenden Methodensignatur genügen [UML-Object 7].

for G **in** *UML* **assert**

[UML-Object 1]:

$$\forall v : V_{Object};\ e : V_{Relationship} \mid v \leftharpoonup_{iRelates} e \bullet v \rightharpoonup_{objIsInstanceOf} \rightharpoonup^{*}_{objClsIsA} \leftharpoonup_{cRelates} \leftharpoonup^{*}_{relClsIsA} \leftharpoonup_{relIsInstanceOf} e ;$$

[UML-Object 2]:

$$\forall v : V_{Object};\ e : E_{cRelates} \mid v \rightharpoonup_{objIsInstanceOf} \rightharpoonup^{*}_{objClsIsA} \omega(e) \bullet first(e.limits) \leq \#\{ r : E_{iRelates} \mid \omega(r) = v \wedge \alpha(r) \rightharpoonup_{relIsInstanceOf} \rightharpoonup^{*}_{relClsIsA} \alpha(e)\} \leq second(e.limits) ;$$

[UML-Object 3]:

$$\forall i : V_{InstanceItem};\ c : V_{ClassItem} \mid i \rightharpoonup_{IsInstanceOf} c \bullet i \rightharpoonup_{hasAttributeValuePair} \leftharpoonup_{identifies} = c \rightharpoonup^{*}_{IsA} \rightharpoonup_{hasAttribute} ;$$

[UML-Object 4]:

$$\forall c : V_{ClassItem} \mid c.abstract \bullet c \leftharpoonup_{IsInstanceOf} = \varnothing ;$$

[UML-Object 5]:

$$\forall rc : V_{RelationshipClass};\ r_1, r_2 : V_{Relationship} \mid r_1 \rightharpoonup_{relIsInstanceOf} rc \leftharpoonup_{relIsInstanceOf} r_2 \wedge r_1 \neq r_2 \bullet r_1 \rightharpoonup_{iRelates} \neq r_2 \rightharpoonup_{iRelates} ;$$

[UML-Object 6]:

$$\forall o : V_{Object};\ oc : V_{ObjectClass}; \mid o \rightharpoonup_{objIsInstanceOf} oc \bullet o \leftharpoonup_{msgGoesTo} \&_{Call} \rightharpoonup_{callsMethod} \subseteq oc \rightharpoonup^{*}_{objClsIsA} \rightharpoonup_{hasOperation} ;$$

[UML-Object 7]:

$$\forall o : V_{Object};\ c : V_{Call}; \mid o \rightharpoonup_{isInputParameter} c \bullet o \rightharpoonup_{objIsInstanceOf} \rightharpoonup^{*}_{objClsIsA} \rightharpoonup_{isInputSort} \rightharpoonup_{isSignature} \leftharpoonup_{callsMethod} c ;$$

$$\forall o : V_{Object};\ c : V_{Call}; \mid o \rightharpoonup_{isOutputParameter} c \bullet o \rightharpoonup_{objIsInstanceOf} \rightharpoonup^{*}_{objClsIsA} \rightharpoonup_{isOutputSort} \rightharpoonup_{isSignature} \leftharpoonup_{callsMethod} c$$

end.

8.2.2 Paradigmenübergreifende Zusicherungen der Prozeß- und Objektsicht

In Aktivitätsdiagrammen kann der Objektfluß zwischen Prozessen modelliert werden. Implementieren diese Prozesse Methoden, so müssen die eingehenden bzw. ausgehenden Objekte auch mit der im Klassendiagramm notierten Methodensignatur bezüglich der Parametertypen als auch deren Reihenfolge verträglich sein [UML-ProcessObject 1].

for G **in** *UML* **assert**

[UML-ProcessObject 1]:

$$\forall p : V_{Process};\ m : V_{Method};\ s : V_{Signature};\ |\ p \rightharpoonup_{implements} m \leftharpoonup_{isSignature} s \bullet$$
$$p \leftharpoonup_{isInputObject} \rightharpoonup_{objIsInstanceOf} = s \leftharpoonup_{isInputSort} \wedge$$
$$\forall io : E_{isInputObject}\ |\ \omega(io) = p \bullet$$
$$\exists is : E_{isInputSort}\ |\ \omega(is) = s \wedge \alpha(io) \rightharpoonup_{objIsInstanceOf} \alpha(is) \bullet$$
$$order_{isInputObject}(io, p) = order_{isInputSort}(is, s)\ ;$$

$$\forall p : V_{Process};\ m : V_{Method};\ s : V_{Signature};\ |\ p \rightharpoonup_{implements} m \leftharpoonup_{isSignature} s \bullet$$
$$p \leftharpoonup_{isOutputObject} \rightharpoonup_{objIsInstanceOf} = s \leftharpoonup_{isOutputSort} \wedge$$
$$\forall oo : E_{isOutputObject}\ |\ \omega(oo) = p \bullet$$
$$\exists os : E_{isOutputSort}\ |\ \omega(os) = s \wedge \alpha(oo) \rightharpoonup_{objIsInstanceOf} \alpha(os) \bullet$$
$$order_{isOutputObject}(oo, p) = order_{isOutputSort}(os, s)$$

end.

8.3 Anwendung des UML-Metaschemas

Die Konzepte der wesentlichen UML-Modellierungsmittel werden im hier vorgestellten UML-Metaschema integriert abgebildet. Im Vergleich zum UML-Metaschema der Object Management Group [OMG, 1999] ist dieses Modell deutlicher von den Konzepten der einzelnen Beschreibungsmittel geprägt. Während im Metaschema der OMG ähnliche Konzepte in verschiedenen Vorkommen kaum unterschieden werden, sind die Konzepte einzelner Beschreibungsmittel im Metaschema *UML* eher auf ihren Anwendungsbezug im jeweiligen Modellierungskontext gerichtet. Die Integration der Teilschemata der einzelnen Modellierungsparadigmen stellt anschließend die Querbezüge der jeweiligen Konzepte her. Hierdurch wirkt das resultierende UML-Metaschema weniger abstrakt.

Wie auch das ARIS-Metaschema kann das UML-Metaschema u. a. zur Einführung und Schulung der UML-Sprachmittel, zur Entwicklung von Werkzeugen zur Modellierung mit der UML[1] (vgl. auch Kapitel 9) oder zum Vergleich der Modellierungsmächtigkeit mit Sprachmitteln anderer Modellierungsansätze z. B. mit den Sprachmitteln des ARIS-Ansatzes (vgl. das ARIS-Metaschema in Kapitel 7) eingesetzt werden.

[1] Auf Basis eines solchen Metaschemas für Klassendiagramme der UML wurde mit dem Meta-CASE-Werkzeug *KOGGE* ein Werkzeug zur Erstellung von UML-Klassendiagrammen und zum Roundtrip Engineering von JAVA-Programmen erstellt (vgl. `http://www.uni-koblenz.de/~case/kogge4/anleitungen/UML/UML.html`; 02.11.1999).

9 Software-Evaluation

Zur Unterstützung der vielfältigen Aufgaben und Geschäftsprozesse von Organisationen bietet der Markt eine große Anzahl unterschiedlicher Softwarewerkzeuge. Die Einschätzung und Auswahl der jeweils am besten geeigneten Lösung zum Einsatz in der Organisation erfordert eine tiefgehende Untersuchung des Leistungsspektrums der angebotenen Produkte, des Zusammenwirkens der Software mit vorhandenen bzw. zusätzlichen Produkten und der Einbindung der Software in die vorhandene organisatorische und technische Umgebung.

Mit *Software-Evaluation* bezeichnet man das strukturierte Vorgehen zur Analyse und Bewertung vorhandener Softwaresysteme für den Einsatz in einem fest umrissenen Anwendungsbereich [Franzke / Winter, 1996]. Ein Vorgehensmodell zur Software-Evaluation wurde in [Dumslaff et al., 1994] entwickelt und u. a. zur Evaluation von Branchenlösungen für das Handwerk [Dumslaff et al., 1992], von Branchenlösungen für die keramische Industrie [Franzke / Zenz, 1995] und von Branchenlösungen für Krankenhausinformationssysteme [Schumm et al., 1995] angewandt. Dieses Vorgehensmodell zur Softwareevaluation basiert auf einer umfassenden Anforderungsanalyse und Modellierung des Einsatzbereichs der Software. Das hierbei resultierende Organisationsmodell dient als werkzeugunabhängiger Vergleichsmaßstab, entlang dessen die angebotenen Softwareprodukte untersucht werden können.

Schwerpunkt der Organisationsmodellierung zur Software-Evaluation sind die zu unterstützenden Geschäftsprozesse. Diese werden mit den Mitteln der Datenflußmodellierung beschrieben. Zentrale Geschäftsprozesse werden zunächst durch ein Kontextdiagramm in ihrer Systemumgebung dargestellt und anschließend durch weitere Datenflußdiagramme verfeinert. Die hierbei gewonnene Verfeinerungshierarchie wird durch Aufgabengliederungspläne abgebildet und liefert somit eine vollständige Liste der zu unterstützenden Funktionen. Die datenbezogene Interaktion dieser Funktionen wird in den Datenflußmodellen abgebildet. Zur Modellierung der in den Geschäftsprozessen benötigten bzw. erzeugten Daten und Objekte werden Objekt-Beziehungsdiagramme verwendet. Die Einbettung der Geschäftsprozesse in die Organisationsstruktur erfolgt durch Organigramme. Diese beschreiben die aufbauorganisatorische Struktur der Organisation und definieren die Zuständigkeiten zur Bearbeitung einzelner Aufgaben und Teilprozesse.

In [Löcher / Pühler, 1997] wurde ein Werkzeug zur Unterstützung der Software-Evaluation (*SET-KOGGE*) entwickelt. Dieses Werkzeug erlaubt sowohl die Erstellung von integrierten Organisationsmodellen mit Hilfe von Datenflußdiagrammen, Aufgabengliederungsplänen, Objekt-Beziehungsdiagrammen und Organigrammen als auch die Dokumentation von Evaluationsergebnissen in der Form von Modellannotationen. Dieses Werkzeug wurde mit Hilfe des Meta-CASE-Werkzeugs *KOGGE* (Koblenzer Generator für graphische Entwurfsumgebungen) [Ebert

et al., 1997b] realisiert. Die Entwicklung eines *KOGGE*-Werkzeugs erfolgt durch Erstellen einer Werkzeugbeschreibung, die ein Konzeptmodell der verwendeten Modellierungstechniken, die Definition der Menüstruktur des Werkzeugs und die Festlegung der Interaktion mit dem Benutzer umfaßt. Diese Werkzeugbeschreibung wird durch das *KOGGE*-Basissystem interpretiert (vgl. zu *KOGGE* auch die Einordnung der *KOGGE*-Modelle als Metamodell in Kapitel 4.3.1, S. 128). Das diesem Werkzeug zugrunde liegende Konzeptmodell der verwendeten Modellierungstechniken wurde aus dem Referenz-Metaschema der Beschreibungsmittel für Organisationen und Softwaresysteme abgeleitet. Dieses Metaschema integriert die Konzepte von Aufgabengliederungsplänen, Organigrammen, Kontextdiagrammen, Datenflußdiagrammen und Objekt-Beziehungsdiagrammen.

9.1 Integration der Metaschemata zur Software-Evaluation

Das Metaschema des Werkzeugs zur Unterstützung der Organisationsmodellierung innerhalb der Software-Evaluation enthält die Referenz-Metaschemata des Aufgabengliederungsparadigmas, des Stellengliederungsparadigmas, des Datenflußparadigmas und des Objekt-Beziehungsparadigmas bzw. Spezialisierungen dieser Referenz-Metaschemata.

Durch die *SET-KOGGE* wird die Modellierung mit einfachen Aufgabengliederungsplänen unterstützt, wie sie auch im ARIS-Ansatz Verwendung finden. Hierbei wird lediglich die Zerlegung von Aufgaben in Teilaufgaben beschrieben. Unterschiedliche Aufgabentypen werden nicht berücksichtigt. Diese Aufgabengliederungspläne (Funktionsbäume) werden durch die Spezialisierung *TDPAris* des Aufgabengliederungsparadigmas (vgl. Kapitel 6.2.1) definiert. Aufbaustrukturen werden durch Abteilungsorganigramme dargestellt, die neben der Untergliederung der Organisation in Abteilungen und Stellen auch die Leitungsbeziehungen zwischen den Stellen abbilden. Solche Organigramme entsprechen der Spezialisierung *PDPDepartment* des Referenz-Metaschemas des Stellengliederungsparadigmas (vgl. Kapitel 6.3.1). Zur Modellierung der Organisation aus Prozeßsicht bietet die *SET-KOGGE* Kontextdiagramme und Datenflußdiagramme des Datenflußparadigmas. Neben den aus der strukturierten Modellierung bekannten Modellierungskonzepten wird auch die Modellierung der Aufgabenträger einzelner Prozesse und die Darstellung von Anwendungsszenarien zur Konkretisierung einzelner Geschäftsprozesse unterstützt. Diese Beschreibungsmittel werden durch das Referenz-Metaschema des Datenflußparadigmas *DFP* abgebildet (vgl. Kapitel 6.4.1). Die Modellierung von Daten- und Objektstrukturen erfolgt durch Objekt-Beziehungsdiagramme des *EER/GRAL*-Ansatzes, deren Metaschema durch das Referenz-Metaschema des Objekt-Beziehungsparadigmas *ORP* definiert ist (vgl. Kapitel 6.5.3).

Der zur Organisationsmodellierung in der *SET-KOGGE* benötigte Anteil des Konzeptmodells integriert diese vier Metaschemata.

for *G* **in** *SET* **include**

TDPAris;	[Spezialisierung des Referenz-Metaschemas des Aufgabengliederungsparadigmas für Funktionsbäume]
PDPDepartment;	[Spezialisierung des Referenz-Metaschemas des Stellengliederungsparadigmas für Abteilungsorganigramme]

DFP; [Referenz-Metaschema des Datenflußparadigmas]

ORP; [Referenz-Metaschema des Objekt-Beziehungsparadigmas]

end .

Abbildung 9.1 bilden den *EER*-Teil des resultierenden Metaschemas ab. Die Integration der vier Metaschemata erfolgt analog zur Definition des integrierten Referenz-Metaschemas entlang der gemeinsamen Konzepte der Teilschemata. Neben diesen Konzepten des Referenz-Metaschemas enthält das Metaschema der *SET-KOGGE* exemplarisch noch einige Konzepte (*Evaluation*, *EvaluationEntry*, *Product*, *SoftwareEvaluation*) zur Dokumentation der Software-Evaluation entlang der modellierten Aufgaben.

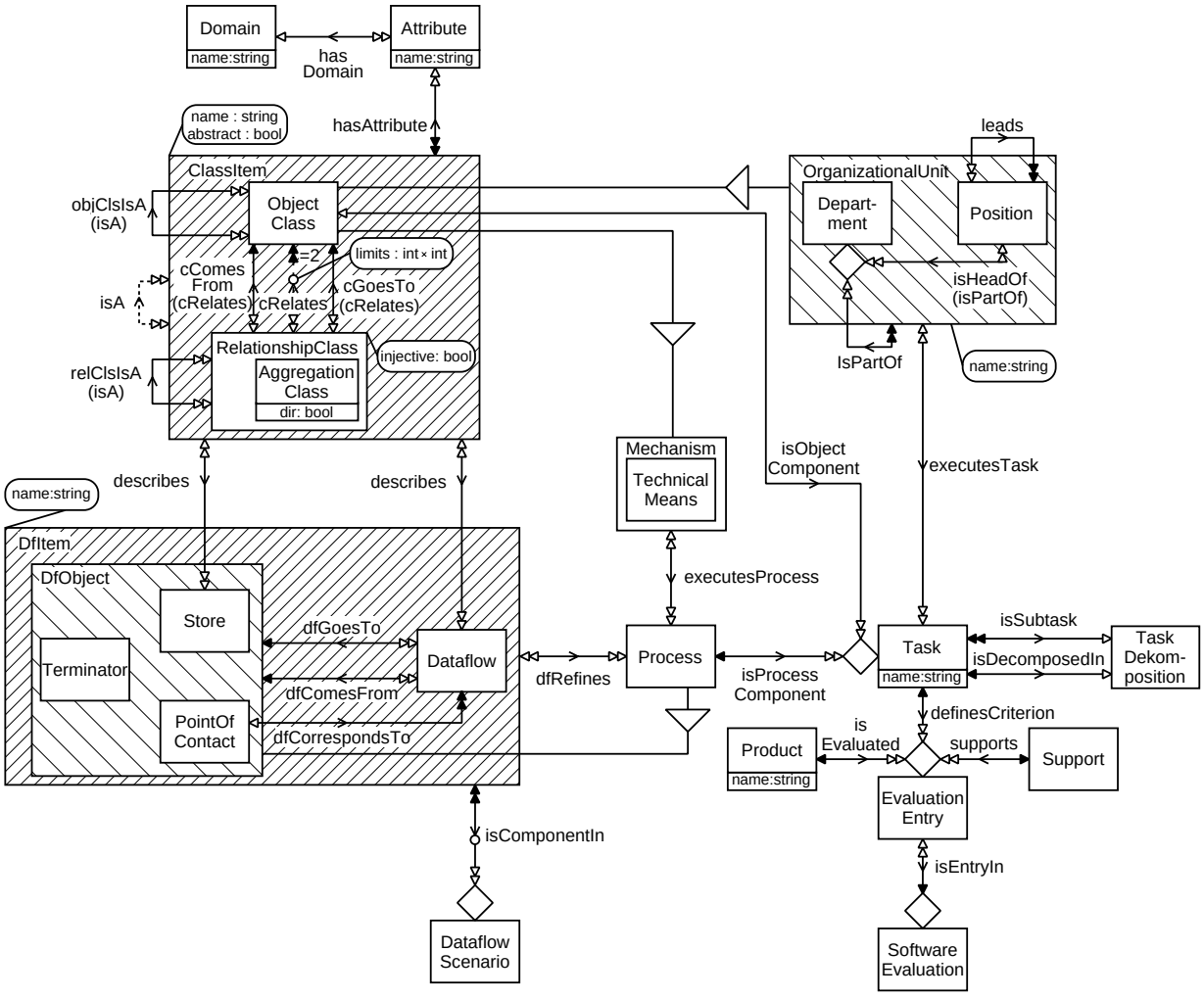


Abbildung 9.1: Spezialisierung des integrierten Referenz-Metaschemas für die Beschreibungsmittel zur Software-Evaluation (*SET*)

Das *SET-KOGGE*-Metaschema integriert das Referenz-Metaschema des Datenflußparadigmas und des Objekt-Beziehungsparadigmas. Datenbezüge in Datenflußdiagrammen werden somit durch die Konzepte des Objektbeziehungsparadigmas abgebildet. Die Zusicherungen des Datenflußparadigmas mit Bezug zur Daten- und Objektmodellierung ([DFP 10] und [DFP 11]) sind daher analog zum integrierten Referenz-Metaschema anzupassen [SET 1]–[SET 3].

for G **in** SET **assert**

[SET 1]:

overwrites [DFP 10] :

$$\forall\, poc : V_{PointOfContact} \bullet$$
$$poc \rightharpoonup_{dfCorrespondsTo} \leftharpoonup_{describes} (\leftharpoonup_{isA} \mid (\leftharpoonup_{cGoesTo} \&_{AggregationClass} \rightharpoonup_{cComesFrom}))^* \rightharpoonup_{describes} (\rightharpoonup_{dfGoesTo} \mid \rightharpoonup_{dfComesFrom})\, poc\ ;$$

[SET 2]:

overwrites [DFP 11] :

$$\forall\, d : V_{Dataflow} \bullet$$
$$d \rightharpoonup_{dfGoesTo} \rightharpoonup_{dfComesFrom} \&_{Store} \leftharpoonup_{describes} (\leftharpoonup_{isA} \mid (\leftharpoonup_{cGoesTo} \&_{AggregationClass} \rightharpoonup_{cComesFrom}))^* \rightharpoonup_{describes} d\ ;$$

[SET 3]:

$$V_{AtomicClass} = V_{AlternativeClass} = V_{IteratedClass} = V_{SequentialClass} = \varnothing$$

end.

9.2 Paradigmenübergreifende *GRAL*-Zusicherungen

Für die vier Modellierungssichten, Aufgabensicht, Aufbausicht, Prozeßsicht und Objektsicht, inkludiert das *SET-KOGGE*-Metaschema jeweils ein Teil-Metaschema. Zusicherungen zur Abstimmung verschiedener Beschreibungparadigmen innerhalb einer Sicht sind daher nicht erforderlich. Die direkten Querbezüge zwischen den Darstellungen durch Aufgabengliederungspläne, Organigramme, Datenflußdiagramme und Objekt-Beziehungsdiagramme wurden bereits entlang der gemeinsamen Konzepte in den Teilmetaschemata hergestellt.

Abzugleichen sind noch die Darstellungen der Prozeßsicht mit den Beschreibungen der Aufgabensicht bezüglich der Verfeinerungsstrukturen, der verwendeten Objekte und der mit der Aufgabenerledigung betrauten Organisationseinheiten.

9.2.1 Paradigmenübergreifende Zusicherungen der Aufgaben- und Prozeßsicht

In Aufgabengliederungsplänen und Datenflußdiagrammen werden Prozesse bzw. Aufgaben in Teilprozesse und -aufgaben zerlegt. Die Aufgabengliederung muß hierbei mit der Gliederung der entsprechenden Prozeßkomponenten der modellierten Aufgaben übereinstimmen [SET-TaskProcess 1]. Ebenso werden Aufgaben in Aufgabengliederungsplänen Objektkomponenten zugeordnet. Diese Objektkomponenten müssen sich in den Ein- bzw. Ausgabedatenflüssen der Datenflußmodellierung der entsprechenden Prozeßkomponente wiederfinden [SET-TaskProcess 2]. Bezüge zu Aufgaben- bzw. Prozeßträgern können sowohl in Aufgabengliederungsplänen als auch in Datenflußdiagrammen hergestellt werden. Auch hier müssen Aufgabenträger der Aufgaben mit denen ihrer Prozeßkomponenten übereinstimmen [SET-TaskProcess 3].

Vergleiche hierzu auch die Zusicherungen [RMS-TaskProcess 1] bis [RMS-TaskProcess 3] des integrierten Referenz-Metaschemas.

for *G* **in** *SET* **assert**

[SET-TaskProcess 1]:

$$\forall\, p : V_{Process};\ t : V_{Task} \mid p \rightharpoonup_{isProcessComponent} t \bullet$$
$$\delta_{\leftharpoonup_{dfRefines}}(p) > 0 \ \wedge\ \delta_{\rightharpoonup_{isDecomposedIn}}(t) > 0 \ \Rightarrow$$
$$p \leftharpoonup_{dfRefines} \&_{Process} = t \rightharpoonup_{isDecomposedIn} \leftharpoonup_{isSubtask} \leftharpoonup_{isProcessComponent};$$

[SET-TaskProcess 2]:

$$\forall\, p : V_{Process};\ t : V_{Task} \mid p \rightharpoonup_{isProcessComponent} t \bullet$$
$$t \leftharpoonup_{isObjectComponent} \subseteq p(\leftharpoonup_{dfComesFrom} \mid \leftharpoonup_{dfGoesTo}) \leftharpoonup_{describes};$$

[SET-TaskProcess 3]:

$$\forall\, p : V_{Process};\ t : V_{Task} \mid p \rightharpoonup_{isProcessComponent} t \bullet$$
$$t \leftharpoonup_{executesTask} = p \leftharpoonup_{executesProcess} \&_{OrganizationalUnit}$$

end.

9.3 Anwendung des Metaschemas zur Software-Evaluation

Das Metaschema *SET* ist ein Teil der Werkzeugbeschreibung des *KOGGE*-Werkzeugs *SET-KOGGE* [Löcher / Pühler, 1997]. Dieses Metaschema legt die Datenstruktur des Werkzeugs zur Speicherung der Organisationsmodelle und der notwendigen Modellannotationen fest und definiert die vom Werkzeug überprüften Regeln zur Konsistenzsicherung der Teilmodelle.

Abbildung 9.2 zeigt einen Bildschirmabzug der *SET-KOGGE* zur Evaluation von Branchenlösungen zur Unterstützung von Krankenhausinformationssystemen. Abgebildet sind hier Teile eines Aufgabengliederungsplans, eines Organigramms, eines Kontextdiagramms und eines Editors zur Erfassung angebotener Softwareunterstützung zu einer Aufgabe.

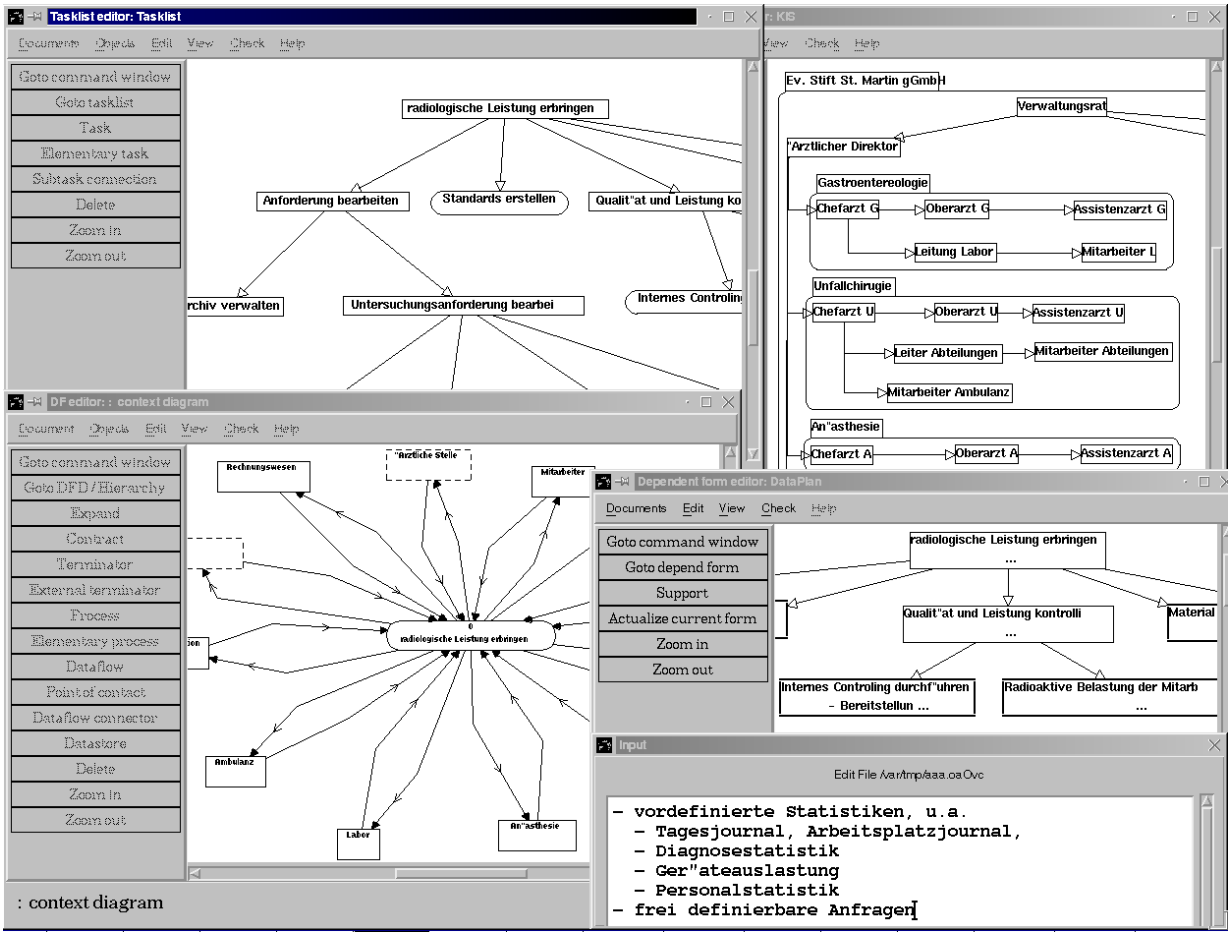


Abbildung 9.2: *SET-KOGGE*

10 Zusammenfassung und Ausblick

Die Modellierung von Organisationen und Softwaresystemen basiert heute auf multiperspektivischen Darstellungen, in denen Teilmodelle unterschiedlicher Sichten miteinander kombiniert werden. Zur Darstellung dieser Teilmodelle verwenden die Methoden der Organisations- und Softwaretechnik eine kaum überschaubare Vielzahl unterschiedlicher visueller Modellierungssprachen, die sich sowohl in ihren konkreten Notationen als auch in den jeweils dargestellten Systemkonzepten und deren Beziehungen zueinander unterscheiden. Modelle konkreter Organisationen oder Softwaresysteme, die entlang solcher Methoden erstellt werden, bestehen aus mehreren Teilmodellen, die jeweils unterschiedliche Systemaspekte besonders herausstellen. Sowohl zur Sicherstellung der Konsistenz der einzelnen Teilmodelle zueinander als auch zur Entwicklung von Modellierungswerkzeugen sind die jeweils in den Methoden zusammengefaßten Beschreibungsmittel zu definieren und die Querbezüge zwischen den beschriebenen Konzepten festzulegen.

Bezüglich der Vorgehensweisen und der eingesetzten Modellierungsmittel weisen die Modellierung und die Gestaltung von Organisationen und Softwaresystemen große Ähnlichkeiten auf. Auch bedingen sich Organisationsentwicklung und Softwareentwicklung gegenseitig. Gestaltungsmaßnahmen innerhalb einer Organisation werden einerseits durch die vorhandenen Softwaresysteme beeinflußt, wirken anderseits aber auch auf die Entwicklung bzw. die Anpassung dieser Softwaresysteme zurück. In ähnlicher Weise wird die Entwicklung von Softwaresystemen durch die Organisation beeinflußt und wirkt ihrerseits wieder auf die Organisation, die i. allg. für den bestmöglichen Einsatz der Software anzupassen ist. Vor dem Hintergrund einer durchgängigen Modellierung von Organisationen und Softwaresystemen bietet sich daher auch eine gemeinsame Betrachtung der visuellen Modellierungssprachen der Organisations- und Softwaretechnik an.

In dieser Arbeit wurde daher ein ***Klassifikationsschema der Beschreibungsmittel*** der Organisations- und Softwaretechnik entwickelt und ein ***integriertes Metaschema*** erstellt, das die konzeptionellen Grundlagen und Zusammenhänge dieser visuellen Modellierungssprachen definiert und als ***Referenz*** u. a. zur Auswahl und Einordnung von Beschreibungsmitteln in Modellierungsmethoden und zur Entwicklung von Modellierungswerkzeugen dienen kann.

Klassifikation der Beschreibungsmittel

Zur Einordnung und Strukturierung der Beschreibungsmittel der Organisations- und Softwaretechnik wurde ein *Klassifikationsschema* entwickelt. Ausgehend von den wesentlichen, zur Modellierung von Organisationen und Softwaresystemen betrachteten Aspekten wurden die zentralen Modellierungssichten *Aufgabensicht*, *Aufbausicht*, *Prozeßsicht* und *Objektsicht* eingeführt.

Beschreibungsmittel dieser Sichten stellen jeweils die Konzepte *Aufgabe*, *Organisationseinheit*, *Prozeß* bzw. *Objekt* besonders heraus. Zur Beschreibung von Organisationen und Softwaresystemen aus diesen Sichten wurden verschiedene *Beschreibungsparadigmen* identifiziert, die unterschiedliche Beziehungsgeflechte zwischen den jeweils modellierten Konzepten betonen.

Das dieser Arbeit zugrunde liegende Klassifikationsschema strukturiert die zur Modellierung von Organisationen und Softwaresystemen eingesetzten Beschreibungsmittel entlang dieser Sichten und Paradigmen.

Durch Abstraktion von konkreten Notationen der einzelnen Beschreibungsmittel erlaubt dieses Klassifikationsschema eine sprachunabhängige Untersuchung und Darstellung der in dieser Arbeit behandelten visuellen Modellierungssprachen. Die Darstellung und Einordnung dieser Beschreibungsmittel erfolgte entlang der in den Paradigmen zusammengefaßten Modellierungskonzepte und deren Beziehungen. Das Klassifikationsschema hat sich in dieser Arbeit sowohl zur Einführung diverser Beschreibungsmittel als auch zur strukturierten Entwicklung des integrierten Referenz-Metaschemas als umfassendes und valides Hilfsmittel erwiesen.

Integriertes Referenz-Metaschema

Die Definition des integrierten Referenz-Metaschemas der visuellen Modellierungssprachen für Organisationen und Softwaresysteme erfolgte mit den Mitteln der graphbasierten Konzeptmodellierung entlang der Beschreibungsparadigmen des Klassifikationsschemas. Zu jedem Beschreibungsparadigma wurde ein *Referenz-Metaschema* entwickelt, das die wesentlichen von ihm betrachteten Modellierungskonzepte und deren Beziehungen abbildet.

Die Referenz-Eigenschaft dieser paradigmenbezogenen Metaschemata wurde durch Ableitung von Metaschemata konkreter Beschreibungsmittel der jeweils betrachteten Paradigmen nachgewiesen. Die Metaschemata der wesentlichen Vertreter der einzelnen Paradigmen konnten durch minimale Einschränkungen und Ergänzungen aus dem entsprechenden Referenz-Metaschema entwickelt werden.

Diese paradigmenbezogenen Metaschemata besitzen bereits die Eigenschaften von Referenzmodellen. Sie sind einerseits so *allgemeingültig*, daß die Ableitung spezialisierter Metaschemata leicht möglich ist. Anderseits sind sie aber auch so *konkret* und *vollständig*, daß diese Ableitungen nur marginale Anpassungen der Referenz-Metaschemata erfordern. Analog zum Nachweis der Anwendbarkeit von Entwurfsmustern wurden zu jedem Referenz-Metaschema mehrere konkrete Anwendungen aufgeführt, die in Einzelfällen bereits den Metaschemata konkreter Beschreibungsmittel entsprachen. Die *Erweiterbarkeit* der Referenz-Metaschemata wird durch den verwendeten Modellierungsansatz *EER/GRAL* sichergestellt. Da die Ableitung mehrerer spezialisierter Metaschemata aus den Referenz-Metaschemata leicht möglich war, kann auch davon ausgegangen werden, daß Spezialisierungen für hier nicht berücksichtigte konkrete Modellierungsmittel durch einfache Anpassungen möglich sind.

Die *Integrationsfähigkeit* der paradigmenbezogenen Referenz-Metaschemata wurde bei der Zusammenfassung zum integrierten Referenz-Metaschema der Beschreibungsmittel für Organisationen und Softwaresysteme gezeigt. Entlang der Modellierungskonzepte, die in den verschiedenen Paradigmen gemeinsam abgebildet werden, wurden die einzelnen Referenz-Metaschemata

vereinigt. Dieses integrierte Referenz-Metaschema definiert die zentralen, heute zur Modellierung von Organisationen und Softwaresystemen dargestellten Konzepte und deren Beziehungen in ihrem multiperspektivischen Kontext.

Das integrierte Referenz-Metaschema wurde zur Ableitung spezialisierter, integrierter Metaschemata konkreter Modellierungsmethoden und -sprachen zur Organisations- und Softwaremodellierung eingesetzt. So wurden hieraus exemplarisch Metaschemata zur Definition der Beschreibungskonzepte des ARIS-Ansatzes, der Unified Modeling Language und der Organisationsmodellierung im Rahmen der Software-Evaluation abgeleitet. Auch diese Spezialisierungen des integrierten Referenz-Metaschemas erforderten nur geringfügige Anpassungen der hierbei eingebundenen paradigmenbezogenen Referenz-Metaschemata und der zur Integration erforderlichen Zusicherungen. In ähnlicher Weise können aus dem integrierten Referenz-Metaschema auch Metaschemata anderer Ansätze und Sprachen zur Modellierung von Organisationen und Softwaresystemen abgeleitet werden.

Neben der zuvor skizzierten Verwendung der Referenz-Metaschemata als *Modellierungsmittel* zur Erstellung spezialisierter Metaschemata konkreter Methoden und Techniken erlaubt das integrierte Referenz-Metaschema die konzeptbasierte Beschreibung und Untersuchung von Modellierungsmethoden. Unabhängig von konkreten Notationen kann die *Schulung* von Modellierungsmethoden entlang der darzustellenden Konzepte und deren Beziehungen erfolgen. Das integrierte Referenz-Metaschema stellt die Modellierungskonzepte in ihrem sichtenübergreifenden Zusammenhang dar. Daher eignet es sich auch zur Festlegung einer einheitlichen und methodenübergreifenden *Terminologie* der insgesamt im Rahmen der Organisations- und Softwaremodellierung zu beschreibenden Konzepte. Ebenso bietet das integrierte Referenz-Metaschema einen von konkreten Modellierungsmethoden unabhängigen Maßstab zum *Vergleich* und zur *Diskussion* unterschiedlicher Modellierungsansätze. Insbesondere die Betrachtung der Abweichungen der Metaschemata konkreter Modellierungsansätze vom Referenz-Metaschema ermöglicht Aussagen über die Mächtigkeit der zu vergleichenden Ansätze hinsichtlich der Unterstützung der multiperspektivischen Modellierung aus den vier Modellierungssichten und den hierbei verwendeten Modellierungsparadigmen.

Das integrierte Referenz-Metaschema liefert darüber hinaus auch einen wesentlichen Beitrag zur *Erstellung von Softwarewerkzeugen* zur Unterstützung der Modellierung von Organisationen und Softwaresystemen. Das Referenz-Metaschema definiert hierzu die Repository-Struktur zur internen Verwaltung der Modelldaten und formalisiert die Konsistenz der Teilmodelle unterschiedlicher Beschreibungsparadigmen zueinander. Die Anwendung des integrierten Referenz-Metaschemas zur Entwicklung von Modellierungswerkzeugen wurde anhand eines Werkzeugs zur Unterstützung der Softwareevaluation gezeigt. Auf Basis des KOGGE Meta-CASE-Ansatzes können in ähnlicher Weise Modellierungswerkzeuge zur Unterstützung weiterer Ansätze zur Organisations- und Softwaremodellierung erstellt werden.

Ausblick

Neben den Anwendungsbereichen, die in der vorliegenden Arbeit diskutiert wurden, eröffnen sowohl das hier entwickelte Referenz-Metaschema der Beschreibungsmittel für Organisationen und Softwaresysteme als auch der hierzu verwendete Ansatz zur Referenz-Metamodellierung weitergehende Anwendungsbereiche im Method-Engineering und im Werkzeugbau.

Auf syntaktischer Ebene dient das integrierte Referenz-Metaschema u. a. zur *Konsolidierung* des Method-Engineerings, und es erlaubt damit die sichten- und paradigmenübergreifende *Integration* der Modellierungsmethoden der Organisations- und Softwaretechnik. Es bildet die Grundlage zur Unterstützung der *Interoperabilität* von Modellierungswerkzeugen und definiert eine Basis zur Formalisierung der *Semantik* von Modellierungssprachen.

Das Referenz-Metaschema legt die Konzepte und Beziehungen der zur Modellierung verwendeten visuellen Sprachen fest und dient hiermit zur *Konsolidierung* der Methoden und Techniken in Organisations- und Softwaretechnik.

Die abstrakte Syntax der Modellierungsmittel wird durch das Referenz-Metaschema, unabhängig von konkreten Notationen, in einer allgemeingültigen Form beschrieben. Durch die Berücksichtigung der heute zur Modellierung von Organisationen und Softwaresystemen verwendeten Beschreibungsmitteln repräsentiert es den aktuellen Stand dieser Modellierungsmittel auf Ebene der abstrakten Syntax.

Wird das hier vorgeschlagene Metaschema als allgemeingültiges Referenz-Metaschema der zur Modellierung eingesetzten Notationen akzeptiert, kann es aufgrund des damit verbundenen normativen Charakters als ein stabiles Fundament zur Entwicklung neuer Beschreibungsformen dienen. Mit Ausnahme weniger, marginaler Anpassungen des Referenz-Metaschemas kann sich die Gestaltung neuer Modellierungsnotationen auf die Entwicklung ergonomischer und an den Modellierungskontext angepaßter konkreter Notationen und auf die Formalisierung der Semantik dieser Modellierungssprachen konzentrieren.

Die Modellierungsansätze der Organisationstechnik und der Softwaretechnik wurden bislang eher unabhängig voneinander betrachtet. Durch das hier entwickelte Referenz-Metaschema wird ein Beitrag zur *Integration* dieser Techniken geleistet.

Diese unabhängige Betrachtung von Organisationen und Softwaresystemen hatte u. a. zur Folge, daß bei der Übertragung von Organisationsmodellen in Softwaremodelle bzw. von Softwaremodellen in Organisationsmodelle weitreichende Überarbeitungen nötig wurden. Auch wenn zur Organisationsmodellierung und zur Softwaremodellierung weiterhin unterschiedliche Notationen verwendet werden, können diese Darstellungen aufgrund der integrativen Ausrichtung des Referenz-Metaschemas auf einem gemeinsamen Metaschema basieren.

Das hier vorgeschlagene Referenz-Metaschema faßt die Konzepte der visuellen Modellierungsprachen der Organisationstechnik und die Konzepte der visuellen Modellierungsprachen der Softwaretechnik in einem gemeinsamen Modell zusammen. Hierdurch werden insbesondere die Querbezüge zwischen Modellen aus Sicht der Organisation i. e. S. und Modellen aus Sicht der Softwaresysteme i. e. S. spezifiziert. Das integrierte Referenz-Metaschema bietet hierdurch eine Grundlage zur Vereinigung der Organisations- und Softwaremodelle und verwischt die Grenzen zwischen Organisations- und Softwaretechnik auf notationeller Ebene.

Zur Modellierung von Organisationen und von Softwaresystemen existiert ein weites Spektrum verschiedener Softwarewerkzeuge, die unterschiedliche Modellierungsmethoden und Notationen unterstützen. Die Übernahme von vorhandenen Modellen in Werkzeuge, die erweiterte Funktionalitäten anbieten oder modernere Notationen bereitstellen, ist kaum möglich, da es z. Z. keinen allgemein akzeptierten Mechanismus zur *Interoperabilität* von unterschiedlichen Modellierungswerkzeugen gibt. Das Referenz-Metaschema kann jedoch als Grundlage dienen, die Interoperabilität dieser Werkzeuge zu ermöglichen.

Die Kombination verschiedener Modellierungswerkzeuge setzt insbesondere Kenntnis der durch diese Werkzeuge abgebildeten Modellierungskonzepte voraus. Die Interoperabilität zwischen Modellierungswerkzeugen erfordert eine Einigung über die jeweils in den Werkzeugen verwendeten Metaschemata und über die Metaschemata, die die gemeinsame Nutzung der Modelldaten ermöglichen. Durch das Referenz-Metaschema wird sowohl eine Unterstützung zur Ableitung der individuellen Metaschemata der einzelnen Werkzeuge geboten, als auch die Festlegung eines den aktuellen Interoperabilitätskontext beschreibenden Metaschemas ermöglicht.

Auf Basis dieser Metaschemata können, mit Hilfe entsprechender Mechanismen zur Schematransformation, Überführungsfunktionen zur Unterstützung der Datenmigration zwischen unterschiedlichen Modellierungswerkzeugen in konkreten Interoperabilitätsszenarien abgeleitet werden. Der Austausch von Modelldaten zwischen unterschiedlichen Werkzeugen erfordert, neben den Modelldaten selbst, auch die schematische Beschreibung dieser Daten durch ein Metaschema. Zum gemeinsamen Austausch von Schemainformationen und Modelldaten wurde in [Ebert et al., 1999a] das *GraX*-Austauschformat (graph exchange format) im Kontext der Interoperabilität von Software-Reengineering-Werkzeugen vorgestellt. In analoger Form können hiermit auch Organisations- und Softwaremodelle, einschließlich der zugehörigen Metaschemata, zwischen Modellierungswerkzeugen ausgetauscht werden.

Die Betrachtungen der Modellierungsmittel in dieser Arbeit bezogen sich ausschließlich auf syntaktische und notationelle Aspekte. Nicht betrachtet wurde die *Semantik* dieser Beschreibungsmittel. Durch die in dieser Arbeit definierten bzw. abgeleiteten Metaschemata wurde lediglich die syntaktische Korrektheit der hierdurch abgebildeten Modelle festgelegt. Die Bedeutung dieser Modelle wurde nur soweit berücksichtigt, wie es für ein intuitives Verstehen der Darstellungstechniken und der Festlegung der abgebildeten Konzepte und deren Beziehungen notwendig war.

Das Referenz-Metaschema bietet jedoch auch die Grundlage zur Formalisierung der Semantik dieser Beschreibungsmittel. In [Ebert / Süttenbach, 1997b], [Süttenbach, 1998], [Süttenbach, 2000] wird ein Ansatz zur operationalen Definition der Semantik visueller Sprachen objektorientierter Methoden vorgestellt, der auf Metaschemata dieser Sprachen basiert.

Ebenso erlaubt der in dieser Arbeit vorgestellte und verwendete Ansatz zur Referenz-Metamodellierung auch die Anwendung in anderen Bereichen der Softwaretechnik, z. B. im *Software-Reengineering*, und eröffnet vielfältige Möglichkeiten zur Weiterentwicklung der *Metatechnologie*.

Die Methoden und Techniken des *Software-Reengineerings* untersuchen Softwaresysteme aus unterschiedlichen Sichten. Hierbei ist ein weites Spektrum zwischen sehr grobgranularen Softwarerepräsentationen zur Analyse auf der Ebene der Softwarearchitektur und sehr feingranularen Betrachtungen zur Analyse von Daten- und Kontrollflußabhängigkeiten auf Ebene abstrakter Syntaxbäume abzubilden. Ebenso sind auch Softwaresysteme unterschiedlicher Programmiersprachen, Datenbanksprachen und Job-Control-Sprachen, die auch kombiniert eingesetzt werden, zu berücksichtigen.

Mit Hilfe des in dieser Arbeit verwendeten Modellierungsansatzes können auch Referenz-Metaschemata für ausgewählte Anwendungsszenarien des Software-Reengineerings entwickelt werden. Im *GUPRO*-Projekt (vgl. u. a. [Ebert et al., 1998], [Kullbach et al., 1998], [Kullbach / Winter, 1999]) wurden bereits solche *EER/GRAL*-Metaschema für integrierte Softwaresysteme auf Architekturebene und für feingranulare Daten- und Kontrollflußanalysen entwickelt. Die-

se Metaschemata können als Ausgangspunkt zur Entwicklung von *Referenz-Metaschemata des Software-Reengineerings* dienen.

Auch der in dieser Arbeit verwendete *EER/GRAL*-Ansatz zur graphbasierten Konzeptmodellierung bietet noch umfangreichen Spielraum zur Weiterentwicklung der hier zur Modellierung bereitgestellten *Metatechnologie*.

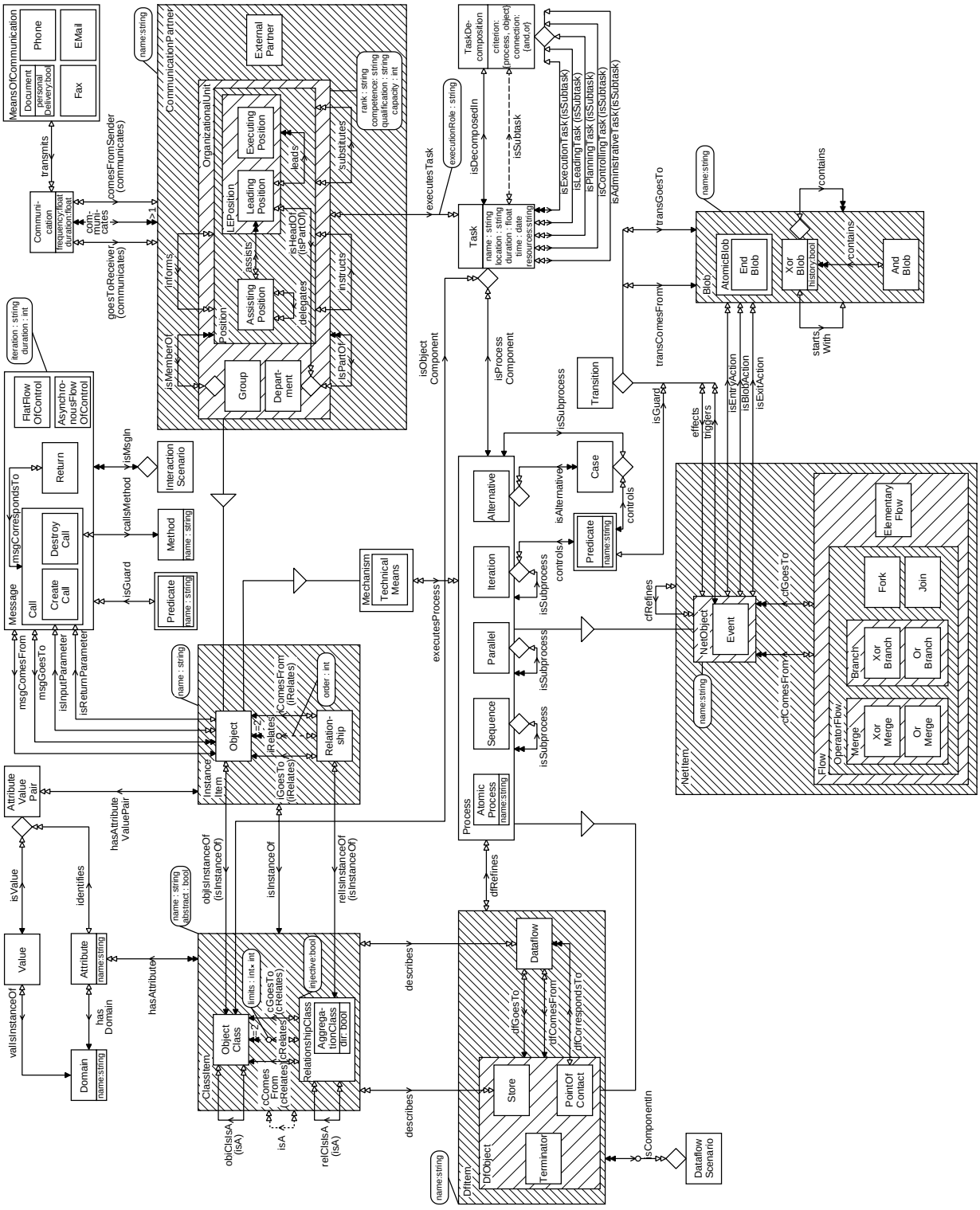
Während die Ableitung spezieller Metamodelle in dieser Arbeit eher intuitiv erfolgte, könnte durch die Formalisierung von Operationen auf Metaschemata die Spezialisierung von Referenz-Metaschemata deutlicher strukturiert und vereinfacht werden. Beispielsweise könnten, ausgehend von Referenz-Metaschemata und zugehörigen Ableitungsfunktionen zur Spezialisierung, auf Basis von Schema-Algebren, Transformationsfunktionen zur Migration von Modelldaten unterschiedlicher Schemata generiert werden. Solche Mechanismen könnten insbesondere im Rahmen der Re-Technologien zur Transformation von Modellen und Programmsystemen eingesetzt werden.

Mit dem in dieser Arbeit vorgestellten und validierten Referenz-Metaschema der Beschreibungsmittel für Organisationen und Softwaresysteme wurde ein Beitrag zur Konsolidierung und Stabilisierung des aktuellen Standes des Method-Engineerings im Kontext der integrierten Modellierung von Organisationen und Softwaresystemen auf notationeller Basis geleistet. Sowohl dieses integrierte Referenz-Metaschema als auch der zu seiner Erstellung verwendete *EER/GRAL*-Ansatz erlauben, über diese Arbeit hinaus, eine weitergehende Anwendung in der Organisations- und Softwaretechnik.

A Formalisierung des Referenz-Metaschemas der Beschreibungsmittel für Organisationen und Softwaresysteme (zusammenfassende Darstellung)

In den folgenden Kapiteln wird die Definition des Referenz-Metaschemas der Beschreibungsmittel für Organisationen und Software-Systeme zusammengefaßt dargestellt. Auf das *EER*-Diagramm in Kapitel A.1 folgen in Kapitel A.2 bis A.6 die *GRAL*-Zusicherungen der Einzelparadigmen und die paradigmenübergreifenden Zusicherungen.

A.1 *EER*-Modell



A.2 *GRAL*-Zusicherungen der Aufgabensicht

A.2.1 *GRAL*-Zusicherungen des Aufgabengliederungsparadigmas

for G **in** *RMS* **assert**
[RMS-TDP 1]:
$isTree(eGraph(E_{isDecomposedIn} \cup E^{-}_{isSubtask}))$
end.

for G **in** *RMS* **define**
$mainTask : V_{Task}$
where
$mainTask = root(eGraph(E_{isDecomposedIn} \cup E^{-}_{isSubtask}))$
end.

A.3 *GRAL*-Zusicherungen der Aufbausicht

A.3.1 *GRAL*-Zusicherungen des Stellengliederungsparadigmas

for G **in** *RMS* **assert**
[RMS-PDP 1]:
$isDag(eGraph(E_{leads}))$;
$isDag(eGraph(E_{instructs}))$;
$isDag(eGraph(E_{substitutes}))$;
$isDag(eGraph(E_{delegates}))$;

[RMS-PDP 2]:
$isTree(eGraph(E^{-}_{isPartOf}))$;

[RMS-PDP 3]:
$\forall\, d : V_{Department};\ h : V_{LeadingPosition} \mid h \rightharpoonup_{isHeadOf} d \bullet$
$d \leftharpoonup^{+}_{isPartOf} \&_{Position} \subseteq h(\rightharpoonup_{leads} | \rightharpoonup_{instructs})^{*}$
end.

A.3.2 *GRAL*-Zusicherungen des Kommunikationsparadigmas

for G **in** *RMS* **assert**

[RMS-CommP 1]:

$$\forall\, c : V_{Communication} \mid \delta_{\rightharpoonup_{comesFromSender}}(c) = 1 \vee \delta_{\rightharpoonup_{goesToReceiver}}(c) = 1 \bullet \delta_{\rightharpoonup^{type}_{communicates}}(c) = 0$$

end.

A.4 *GRAL*-Zusicherungen der Prozeßsicht

A.4.1 *GRAL*-Zusicherungen des Datenflußpardigmas

for G **in** *RMS* **assert**

[RMS-DFP 1]:

$$\{s_1, s_2 : V_{Store} \mid s_1 \leftharpoonup_{dfComesFrom} \rightharpoonup_{dfGoesTo} s_2\} = \varnothing$$
$$\{t_1, t_2 : V_{Terminator} \mid t_1 \leftharpoonup_{dfComesFrom} \rightharpoonup_{dfGoesTo} t_2\} = \varnothing$$
$$\{p_1, p_2 : V_{PointOfContact} \mid p_1 \leftharpoonup_{dfComesFrom} \rightharpoonup_{dfGoesTo} p_2\} = \varnothing\,;$$

[RMS-DFP 2]:

$$\forall\, p : V_{Process} \mid \delta_{\leftharpoonup_{dfRefines}}(p) \neq 0 \bullet 3 \leq \delta_{\leftharpoonup_{dfRefines}}(p) \leq 6\,;$$

[RMS-DFP 3]:

$$isDag(eGraph(E_{dfRefines}))\,;$$

[RMS-DFP 4]:

$$\forall\, a, b : V_{DfItem} \mid a \rightharpoonup_{dfRefines} \leftharpoonup_{dfRefines} b \bullet a \rightharpoonup_{dfRefines} = b \rightharpoonup_{dfRefines}\,;$$

[RMS-DFP 5]:

$$\forall\, a, b : V_{DfObject} \mid a \leftharpoonup_{dfComesFrom} \rightharpoonup_{dfGoesTo} b \bullet a \rightharpoonup_{dfRefines} = b \rightharpoonup_{dfRefines}\,;$$

[RMS-DFP 6]:

$$\forall\, c : E_{dfCorrespondsTo} \bullet \exists\, p : V_{Process} \mid \delta_{\leftharpoonup_{dfRefines}}(p) > 0 \bullet \omega(c)\ (\rightharpoonup_{dfGoesTo} | \rightharpoonup_{dfComesFrom})\ p\,;$$

[RMS-DFP 7]:

$$\forall\, d : V_{Dataflow};\ p : V_{Process} \mid \delta_{\leftharpoonup_{dfRefines}}(p) > 0 \wedge d \rightharpoonup_{dfGoesTo} p \bullet \exists_1\, poc : V_{PointOfContact} \mid d \leftharpoonup_{dfCorresponds} poc \rightharpoonup_{dfRefines} p \bullet poc \leftharpoonup_{dfComesFrom} \rightharpoonup_{dfGoesTo} \rightharpoonup_{dfRefines} p\,;$$

$$\forall\, d : V_{Dataflow};\ p : V_{Process} \mid \delta_{\leftharpoondown_{dfRefines}}(p) > 0 \wedge d \rightharpoonup_{dfComesFrom} p \bullet$$
$$\exists_1\, poc : V_{PointOfContact} \mid d \leftharpoondown_{dfCorresponds} poc \rightharpoonup_{dfRefines} p \bullet$$
$$poc \leftharpoondown_{dfGoesTo} \rightharpoonup_{dfComesFrom} \rightharpoonup_{dfRefines} p\ ;$$

[RMS-DFP 8]:

$$\forall\, p : V_{Process};\ d_1, d_2 : V_{Dataflow};\ poc_1, poc_2 : V_{PointOfContact} \mid$$
$$d_1 \neq d_2 \wedge d_1\ (\rightharpoonup_{dfGoesTo} | \rightharpoonup_{dfComesFrom})\ p \leftharpoondown_{dfRefines} poc_1 \rightharpoonup_{dfCorrespondsTo} d_1 \wedge$$
$$d_2\ (\rightharpoonup_{dfGoesTo} | \rightharpoonup_{dfComesFrom})\ p \leftharpoondown_{dfRefines} poc_2 \rightharpoonup_{dfCorrespondsTo} d_2 \bullet$$
$$poc_1 \neq poc_2\ ;$$

[RMS-DFP 9]:

$$\forall\, d : V_{Dataflow};\ p : V_{Process} \mid d \rightharpoonup_{dfRefines} p \bullet$$
$$d\ (\rightharpoonup_{dfGoesTo} | \rightharpoonup_{dfComesFrom})\ \rightharpoonup_{dfRefines} p\ ;$$

[RMS-DFP 10]:

$$\forall\, poc : V_{PointOfContact} \bullet$$
$$poc \rightharpoonup_{dfCorrespondsTo} \leftharpoondown_{describes}$$
$$(\leftharpoondown_{isA} \mid (\leftharpoondown_{cGoesTo}\ \&_{AggregationClass}\ \rightharpoonup_{cComesFrom}))^*$$
$$\rightharpoonup_{describes}\ (\rightharpoonup_{dfGoesTo} | \rightharpoonup_{dfComesFrom})\ poc\ ;$$

[RMS-DFP 11]:

$$\forall\, d : V_{Dataflow} \bullet$$
$$d \rightharpoonup_{dfGoesTo} \rightharpoonup_{dfComesFrom}\ \&_{Store}\ \leftharpoondown_{describes}$$
$$(\leftharpoondown_{isA} \mid (\leftharpoondown_{cGoesTo}\ \&_{AggregationClass}\ \rightharpoonup_{cComesFrom}))^*\ \rightharpoonup_{describes}\ d$$

end .

A.4.2 *GRAL*-Zusicherungen des Zustandsübergangsparadigmas

for G **in** *RMS* **assert**

[RMS-STP 1]:

$$\forall\, e : V_{EndBlob} \bullet \delta_{\leftharpoondown_{transComesFrom}}(e) = 0\ ;$$

[RMS-STP 2]:

$$isTree(eGraph(E_{contains}))\ ;$$

[RMS-STP 3]:

$$type(root(eGraph(E_{contains})) = XorBlob\ ;$$

[RMS-STP 4]:

$$\delta_{\leftharpoondown_{transComesFrom}}(root(eGraph(E_{contains}))) = \delta_{\leftharpoondown_{transGoesTo}}(root(eGraph(E_{contains}))) = 0\ ;$$

[RMS-STP 5]:

$$\exists_1\, b : V_{Blob} \bullet b \leftharpoondown_{contains} root(eGraph(E_{contains})) \rightharpoonup_{startsWith} b\ ;$$

[RMS-STP 6]:

$$\forall\, x : V_{XorBlob} \mid \delta_{\leftharpoondown_{transGoesTo}}(x) > 0 \vee x \leftharpoondown_{contains} \&_{AndBlob} \neq \varnothing \bullet \delta_{\rightharpoonup_{startsWith}}(x) = 1\,;$$

[RMS-STP 7]:

$$\forall\, e : E_{startsWith} \bullet \alpha(e) \rightharpoonup_{contains} \omega(e)\,;$$

[RMS-STP 8]:

$$\forall\, x, y : V_{XorBlob} \mid x \leftharpoondown_{contains} \&_{AndBlob} \rightharpoonup_{contains} y \wedge x \neq y \bullet \neg\, (x \rightharpoonup^*_{contains} \leftharpoondown_{transComesFrom} \rightharpoonup_{transGoesTo} \leftharpoondown^*_{contains} y)\,;$$

[RMS-STP 9]:

$$\begin{aligned}
&\forall\, t_1, t_2 : V_{Transition};\ a, b, c, d : V_{Blob} \mid t_1 \neq t_2 \wedge \\
&\qquad t_1 \leftharpoondown_{triggers} \&_{Event} \rightharpoonup_{triggers} t_2 \wedge \\
&\qquad a \leftharpoondown_{transComesFrom} t_1 \rightharpoonup_{transGoesTo} b \wedge \\
&\qquad c \leftharpoondown_{transComesFrom} t_2 \rightharpoonup_{transGoesTo} d \bullet \\
&\quad lca(a,b) = lca(c,d) \wedge \\
&\quad (a = c \vee \\
&\qquad (\neg\, (a = c \vee a \rightharpoonup^*_{contains} c \vee a \leftharpoondown^*_{contains} c) \wedge \\
&\qquad\quad a \leftharpoondown^*_{contains} \&_{AndBlob} \rightharpoonup^*_{contains} c)) \\
&\quad \Rightarrow \\
&\quad (b = d \vee \\
&\qquad (\neg\, (b = d \vee b \rightharpoonup^*_{contains} d \vee b \leftharpoondown^*_{contains} d) \wedge \\
&\qquad\quad b \leftharpoondown^*_{contains} \&_{AndBlob} \rightharpoonup^*_{contains} d))
\end{aligned}$$

end.

hierbei gilt:

$$\begin{aligned}
&lca : V_{Blob} \times V_{Blob} \to V_{Blob} \\
&\hline \\
&lca = \lambda v, w : V_{Blob} \bullet \mu x : V_{Blob} \bullet v \leftharpoondown^*_{contains} x \rightharpoonup^*_{contains} w \wedge \\
&\qquad\qquad \forall y : V_{Blob} \mid v \leftharpoondown^*_{contains} y \rightharpoonup^*_{contains} w \bullet x \leftharpoondown^*_{contains} y
\end{aligned}$$

A.4.3 *GRAL*-Zusicherungen des Netzparadigmas

for *G* **in** *RMS* **assert**

[RMS-NP 1]:

$$\forall\, f : V_{ElementaryFlow} \bullet \delta_{\rightharpoonup_{cfComesFrom}}(f) = 1 \wedge \delta_{\rightharpoonup_{cfGoesTo}}(f) = 1\,;$$

[RMS-NP 2]:

$$\forall\, f : V_{Branch} \cup V_{Fork} \bullet \delta_{\rightharpoonup_{cfComesFrom}}(f) = 1 \wedge \delta_{\rightharpoonup_{cfGoesTo}}(f) > 1\,;$$

[RMS-NP 3]:

$$\forall\, f : V_{Merge} \cup V_{Join} \bullet \delta_{\rightharpoonup_{cfComesFrom}}(f) > 1 \wedge \delta_{\rightharpoonup_{cfGoesTo}}(f) = 1\,;$$

[RMS-NP 4]:

$$isDag(eGraph(E_{cfRefines}))\ ;$$

[RMS-NP 5]:

$$\forall\, a, b : V_{NetItem} \mid a \rightharpoonup_{cfRefines} \leftharpoonup_{cfRefines} b \bullet a \rightharpoonup_{cfRefines} = b \rightharpoonup_{cfRefines}\ ;$$

[RMS-NP 6]:

$$\forall\, a, b : V_{NetObject} \mid a \leftharpoonup_{cfComesFrom} \rightharpoonup_{cfGoesTo} b \bullet a \rightharpoonup_{cfRefines} = b \rightharpoonup_{cfRefines}\ ;$$

[RMS-NP 7]:

$$\forall\, p_1 : V_{Process} \mid \delta_{\leftharpoonup_{cfRefines}} > 0 \bullet$$
$$p_1 \leftharpoonup_{executesProcess} = \bigcup \{p_2 : V_{Process} \mid p_1 \leftharpoonup_{cfRefines} p_2 \bullet p_2 \leftharpoonup_{executesProcess}\}$$

end.

A.4.4 *GRAL*-Zusicherungen des Kontrollflußparadigmas

for *G* **in** *RMS* **assert**

[RMS-CP 8]:

$$isDag(eGraph(E_{isSubprocess} \cup E_{isAlternative}))$$

end.

A.5 *GRAL*-Zusicherungen der Objektsicht

A.5.1 *GRAL*-Zusicherungen des Objekt-Instanzpardigmas

for *G* **in** *RMS* **assert**

[RMS-OIP 1]:

$$\forall\, e : E_{iRelates} \bullet 1 \leq e.order \leq \delta_{\leftharpoonup_{iRelates}}(\omega(e))\ ;$$

[RMS-OIP 2]:

$$\forall\, e_1, e_2 : E_{iRelates} \mid e_1 \neq e_2 \wedge \omega(e_1) = \omega(e_2) \bullet e_1.order \neq e_2.order\ ;$$

[RMS-OIP 3]:

$$\forall\, a : V_{Attribute};\ v : V_{Value} \mid a \rightharpoonup_{identifies} \leftharpoonup_{isValue} v \bullet$$
$$a \rightharpoonup_{hasDomain} \leftharpoonup_{valIsInstanceOf} v$$

end.

A.5.2 *GRAL*-Zusicherungen des Objekt-Interaktionsparadigmas

for *G* **in** *RMS* **assert**

[RMS-IAP 1]:

$$\forall\, c : V_{Call};\ r : V_{Return} \mid r \rightharpoonup_{msgCorrespondsTo} c \bullet$$
$$c \rightharpoonup_{msgComesFrom} = r \rightharpoonup_{msgGoesTo} \wedge\ c \rightharpoonup_{msgGoesTo} \supseteq r \rightharpoonup_{msgComesFrom}\ ;$$

[RMS-IAP 2]:

$$\forall\, e_1, e_2 : E_{isMsgIn} \mid \omega(e_1) = \omega(e_2) \wedge \alpha(e_1) \leftharpoonup_{msgCorrespondsTo} \alpha(e_2) \bullet order_{isMsgIn}(e_1, \omega(e_1)) < order_{isMsgIn}(e_2, \omega(e_2)) ;$$

[RMS-IAP 3]:

$$\forall\, e_1, e_2 : E_{isMsgIn} \mid \omega(e_1) = \omega(e_2) \wedge \alpha(e_1) \leftharpoonup_{msgCorrespondsTo} \alpha(e_2) \bullet \neg\ \exists\, e_3 : E_{isMsgIn} \mid \omega(e_1) = \omega(e_2) = \omega(e_3) \wedge order_{isMsgIn}(e_1, \omega(e_1)) < order_{isMsgIn}(e_3, \omega(e_3)) < order_{isMsgIn}(e_2, \omega(e_2)) \bullet \alpha(e_1) \rightharpoonup_{msgComesFrom} (\leftharpoonup_{msgComesFrom} | \leftharpoonup_{msgGoesTo}) \alpha(e_3)$$

end.

A.5.3 *GRAL*-Zusicherungen des Objekt-Beziehungsparadigmas

for *G* **in** *RMS* **assert**

[RMS-ORP 1]:

$$\forall\, e, f : V_{RelationshipClass};\ a, b, c, d : V_{ObjectClass} \mid e \rightharpoonup_{relClsIsA} f \wedge a \leftharpoonup_{cComesFrom} e \rightharpoonup_{cGoesTo} b \wedge c \leftharpoonup_{cComesFrom} f \rightharpoonup_{cGoesTo} d \bullet a \rightharpoonup^{*}_{objClsIsA} c \wedge b \rightharpoonup^{*}_{objClsIsA} d ;$$

[RMS-ORP 2]:

$$\forall\, e, f : E_{cRelates} \mid \alpha(e) \rightharpoonup_{relClsIsA} \alpha(f) \wedge \omega(e) \rightharpoonup^{*}_{objClsIsA} \omega(f) \bullet first(e.limits) \geq first(f.limits) \wedge second(e.limits) \leq second(f.limits) ;$$

[RMS-ORP 3]:

$$\forall\, t_1 : V_{ClassItem} \mid t_1.abstract = \mathsf{true} \bullet \exists\, t_2 : V_{ClassItem} \mid t_2 \rightharpoonup^{+}_{isA} t_1 \bullet t_2.abstract = \mathsf{false}$$

end.

A.6 Sichten- und Paradigmenübergreifende *GRAL*-Zusicherungen

Paradigmenübergeifende Zusicherungen der Prozeßsicht

for *G* **in** *RMS* **assert**

[RMS-Process 1]:

$$\forall\, p : V_{Process} \bullet$$

[Datenfluß- und Netzparadigma]

$$(\delta_{\leftharpoonup_{dfRefines}}(p) > 0 < \delta_{\leftharpoonup_{cfRefines}}(p) \Rightarrow p \leftharpoonup_{dfRefines} \&_{Process} = p \leftharpoonup_{cfRefines} \&_{Process}) \wedge$$

[Datenfluß- und Kontrollflußparadigma]

$$(\delta_{\leftarrow_{dfRefines}}(p) > 0 \wedge type(p) \in \{Sequence, Parallel, Iteration, Alternative\} \Rightarrow p \leftarrow_{dfRefines} \&_{Process} = p\,[\leftarrow_{isAlternative}]\ \leftarrow_{isSubprocess})\ \wedge$$

[Netz- und Kontrollflußparadigma]

$$(\delta_{\leftarrow_{cfRefines}}(p) > 0 \wedge type(p) \in \{Sequence, Parallel, Iteration, Alternative\} \Rightarrow p \leftarrow_{cfRefines} \&_{Process} = p\,[\leftarrow_{isAlternative}]\ \leftarrow_{isSubprocess})$$

end.

Paradigmenübergeifende Zusicherungen der Objektsicht

for *G* **in** *RMS* **assert**

[RMS-Object 1]:

$$\forall v : V_{Object};\ e : V_{Relationship} \mid v \leftarrow_{iComesFrom} e \bullet v \rightarrow_{objIsInstanceOf} \rightarrow^*_{objClsIsA} \leftarrow_{cComesFrom} \leftarrow^*_{relClsIsA} \leftarrow_{relIsInstanceOf} e;$$

$$\forall v : V_{Object};\ e : V_{Relationship} \mid v \leftarrow_{iGoesTo} e \bullet v \rightarrow_{objIsInstanceOf} \rightarrow^*_{objClsIsA} \leftarrow_{cGoesTo} \leftarrow^*_{relClsIsA} \leftarrow_{relIsInstanceOf} e;$$

[RMS-Object 2]:

$$\forall v : V_{Object};\ e : E_{cComesFrom} \mid v \rightarrow_{objIsInstanceOf} \rightarrow^*_{objClsIsA} \omega(e) \bullet first(e.limits) \leq \#\{ r : E_{iComesFrom} \mid \omega(r) = v \wedge \alpha(r) \rightarrow_{relIsInstanceOf} \rightarrow^*_{relClsIsA} \alpha(e)\} \leq second(e.limits);$$

$$\forall v : V_{Object};\ e : E_{cGoesTo} \mid v \rightarrow_{objIsInstanceOf} \rightarrow^*_{objClsIsA} \omega(e) \bullet first(e.limits) \leq \#\{ r : E_{iGoesTo} \mid \omega(r) = v \wedge \alpha(r) \rightarrow_{relIsInstanceOf} \rightarrow^*_{relClsIsA} \alpha(e)\} \leq second(e.limits);$$

[RMS-Object 3]:

$$\forall i : V_{InstanceItem};\ c : V_{ClassItem} \mid i \rightarrow_{IsInstanceOf} c \bullet i \rightarrow_{hasAttributeValuePair} \leftarrow_{identifies} = c \rightarrow^*_{IsA} \rightarrow_{hasAttribute};$$

[RMS-Object 4]:

$$\forall c : V_{ClassItem} \mid c.abstract \bullet c \leftarrow_{IsInstanceOf} = \varnothing;$$

[RMS-Object 5]:

$$\forall rc : V_{RelationshipClass};\ r_1, r_1 : V_{Relationship} \mid rc.injective \wedge r_1 \rightarrow_{relIsInstanceOf} rc \leftarrow_{relIsInstanceOf} r_2 \wedge r_1 \neq r_2 \bullet r_1 \rightarrow_{iGoesTo} \neq r_2 \rightarrow_{iGoesTo} \vee r_1 \rightarrow_{iComesFrom} \neq r_2 \rightarrow_{iComesFrom}$$

end.

Paradigmenübergeifende Zusicherungen der Aufgaben- und Prozeßsicht

for G **in** *RMS* **assert**

[RMS-TaskProcess 1]:

$$\forall\, p : V_{Process};\ t : V_{Task} \mid p \rightharpoonup_{isProcessComponent} t \bullet$$

[Datenflußparadigma]

$$(\delta_{\leftharpoonup_{dfRefines}}(p) > 0 \ \wedge\ \delta_{\rightharpoonup_{isDecomposedIn}}(t) > 0 \ \Rightarrow$$
$$p \leftharpoonup_{dfRefines} \&_{Process} = t \rightharpoonup_{isDecomposedIn} \leftharpoonup_{isSubtask} \leftharpoonup_{isProcessComponent}) \ \wedge$$

[Netzparadigma]

$$(\delta_{\leftharpoonup_{cfRefines}}(p) > 0 \ \wedge\ \delta_{\rightharpoonup_{isDecomposedIn}}(t) > 0 \ \Rightarrow$$
$$p \leftharpoonup_{cfRefines} \&_{Process} = t \rightharpoonup_{isDecomposedIn} \leftharpoonup_{isSubtask} \leftharpoonup_{isProcessComponent}) \ \wedge$$

[Kontrollflußparadigma]

$$(type(p) \in \{Sequence, Parallel, Iteration, Alternative\} \ \wedge$$
$$\delta_{\rightharpoonup_{isDecomposedIn}}(t) > 0 \ \Rightarrow$$
$$p\,[\leftharpoonup_{isAlternative}]\ \leftharpoonup_{isSubprocess} =$$
$$t \rightharpoonup_{isDecomposedIn} \leftharpoonup_{isSubtask} \leftharpoonup_{isProcessComponent});$$

[RMS-TaskProcess 2]:

$$\forall\, p : V_{Process};\ t : V_{Task} \mid p \rightharpoonup_{isProcessComponent} t \ \wedge$$
$$p(\leftharpoonup_{dfComesFrom} \mid \leftharpoonup_{dfGoesTo})\ \leftharpoonup_{describes} \neq \varnothing \bullet$$
$$t \leftharpoonup_{isObjectComponent} \subseteq p(\leftharpoonup_{dfComesFrom} \mid \leftharpoonup_{dfGoesTo})\ \leftharpoonup_{describes};$$

[RMS-TaskProcess 3]:

$$\forall\, p : V_{Process};\ t : V_{Task} \mid p \rightharpoonup_{isProcessComponent} t \bullet$$
$$t \leftharpoonup_{executesTask} = p \leftharpoonup_{executesProcess} \&_{OrganizationalUnit}$$

end.

Paradigmenübergeifende Zusicherungen der Aufbau- und Prozeßsicht

for G **in** *RMS* **assert**

[RMS-PositionProcess 1]:

$$\forall\, p_1, p_2 : V_{Process};\ s_1, s_2 : V_{OrganizationalUnit} \mid s_1 \neq s_2 \ \wedge$$
$$s_1 \rightharpoonup_{executesProcess} p_1 \ \wedge\ s_2 \rightharpoonup_{executesProcess} p_2 \bullet$$
$$p_1 \leftharpoonup_{dfComesFrom} \rightharpoonup_{dfGoesTo} p_2 \Rightarrow s_1 \leftharpoonup_{communicates} \rightharpoonup_{communicates} s_2$$

end.

Paradigmenübergeifende Zusicherungen der Aufbau- und Objektsicht

for G **in** *RMS* **assert**

[RMS-PositionObject 1]:

$$\forall\, s_1, s_2 : V_{OrganizationalUnit} \bullet$$
$$s_1 \leftharpoonup_{msgComesFrom} \rightharpoonup_{msgGoesTo} s_2 \Rightarrow s_1 \leftharpoonup_{communicates} \rightharpoonup_{communicates} s_2$$

end.

B Glossar

B.1 Grundlegende Definitionen

Konzeptmodellierung

Abstraktion der Realität durch mentale Repräsentationen (Konzepte) realen oder abstrakten Dinge der zu modellierenden Diskurswelt und den hierzwischen bestehenden Beziehungen. Die Konzepte werden in einer symbolischen Form notiert.

Metaaktivitätsmodell

Teil eines ⇝*Metamodells*, durch den das Modellierungsvorgehen beschrieben wird.

Metamodell

Konzeptionelle Beschreibung einer Modellierung, die sowohl die verwendeten Modellierungskonzepte als auch ihre Verwendung verdeutlicht. Hierbei wird die Vorgehensweise der Modellierung in einem ⇝*Metaaktivitätsmodell* und die Modellierungsmittel durch ihre abstrakte Syntax in einem ⇝*Metaschema* beschrieben.

Metaschema

Teil eines ⇝*Metamodells*, durch den die zur Modellierung verwendeten Sprachmittel, unabhängig von ihrer konkreten Notation, durch ihre abstrakte Syntax beschrieben werden.

Modell

Zielgerichtetes Abbild eines Systems, das zum einen ähnliche Beobachtungen und Aussagen ermöglicht wie dieses System, zum anderen diese Realität durch Abstraktion auf die jeweils relevanten Aspekte vereinfacht.

Organisation

1. System formaler Regeln, das als Instrument zur Steuerung sozio-technischer Systeme dient (organisatatorische Struktur).
2. Organisatatorische Struktur und deren Einbettung in das umgebende sozio-technische System.

Organisationstechnik

1. Entwicklung von Prinzipien, Methoden, und Werkzeugen zur Organisationsgestaltung.
2. Anwendung dieser Mittel zur Durchführung gezielter organisatorischer Änderungen und zum Betrieb diese Organisationen.

Referenzmodell

Ausgewiesenes Modell, das *charakteristische Eigenschaften* einer Klasse gleichartiger Systeme beschreibt und als *Bezugspunkt* für *spezielle Modelle* dient.

Softwaretechnik

1. Entwicklung von Prinzpien, Methoden und Techniken zur Software-Erstellung.
2. Anwendung dieser Mittel zur Erstellung, zum Betrieb und zur Wartung von umfangreichen Softwareprodukten.

Visuelle Modellierungssprachen des Requirements-Engineering

der Organisationstechnik:
Hilfsmittel zur Erhebung und Beschreibung organisatorischer Zusammenhänge bezogen auf die statische und dynamische Organisationsstruktur und das umgebende sozio-technische System.

der Softwaretechnik:
Hilfsmittel zur Erhebung und Beschreibung von Softwaresystemen bezogen auf ihre statische und dynamische Struktur und das umgebende Anwendungssystem.

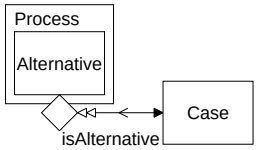
B.2 Konzepte des Referenz-Metaschemas

Das folgende graphische Glossar stellt die objektartigen Konzepte des Referenz-Metaschemas der Beschreibungsmittel für Organisationen und Softwaresysteme in ihrem Schemakontext dar, und definiert somit die Einbettung der durch diese Konzepte modellierten Begriffe in ihr Begriffsumfeld.

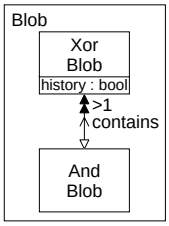
AggregationClass

⇝ RelationshipClass

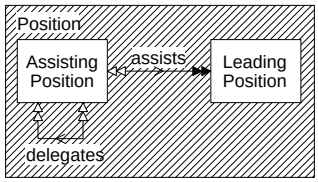
Alternative



AndBlob



AssistingPosition



AsynchroneousFlowOfControl

⇝ Message

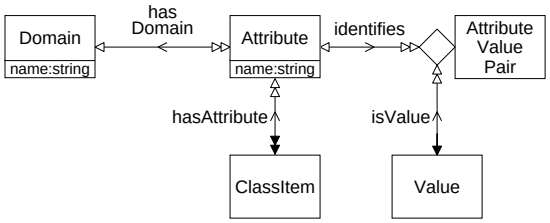
AtomicProcess

⇝ Process

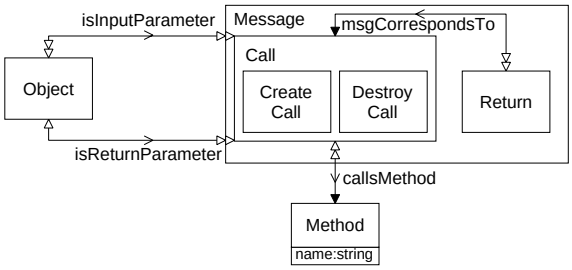
AtomicBlob

⇝ Blob

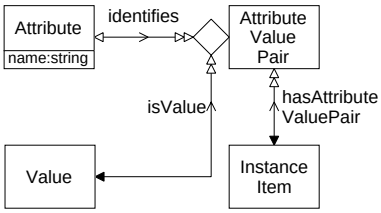
Attribute



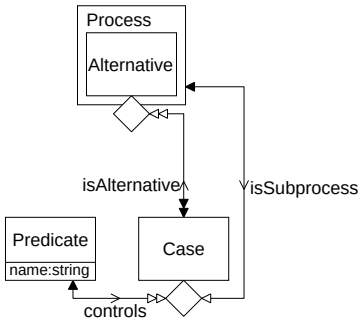
Call



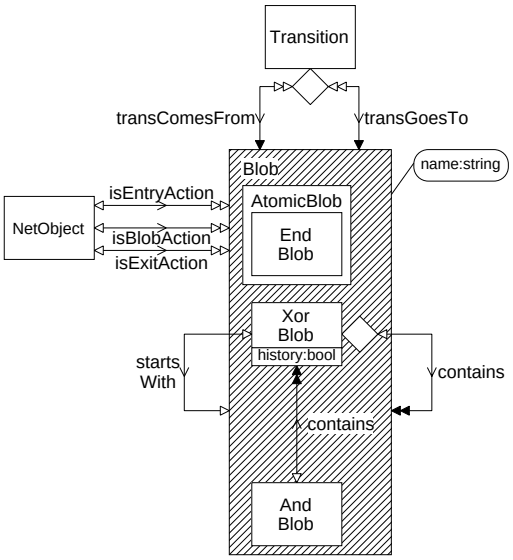
AttributeValuePair



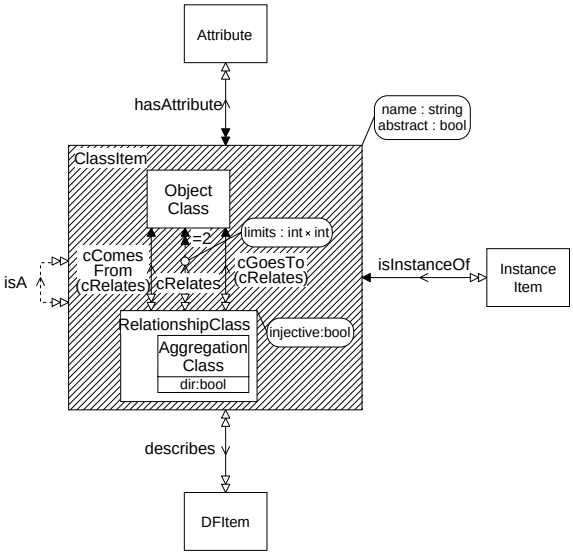
Case



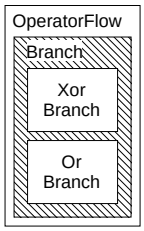
Blob



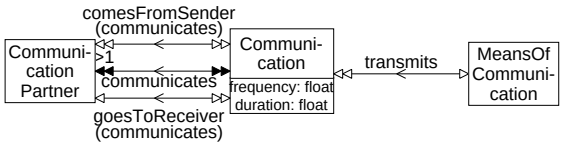
ClassItem



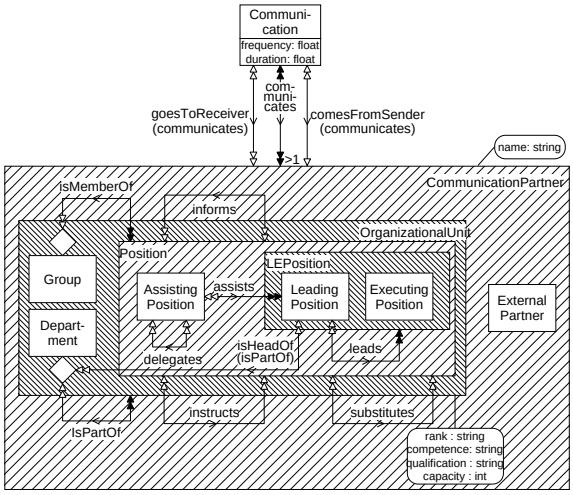
Branch



Communication



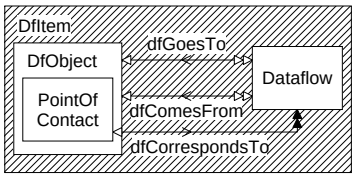
CommunicationPartner



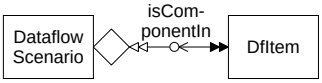
CreateCall

⇝ Call

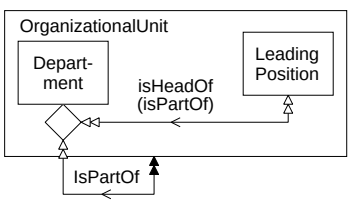
Dataflow



DataflowScenario



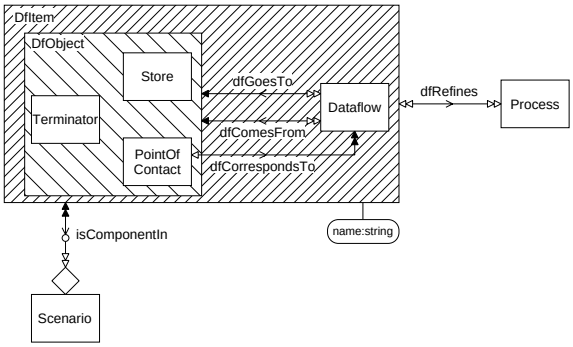
Department



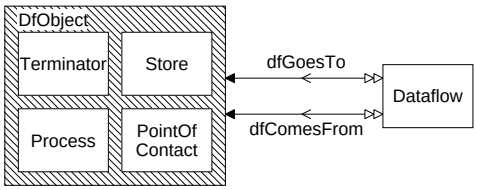
DestroyCall

⇝ Call

DfItem



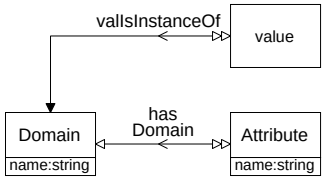
DfObject



Document

⇝ MeansOfCommunication

Domain



EMail

⇝ MeansOfCommunication

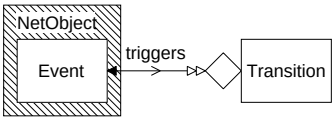
EndBlob

⇝ Blob

ElementaryFlow

⇝ Flow

Event



ExecutingPosition

⇝ LEPosition

ExternalPartner

⇝ CommunicationPartner

Fax

⇝ MeansOfCommunication

FlatFlowOfControl

⇝ Message

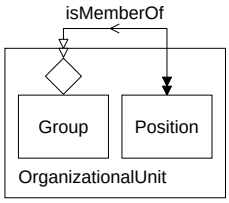
Flow

⇝ NetObject

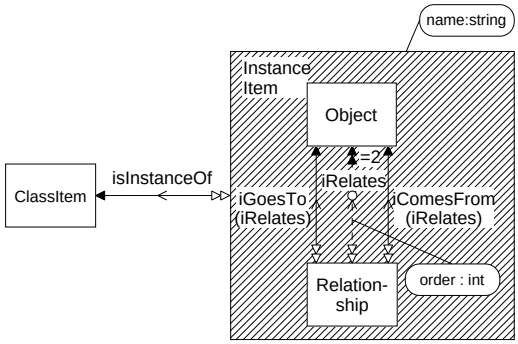
Fork

⇝ OperatorFlow

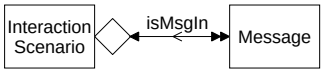
Group



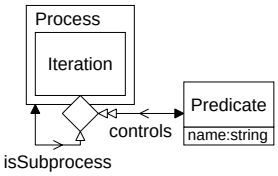
InstanceItem



InteractionScenario



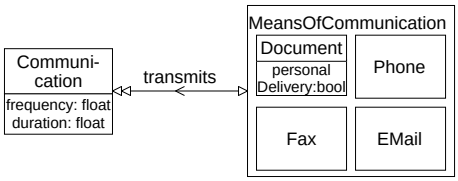
Iteration



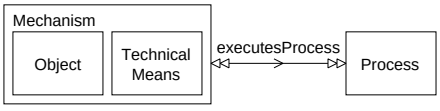
Join

⇝ OperatorFlow

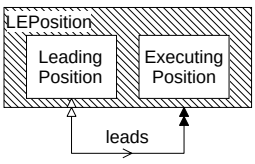
MeansOfCommunication



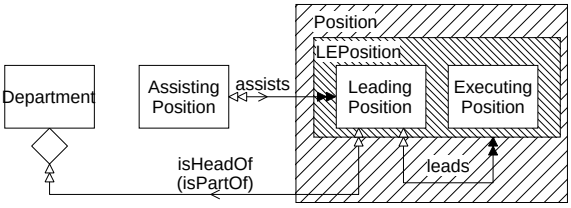
Mechanism



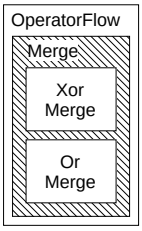
LEPosition



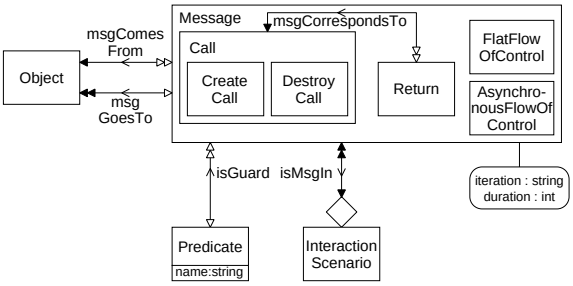
LeadingPosition



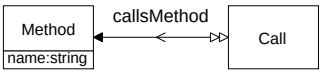
Merge



Message



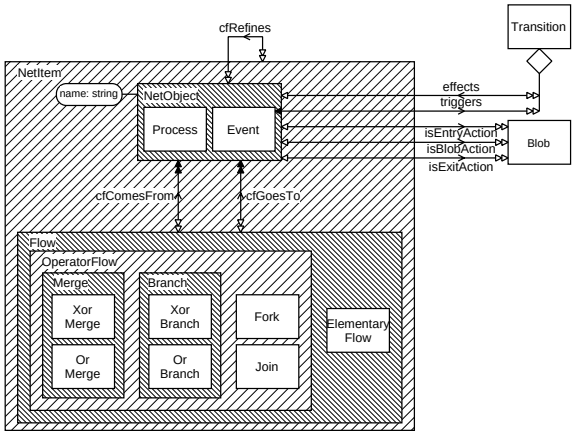
Method



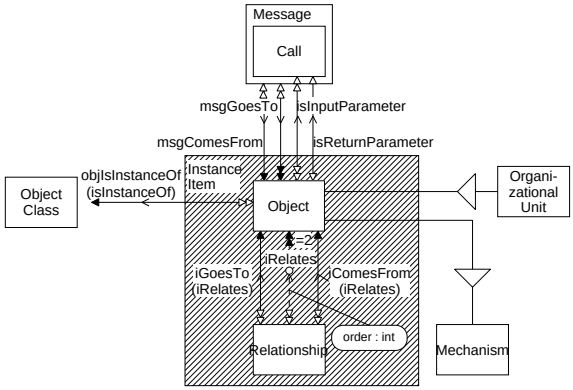
NetItem

⇝ NetObject

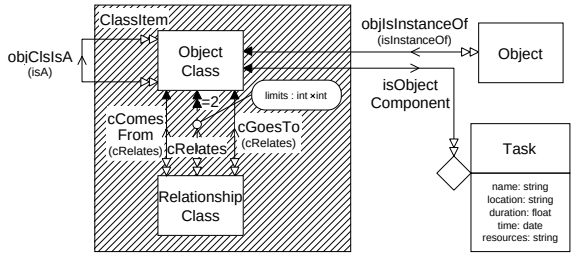
NetObject



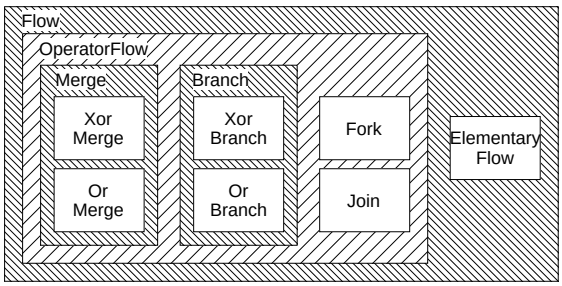
Object



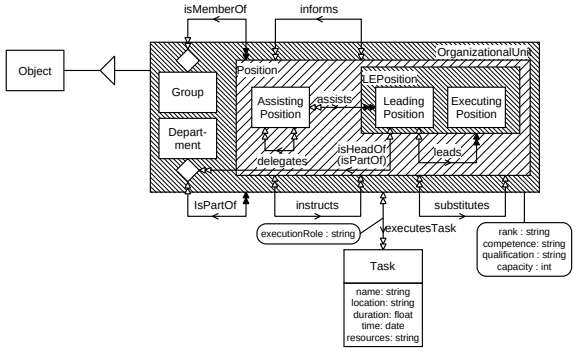
ObjectClass



OperatorFlow



OrganizationalUnit



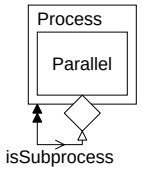
OrBranch

⇝ Branch

OrMerge

⇝ Merge

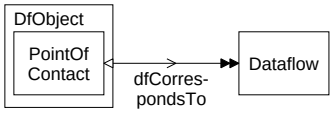
Parallel



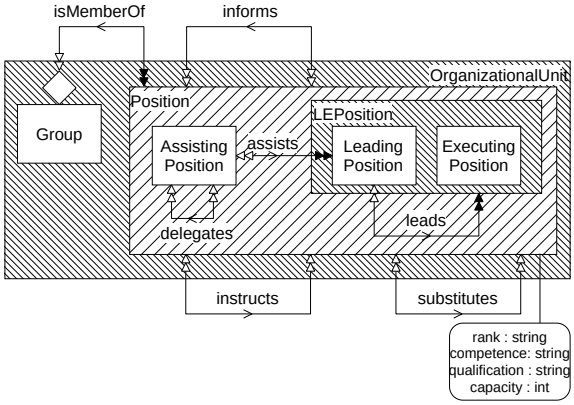
Phone

⇝ MeansOfCommunication

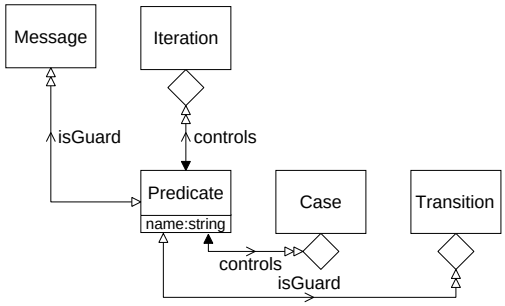
PointOfContact



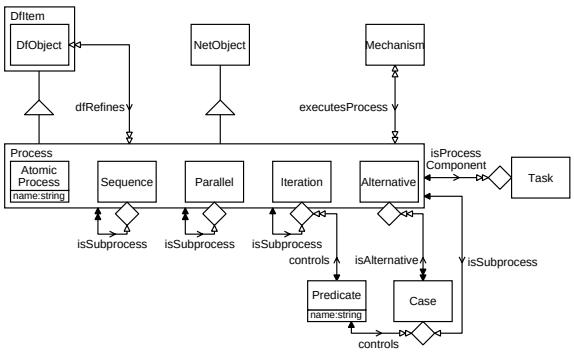
Position



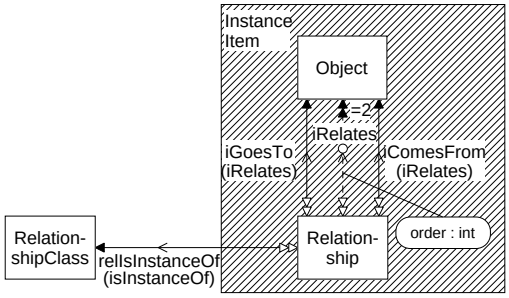
Predicate



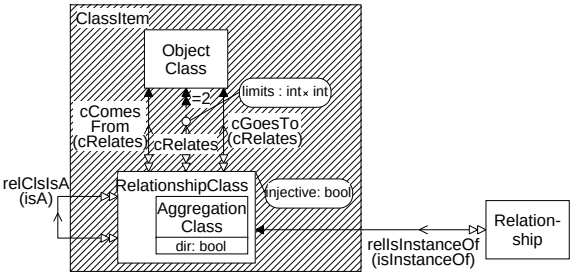
Process



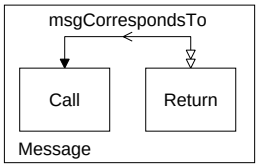
Relationship



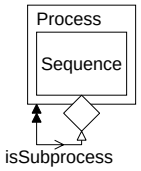
RelationshipClass



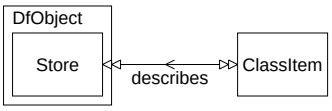
Return



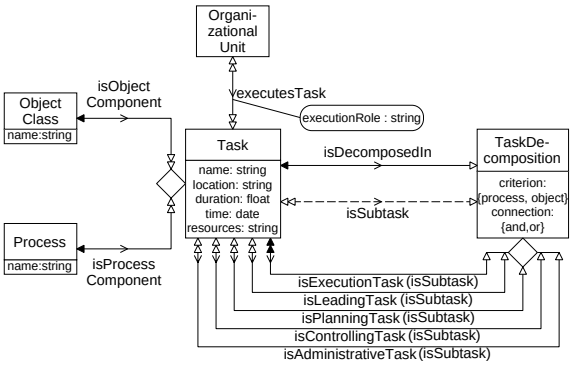
Sequence



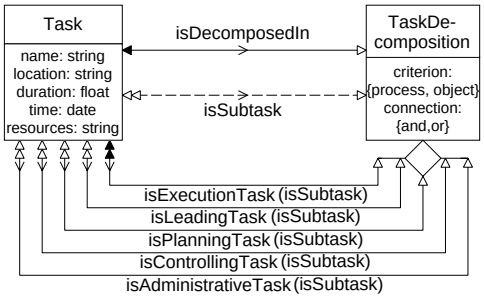
Store



Task



TaskDecomposition



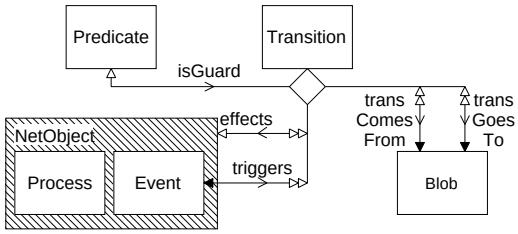
TechnicalMeans

⇝ Mechanism

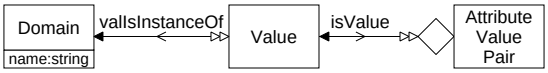
Terminator

⇝ DfObject

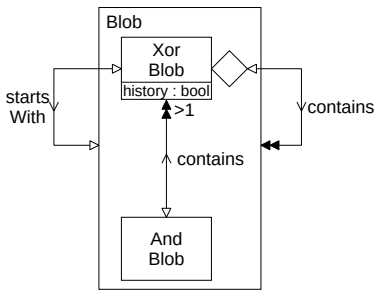
Transition



Value



XorBlob



XorBranch

⇝ Branch

XorMerge

⇝ Merge

Literaturverzeichnis

[Acker, 1973] H. B. Acker. Organisationsanalyse, Verfahren und Techniken praktischer Organisationsarbeit. Gehlen, Baden-Baden, 9. Auflage, 1973.

[Andersen et al., 1991] R. Andersen, J. A. Bubenko, A. Sølvberg (eds.). Advanced Information Systems Engineering, Third International Conference CAiSE'91, Trondheim, Norway, May 1991, Proceedings, LNCS 498. Springer, Berlin, 1991.

[ANSI/IEEE Std. 729, 1983] IEEE Standard Glossary of Software Engineering Terminology (ANSI/IEEE Std. 729-1983). The Institute of Electrical and Electronics Engineering, New York, Spring 1983.

[Apostel, 1960] L. Apostel. Towards the formal study of models in a non formal science. Synthese, 12:125–161, 1960.

[Arnold, 1993a] R. S. Arnold. A Roadmap Guide to Software Reengineering Technology. in [Arnold, 1993b], 3–22. 1993.

[Arnold, 1993b] R. S. Arnold (ed.). Software Reengineering. IEEE Computer Society Press, Los Alamitos, 2. edition, 1993.

[Arthur Young International, 1988] Information Engineering, Computer System Methodology: Planning, June 1988.

[Auramäki et al., 1987] E. Auramäki, M. Leppänen, V. Savolainen. Universal Framework for Information Activities. Data Base, 11–20, Fall/Spring 1987.

[Balzert / Balzert, 1994] H. Balzert, H. Balzert. Werkzeuge für die objektorientierte Softwareentwicklung. CD-ROM in [Balzert, 1994], 1994.

[Balzert, 1988] H. Balzert. Die Entwicklung von Software–Systemen, Prinzipien, Methoden, Sprachen, Werkzeuge. Bibliographisches Institut, Zürich, 1988.

[Balzert, 1993] H. Balzert (Hrsg.). CASE Systeme und Werkzeuge. BI Wissenschaftsverlag, Mannheim, 5. Auflage, 1993.

[Balzert, 1994] H. Balzert. Methoden der objektorientierten Systemanalyse. Bibliographisches Institut, Mannheim, 1994.

[Balzert, 1996a] H. Balzert. Lehrbuch der Softwaretechnik, Softwareentwicklung, Band 1. Spektrum, Heidelberg, 1996.

[Balzert, 1996b] H. Balzert. SWT CD ROM. CD-ROM in [Balzert, 1996a], 1996.

[Balzert, 1998a] H. Balzert. Lehrbuch der Softwaretechnik, Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung, Band 2. Spektrum, Heidelberg, 1998.

[Balzert, 1998b] H. Balzert. SWT 2 CD ROM V1.0. CD-ROM in [Balzert, 1998a], 1998.

[Batini et al., 1992] C. Batini, S. Ceri, S. B. Navathe. Conceptual Database Design. Benjamin/Cummings, Redwood City, 1992.

[Bauer, 1993] F. L. Bauer. Software Engineering, Wie es begann. Informatik Spektrum, 16(5):259–260, Oktober 1993.

[Baumgarten, 1996] B. Baumgarten. Petri-Netze, Grundlagen und Anwendungen. Spektrum, Heidelberg, 2. Auflage, 1996.

[Becker / Schütte, 1997] J. Becker, R. Schütte. Referenz-Informationsmodelle für den Handel: Begriff, Nutzen und Empfehlung für die Gestaltung und unternehmensspezifische Adaption von Referenzmodellen. in [Krallmann, 1997], 427–448. 1997.

[Becker / Vossen, 1996a] J. Becker, G. Vossen. Geschäftsprozeßmodellierung und Workflow-Management: Eine Einführung. in [Becker / Vossen, 1996b], 17–26. 1996.

[Becker / Vossen, 1996b] J. Becker, G. Vossen (Hrsg.). Geschäftsprozeßmodellierung und Workflows, Modelle, Methoden, Werkzeuge. Thomson, Bonn, 1996.

[Becker et al., 1995] J. Becker, M. Rosemann, R. Schütte. Grundsätze ordnungsmäßiger Modellierung. Wirtschaftsinformatik, 37(5):435–445, 1995.

[Becker et al., 1997] J. Becker, M. Rosemann, R. Schütte (Hrsg.). Entwicklungsstand und Entwicklungsperspektiven der Referenzmodellierung, Münster, März 1997. Westfälische Wilhelms Universität, Münster, Institut für Wirtschaftsinformatik.

[Berge, 1976] C. Berge. Graphs and Hypergraphs. North-Holland, Amsterdam, 2. edition, 1976.

[Bernus et al., 1998] P. Bernus, K. Mertins, G. Schmidt (eds.). Handbook on Architectures of Information Systems, in: International Handbooks of Information Systems. Springer, Berlin, 1998.

[Berztiss / Matjasko, 1995] A. T. Berztiss, K. J. Matjasko. Queries and the Incremental Construction of Conceptual Models. in [Kangassalo et al., 1995], 175–185. 1995.

[Bihr / Seelos, 1997] H. Bihr, H.-J. Seelos. Entwicklung eines Referenzdatenmodells für Krankenhäuser. Wirtschaftsinformatik, 39(4):367–371, August 1997.

[Blaha, 1992] M. Blaha. Models of models. Journal of Object-Oriented Programming, 5(5):13–18, September 1992.

[Blum, 1980a] E. Blum. Möglichkeit und Grenzen des Organigramms unter besonderer Berücksichtigung großbetrieblicher und mehrdimensionaler Organisationsstrukturen, Teil 1: Konventionelle Organigramme. Zeitschrift für Organisation, (1):42–51, 1980.

[Blum, 1980b] E. Blum. Möglichkeit und Grenzen des Organigramms unter besonderer Berücksichtigung großbetrieblicher und mehrdimensionaler Organisationsstrukturen, Teil 2: Organigramme für mehrdimensionale Organisationsstrukturen. Zeitschrift für Organisation, 84–91, 1980.

[Blum, 1991] E. Blum. Betriebsorganisation, Methoden und Techniken, Organisation als Gestaltungsprozeß, Erhebungs- und Darstellungstechniken, Problemanalyse/Alternativensuche, interne Kontrolle. Gabler, Wiesbaden, 3. Auflage, 1991.

[Boehm, 1976] B. W. Boehm. Software Engineering. IEEE Transactions on Computers, 25(12):1226–1241, December 1976.

[Boehm, 1979] B. W. Boehm. Guidelines for Verifying and Validating Software Requirements and Design Specifications. in [Samet, 1979], 711–719. 1979.

[Boehm, 1981] B. W. Boehm. Software Engineering Economics. Prentice Hall, Englewood Cliffs, 1981.

[Boehm, 1988] B. W. Boehm. A Spiral Model of Software Development and Enhancement. IEEE Computer, 21(5):61–72, 1988.

[Booch et al., 1999] G. Booch, J. Rumbaugh, I. Jacobson. The Unified Modeling Language User Guide. Addison Wesley, Reading, 1999.

[Booch, 1994] G. Booch. Object Oriented Design with Applications. Benjamin/Cummings, Redwood City, 2. edition, 1994.

[Booch, 1996] G. Booch. Konsolidierung der Methoden. Object Spectrum, (4):57 – 60, Juli/August 1996.

[Bosch / Mitchel, 1997] J. Bosch, S. Mitchel (eds.). Object-Oriented Technology, ECOOP'97-Workshop Reader, LNCS 1357. Springer, Berlin, 1997.

[Braek / Haugen, 1993] R. Braek, O. Haugen. Engineering Real Time Systems. Prentice-Hall, Englewood Cliffs, 1993.

[Brinkkemper et al., 1989] S. Brinkkemper, M. Geruts, I. van de Kamp, J. Acohen. On a Formal Approach to the Methodology of Information Planning. Technical Report 89-12, University of Nijmegen, Department of Informatics, Nijmegen, August 1989.

[Brinkkemper et al., 1990] S. Brinkkemper, M. de Lange, R. Looman, F. H. G. C. van der Steen. On the Derivation of Method Comparison by Meta-Modeling. ACM SIGSOFT Software Engineering Notes, 15(1):49–58, 1990.

[Brinkkemper et al., 1996] S. Brinkkemper, K. Lyytinen, R. Welke (eds.). Proceedings of the IFIP TC8 Working Conference on Method Engineering: Principles of Method Construction and Tool Support. Chapman & Hall, 1996.

[Brinkkemper, 1990] S. Brinkkemper. Formalisation of Information Systems Modeling. Thesis Publishers, Amsterdam, 1990.

[Brinkkemper, 1996] S. Brinkkemper. Method Engineering: Engineering of Information Systems Developement Methods and Tools. Information & Software Technology, 38(4):275–280, 1996.

[Brodie et al., 1986] M. Brodie, J. Mylopoulos, J. W. Schmidt (eds.). On Conceptual Modeling, Perspectives from Artificial Intelligence, Databases and Programming Languages. Springer, New York, 2. edition, 1986.

[Brombacher, 1991] R. Brombacher. Effizientes Informationsmanagement, Die Herausforderung von Gegenwart und Zukunft. Schriften zur Unternehmensführung, 44:111–134, 1991.

[Büchli / Chrobok, 1997] R. Büchli, R. Chrobok. Organisations- und Planungstechniken im Unternehmen, Methoden, Instrumente und Handlungsempfehlungen nach dem ganzheitlichen Organisationsmodell GOM. Schäffer Poeschel, Stuttgart, 2. Auflage, 1997.

[Bußmann, 1990] H. Bußmann. Lexikon der Sprachwissenschaft. Kröner, Stuttgart, 2. Auflage, 1990.

[Capellmann / Franzke, 1991] C. Capellmann, A. Franzke. GRAL Eine Sprache für die graphbasierte Modellierung. Diplomarbeit D 143, Universität Koblenz-Landau, Fachbereich Informatik, Koblenz, 1991.

[Carnap, 1968] R. Carnap. Logische Syntax der Sprache. Springer, Wien, 2. Auflage, 1968.

[Carstensen et al., 1994] M. Carstensen, J. Ebert, A. Winter. Deklarative Beschreibung von Graphsprachen (Erweiterte Kurzfassung). in [Simon, 1994]. 1994.

[Carstensen, 1996] M. Carstensen. Konzept eines Generators für CASE-Umgebungen. Manuskript, August 1996.

[Chapin, 1970] N. Chapin. Flowcharting with the ANSI Standard: A Tutorial. ACM Computing Surveys, 2:119–146, 1970.

[Chen, 1976] P. P. Chen. The Entity–Relationship Model, Toward a Unified View of Data. ACM Transactions on Database Systems, 1(1):9–36, March 1976.

[Chen, 1983] P. P. Chen (ed.). Entity-Relationship Approach to Information Modeling. Elsevier, Amsterdam, 1983.

[Coad / Yourdon, 1991] P. Coad, E. Yourdon. Object-Oriented Analysis. Yourdon Press, Englewood Ciffs, 2. edition, 1991.

[Coleman et al., 1994] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, P. Jeremaes. Object-Oriented Development, The Fusion Method. Prentice Hall, London, 1994.

[Constantopoulos et al., 1996] P. Constantopoulos, J. Mylopoulos, Y. Vassiliou, (eds.). Advanced Inforamtion Systems Engineering, 8th International Confernce, CAiSE'96, Heraklion, LNCS 1080. Springer, Berlin, 1996.

[Cook / Daniels, 1994] S. Cook, J. Daniels. Designing Object Systems: Object Modeling with Syntropy. Prentice Hall, 1994.

[Coy, 1989] W. Coy. Brauchen wir eine Theorie der Informatik? Informatik-Spektrum, 12(5):256–266, Oktober 1989.

[Dahm / Widmann, 1998] P. Dahm, F. Widmann. Das Graphenlabor, Version 4.2. Fachbericht Informatik 11/98, Universität Koblenz-Landau, Institut für Informatik, Koblenz, 1998.

[Dahm et al., 1998a] P. Dahm, J. Ebert, A. Franzke, M. Kamp, A. Winter. TGraphen und EER-Schemata, Formale Grundlagen. in [Ebert et al., 1998], 51–65. 1998.

[Dahm et al., 1998b] P. Dahm, M. Kamp, F. Moskopp. Das Schemamodul, Schnittstellenbeschreibung. Projektbericht 1/98, Universität Koblenz-Landau, Institut für Softwaretechnik, Koblenz, 1998.

[Davenport, 1993] T. H. Davenport. Process Innovation, Reengineering Work through Information Technologie. Harvard Business School Press, Boston, 1993.

[Day / Zimmermann, 1983] J. D. Day, H. Zimmermann. The OSI Reference Model. Proceedings of the IEEE, 71:1334–1340, December 1983.

[DeMarco, 1978] T. DeMarco. Structured Analysis and System Specification. Yourdon Inc., New York, 1978.

[Denert, 1991] E. Denert. Software Engineering. Springer, Heidelberg, 1991.

[Derungs et al., 1996] M. Derungs, P. Vogler, H. Österle. Metamodell Workflow. Version 1.5. Bericht IM HSG/CC PSI/3, Institut für Wirtschaftsinformatik, Universität St. Gallen, 24. Mai 1996.

[Dijkstra, 1968] E. W. Dijkstra. Go To Statement Considererd Harmfull. Communication of the ACM, 11(3):147–148, March 1968.

[Diller, 1990] A. Diller. Z, An Introduction to Formal Methods. Wiley, 1990.

[DIN 2330, 1993] Begriffe und Benennungen, Allgemeine Grundsätze. Beuth, Berlin, Dezember 1993.

[DIN 66001, 1983] Sinnbilder und ihre Anwendung. Beuth, Berlin, Dezember 1983.

[DIN 66261, 1985] Sinnbilder für Struktogramme nach Nassi-Shneiderman. Beuth, Berlin, November 1985.

[DIN 69900, 1987] Netzplantechnik. Beuth, Berlin, August 1987.

[DIVO, 1966] DIVO (Hrsg.). MPM: Die Metra-Potential-Methode, in: DIVO Information, Reihe 2, Sonderheft 1. DIVO Institut für Wirtschaftsforschung, Sozialforschung und angewandte Mathematik, 2. Auflage, Januar 1966.

[Dörner, 1993] D. Dörner. Modellbildung und Simulation. in [Roth, 1993], 327–340. 1993.

[Drüke, 1996] M. Drüke. Dokumentation für den Datenflußdiagramm-Editor. Studienarbeit S 429, Universität Koblenz-Landau, Fachbereich Informatik, Koblenz, Mai 1996.

[Dumslaff et al., 1992] U. Dumslaff, M. Mertesacker, A. Winter. Konzeptualisierung und Durchführung der Branchensoftwareanalyse. in [Ebert et al., 1992], 235–333. 1992.

[Dumslaff et al., 1994] U. Dumslaff, J. Ebert, M. Mertesacker, A. Winter. Ein Vorgehensmodell zur Software-Evaluation. HMD, 31(175):89–105, Januar 1994.

[Ebert / Engels, 1994] J. Ebert, G. Engels. Design Representation. in [Marciniak, 1994], 382–394. 1994.

[Ebert / Franzke, 1995] J. Ebert, A. Franzke. A Declarative Approach to Graph Based Modeling. in [Mayr et al., 1995], 38–50. 1995.

[Ebert / Süttenbach, 1997a] J. Ebert, R. Süttenbach. An OMT Metamodel. Fachbericht Informatik 13/97, Universität Koblenz-Landau, Institut für Informatik, Koblenz, 1997.

[Ebert / Süttenbach, 1997b] J. Ebert, R. Süttenbach. Integration of Z-based Semantics of OO-Notations. in [Bosch / Mitchel, 1997]. 1997.

[Ebert et al., 1992] J. Ebert, D. Euler, M. Twardy (Hrsg.). Computerunterstützte Auftragsabwicklung im Handwerk, Untersuchung von Problemfeldern und Konzeptualisierung von Bildungsmaßnahmen für die Bereiche der Energie- und Fertigungstechnik. Carl, Laasphe, 1992.

[Ebert et al., 1996a] J. Ebert, R. Gimnich, A. Winter. Wartungsunterstützung in heterogenen Sprachumgebungen, Ein Überblick zum Projekt GUPRO. in [Lehner, 1996], 263–275. 1996.

[Ebert et al., 1996b] J. Ebert, A. Winter, P. Dahm, A. Franzke, R. Süttenbach. Graph Based Modeling and Implementation with EER/GRAL . in [Thalheim, 1996], 163–178. 1996.

[Ebert et al., 1997a] J. Ebert, A. Franzke, M. Kamp, D. Polock, F. Widmann. TGREP – Graphklasse zur Repräsentation von TGraph–bezogenen Ausdrücken und Prädikaten. Projektbericht 12/97, Universität Koblenz-Landau, Institut für Softwaretechnik, Koblenz, 1997.

[Ebert et al., 1997b] J. Ebert, R. Süttenbach, I. Uhe. Meta-CASE in Practice: a Case for KOGGE. in [Olivé / Pastor, 1997], 203–216. 1997.

[Ebert et al., 1998] J. Ebert, R. Gimnich, H. H. Stasch, A. Winter (Hrsg.). GUPRO, Generische Umgebung zum Programmverstehen. Fölbach, Koblenz, 1998.

[Ebert et al., 1999a] J. Ebert, B. Kullbach, A. Winter. GraX – An Interchange Format for Reengineering Tools. in [WCRE, 1999], 89–98. 1999.

[Ebert et al., 1999b] J. Ebert, R. Süttenbach, I. Uhe. JKogge: a Component-Based Approach for Tools in the Internet. in [STJA, 1999]. 1999.

[Ebert, 1981] J. Ebert. Effiziente Graphenalgorithmen. Akademische Verlagsgesellschaft, 1981.

[Ebert, 1993] J. Ebert. Efficient Interpretation of State Charts. in [Ésik, 1993], 121–221. 1993.

[Ebert, 1997] J. Ebert. MetaCASE: Generierung und Anpassung von CASE-Werkzeugen. in [Gens, 1997], 191–196. 1997.

[ECMA, 1991] A Reference Model for Computer Assisted Software Engineering Environments Frameworks. Technical Report TR/55, European Computer Manufacturers Association, 111 Rue du Rhône, CH-1204 Genf, 1991.

[Ehrig / Mahr, 1985] H. Ehrig, B. Mahr. Fundamentals of Algebraic Specification. Springer, Berlin, 1985.

[Ehrig et al., 1991] H. Ehrig, H.-J. Kreowski, G. Rozenberg (eds.). Graph Grammars and their Application to Computer Science, LNCS 532. Springer, Berlin, 1991.

[EIA/IS-107, 1994] CDIF Framework for Modeling and Extensibility. (Extract of Interim Standard) http://www.eigroup.org/cdif/how-to-obtain-standards.html (20.08.1999) EIA/IS-107, Electronic Industries Association, Arlington, January 1994.

[EIA/IS-108, 1994] CDIF Transfer Format – General Rules for Syntaxes and Encodings. (Extract of Interim Standard) http://www.eigroup.org/cdif/how-to-obtain-standards.html (20.08.1999) EIA/IS-108, Electronic Industries Association, Arlington, January 1994.

[EIA/IS-109, 1994] CDIF Transfer Format – Transfer Format Syntax SYNTAX.1. (Extract of Interim Standard) http://www.eigroup.org/cdif/how-to-obtain-standards.html (20.08.1999) EIA/IS-109, Electronic Industries Association, Arlington, January 1994.

[EIA/IS-110, 1994] CDIF Transfer Format – Transfer Format Encoding ENCODING.1. (Extract of Interim Standard) http://www.eigroup.org/cdif/how-to-obtain-standards.html (20.08.1999) EIA/IS-110, Electronic Industries Association, Arlington, January 1994.

[EIA/IS-111, 1994] CDIF Integrated Meta-model, Foundation Subject Area. (Extract of Interim Standard) http://www.eigroup.org/cdif/how-to-obtain-standards.html (20. 08.1999) EIA/IS-111, Electronic Industries Association, Arlington, January 1994.

[EIA/IS-112, 1995] CDIF Integrated Meta-model, Common Subject Area. (Extract of Interim Standard) http://www.eigroup.org/cdif/how-to-obtain-standards.html (20. 08.1999) EIA/IS-112, Electronic Industries Association, Arlington, December 1995.

[EIA/IS-113, 1994] CDIF Integrated Meta-model, Data Modeling Subject Area. (Extract of Interim Standard) http://www.eigroup.org/cdif/how-to-obtain-standards.html (20. 08.1999) EIA/IS-113, Electronic Industries Association, Arlington, December 1994.

[EIA/IS-115, 1995] CDIF Integrated Meta-model Data Flow Model, Subject Area. (Extract of Interim Standard) http://www.eigroup.org/cdif/how-to-obtain-standards.html (20.08.1999) EIA/IS-115, Electronic Industries Association, Arlington, December 1995.

[Elmasri / Navathe, 2000] R. A. Elmasri, S. B. Navathe. Fundamentals of Database Systems. Addison Wesley, Reading, 3. edition, 2000.

[Elmasri / Wiederhold, 1983] R. A. Elmasri, G. Wiederhold. GORDAS: A formal high-level Query Language for the Entity-Relationship Model. in [Chen, 1983], 49–72. 1983.

[Elmasri et al., 1993] R. A. Elmasri, V. Kouramajian, B. Thalheim (eds.). Entity-Relationship Approach, ER'93. 12th International Conference on the Entity-Relationship Approach, Arlington, Texas, LNCS 823. Springer, Berlin, December 1993.

[Embley / Goldstein, 1997] D. W. Embley, R. C. Goldstein (eds.). Conceptual Modeling, ER'97, LNCS 1331. Springer, Berlin, 1997.

[Engel, 1993] A. Engel. Organisationsforschung und Informatik. Vorlesungsskript, Institut für Sozialwissenschaftliche Informatik, Universität Koblenz-Landau, August 1993.

[Engel, 1995] A. Engel. Vorgangsbearbeitung im Informationsverbund. in [Huber-Wäschle et al., 1995], 118–126. 1995.

[Engel, 1996] A. Engel. Verwaltungsreorganisation mit Referenzmodellen. Ein Beitrag zur Konzeption einer Vorgangsunterstützungsumgebung für die Öffentliche Verwaltung. in [Scheer, 1996], 457–483. 1996.

[Engels et al., 1992] G. Engels, C. Lewerentz, M. Nagl, W. Schäfer, A. Schürr. Building Integrated Software Development Environments. Part 1: Tool Specification. ACM Transactions on Software Engineering and Methodology, 1(2):135–167, 1992.

[Ernst and Young, 1990] NAVIGATOR System Series, Reference Manual, Analysis Phase Manual, Release 1.0. 1990.

[Ernst, 1997] J. Ernst. Introduction to CDIF. `http://www.eigroup.org/cdif/intro.html` (19.07.1999), September 1997.

[Ésik, 1993] Z. Ésik (eds.). Fundamentals of Computation Theory, 9th International Conference FCT'93, Szeded, Hungary, 23–27 August, 1993, LNCS 710. Spinger, Berlin, 1993.

[Euler, 1989] D. Euler. Gestaltung von computerunterstützten Informationssystemen: von der bildungspolitischen Programmatik zur didaktischen Konzeption. Kölner Zeitschrift für Wirtschaft und Pädagogik, (7):63–107, November 1989.

[Even, 1979] S. Even. Graph Algorithms. Pitman, Maryland, 1979.

[Eversheim, 1996] W. Eversheim (Hrsg.). Prozeßorientierte Unternehmensorganisation: Konzepte und Methoden zur Gestaltung schlanker Organisationen. Springer, Berlin, 2. Auflage, 1996.

[Falkenberg et al., 1995] E. D. Falkenberg, W. Hesse, A. Olivé (eds.). Information System Concepts, Towards a Consolidation of Views. London, 1995.

[Falkenberg et al., 1998] E. D. Falkenberg, W. Hesse, P. Lindgreen, B. E. Nilsson, J. L. H. Oei, C. Rolland, R. K. Stamper, F. J. M. van Assche, A. A. Verrijn-Stuart, K. Voss. FRISCO, A Framework of Information System Concepts, The FRISCO Report (Web edition). Department of Computer Science, University of Leiden, `ftp://ftp.leidenuniv.nl/pub/rul/fri-full.zip` (15.11.1999), 1998.

[Färberböck et al., 1991] H. Färberböck, T. Gutzwiller, M. Heym. Ein Vergleich von Requirements Engineering Methoden auf Metamodell-Basis. in [Timm, 1991], 40–66. 1991.

[Fayol, 1919] H. Fayol. Allgemeine und industrielle Verwaltung. München, 1919.

[Ferstl / Sinz, 1996] O. K. Ferstl, E. J. Sinz. Geschäftsprozeßmodellierung im Rahmen des Semantischen Objektmodells. in [Becker / Vossen, 1996b], 47–61. 1996.

[Flatscher, 1996] R. G. Flatscher. An Overview of the Architecture of EIA'a CASE Data Interchange Format (CDIF). Rundbrief des GI-Fachausschuß 5.2 `http://www.eigroup.org/cdif/online.html` (19.07.1999), 3(1):26–30, Sommer 1996.

[Flatscher, 1998] R. G. Flatscher. Meta-Modellierung in EIA/CDIF. ADV-Verlag, Wien, 1998.

[Floyd et al., 1997] C. Floyd, A. Krabbel, S. Ratuski, I. Wetzel. Zur Evolution der evolutionären Systementwicklung: Erfahrungen aus einem Krankenhausprojekt. Informatik Spektrum, 20(1):13–20, Februar 1997.

[Fowler / Scott, 1998] M. Fowler, K. Scott. UML konzentriert, Die neue Standard-Objektmodellierungssprache anwenden. Addison Wesley, Bonn, 1998.

[Frank / Prasse, 1997a] U. Frank, M. Prasse. Ein Bezugsrahmen zur Beurteilung objektorientierter Modellierungssprachen – Veranschaulicht am Beispiel von OML und UML. Arbeitsbericht 6, Institut für Wirtschaftsinformatik, Universität Koblenz-Landau, September 1997.

[Frank / Prasse, 1997b] U. Frank, M. Prasse. Zur Standardisierung objektorientierter Modellierungssprachen: Eine kritische Betrachtung des State of the Art am Beispiel der Unified Modeling Language. Rundbrief des GI-Fachausschußes 5.2, 4(1), September 1997.

[Frank, 1994] U. Frank. Multiperspektivische Unternehmensmodellierung. Theoretischer Hintergrund und Entwurf einer objektorientierten Entwurfsumgebung. Bericht 225, GMD, München/Wien, 1994.

[Frank, 1997a] U. Frank. Enriching Object-Oriented Methods with Domain Specific Knowledge: Outline of a Method for Enterprise Modeling. Arbeitsbericht 4, Institut für Wirtschaftsinformatik, Universität Koblenz-Landau, Koblenz, Juli 1997.

[Frank, 1997b] U. Frank. Towards a Standardization of Object-Oriented Modeling Languages. Arbeitsbericht 3, Institut für Wirtschaftsinformatik, Universität Koblenz-Landau, Koblenz, Mai 1997.

[Frank, 1998] U. Frank. The MEMO Object Modeling Language (MEMO-OML). Arbeitsbericht 10, Institut für Wirtschaftsinformatik, Universität Koblenz-Landau, Juni 1998.

[Franzke / Winter, 1996] A. Franzke, A. Winter. Softwareevaluation mit Mitteln des RequirementsEngineering. EMISA FORUM, (1):71–74, 1996.

[Franzke / Zenz, 1995] A. Franzke, I. Zenz. Studie: Auswahl eines Standardsoftwarepakets für die keramische Industrie. Projektbericht, Institut für Softwaretechnik, Universität Koblenz-Landau, Juni 1995.

[Franzke, 1997] A. Franzke. GRAL: A Reference Manual. Fachbericht Informatik 3/97, Universität Koblenz-Landau, Fachbereich Informatik, Koblenz, 1997.

[Frege, 1891] G. Frege. Funktion und Begriff. Vortrag, gehalten in der Sitzung vom 9.1.1891 in der Jenaischen Gesellschaft für Medizin und Naturwissenschaft (in [Frege, 1994][S. 18–39]), 1891.

[Frege, 1994] G. Frege. Funktion, Begriff, Bedeutung. Vandenhoeck & Ruprecht, Göttingen, 7. Auflage, 1994.

[Gaitanides et al., 1994a] M. Gaitanides, R. Scholz, A. Vrohlings. Prozeßmanagement, Grundlagen und Zielsetzungen. in [Gaitanides et al., 1994b], 1–19. 1994.

[Gaitanides et al., 1994b] M. Gaitanides, R. Scholz, A. Vrohlings, M. Raster (Hrsg.). Prozeßmanagement: Konzepte, Umsetzungen und Erfahrungen des Reengineering. Hanser, München, 1994.

[Gaitanides, 1983] M. Gaitanides. Prozeßorganisation, Entwicklung, Ansätze und Programme prozeßorientierter Organisationsgestaltung. Vahlen, München, 1983.

[Gamma et al., 1994] E. Gamma, R. Helm, R. Johnson, J. Vlissides. Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, 1994.

[Gamma, 1996] E. Gamma. Java und Design Patterns, Eine vielversprechende Kombination. Java Spektrum, 19–24, September/Oktober 1996.

[Gane / Sarson, 1979] C. Gane, T. Sarson. Structured Systems Analysis, Tools & Techniques. Prentice-Hall, Englewood Cliffs, 1979.

[Gane, 1990] C. Gane. Computer Aided Software Engineering. The Methodologies, the Products, and the Future. Prentice Hall, Englewood Cliffs, 1990.

[Genrich / Lautenbach, 1981] H. J. Genrich, K. Lautenbach. System Modeling with High-Level Petri Nets. Theoretical Computer Science, 13:109–136, 1981.

[Gens, 1997] W. Gens (Hrsg.). STJA'97, Smalltalk und Java in Industrie und Ausbildung. Technische Universität Illmenau, Illmenau, 10.-11. September 1997.

[Gigch, 1991] J. P. van Gigch. System Design, Modeling and Metamodeling. Plenum Press, New York, 1991.

[Göhner, 1984] P. Göhner. Methoden zur Entwicklung von Realzeitsystemen und ihre praktische Anwendung in EPOS. in [Trauboth / Jaeschke, 1984], 325–335. 1984.

[Goor et al., 1992] G. van den Goor, S. Hong, S. Brinkkemper. A Comparison of Six Object-Oriented Analysis and Design Methods. Technical report, Method Engineering Institute, University of Twente, 1992.

[Göttler, 1988] H. Göttler. Graph Grammars used in Software Engineering, IFB 178. Springer, Berlin, 1988.

[Graham, 1994] I. Graham. Object-Oriented Methods. Addison-Wessley, Wokingham, 2. edition, 1994.

[Grochla, 1974] E. Grochla. Integrierte Gesamtmodelle der Datenverarbeitung, Entwicklung und Anwendung des Kölner Integrationsmodells (KIM), in: Reihe Betriebsinformatik. Hanser, München, Wien, 1974.

[Grochla, 1978] E. Grochla. Einführung in die Organisationstheorie. Poeschel, Stuttgart, 1978.

[Grochla, 1980] E. Grochla (Hrsg.). Handwörterbuch der Organisation. Poeschel, Stuttgart, 2. Auflage, 1980.

[Grochla, 1982] E. Grochla. Grundlagen der organisatorischen Gestaltung. Poeschel, Stuttgart, 1982.

[Grupp, 1990] B. Grupp. Darstellungstechniken für Organisatoren, Programmierer, EDV-Anwender, Revisoren. Forkel, Wiesbaden, 1990.

[Gutzwiller, 1994] T. A. Gutzwiller. Das CC RIM-Referenzmodell für den Entwurf von betrieblichen, transaktionsorientierten Informationssystemen, in: Betriebs- und Wirtschaftsinformatik. Physika, Heidelberg, 1994.

[Haines, 1996] M. N. Haines. Ein Referenzmodell für Krankenhausinformationssysteme. Diplomarbeit, Institut für Informatik, Universität Koblenz-Landau, Koblenz, September 1996.

[Hammer / Champy, 1996] M. Hammer, J. Champy. Business Reengineering, Die Radikalkur für das Unternehmen. Campus, Frankfurt, 6. Auflage, 1996.

[Harary, 1976] F. Harary. Graphentheorie. Oldenburger, München, 1976.

[Harel, 1987] D. Harel. Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming, 8:231–274, 1987.

[Harel, 1988] D. Harel. On Visual Formalisms. Communication of the ACM, 31(5):514–530, May 1988.

[Hars, 1994] A. Hars. Referenzdatenmodelle: Grundlagen effizienter Datenmodellierung. Gabler, Wiesbaden, 1994.

[Hasselbring, 1997] W. Hasselbring (Hrsg.). Erfolgsfaktor Softwaretechnik für die Entwicklung von Krankenhausinformationssystemen, in: Informatik für Systementwickler, Band 4. Krehl, Münster, 1997.

[Haux et al., 1998] R. Haux, A. Lagemann, P. Knaup, P. Schmücker, A. Winter. Management von Informationssystemen, Analyse, Bewertung, Auswahl, Bereitstellung und Einführung von Informationssystemkomponenten am Beispiel von Krankenhausinformationssystemen. Teubner, Stuttgart, 1998.

[Heinrich, 1997] L. J. Heinrich. Grundlagen der Wirtschaftsinformatik. in [Rechenberg / Pomberger, 1997], 859–874. 1997.

[Hess et al., 1995] T. Hess, L. Brecht, H. Österle. Metamodell Prozessentwurf. Version 1.5. Bericht IM HSG/CC PRO/13, Institut für Wirtschaftsinformatik, Hochschule St. Gallen, 18. April 1995.

[Hesse et al., 1984] W. Hesse, H. Keutgen, A. Luft, D. Rombach. Ein Begriffssystem für die Softwaretechnik, Vorschlag zur Terminologie. Informatik Spektrum, 7:200–213, 1984.

[Heym / Österle, 1993] M. Heym, H. Österle. Computer-aided Methodology Engineering. Information and Software Technology, 35(6/7):345–354, June/July 1993.

[Heym, 1995] M. Heym. Prozeß- und Methodenmanagement für Informationssysteme. Überblick und Referenzmodell. Springer, Berlin, 1995.

[Hill et al., 1994] W. Hill, R. Fehlbaum, P. Ulrich. Organiationslehre 1, Ziele, Instrumente und Bedingungen der Organisation sozialer Systeme. Haupt, Bern, 5. Auflage, 1994.

[Hoffmann, 1980] F. Hoffmann. Begriff der Organisation. in [Grochla, 1980], 1425–1431. 1980.

[Hollingsworth, 1994] D. Hollingsworth. The Workflow Reference Model. Technical Report TC00-1003, Workflow Management Coalition, Brussels, 29. November 1994.

[Hopcroft / Ullmann, 1979] J.E. Hopcroft, J.D. Ullmann. Introduction to Automata Theory, Languages and Computation. Addison Wessley, Reading, Massachusets, 1979.

[Hoppen, 1992] D. Hoppen. Organisation und Informationstechnologie, Grundlagen für ein Konzept zur Organisationsgestaltung. Dr. Kovač, Hamburg, 1992.

[Hub / Fischer, 1977] H. Hub, W. Fischer. Techniken der Aufbauorganisation. W. Kohlhammer, Stuttgart, 1977.

[Huber-Wäschle et al., 1995] F. Huber-Wäschle, H. Schauer, P. Widmayer (Hrsg.). GISI 95, Herausforderungen eines globalen Informationsverbundes für die Informatik, in: Informatik aktuell. Springer, Berlin, 1995.

[Hull / King, 1987] R. Hull, R. King. Semantic Database Modeling: Survey, Applications, and Research Issues. ACM Computing Surveys, 19(3):201–260, September 1987.

[Hunt, 1999] J. Hunt. Product Roundup, Mad About Modeling. Application Development `http://www.appdevadvisor.com/html/prd_roundup/index.shtml`, (22.06.1999), May 1999.

[IDS, 1995] ARIS Methoden. Handbuch, IDS Prof. Scheer GmbH, Saarbrücken, April 1995.

[IDS, 1998] ARIS Toolset 4.0, Whitepaper. `http://www.ids-scheer.de/support.htm` (25. 06.1999), IDS Prof. Scheer GmbH, 20. Juli 1998.

[Imhoff / Paczkowski, 1997] B. Imhoff, J. Paczkowski. Erstellung und Validierung eines Referenzmodells für Krankenhausinformationssysteme. in [Hasselbring, 1997], 55–63. 1997.

[Imhoff et al., 1996] B. Imhoff, C. Jostes, G. Mies. Datenmodell Bundeswehrkrankenhaus, Studienergebnis. Abschlußbericht, Sanitätsamt der Bundeswehr, Abteilung IV, Medizinische Informatik und Informationstechnik, Bonn, Oktober 1996.

[ISG, 1999] OEW-Handbuch. Programmdokumentation, Object Engineering Workbench Version 3.0.3, Demo Version, Innovative Software GmbH, Frankfurt, 1999.

[ISO/IEC DIS 15474, 1998] Information Technology, Software Engineering Data Definition and Interchange, Overview and Framework. Technical Report DIS 15475, 1998.

[ISO/IEC DIS 15475, 1998] Information Technology – Software Engineering Data Definition and Interchange – Transfer Format. Technical Report DIS 15475, 1998.

[ISO/IEC DIS 15476, 1998] Information Technology – Software Engineering Data Definition and Interchange – Integrated Meta-model. Technical Report DIS 15476, 1998.

[ITU, 1988] ITU-T Recommendation B.17: Adoption of the CCITT Specification and description Language (SDL). ITU, 1988.

[ITU, 1996] ITU-TS Recommendation Z.120: Message Sequence Chart (MSC). ITU, Geneva, 1996.

[Jablonski / Bussler, 1996] S. Jablonski, C. Bussler. Workflow-Management: Modeling Concepts, Architectures and Implementation. International Thomson Computer Press, London, 1996.

[Jablonski et al., 1997] S. Jablonski, M. Böhm, W. Schulze (Hrsg.). Workflow-Management, Entwicklung von Anwendungen und Systemen, Facetten einer neuen Technologie. dpunkt, Heidelberg, 1997.

[Jackson, 1975] M. A. Jackson. Principles of Program Design. Academic Press, London, 1975.

[Jackson, 1983] M. A. Jackson. System Developement. Prentice Hall, Englewood Cliffs, 1983.

[Jacobson et al., 1993] I. Jacobson, M. Christerson, P. Jonsson, G. Övergaard. Object-Oriented Software Engineering, A Use Case Driven Approach. Addison Wessley, Wokingham, 4. edition, 1993.

[Jacobson et al., 1994] I. Jacobson, M. Ericsson, A. Jacobson. The Object Advantage, Business Process Reengineering with Object Technology. Addison Wesley, Wokingham, 1994.

[James Martin Associates, 1987] ISP - Information Strategie Planing Handbook. Dublin, 1987.

[James Martin Associates, 1989] Business Area Analysis Handbook. Ashford, 1989.

[Jarke et al., 1995] M. Jarke, R. Gallersdörfer, M. A. Jeusfeld, M. Staudt, S. Eherer. ConceptBase, a Deductive Object Base for Meta Data Management. Journal of intelligent Information Systems, Special Issue on Advances in Deductive Object-Oriented Databases, 4(2):167–192, 1995.

[Jensen, 1998] K. Jensen. An Introduction to the Practical Use of Coloured Petri Nets. in [Reisig / Rozenberg, 1998], 237–292. 1998.

[Johansson et al., 1993] H. Johansson, P. McHugh, J. Pendlebury, W. Wheeler. Business Process Reengineering, Breakpoint Strategies for Market Dominance. Wiley, Chichester, 1993.

[Jones, 1990] C. B. Jones. Systematic Software Development Using VDM. Prentice-Hall, Hempel Hempstead, 2. edition, 1990.

[Jordt / Gscheidle, (o.J.)] A. Jordt, K. Gscheidle. Fernkurs für Organisation, Lehrbriefe 1–6. Gabler, Wiesbaden, (o.J.).

[Joschke, 1980] H. K. Joschke. Darstellungstechniken. in [Grochla, 1980], 431–462. 1980.

[Kamp, 1998] M. Kamp. GReQL, Eine Anfragesprache für das GUPRO-Repository, Sprachbeschreibung (Version 1.2). in [Ebert et al., 1998], 173–202. 1998.

[Kangassalo et al., 1995] H. Kangassalo, H. Jaakkola, S. Ohsuga, B. Wangler (eds.). Information Modeling and Knowledge Bases VI, in: Frontiers in Artificial Intelligence and Applications, Volume 26. IOS Press, Amsterdam, 1995.

[Kaucky, 1988] G. Kaucky. Informationstechnologie und organisatorische Änderungen. Deutscher Universitäts-Verlag, Wiesbaden 1988, 1988.

[Keller et al., 1992] G. Keller, M. Nüttgens, A.-W. Scheer. Semantische Prozeßmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten“ (EPK). IWi-Heft Heft 89, Institut für Wirtschaftsinformatik, Saarbrücken, Januar 1992.

[Kelley, 1961] J.E. Kelley. Critical-Path Planning and Scheduling: Mathematical Basis. Operations Research, 9:296–320, 1961.

[Kelter, 1988] U. Kelter. PCTE, Weiterentwicklung und Standardisierung. Softwaretechnik-Trends, 8(1):33–42, August 1988.

[Kelter, 1989] U. Kelter. PCTE. Informatik Spektrum, 12(3):165–166, Juni 1989.

[Kelly et al., 1996] S. Kelly, K. Lyytinen, M. Rossi. MetaEdit+, A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment. in [Constantopoulos et al., 1996], 1–21. 1996.

[KGSt, 1995] Kommunales Informationsmodell KIM. Bericht 12/1995, Kommunale Gemeinschaftsstelle (KGSt), Köln, Dezember 1995.

[Kieser / Kubicek, 1992] A. Kieser, H. Kubicek. Organisation. Walter de Gruyter, Berlin, New York, 3. Auflage, 1992.

[King et al., 1988] S. King, I. H. Sørensen, J. Woodcock. Z: Grammar and Concrete and Abstract Syntaxes, Version 2.0. Technical Monograph PRG-68, Oxford University Compution Labaratory, Programming Research Group, Oxford, July 1988.

[Kling, 1993] R. Kling. Organizational Analysis in Computer Science. The Information Society, 9(2):71–87, March-May 1993.

[KnowledgeWare, 1987] Information Engineering Workbench: Planning, Analysis and Design Workstation Guide, ESP release 4.0, 1987.

[Knuth, 1974] D. E. Knuth. Structured Programming with go to Statements. Computing Surverys, 6(4):261–301, December 1974.

[Kölzer / Uhe, 1997] A. Kölzer, I. Uhe. Benutzerhandbuch für das KOGGE-Tool BONsai, Version 3.0. Projektbericht 2/97, Universität Koblenz-Landau, Institut für Softwaretechnik, Koblenz, 1997.

[KoopA ADV, 1997] Handlungsleitfaden „IT-gestützte Vorgangsbearbeitung“. Schriftenreihe der KBST 35, Bundesministerium des Innern, Arbeitsgruppe O I 3 (KBSt), Kooperationsausschuß ADV, Arbeitsgruppe IT-gestützte Vorgangsbearbeitung, Redaktion: A. Engel, Forschungsstelle für Verwaltungsinformatik, Universität Koblenz-Landau, 1997.

[Kornwachs, 1997] K. Kornwachs. Um wirklich Informatiker zu sein, genügt es nicht, Informatiker zu sein. Informatik Spektrum, 20(2):79–87, April 1997.

[Kosiol, 1976] E. Kosiol. Organisation der Unternehmung. Th. Gabler, Wiesbaden, 2. Auflage, 1976.

[Krabbel et al., 1996] A. Krabbel, I. Wetzel, S. Ratuski. Objektorientierte Analysetechniken für übergreifende Aufgaben. Softwaretechnik-Trends, 16(3):65–72, September 1996.

[Krabbel et al., 1997] A. Krabbel, I. Wetzel, S. Ratuski. Anforderungsermittlung für Krankenhausinformationssysteme: Definition von Kernsystem und Ausbaustufen. in [Hasselbring, 1997], 1–8. 1997.

[Krallmann / Wood, 1998] H. Krallmann, G. Wood. Bonapart. in [Bernus et al., 1998], 568–587. 1998.

[Krallmann, 1997] H. Krallmann (Hrsg.). Wirtschaftsinformatik ’97. Heidelberg, 1997.

[Krogstie et al., 1995] J. Krogstie, O. I. Lindland, G. Sindre. Defining Quality Aspects for Conceptual Models. in [Falkenberg et al., 1995], 216–231. 1995.

[Krüger, 1981] Wilfried Krüger. Aufgabenanalyse: Renaissance einer Organisationstechnik. Zeitschrift für Organisation, (4):185–198, 1981.

[Kuhn, 1979] T. S. Kuhn. Die Struktur wissenschaftlicher Revolutionen. Suhrkamp, Frankfurt, 4. Auflage, 1979.

[Kullbach / Winter, 1999] B. Kullbach, A. Winter. Querying as an Enabling Technology in Software Reengineering. in [Nesi / Verhoef, 1999], 42–50. 1999.

[Kullbach et al., 1998] B. Kullbach, A. Winter, P. Dahm, J. Ebert. Program Comprehension in Multi-Language Systems. in [WCRE, 1998], 135–143. 1998.

[Laender / Flynn, 1993] A. H. Laender, D. J. Flynn. A semantic Comparison of the Modeling Capabilities of the ER and NIAM Models. in [Elmasri et al., 1993], 242–256. 1993.

[Langer et al., 1997] P. Langer, C. Schneider, J. Wehler. Prozeßmodellierung mit Ereignisgesteuerten Prozeßketten (EPKs) und Petri-Netzen. Wirtschaftsinformatik, 39(5):479–489, 1997.

[Leavitt / Bahrami, 1988] H. J. Leavitt, H. Bahrami. Managerial Psychology. The University of Chicago Press, Chicago, 5. edition, 1988.

[Leavitt, 1965] H. J. Leavitt. Applied Organizational Change in Industry: Structural, Technological and Humanistic Approaches. in [March, 1965], 1144–1170. 1965.

[Lehner et al., 1991] F. Lehner, W. Auer-Rizzi, R. Bauer, K. Breit, J. Lehner, G. Reber. Organisationslehre für Wirtschaftsinformatiker. Hanser, München, Wien, 1991.

[Lehner et al., 1995] F. Lehner, R. Maier, K. Hildebrand. Wirtschaftsinformatik. Theoretische Grundlagen. Hanser, München, 1995.

[Lehner, 1995] F. Lehner. Modelle und Modellierung. in [Lehner et al., 1995], 73–164. 1995.

[Lehner, 1996] F. Lehner (Hrsg.). Softwarewartung und Reengineering, Erfahrungen und Entwicklungen. Gabler, Wiesbaden, 1996.

[Lindland et al., 1994] O. I. Lindland, G. Sindre, A. Sølvberg. Understanding Quality in Conceptual Modeling. IEEE Software, (2):42–49, March 1994.

[Ling et al., 1998] T. W. Ling, S. Ram, M. L. Lee (eds.). Conceptual Modeling, ER'98, LNCS 1507. Springer, Berlin, 1998.

[Löcher / Pühler, 1997] M. Löcher, T. Pühler. Entwicklung eines Softwareevaluationstools. Studienarbeit, Universität Koblenz-Landau, Fachbereich Informatik, Koblenz, Juli 1997.

[Longworth / Nicholls, 1986] G. Longworth, D. Nicholls. SSADM Manual. NCC Publications, Manchester, 1986.

[Loucopoulos / Zicari, 1992] P. Loucopoulos, R. Zicari (eds.). Conceptual Modeling, Databases, and CASE. Wiley, New York, 1992.

[Loucopoulos, 1992] P. Loucopoulos. Conceptual Modeling. in [Loucopoulos / Zicari, 1992], 1–26. 1992.

[Loucopoulos, 1994] P. Loucopoulos (eds.). Entity-Relationship Approach, ER'94. Business Modeling and Re-Engineering. 13th International Conference on the Entity-Relationship Approach, LNCS 881. Springer, Berlin, 13.-16. December 1994.

[Lyytinen / Tahvanainen, 1992] K. Lyytinen, V.-P. Tahvanainen (eds.). Next Generation CASE Tools. IOS Press, Amsterdam, 1992.

[Malcolm et al., 1959] D. G. Malcolm, J. H. Roseboom, C. E. Clark, W. Fazar. Application of a Technique for Resarch and Development Program Evaluation. Operations Research, 7:646–669, 1959.

[Marca / McGowan, 1988] D. A. Marca, C. L. McGowan. SADT, Structured Analysis and Design Technique. McGraw Hill, New York, 1988.

[March, 1965] J. G. March (ed.). Handbook of Organizations. Chicago, 1965.

[Marciniak, 1994] J. J. Marciniak (ed.). Encyclopedia of Software Engineering. Wiley, New York, 1994.

[Marent, 1995] C. Marent. Branchenspezifische Referenzmodelle für betriebswirtschaftliche IV-Anwendungsbereiche. Wirtschaftsinformatik, 37(3):303–313, 1995.

[Martin / McClure, 1985a] J. Martin, C. McClure. Diagramming Techniques for Analysts and Programmers. Prentice Hall, Englewood Cliffs, 1985.

[Martin / McClure, 1985b] J. Martin, C. McClure. Structured Techniques for Computing. Prentice Hall, Englewood Cliffs, 1985.

[Martin / Odell, 1992] J. Martin, J. J. Odell. Object-Oriented Analysis and Design. Prentice Hall, Englewood Cliffs, 1992.

[Martin, 1982] J. Martin. Strategic Data Planning. Prentice Hall, Englewood Cliffs, 1982.

[Marx, 1998] T. Marx. NetCase, Softwareentwurf und Workflow-Modellierung mit Petri-Netzen. Shaker, Aachen, 1998.

[Mayntz, 1985] R. Mayntz. Soziologie der öffentlichen Verwaltung. Müller, Heidelberg, 3. Auflage, 1985.

[Mayr et al., 1995] E. Mayr, G. Schmidt, G. Tinhofer (eds.). Graphtheoretic Concepts in Computer Science, LNCS 903. Springer, Berlin, 1995.

[McDermid / Rook, 1991] J. A. McDermid, P. Rook. Software Development Process Models. in [McDermid, 1991], 15/1–15/36. 1991.

[McDermid, 1991] J. A. McDermid. Software Engineer's Reference Book. Butterworth-Heinemann, Oxford, 1991.

[Mehlhorn, 1984] K. Mehlhorn. Graph Algorithms and NP-Completeness, in: Data structures and algorithms, Volume 2. Springer, Berlin, 1984.

[Menzel / Mayer, 1998] C. Menzel, R. J. Mayer. The IDEF Family of Languages. in [Bernus et al., 1998], 208–241. 1998.

[Menzl / Nauer, 1974] A. Menzl, E. Nauer. Das Funktionendiagramm, ein flexibles Organisations- und Führungsmittel. Paul Haupt, Bern, 2. Auflage, 1974.

[Mišić / Moser, 1997] V. B. Mišić, S. Moser. Formal Approach to Metamodeling: A Generic Object-Oriented Perspective. in [Embley / Goldstein, 1997], 243–256. 1997.

[Micro Tool, 1997] case/4/0/ im Überblick. `http://www.microTool.de/prod.d/pdf/case40/faltblatt42.pdf` (24.06.1999), Micro Tool GmbH, Berlin, 1997.

[Micro Tool, 1998] case/4/0/ Quicktour. `http://www.microTool.de/prod.d/pdf/case40/quicktour43.pdf` (24.06.1999), Micro Tool GmbH, Berlin, 1998.

[Microsoft, 1998] PROJECT-98, Das Handbuch. Microsoft Press, Redmond, 1998.

[Miller, 1956] G. A. Miller. The magical Number Seven, plus or minus two: Some Limits on our Capacity for processing Information. Psychology Review, 63(2):81–97, March 1956.

[Mittelstraß, 1984] J. Mittelstraß (Hrsg.). Enzyklopädie Philosophie und Wissenschaftstheorie. Bibliographisches Institut, Mannheim, 1984.

[Moody / Shanks, 1994] D. L. Moody, G. G. Shanks. What Makes a Good Data Model? Evaluating the Quality of Entity Relationship Models. in [Loucopoulos, 1994], 94–111. 1994.

[Münch et al., 1998] M. Münch, A. Schürr, A. J. Winter. Integrity Constraints in the Multiparadigm Language PROGRES. Technical Report DSSE-TR-98-2, Department of Electronics and Computer Science, University of Southampton, Southampton, February 1998.

[Mylopoulos / Levesque, 1996] J. Mylopoulos, H. J. Levesque. An Overview of Knowledge Representation. in [Brodie et al., 1986], 3–17. 1996.

[Mylopoulos et al., 1990] J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarkis. Telos: Representing Knowledge About Information Systems. ACM Transactions on Information Systems, 8(4):325–262, October 1990.

[Mylopoulos, 1992] J. Mylopoulos. Conceptual Modeling with TELOS. in [Loucopoulos / Zicari, 1992], 49–68. 1992.

[Nagl, 1979] M. Nagl. Graph-Grammatiken, Theorie, Implementierung, Anwendung. Vieweg, Braunschweig, 1979.

[Nagle et al., 1992] T. Nagle, J. Nagle, L. Gerholz, P. Eklund (eds.). Conceptual Structures: Current Research and Practice. Ellis, Horwood, 1992.

[Nassi / Shneiderman, 1973] I. Nassi, B. Shneiderman. Flowchart Techniques for Structured Programming. ACM SIGPLAN Notices, (8):12–26, 1973.

[Naur / Randell, 1969] P. Naur, B. Randell (eds.). Software Engineering, Report on a Conference sponsored by the NATO Science Committee, Garmisch, Germany, October 7-11, 1968. Scientific Affairs Division NATO, Brussels, January 1969.

[Nesi / Verhoef, 1999] P. Nesi, C. Verhoef (eds.). Proceedings of the 3nd European Conference on Software Maintenance and Reengineering. IEEE Computer Society, Los Alamitos, 1999.

[Nordsieck, 1962] F. Nordsieck. Die schaubildliche Erfassung und Untersuchung der Betriebsorganisation. Poeschel, Stuttgart, 6. Auflage, 1962.

[Nordsieck, 1972] F. Nordsieck. Betriebsorganisation, Lehre und Technik. Poeschel, Stuttgart, 2. Auflage, 1972.

[Oestereich, 1998] B. Oestereich. Objektorientierte Geschäftsprozeßmodellierung mit der UML. Objektspektrum, (2):48–52, März/April 1998.

[Ogden / Richards, 1923] C. K. Ogden, I. A. Richards. The Meaning of Meaning, A Study of the Influence of Language upon Thought and of the Science of Symbolism. Harcourt, Base and Company, New York, 1923.

[Olivé / Pastor, 1997] A. Olivé, J. A. Pastor (eds.). Advanced Information Systems Engineering, 9th international Conference, CAiSE'97, LNCS 1250. Springer, Berlin, 1997.

[Olle et al., 1982] T. W. Olle, H. G. Sol, A. A. Verrijn-Stuart (eds.). Information Systems Methodologies. North Holland, Amsterdam, 1982.

[Olle et al., 1987] T. W. Olle, G. H. Sol, A. A. Verrijn-Stuart (eds.). Information Systems Design Methodologies: A Comparative Review. Proceedings of the IFIP WG 8.1 Working Conference on Comparative Review of Information Systems Design Methodologies, Noordwijkerhout, 19-14. May 1982. Elsevier, North-Holland, 3. edition, 1987.

[Olle et al., 1991] T. W. Olle, J. Hagelstein, I. G. Macdonald, C. Rolland, H. G. Sol, F. J. M. van Assche, A. A. Verrijn-Stuart. Information Systems Methodologies. A Framework for Understanding. Addison-Wesley, Wokingham, 2. edition, 1991.

[OMG, 1999] Unified Modeling Language Specification, Version 1.3a1. CD-ROM in [Rumbaugh et al., 1999], January 1999.

[Österle / Gutzwiller, 1992] H. Österle, T. Gutzwiller. Konzepte angewandter Analyse- und Design-Methoden. Ein Referenz-Metamodell für die Analyse und das System-Design, Band 1. Angewandte Informationstechnik, Hallbergmoos, 1992.

[Papazoglou, 1995] M. P. Papazoglou (ed.). OOER'95: Object-Oriented and Entity Relationship Modeling, LNCS 1021. Springer, Berlin, 1995.

[Partsch, 1998] H. Partsch. Requirements-Engineering systematisch. Springer, Berlin, 1998.

[PERT, 1958] PERT Summary Report, Phase 1. Technical report, Special Projects Office, Bueau of Naval Weapons, Department of the Navy, U.S. Government Printing Office, Washington, D.C., July 1958.

[Petri, 1962] C. A. Petri. Kommunikation mit Automaten. Schriften des Institutes für instrumentelle Mathematik, Bonn, 1962.

[Popkin, 1999] Enterprise Modeling, Aligning Business and Information Technology. White Paper, Popkin Software and Systems Ltd, Leamington, 1999.

[Porter, 1985] M. E. Porter. Competitive Advantage. Free Press, New York, 1985.

[Pressman, 1997] R. S. Pressman. Software Engineering, A Practitioners's Approach. McCraw-Hill, New York, 4. edition, 1997.

[Pro Ubis, 1999a] Referenz-Handbuch Bonapart 2.3. `http://www.proubis.de/bonapartcd/` (21.06.1999), Pro Ubis GmbH, Berlin, 1999.

[Pro Ubis, 1999b] Tutorial Bonapart 2.3. `http://www.proubis.de/bonapartcd/` (21.06. 1999), Pro Ubis GmbH, Berlin, 1999.

[Raet BV, 1988] Information Planning, De aanpak van Raet, Juni 1988.

[Raue, 1996] H. Raue. Wiederverwendbare betriebliche Anwendungssysteme, Grundlagen und Methoden ihrer objektorientierten Entwicklung. Deutscher Universitäts-Verlag, Wiesbaden, 1996.

[Rechenberg / Pomberger, 1997] P. Rechenberg, G. Pomberger (Hrsg.). Informatik Handbuch. Hanser, München, 1997.

[REFA, 1992] REFA. Methodenlehre der Betriebsorganisation, Aufbauorganisation. Hanser, München, 1992.

[Reisig / Rozenberg, 1998] W. Reisig, G. Rozenberg (eds.). Lectures on Petri Nets II: Applications, Advances in Petri Nets, LNCS 1492. Springer, Berlin, 1998.

[Reisig, 1992] W. Reisig. Petrinetze, Eine Einführung. Springer, Berlin, 2. Auflage, 1992.

[Rolf, 1998a] A. Rolf. Grundlagen der Organisations- und Wirtschaftsinformatik. Springer, Berlin, 1998.

[Rolf, 1998b] A. Rolf. Herausforderungen für die Wirtschaftsinformatik. Informatik Spektrum, 21(5):259–264, Oktober 1998.

[Rolland / Cauvet, 1992] C. Rolland, C. Cauvet. Trends and Perspectives in Conceptual Modeling. in [Loucopoulos / Zicari, 1992], 27–48. 1992.

[Rosemann / Schütte, 1997] M. Rosemann, R. Schütte. Grundsätze ordnungsmäßiger Referenzmodellierung. in [Becker et al., 1997], 16–33. 1997.

[Rosemann / zur Mühlen, 1995] M. Rosemann, M. zur Mühlen. Der Lösungsbeitrag von Metadatenmodellen beim Vergleich von Workflowmanagementsystemen. Arbeitsbericht 48, Institut für Wirtschaftsinformatik, Westfälische Wilhelms-Universität Münster, Juni 1995.

[Rosemann / zur Mühlen, 1998] M. Rosemann, M. zur Mühlen. Modellierung der Aufbauorganisation in Workflow-Management-Systemen: Kritische Bestandsaufnahme und Gestaltungsvorschläge. EMISA-Forum, (1):78–85, 1998.

[Rosemann, 1996] M. Rosemann. Komplexitätsmanagement in Prozeßmodellen: methodenspezifische Gestaltungsempfehlungen für die Informationsmodellierung. Gabler, Wiesbaden, 1996.

[Ross, 1977] D. T. Ross. Structured Analysis (SA): A Language for Communicating Ideas. IEEE Transactions on Software Engineering, 3(1):16–34, January 1977.

[Roth, 1993] E. Roth (Hrsg.). Sozialwissenschaftliche Methoden. Lehr- und Handbuch für Forschung und Praxis. Oldenbourg, München, 3. Auflage, 1993.

[Roy, 1962] B. Roy. Cheminement et connexité dans les graphs. Applications aux problémes d' ordonnancement. METRA: Série Spéciale No. 1, Mai 1962.

[Royce, 1970] W. W. Royce. Managing the Development of large Software Systeme. Proceedings of IEEE WESCON 1970 (Reprint in [Thayer, 1988, 118–127]), 1–9, 1970.

[Rozenberg, 1997] G. Rozenberg (ed.). Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations. World Scientific, 1997.

[Rudolph et al., 1996] E. Rudolph, P. Graubmann, J. Grabowski. Tutorial on Message Sequence Charts. Computer Networks and ISDN Systems, 28(12):1629–1641, 1996.

[Rumbaugh et al., 1991] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen. Object-Oriented Modeling and Design. Prentice Hall, Englewood Cliffs, 1991.

[Rumbaugh et al., 1999] J. Rumbaugh, I. Jacobson, G. Booch. The Unified Modeling Language Reference Manual. Addison Wesley, Reading, 1999.

[Rumbaugh, 1995] J. Rumbaugh. What is a Method. Journal of Object-Oriented Programming, 8(6):10–26, 1995.

[Saeki, 1995] M. Saeki. Object-Oriented Meta Modeling. in [Papazoglou, 1995], 250–259. 1995.

[Samet, 1979] P. A. Samet (ed.). EURO IFIP 79. North Holland, Amsterdam, 1979.

[Scheer, 1992] A.-W. Scheer. Architektur integrierter Informationssysteme, Grundlagen der Unternehmensmodellierung. Springer, Berlin, 2. Auflage, 1992.

[Scheer, 1994] A.-W. Scheer. Wirtschaftsinformatik, Referenzmodelle für industrielle Geschäftsprozesse. Springer, Berlin, 5. Auflage, 1994.

[Scheer, 1996] A.-W. Scheer (Hrsg.). Rechnungswesen und EDV 96. Kundenorientierung in Industrie, Dienstleistung und Verwaltung. 17. Saarbrücker Arbeitstagung. Physica, Heidelberg, 1996.

[Scheer, 1997] A.-W. Scheer. ARIS House of Business Engineering, Konzept zur Beschreibung und Ausführung von Referenzmodellen. in [Becker et al., 1997], 3–15. 1997.

[Scheer, 1998] A.-W. Scheer. ARIS. in [Bernus et al., 1998], 541–565. 1998.

[Schmidt, 1980] G. Schmidt. Die Entwicklung von Organisationsmethoden und Techniken. Zeitschrift für Organisation, 322–335, 1980.

[Schmidt, 1989] G. Schmidt. Methode und Technik der Organisation, in: Schriftenreihe „Der Organisator“, Band 1. Verlag Dr. Götz Schmidt, Gießen, 8. Auflage, 1989.

[Scholz-Reiter, 1990] B. Scholz-Reiter. CIM-Informations- und Kommunikationssyteme, Darstellung von Methoden und Konzeption eines rechnergestützten Werkzeugs für die Planung. Oldenbourg, München, 1990.

[Schulz, 1988] A. Schulz. Software Entwurf, Methoden und Werkzeuge. Oldenburg, München, 1988.

[Schumm et al., 1995] T. Schumm, C. Thomann, A. Winter. Evaluation von Krankenhausinformationssystemen. Projektbericht 4/95, Institut für Softwaretechnik, Universität Koblenz-Landau, Juli 1995.

[Schürr / Westfechtel, 1992] A. Schürr, B. Westfechtel. Graphgrammatiken und Graphersetzungssyteme. Aachener Informatik Berichte AIB 92–15, Fachgruppe Informatik, RWTH Aachen, 1992.

[Schürr, 1991a] A. Schürr. Operationales Spezifizieren mit Programmierten Graphersetzungssystemen: Formale Definitionen, Anwendungen und Werkzeuge. Deutscher Universitätsverlag, Braunschweig, 1991.

[Schürr, 1991b] A. Schürr. PROGRES: A VHL-Language Based on Graph Grammars. in [Ehrig et al., 1991], 641–659. 1991.

[Schürr, 1994] A. Schürr. PROGRES, A Visual Language and Environment for PROgramming with Graph REwriting Systems. Aachener Informatik Berichte AIB 94–11, Lehrstuhl für Informatik III, RWTH Aachen, 1994.

[Schütte / Rotthowe, 1998] R. Schütte, T. Rotthowe. The Guidelines to Modeling, An Approach to Enhance the Quality in Information Models. in [Ling et al., 1998], 240–254. 1998.

[Schütte, 1996] R. Schütte. Referenzprozeßmodelle für Handelsunternehmen. HMD, 33(192): 72–87, 1996.

[Schwarz, 1980] H. Schwarz. Stelle. in [Grochla, 1980], 2113–2118. 1980.

[Schwarze, 1994] J. Schwarze. Netzplantechnik, Eine Einführung in das Projektmanagement. Neue Wirtschafts-Briefe, Herne, 7. Auflage, 1994.

[Seubert, 1997] M. Seubert. Business Objekte und objektorientiertes Prozeßdesign. in [Becker et al., 1997], 46–64. 1997.

[Shlaer / Mellor, 1988] S. Shlaer, S. J. Mellor. Object-Oriented Systems Analysis: Modeling the World in Data. Yourdon Press, Englewood Cliffs, 1988.

[Simon et al., 1997] C. Simon, H. Ridder, T. Marx. The Petri Net Tools Neptun and Posidon. Fachbericht Informatik 15/97, Universität Koblenz-Landau, Institut für Informatik, Koblenz, 1997.

[Simon, 1994] F. Simon. Tagungsband zum Workshop „Deklarative Programmierung und Spezifikation“, der GI-Fachgruppe 2.1.4 Alternative Konzepte für Sprachen und Rechner, 9.-11. Mai 1994, Bad Honnef. Fachbericht Nr. 9412, Institut für Informatik und praktische Mathematik, Universität Kiel, Kiel, September 1994.

[Smith / Smith, 1977] J. M. Smith, D. C. P. Smith. Databases Abstractions: Aggregation. Communications of the ACM, 20:405–413, 1977.

[Smith, 1990] A. Smith. Der Wohlstand der Nationen, Eine Untersuchung seiner Natur und seiner Ursachen. dtv, 5. Auflage, 1990.

[Sommerville, 1996] I. Sommerville. Software Engineering. Addison-Wesley, Wokingham, 5. edition, 1996.

[Sowa / Zachman, 1992] J. F. Sowa, J. A. Zachman. Extending and formalizing the framework for information systems architecture. IBM Systems Journal, 31(3):590–616, 1992.

[Sowa, 1984] J. F. Sowa. Conceptual Structures. Information, Processing in Mind and Machine, in: The Systems Programming Series. Addison-Wesley, Reading, 1984.

[Sowa, 1992] J. F. Sowa. Conceptual Graphs Summary. in [Nagle et al., 1992], 3–51. 1992.

[Spivey, 1992] J. M. Spivey. The Z Notation: A Reference Manual, in: International Series in Computer Science. Prentice Hall, Englewood Cliffs, 2. edition, 1992.

[Staff, 1996] J. Staff. MS Project 4.1. Tewi, München, 1996.

[Staud, 1999] J. Staud. Geschäftsprozeßanalyse mit Ereignisgesteuerten Prozeßketten, Grundlage des Business Reengineering für SAP R/3 und andere Betriebswirtschaftliche Standardsoftware. Springer, Berlin, 1999.

[Stegmüller, 1973] W. Stegmüller. Probleme und Resultate der Wissenschaftstheorie und analytischen Philosophie, Band 2, Theorie und Erfahrung. Springer, Berlin, 2. Auflage, 1973.

[Steinebach, 1983] N. Steinebach. Verwaltungsbetriebslehre, in: Das Verwaltungstudium in Grundrissen, Band 6. Walhalla und Praetoria, Regensburg, 2. Auflage, 1983.

[STJA, 1999] STJA. STJA 99, 5. Fachkonferenz Smalltalk und Java in Industrie und Ausbildung. CD-ROM, 1999.

[Strahringer, 1996] S. Strahringer. Metamodellierung als Instrument des Methodenvergleichs, Eine Evaluierung am Beispiel objektorientierter Analysemethoden. Shaker, Aachen, 1996.

[Süttenbach / Ebert, 1997] R. Süttenbach, J. Ebert. A Booch Metamodel. Fachbericht Informatik 5/97, Universität Koblenz-Landau, Institut für Informatik, Koblenz, 1997.

[Süttenbach, 1998] R. Süttenbach. Formalizing Visual Languages of Object-Oriented Methods. in [Thurner / Erni, 1998]. 1998.

[Süttenbach, 2000] R. Süttenbach. Formalisierung visueller Modellierungssprachen objektorientierter Methoden. Abstrakte Syntax und Semantik. erscheint 2000.

[Tanenbaum, 1990] A. S. Tanenbaum. Computer Netzwerke. Wolfram, Attenkirchen, 2. Auflage, 1990.

[Taylor, 1919] F. W. Taylor. Die Grundsätze wissenschaftlicher Betriebsführung. Oldenbourg, München, 1919.

[Teorey et al., 1989] T. J. Teorey, G. Wei, D. L. Bolton, J. A. Koenig. ER Model Clustering as an Aid for User Communication and Documentation in Database Design. Communications of the ACM, 32(8):975–987, August 1989.

[Thalheim, 1996] B. Thalheim (ed.). Conceptual Modeling, ER'96, LNCS 1157. Springer, Berlin, 1996.

[Thayer, 1988] R. H. Thayer (ed.). IEEE Tutorial on Software Engineering Project Management. Computer Society of the IEEE, Los Angeles, 1988.

[Thurner / Erni, 1998] V. Thurner, A. Erni. CAiSE'98 5th Doctoral Consortium on Advanced Information Systems Engineering, Proceedings. Technical Report, Eidgenössische Technische Hochschule Zürich, Zürich, May 1998.

[Timm, 1991] M. Timm (Hrsg.). Requirements Engineering '91. „Structured Analysis“ und verwandte Ansätze, Marburg, 10./11. April 1991, Proceedings, IFB 273. Springer, Berlin, 1991.

[Tolvanen et al., 1996] J.-P. Tolvanen, M. Rossi, H. Liu. Method Engineering: Current Research Directions and Implications for Future Research. in [Brinkkemper et al., 1996], 296–317. 1996.

[Tolvanen, 1998] J.-P. Tolvanen. Incremental Method Engineering with Modeling Tools, in: Jyväskylä Studies in Computer Science, Economics and Statistics, Volume 47. University of Jyväskylä, Jyväskylä, 1998.

[Trauboth / Jaeschke, 1984] H. Trauboth, A. Jaeschke (Hrsg.). Prozeßrechner 1984, Prozeßdatenverarbeitung im Wandel, 4. GI/GMR/KfK-Fachtagung, Karlsruhe, 26.-28. September 1984, IFB 86. Springer, Berlin, 1984.

[Troitzsch, 1990] K. G. Troitzsch. Modellbildung und Simulation in den Sozialwissenschaften. Westdeutscher Verlag, Opladen, 1990.

[Turner et al., 1987] W. S. Turner, R. P. Langerhorst, G. E. Hice, H. B. Eilers, A. A. Uijtenbroek. System Development Methodology (SDM II). North Holland and Pandata, 1987.

[Verheijen / van Bekkum, 1982] G. M. A. Verheijen, J. van Bekkum. NIAM: An Information Analysis Method. in [Olle et al., 1982], 537–589. 1982.

[Verhoef et al., 1991] T. F. Verhoef, A. H. M. ter Hofstede, G. M. Wijers. Structuring Modeling Knowledge for CASE Shells, SOCRATES Project. in [Andersen et al., 1991], 501–524. 1991.

[Versteegen / Versteegen, 1998] C. Versteegen, G. Versteegen. UML inside, CASE-Markt: Stand der Dinge. iX, (9):85–89, September 1998.

[Versteegen, 1998] G. Versteegen. Objektorientierte Geschäftsprozeßmodellierung mit der UML: die „Innovator Business Workbench“. Objektspektrum, (1):62–67, Januar/Februar 1998.

[Vescoukis et al., 1996] V. Vescoukis, N. Papaspyrou, E. Theodorakis (eds.). CAiSE'96, Software engineering Challenges in Modern Information Systems, 3rd Doctoral Consortium, Athens, May 1996. Software Engineering Laboratory, National Technical University of Athens.

[Voßbein, 1987] R. Voßbein. Organisation. Oldenbourg, München, 2. Auflage, 1987.

[Vossen, 1994] G. Vossen. Datenmodelle, Datenbanksprachen und Datenbank-Management-Systeme. Addison-Wesley Publishing Company, Bonn, 2. Auflage, 1994.

[Ward / Mellor, 1985] P. T. Ward, S. J. Mellor. Structured Development for Real-Time Systems, Volume 1 Introduction and Tools. Prentice-Hall, Englewood Cliffs, 1985.

[Warnier, 1974] J. D. Warnier. Logical Construction of Programs. Stefert Kroese, Leiden, 1974.

[WCRE, 1998] Fifth Working Conference on Reverse Engineering. IEEE Computer Society, Los Alamitos, 1998.

[WCRE, 1999] Sixth Working Conference on Reverse Engineering. IEEE Computer Society, Los Alamitos, 1999.

[WfMC, 1996] Terminology & Glossary. Technical Report WFMC-TC-1011, Workflow Management Coalition, Brussels, June 1996.

[Wieringa, 1998] R. J. Wieringa. A Survey of Structured and Object-Oriented Software Specification Methods and Techniques. ACM Computing Surveys, 30(4):459–527, December 1998.

[Wijers et al., 1992] G. M. Wijers, A. H. M. ter Hofstede, N. E. van Oostermon. Representation of Information Modeling Knowledge. in [Lyytinen / Tahvanainen, 1992], 167–223. 1992.

[Willke, 1982] H. Willke. Systemtheorie, Eine Einführung in die Grundprobleme. Gustav Fischer, Stuttgart, 1982.

[Wilson, 1976] R. J. Wilson. Einführung in die Graphentheorie. Vandenhoeck & Ruprecht, Göttingen, 1976.

[Winter / Ebert, 1996] A. Winter, J. Ebert. Ein Referenz-Schema zur Organisationsbeschreibung. in [Becker / Vossen, 1996b], 101–123. 1996.

[Winter / Ebert, 1997] A. Winter, J. Ebert. Referenzmodelle für Krankenhausinformationssysteme und deren Anwendung. in [Zwierlein, 1997], 548–562. 1997.

[Winter et al., 1999] A. Winter, A. Winter, K. Becker, O. Bott, B. Birgl, S. Gräber, W. Hasselbring, R. Haux, C. Jostes, O.-S. Penger, H.-U. Prokosch, J. Ritter, R. Schütte, A. Terstappen. Referenzmodelle für die Unterstützung des Managements von Krankenhausinformationssystemen. Informatik, Biometrie und Epidemiologie in Medizin und Biologie, 30(4):173–189, 1999.

[Winter, 1992] A. Winter. Spezifikation der abstrakten und konkreten Syntax für die Modellierung nach dem Entity-Relationship-Paradigma. Diplomarbeit D 165, Universität Koblenz-Landau, Fachbereich Informatik, Koblenz, März 1992.

[Winter, 1996] A. Winter. A Reference-Scheme for Describing Organizations. in [Vescoukis et al., 1996], 68–71. 1996.

[Wintraeken, 1985] J. J. V. R. Wintraeken. Informatie-analyse volgens NIAM. Academic Service, Den Haag, 1985.

[Wirfs-Brock et al., 1990] R. Wirfs-Brock, B. Wilkerson, L. Wiener. Designing Object-Oriented Software. Prentice Hall, Englewood Cliffs, 1990.

[Wordsworth, 1993] J. B. Wordsworth. Software Development with Z, A Practical Approach to Formal Methods in Software Enginering. Addison-Wesley, 1993.

[Yourdon, 1989] E. Yourdon. Modern Structured Analysis. Prentice Hall, Englewood Cliffs, 1989.

[Zamperoni / Löhr-Richter, 1993] A. Zamperoni, P. Löhr-Richter. Enhancing the Quality of Conceptual Database Specification through Validation. in [Elmasri et al., 1993], 85–98. 1993.

[Zemanek, 1971] H. Zemanek. Was ist Informatik?, in: Rektorat der Technischen Hochschule in Wien (Hrsg.): Informatik, Aspekte und Studienmodelle. Wien, New York, 1971.

[Zimmermann, 1971] H. J. Zimmermann. Netzplantechnik. Walter de Gruyter, Berlin, New York, 1971.

[Zinnen, 1995] H. Zinnen. Polymorphie und Typanalysen in PROGRES. Diplomarbeit, RWTH Aachen, Lehrstuhl für Informatik III, Aachen, 1995.

[Zündorf, 1996] A. Zündorf. PROgrammierte GRaph ErsetzungsSysteme, Spezifikation, Implementierung und Anwendung einer integrierten Entwicklungsumgebung. Deutscher Universitäts-Verlag, 1996.

[Zwierlein, 1997] E. Zwierlein (Hrsg.). Klinikmanagement, Erfolgsstrategien für die Zukunft. Urban & Schwarzenberg, München, 1997.

Andreas Winter studierte von 1984 bis 1992 Informatik mit Anwendungsschwerpunkt sozialwissenschaftliche Informatik (Verwaltungsinformatik) an der Universität Koblenz-Landau. Anschließend wurde er wissenschaftlicher Mitarbeiter der Universität Koblenz-Landau in der Forschungsgruppe Softwaretechnik bei Prof. Dr. Jürgen Ebert. Von September 1992 bis August 1994 wurden seine wissenschaftlichen Arbeiten durch ein Graduiertenstipendium des Landes Rheinland-Pfalz unterstützt.

Seine wissenschaftlichen Interessen beziehen sich sowohl auf die formalen Grundlagen als auch auf die interdisziplinäre Anwendung der Softwaretechnik. Die Schwerpunkte seiner wissenschaftlichen Arbeiten sind in den Bereichen der Metamodellierung visueller und textueller Sprachen, der Methoden und Techniken des Software-Reengineerings und der Modellierung von Informationssystemen (insbesondere Krankenhausinformationssysteme) anzusiedeln.