KONZEPTION UND ENTWURF EINER KOMPONENTE FÜR ORGANISATIONSMODELLE

Diplomarbeit an der Universität Ulm
Fakultät für Informatik

vorgelegt von
Marco Berroth

Gutachter:
Prof. Dr. Peter Dadam, Universität Ulm
Dr. Manfred Reichert, Universität Twente

Juni 2005

Inhaltsverzeichnis

Kurzfassung

Im Umfeld von Workflow-Management-Systemen (WfMS) kommt der Organisationsverwaltung die Aufgabe zu, für anfallende Arbeitsschritte nach einem festgelegten Verfahren die Menge der potentiellen Bearbeiter zu bestimmen. Die dazu benötigten Daten bezieht die Organisationsverwaltung aus dem Organisationsmodell, das diese Informationen beherbergt. Ein Organisationsmodell bietet die Möglichkeit, die Aufbaustruktur einer Organisation hinreichend genau zu formalisieren, um eine Qualifikation für eine Aufgabe nach beliebigen Kriterien definieren zu können.

Da die Aufbaustruktur einer Organisation kein unabänderliches Gebilde darstellt, sondern sich ständig im Wandel befindet, muß eine Organisationsverwaltung auch Operationen zur Verfügung stellen, mit denen die Organisationsstruktur unter Einhaltung von Korrektheitsbedingungen manipuliert werden kann.

Diese Arbeit betrachtet die Möglichkeiten, die verfügbare Systeme für die Organisationsverwaltung bieten und zeigt deren Grenzen auf. Im Anschluß an diese Betrachtung werden neue, ausdrucksmächtigere Konzepte vorgestellt und bis zu einem konkreten Entwurf entwickelt. Besondere berücksichtigt werden dabei Aspekte wie:

- Flexibilität und Anpaßbarkeit an reale Organisationen
- Hohe Ausdrucksmächtigkeit bei der Bestimmung von Aufgabenträgern
- Zusammenwirken autonomer Teilsysteme in einem Workflow-Management-System
- Evaluierung von möglichen Datenhaltungssystemen für die Organisationsverwaltung (LDAP)

Ziel dieser Arbeit ist der Entwurf einer Organisationsmodell-Komponente für den zweiten Prototypen des ADEPT – Workflow-Management-Systems der Universität Ulm.

1 Einleitung

1.1 Motivation

Unter dem Druck des Marktes ist es heutzutage für eine Organisation eine große Herausforderung, ihr Unternehmensziel mit möglichst hoher Effizienz zu erreichen. Schon ein kleiner Vorsprung gegenüber dem Konkurrenten kann dabei für die zukünftige Position am Markt entscheidend sein.
Effizienz in diesem Sinne ist das Verhältnis zwischen einem in gewünschter Qualität existierendem Nutzen und dem Aufwand, der erbracht werden muß, um diesen Nutzen zu erreichen.

In diese Definition fallen sowohl materielle Produkte und Erzeugnisse als auch Dienstleistungen. Gemeinsam ist allen, daß sie durch einen Wertschöpfungsprozeß geschaffen werden. Die Effizienz ist also auch ein Maß dafür, wie gut die vorhandenen Ressourcen in diesen Prozeß integriert sind.

Die Planung dieser Prozesse ist umso schwieriger, je größer das Unternehmen und je komplexer die Prozesse sind. Schon seit längerem werden computergestützte Systeme eingesetzt, um diese Komplexität handhabbar zu machen. Die einfacheren Systeme unterstützen dabei die Modellierung und Ausführung von Geschäftsprozessen innerhalb einer einzigen Applikation, teilweise allerdings nur in sehr eingeschränktem Rahmen. Ein Bestreben der Softwarebranche, insbesondere auch der aktuellen Forschung ist es, einen solchen als Workflow bezeichneten formalisierten Prozeß nicht nur innerhalb einer einzigen Applikation darstellen zu können, sondern von Applikationen unabhängig, die komplette Struktur der Organisation zu modellieren, und auf Basis dieser Struktur Workflows auszuführen, deren einzelne Prozeßschritte unter Zuhilfenahme beliebiger Applikationen bearbeitet werden können. System, die ein solches Vorgehen unterstützen, werden Workflow-Management-Systeme genannt (WfMS).

Der Einsatz eines WfMS bedeutet letztendlich, daß bei der Entwicklung von Applikationen, die ein Workflow-Management-System nutzen, die Aspekte der Prozeßlogik aus der Applikation entfernt werden und stattdessen die Routingeigenschaften des WfMS genutzt werden. Eine Applikation stellt also nur noch Schnittstellen zur Verfügung, über die das WfMS die zum Ablauf benötigten Informationen extrahieren kann. Alle prozeßrelevanten Daten stehen somit unter Kontrolle des WfMS. Damit sind die einbezogenen Applikationen entkoppelt von Änderungen am Prozeß oder an der Organisation. Änderungen werden für diese unsichtbar im WfMS vorgenommen.

Workflow-Management-Systeme haben ihren Ursprung in der Idee der Büroautomation. Dieser Gedanke, der schon in den 1970er Jahren aufkam, wird dadurch erweitert, daß diese Systeme aufgrund der Verfügbarkeit der Hardware nun nicht mehr nur für Büroanwendungen, sondern unternehmensweit eingesetzt werden können. Dazu trägt vor allem auch die schon erwähnte Sichtweise der Integration verschiedener Applikationen bei.
Da Workflow-Management-Systeme den Kontroll- und Datenfluß zwischen den an den Prozessen Beteiligten unterstützen, können sie zu höherer Qualität und schnellerer Ausführung der Geschäftsprozesse beitragen und somit auch zur Effizienz der Unternehmen.

Oftmals ist die Erstinstallation eines WfMS in einer Organisation mit erheblichem Aufwand verbunden, da sowohl die Struktur der Organisation als auch die Geschäftsprozesse erstmals

formal beschrieben werden müssen. Meist geht damit eine komplette Neumodellierung bestehender Prozesse (Business Process Reengineering[1]) einher, da bis dahin in der Regel die prozeßorientierte Sichtweise zu wenig ausgeprägt war.

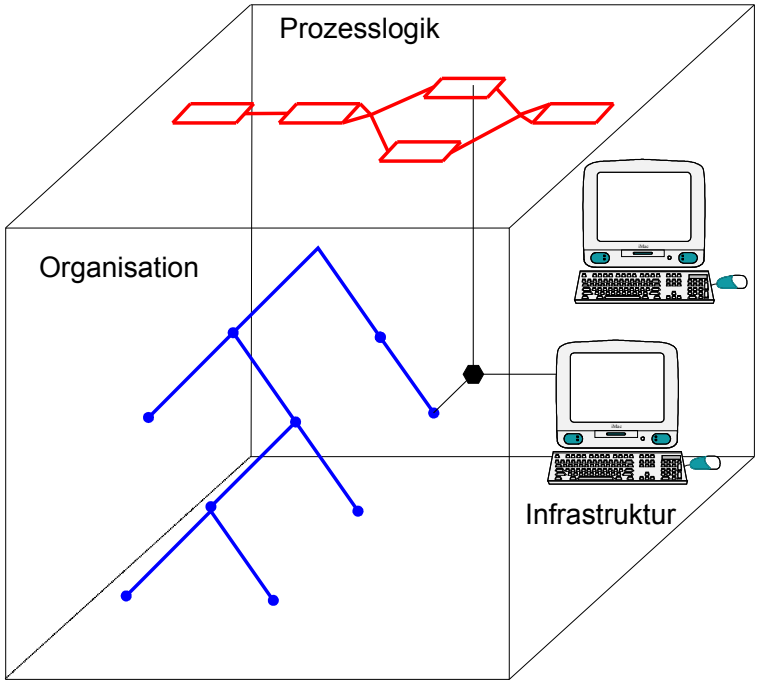


Abbildung 1-1: *Drei Dimensionen eines Workflows (aus [MQWF04])*

Um die Ausführung von Prozessen durch ein Workflow-Management-System zu ermöglichen, muß ein Unternehmen formal in drei Dimensionen beschrieben werden. Diese drei Sichten sind laut [MQWF04]:

- Identifizierung und Beschreibung der Abläufe und Datenflüsse
- Organisationsstruktur, auf der die Prozesse ausgeführt werden
- (IT-) Infrastruktur und Ressourcen, unter deren Zuhilfenahme die Prozesse ausgeführt werden

Abbildung 1-1 veranschaulicht diese Sichtweise.

Von solchen Workflow-Management-Systemen wird ein hoher Grad an Flexibilität gefordert. Das WfMS soll die Möglichkeit bieten, jegliche Organisationsstruktur und jegliche Geschäftsprozesse abzubilden. Berücksichtigt werden sollen außerdem auch die vielfältigen Abweichungen von definierten Prozessen und Änderungen von Organisationsstrukturen. Als Beispiel zu nennen sind hier Abweichungen von Standardverfahren bei Notfallsituationen im klinischen Bereich oder bei Nichtverfügbarkeit von benötigten Ressourcen im Allgemeinen. Schließlich soll die Einführung eines solchen Systems nicht als Einschränkung empfunden werden.

Weiterhin ist nicht absehbar, welche Mengen von verschiedenen Prozessen und wieviele zeitgleich ablaufenden Prozessen ein WfMS zu bewältigen hat. Die Zahl der Benutzer eines

1 [HaCh96], [GrKe98]

solchen Systems oder der Grad an Dynamik einer Organisationsstruktur, die auf ein WfMS abgebildet werden soll, läßt erahnen, daß die Anforderung an die Skalierbarkeit eines solchen Systems eine hohe Priorität hat. Bei vielen dieser Aspekte steht die Forschung erst am Anfang, und so bietet der Bereich der Workflow-Management-Systeme noch viele weitere Betätigungsfelder.

1.2 Organisatorische Aspekte

Wie schon angedeutet, sind die beiden Bereiche Prozeßunterstützung und Organisationsstruktur-Management für ein WfMS von besonderer Wichtigkeit. Damit bildet ein WfMS sowohl die Ablauforganisation ab, welche die räumliche und zeitliche Reihenfolge zusammenhängender Prozeßschritte regelt als auch die Aufbauorganisation, die eine klare Verteilung und Abgrenzung der betrieblichen Aufgaben, Zuständigkeiten und Verantwortung beschreibt.

Durch die prozeßunterstützenden Komponenten ist das System in der Lage, Vorlagen von Geschäftsprozessen zu instantiieren und auszuführen. Kommt es bei der Ausführung einer Prozeßinstanz zur Aktivierung eines Prozeßschrittes, ermittelt das WfMS die für diesen Schritt vorgesehenen Bearbeiter.

Die Informationen, welche Mitarbeiter sich für die Bearbeitung einer Aufgabe qualifizieren, sind an den jeweiligen Prozeßschritten als Zuordnungsanweisung hinterlegt. Diese Anweisungen referenzieren ein konkretes Organisationsmodell (Org.Modell). Zur Laufzeit der Prozeßinstanz werden die konkreten Bearbeiter einer Aufgabe durch die Org.Modell-Komponente aus der Zuordnungsanweisung ermittelt.
Grundlage für diese Entscheidung ist ein Organisationsmodell, das die Struktur der Organisation beschreibt. In diesem Modell sind die organisatorische Struktur (Abteilungen, Projekte, Fähigkeiten und Rollen von Mitarbeitern...) und die organisatorische Population (Mitarbeiter und Rechnerprozesse) entsprechend ihrer Position in der Organisation abgebildet. Hinzu kommen die Beziehungen, die diese Typen untereinander verbinden.
Beispiele für diese Beziehungen sind die Vorgesetztenverhältnisse von Mitarbeitern, Strukturierung von Abteilungen in Unterabteilungen und Projekten in Teilprojekte, Stellenbesetzungen durch Mitarbeiter usw.

Für die Zuordnung von Aufgaben zu Mitarbeitern sollen aber nicht nur allein Informationen aus dem Org.Modell herangezogen werden können, sondern auch Informationen aus dem bisherigen Verlauf des Prozesses. Ein Beispiel ist, daß ein Prozeßschritt nicht vom selben Vorgänger bearbeitet werden darf, wie ein bestimmter vorangegangener Prozeßschritt. Damit läßt sich auch ein sogenanntes Vier-Augen-Prinzip im System abbilden. In diesem Punkt unterscheidet sich das zu entwickelnde Konzept von den meisten verfügbaren Systemen.

Da sich die Struktur eines Unternehmens in ständigem Wandel befindet, sollte ein geeignetes Org.Modell alle möglichen Arten von Änderungen an der Organisation nachvollziehen können. Unter diese Änderungen fallen einfache Änderungen wie Einstellungen, Entlassungen und Versetzungen von Mitarbeitern bis hin zu komplexen wie der Fusion von Konzernen. Dabei soll die Integrität und die Konsistenz des Datenbestandes des Org.Modells jederzeit gewährleistet sein. Selbst laufende Prozeßinstanzen sollen durch Änderungen am Org.Modell nicht in ihrer Ausführung beeinträchtigt werden. Auch diese Fähigkeit findet sich in keinem der bis jetzt verfügbaren Systeme.

1.3 Zielsetzung und Aufgabenstellung der Diplomarbeit

Ziel dieser Arbeit ist die Konzeption einer Org.Modell-Komponente für den zweiten Prototypen des ADEPT Workflow-Management-Systems der Universität Ulm (Abteilung DBIS) [ADEPT]. Zu einzelnen Aspekten des Organisationsmanagements existieren bereits Arbeiten, deren Ergebnisse geprüft und in die Gesamtkonzeption mit eingehen sollen.
Dazu sind die zum Teil schon existierenden Konzepte mit neu zu gewinnenden Erkenntnissen in einem Entwurf für eine konkrete Implementierung in Software zu vereinen. Hierzu gehört auch der Entwurf einer graphischen Benutzerschnittstelle.

Dabei sind folgende Aspekte zu behandeln:

- Organisationsstrukturmodell (bzw. Organisationsstrukturmetamodell)
- Änderungsoperationen des Modells
- Bestimmung von Aufgabenträgern
- Auflösung von Zuordnungsanweisungen
- Anpassungen von Zuordnungsanweisungen nach Änderungen am Organisationsmodell
- Intelligente Unterstützung bei der Modellierung von Zuordnungsanweisungen

Zur Speicherung des Datenbestandes ihrer Organisationsstruktur kommt in vielen Unternehmen ein X.500 kompatibler Verzeichnisdienst wie LDAP zum Einsatz. Im Rahmen dieser Arbeit soll auch geprüft werden, ob sich ein Verzeichnisdienst für die Datenhaltung der Org.Modell-Komponente eignet. Durch eine Integration oder Mitbenutzung eines vorhandenen Verzeichnisses könnte somit vermieden werden, daß bei der Installation eines WfMS die komplette Organisationsstruktur neu auf das Datenmodell der Org.Modell-Komponente abgebildet werden muß. Um die Entscheidung über die Eignung eines Verzeichnisdienstes für dieses Vorhaben fällen zu können, sollen die grundlegenden Eigenschaften eines Verzeichnisdienstes beschrieben werden.

1.4 Aufbau der Arbeit

Diese Arbeit gliedert sich in folgende grundlegende Teile: In Kapitel 2 werden die allgemeinen Anforderungen formuliert, denen die Org.Modell-Komponente genügen soll. Dazu wird auch eine Betrachtung existierender Systeme vorgenommen. An dieses Kapitel schließt sich das Kapitel –Organisationsstrukturmetamodell– an, in dem ein ausdrucksmächtiges Metamodell für die Organisationsstruktur entwickelt wird. Kapitel 4 beschreibt die auf dem Metamodell angebotenen Möglichkeiten zur Manipulation der Organisationsstruktur. In Kapitel 5 wird die Zuordnung von Aufgaben zu einer Menge von potentiellen Bearbeitern behandelt. Kapitel 6 beschäftigt sich mit den Auswirkungen von Änderungsoperationen auf die Bearbeiterzuordnungen. Implementierungsaspekte, wie die Wahl eines geeigneten Datenhaltungssystems (LDAP vs. RDBMS) und der Entwurf der Org.Modell-Komponente finden sich in Kapitel 7. Weitergehende Aspekte, die zukünftig weiterverfolgt werden können, sind in Kapitel 8 aufgeführt. Kapitel 9 bietet abschließend eine Zusammenfassung der Resultate.

2 Anforderungen an die Organisationsmodell-Komponente

In diesem Kapitel werden die Anforderungen an eine Organisationsmodell-Komponente beschrieben. Dabei werden die Anforderungen in allgemeiner Form durch Aufgaben beschrieben, die eine Org.Modell-Komponente zu erfüllen hat.
Im Anschluß werden die Organisationsmodelle existierender Systeme unter verschiedenen Gesichtspunkten diskutiert. Zuletzt werden aus dieser Diskussion die konkreten Anforderungen an die Org.Modell-Komponente hergeleitet.

Die Org.Modell-Komponente verwaltet den einer Organisation bezüglich ihrer Struktur zugrundeliegenden Datenbestand. Damit ist auch die Aufrechterhaltung der Konsistenzbedingungen mit eingeschlossen. Die geforderten Konsistenzregeln des Modells werden definiert durch die Festlegung des Metamodells einer Organisation (Organisationsmetamodell).

Da es sich bei der Org.Modell-Komponente nicht um einen Kernteil des Workflow-Management-Systems, sondern um ein unterstützende Komponente handelt, die mit dem WfMS interagiert, bietet es sich an, diese möglichst unabhängig vom WfMS zu konzipieren. Dadurch ist es möglich, das Org.Modell als eigenen Server zu betreiben. Der Zugriff auf den Datenbestand des Org.Modells – insbesondere der schreibende Zugriff – sollte auf die Org.Modell-Komponente beschränkt sein. Diese Anforderung ergibt sich als Folge der Forderung, daß die Org.Modell-Komponente die Konsistenz und Integrität der Daten sicherstellen soll. Die Komponente sollte aus diesem Grund Schnittstellen zur Verfügung stellen, über die andere Komponenten auf dem Datenbestand operieren können.

Da ein Organisationsmodell die Aufbaustruktur einer Organisation abbildet und diese ständig Änderungen unterworfen ist, soll die Org.Modell-Komponente fähig sein, diese Änderungen nachzuvollziehen. Für den manipulativen Zugriff exportiert das Datenmodell dazu einen minimalen Satz an Änderungsoperationen. Änderungsoperationen sind dabei das Mittel, um jegliche Änderungen am Organisationsmodell vorzunehmen, von der Einstellung und Entlassung einzelner Mitarbeiter bis hin zu Änderungen, die die komplette Struktur einer Organisation betreffen. Diese Änderungsoperationen gehören dann auch zu den nach außen angebotenen Operationen der Org.Modell-Komponente.

Eine weitere Aufgabe der Org.Modell-Komponente ist die Auflösung von Bearbeiterformeln in die resultierende Menge potentieller Bearbeiter (Policy Resolution). Dabei ist das Org.Modell für die Erstellung der Bearbeiterformeln selbst als auch für deren Auflösung verantwortlich. Die Bearbeiterformeln werden bei den Aktivitätenvorlagen oder deren Instanzen, den Aktivitäten, hinterlegt.

Im Rahmen von Änderungsoperationen kommt es unter bestimmten Bedingungen zu Situationen, in denen Bearbeiterformeln nicht mehr gültig sind, bzw. nicht mehr der ursprünglichen Intention entsprechen. In diesem Fall muß die Org.Modell-Komponente Anpassungen an den Bearbeiterformeln entweder automatisch oder mit Hilfe des Modellierers vornehmen können.

Noch eine weitere wichtige Aufgabe fällt der Org.Modell-Komponente im Rahmen der Bearbeiterformeln zu. Das Gesamtsystem soll den Modellierer eines Workflows auch bei der Festlegung von Bearbeiterzuordnungen möglichst intelligent unterstützen. Ein Tool zur Modellierung von Bearbeiterzuordnungen sollte von vorne herein ausschließen können, daß syntaktisch nicht korrekte Formeln entstehen. Auch ist es bis zu einem gewissen Grad möglich,

unsinnige Semantik zu verhindern. Dazu muß die Modellierungskomponente sowohl von den Daten des Org.Modells selbst als auch von deren Struktur Kenntnis haben. Die Org.Modell-Komponente muß also auch für diese Anforderung eine entsprechende Schnittstelle anbieten. Denkbar ist auch, die Unterstützung bei der Definition von Bearbeiterzuordnungen in die Org.Modell-Komponente selbst zu integrieren. Auch diese Überlegung ist ein Thema dieser Arbeit.

2.1 Betrachtung existierender Lösungen

Im folgenden werden Organisationsmodelle von verfügbaren WfMS kurz betrachtet, um sich ein Bild von deren Ausdrucksmächtigkeit verschaffen zu können. Dazu werden die Modelle unter sechs Gesichtspunkten gruppiert. Da von allen Modellen eine gewisse Grundfunktionalität angenommen wird, werden hier nur die Besonderheiten einzelner Systeme erwähnt. Diese Vergleiche finden sich in ausführlicherer Form in [Buss98]. Zu den dort aufgeführten Systemen kommen in der vorliegenden Arbeit noch die Systeme *ADEPT*, *Staffware* und *Websphere MQ Workflow,* ein Nachfolger des früheren *FlowMark,* hinzu.

Um im folgenden den Lesefluß zu erleichtern, wird die Literatur, die für die erwähnten Systeme relevant ist, en bloc angegeben:

ADEPT	[ADEPT], [Spar01]
COSA	[Buss96], [COAH94], [COPH94], [CORH04]
DOMINO	[KHK+91], [Krei91], [KrHW93], [KrSW87], [WoKr93]
E.C.H.O	[ECAG92], [VanH93], [Vuur92]
FlowMark	[Buss96], [IBMW95], [IBPG95], [LeAl94], [Zern95]
OfficeTalk	[Cook80], [ElBe82], [Elli83], [ElNu80], [ElWa83]
Oracle Workflow	[OrWG96]
ORM	[Buss96], [ORGR94], [ORRE94], [Rupi92], [Rupi94]
Regatta	[SIM+94a], [SIM+94b], [SMM+94], [Swen93a], [Swen93b], [Swen93c]
Staffware	[Staff99]
MQ Workflow	[MQWF04]

2.1.1 Geltungsbereich der Organisationsstruktur

Eine Organisationsstruktur kann global für alle Prozesse Gültigkeit besitzen oder auch nur für einen einzelnen Prozeß.
Für die meisten WfMS gilt aber, daß für alle laufenden Instanzen ein einziges Organisationsmodell zur Verfügung steht. Damit ist ein Organisationsmodell global zu allen Workflows. *Regatta* ermöglicht für Workflowinstanzen die Definition von Rollen, die nur für die jeweilige Instanz sichtbar sind. Beim Start einer Instanz wird diese Rolle durch eine Menge von Personen ersetzt. Somit ist zumindest in Bezug auf die Rollen die Organisationsstruktur workflowlokal. Auch *E.C.H.O* und *DOMINO* nutzen ein ähnliches Konzept.

2.1.2 Art der Organisationsstruktur

Für eine Organisationsstruktur können verschiedene Modellierungselemente zur Verfügung stehen. In diesem Zusammenhang wird geprüft, in welchem Umfang Elemente wie Stelle, Rolle, Fähigkeiten usw. vorhanden sind.

Der Umfang des Schemas eines Organisationsmetamodells ist maßgebend für dessen Ausdrucksmächtigkeit. Keines der verfügbaren System bietet nur personenbezogene Bearbeiterzuordnung an. Mindestens die Entitäten Rolle und Stelle finden sich bei den allermeisten WfMS. Damit wird eine dynamische Zuordnung möglich, die auch nach Änderungen wie Einstellungen und Entlassungen nicht extra angepaßt werden muß.
OfficeTalk und *Oracle Workflow* als Vertreter der rollenbasierten Regeln kommen dabei mit den drei Relationen *actor*, *role* und *player* aus. *Regatta* kennt dagegen nur das Konzept der workflowlokalen Rollen und verfügt somit über eine geringere Ausdrucksmächtigkeit.

Staffware bietet an, die Organisation in Gruppen zu gliedern. Bearbeiter können in beliebigen Gruppen Mitglied sein und beliebige Rollen ausführen. Ein umfangreicheres Schema bezüglich der Organisationsstruktur besitzt dagegen *COSA*. Hier können Mitarbeiter zu Gruppen eines bestimmten Typs zusammengefaßt werden. Auch die Modellierung von Vorgesetztenverhältnissen sowie von Vertreterregelungen ist hier möglich. Bei der Bearbeiterzuordnung wird dabei zuerst eine Auswahl auf Ebene der Gruppen durchgeführt und danach auf der Ebene einzelner Personen in diesen Gruppen.

DOMINO bietet die Entitäten Organisationseinheit, Projekte, Funktion und Mitarbeiter, mit denen sich Hierarchien abbilden lassen.
Ein Schema mit Modellierungselementen wie Person, Ebene, Rolle und Organisationseinheit bietet *WebSphere MQ Workflow*[2] von IBM. Auch hier lassen sich Hierarchien definieren und Aufgaben durch Rollen indirekt auf Personen verteilen. Allerdings sind im System dazu schon bestimmte Rollen wie *Coordinator*, *System Administrator* und *Manager* fest vorgegeben. Auch ADEPT bietet ein reichhaltiges Repertoire von Modellierungsaspekten für das Organisationsmodell. Hier stehen die Elemente *Organisationsgruppe*, *Organisationseinheit*, *Arbeitsgruppe*, *Rolle*, *Stelle*, *Fähigkeit*, *Mitarbeiter*, *Vertreterregelung* und *Aufgabenkategorie* zur Verfügung, die sich auch alle durch Bearbeiterzuordnung referenzieren lassen.

2.1.3 Dynamische Fähigkeiten der Organisationsstrukturverwaltung

Eine Organisation ist einem ständigen Wandel unterworfen. Einfache Änderungen sind die Anpassung von personenbezogenen Daten. Über Entlassungen und Einstellungen von Mitarbeitern bis hin zur kompletten Umstrukturierung wirken auch komplexe Änderungen auf das Modell ein. Es ist also eine wichtige Eigenschaft eines Organisationsmodells, ob es Operationen anbietet, um diese Einflüsse nachzuvollziehen. Die Eigenschaft der Anpassungsfähigkeit läßt sich noch weiter verfeinern nach dem Kriterium, ob diese Änderungen nur zur Definitionszeit oder auch zur Laufzeit des Systems möglich sind und ob dabei nur Änderungen an vorhandenen Ausprägungen möglich sind (Änderungen von Attributwerten) oder auch am Schema (Hinzufügen / Löschen einer Organisationseinheit).

Änderungen an vorhandenen Ausprägungen sind bei allen schon genannten Systemen möglich.

Regatta läßt generell alle Änderungen an der Organisationsstruktur zu, da der Kontext des Organisationsmodells sich nur auf eine Instanz bezieht (Organisationsmodell ist workflowlokal).

Teilweise lassen die Systeme zur Laufzeit nur solche Änderungen zu, die die Ausführung der laufenden Prozesse nicht beeinträchtigen. So sind meist nur Operationen erlaubt, die neue

2 Früher bekannt unter dem Namen IBM MQSeries Workflow

Attribute oder Instanzen von Entitäten hinzufügen, aber keine Löschoperationen. Bei *FlowMark* sind zwar auch Löschoperationen zur Laufzeit beschrieben, allerdings wird von deren Anwendung abgeraten, falls sich diese auf Ausprägungen beziehen, die von laufenden Prozessen referenziert werden.
Überlegungen zu kontrollierten Änderungen am Organisationsmodell wurden auch um Rahmen des *ADEPT*-Projekts angestellt. Die Erkenntnisse dieser Untersuchung sind aber bis jetzt noch nicht implementiert.

Für alle bekannten Systeme gilt, daß sie keine Versionierung über die durch die Operationen entstehenden Versionen des Organisationsmodells anbieten.

2.1.4 Eigenständigkeit der Organisationsstrukturverwaltung

Die Eigenständigkeit der Organisationsstrukturverwaltung betrifft die Fähigkeit der Systeme, Zugang zu den verwalteten Daten sowie Änderungen derselben nur einer Gruppe von autorisierten Benutzern zu erlauben. Diese Forderung ist nicht unerheblich, da der Org.Modell-Komponente je nach Ausdrucksmächtigkeit auch die gesamte Personalstammdatenverwaltung unterliegen kann. Unter Eigenständigkeit ist dabei zu verstehen, daß ein System für sämtliche Zugriffe geschützte Schnittstellen anbietet.

Oracle Workflow beschränkt sich dabei auf die Zugriffsbeschränkungen, die schon durch die Tabellen der zugrundeliegenden Datenbank auferlegt sind. Für *Regatta* entfällt die Diskussion dieses Punktes, da ein Organisationsmodell sich hier nur auf eine Prozeßinstanz bezieht, und die Berechtigungen über die Regelungen für die Prozeßinstanz erfolgen.

WebSphere MQ Workflow erlaubt es, für jeden Benutzer einzeln Rechte für den Zugriff zu erteilen. Mit dieser Erlaubnis kann der Benutzer dann aber auf alle Daten der Organisationsstrukturverwaltung zugreifen. Die Möglichkeit, den Zugriff feingranularer zu verwalten, bietet *MQ Workflow* nicht.

ORM dagegen erlaubt die Formulierung von Zugriffsrechten sogar bis auf Ebene der einzelnen ausführbaren Operationen. *ORM* ist zudem die einzige erhältliche Org.Modell-Komponente, die in einer Form zur Verfügung steht, in der sie unabhängig vom WfMS (*WorkParty*) durch andere Applikationen genutzt werden kann[3].

2.1.5 Dokumentation der Semantik

Die Dokumentation der Semantik beschreibt die Ausführlichkeit und Präzision, in der die Dokumentationen der Systeme die implementierte Semantik beschreiben. Dies ist aufgrund der vielen möglichen Deutungen der Beziehungen der Objekte, der Konsistenzregeln und der Bearbeiterzuordnungen auf jeden Fall wünschenswert.
Dieser Teil der Dokumentation sollte, um Mißverständnisse von vorne herein auszuschließen, in Form eines Algorithmus, als logische Prädikate, als SQL-Statement oder ähnliches formal definiert sein. Falls es sich dabei nur um Text handelt, sollte dieser zumindest sehr präzise beschrieben sein.
FlowMark, *WebSphere MQ Workflow*, *COSA*, *E.C.H.O*, *OfficeTalk* und *Oracle Workflow* beschreiben die Semantik ihrer Modelle nur textuell. Für *ADEPT* ist die Semantik in Arbeiten zum Projekt zum Teil textuell, zum Teil formal beschrieben. Bei *ORM* liegen die Definitionen

3 *ORM* (Organisation Ressource Management) ist die Org.Modell-Komponente des WfMS *WorkParty*

der Datenstrukturen und der Schnittstellen formal vor. *DOMINO* beschreibt die Semantik seines Organisationsmodells in Form von Horn-Klauseln.

2.1.6 Autonomie

Das Kriterium der Autonomie beschreibt, inwieweit die Org.Modell-Komponente als Bestandteil des WfMS fest in das Gesamtsystem integriert ist, oder ob die Org.Modell-Komponente als eigenständige Komponente implementiert ist, die gegebenenfalls sogar gegen eine andere ausgetauscht werden kann. Wie bei den meisten Systemen ist auch bei *ADEPT* die Org.Modell-Komponente als fester Bestandteil im WfMS integriert.

Die einzige Org.Modell-Komponente, die komplett vom WfMS getrennt werden kann und gegebenenfalls ausgetauscht bzw. an andere Stelle eingesetzt werden kann, ist *ORM*. Alle anderen Systeme zeigen keinen Hinweis darauf, daß sie bezüglich der Architektur eine derartig strenge Trennung von Organisationsverwaltung und Prozeßverwaltung vornehmen.

2.1.7 Zusammenfassung

In den vorigen Abschnitten wurden Organisationsmodelle verfügbarer Workflow-Management-Systeme unter verschiedenen Aspekten besprochen. Dabei wurden im Text nur die Besonderheiten einiger Systeme erwähnt. Tabelle 2-1 gibt einen Überblick über die ausgewählten Systeme und zeigt für jedes System die Eigenschaften in den angesprochenen Aspekten *Geltungsbereich*, *Art* und *Dynamik* der Organisationsmodelle sowie *Zugriffs-beschränkungen*, *Dokumentation der Semantik* und *Autonomie*.

	Geltungsbereich	Art	Dynamik (Änderungen) Modellierzeit (Schema/Ausprägungen) Laufzeit (Schema/Ausprägungen)	Eigenständigkeit/ Zugriffsbeschränkungen	Dokumentation der Semantik	Autonomie
ADEPT	wf-global	Org.Gruppe, Org.Einheit, Arbeitsgruppe, Rolle, Stelle, Fähigkeit, Mitarbeiter, Vertreterregel Aufgabenkate-gorie	X / X / - / -	keine feingranulare Rechtevergabe	textuell	monolithisch
COSA	wf-global	users, user_groups, group_types, substitution_plan	X / X / - / X	keine feingranulare Rechtevergabe	textuell	monolithisch
DOMINO	Org.-Struktur global, Rollen lokal	user, function, group, project,	X / X / - / X	k.A.	Hornklauseln	monolithisch
E.C.H.O	Org.-Struktur global, Rollen lokal	user, group, role, organisation	X / X / - / X	k.A.	textuell	monolithisch
FlowMark / MQ Workflow	wf-global	person, level, role, orgunit	X / X / - / X	keine feingranulare Rechtevergabe	textuell	monolithisch
OfficeTalk	wf-global	actor, role, player	X / X / - / X	k.A.	textuell	monolithisch
Oracle Workflow	wf-global	actor, role, player	X / X / - / X	zugrunde-liegende DB	textuell	monolithisch
ORM (Work Party)	wf-global	person, level, role, orgunit	X / X / - / X	Rechtevergabe für Änderungen an Schema und Ausprägung der Organisations-struktur	textuell Daten-strukturen + Schnittstellen formal beschrieben	standalone
Regatta	wf-lokal	role	X / X / - / X	entfällt, da wf-lokal	textuell	monolithisch
Staffware	wf-global	user, group, role	X / X / - / X	keine feingranulare Rechtevergabe	textuell	monolithisch

Tabelle 2-1: *Vergleich der Organisationsmodelle verfügbarer Systeme*

2.2 Anforderungen im Detail

Das Organisationsmetamodell stellt die Grundlage dar, auf welcher eine Organisation modelliert werden kann. Dieses Metamodell definiert gewissermaßen die Struktur des Organisationsmodells und die damit verbundenen Ausdrucksmöglichkeiten.
Die Anforderungen, die an das Organisationsmetamodell gestellt werden, haben ihren Ursprung zum einen in den Einschränkungen der Modelle existierender Systeme (siehe Abschnitt 2.1), zum anderen sollen die Fähigkeiten des Metamodells den Anwendungsfällen aus [Kony96] gerecht werden. [Kony96] beschreibt Abläufe und Organisationsstrukturen der Universitäts-Frauenklinik Ulm, und formuliert auf dieser Basis Anforderungen an ein Organisationsmodell.
Die Kategorisierung der Anforderung erfolgt nach denselben Gesichtspunkten, nach denen auch die existierenden Systeme in Abschnitt 2.1 beurteilt wurden.

Geltungsbereich der Organisationsstruktur

- Die Organisationsstruktur soll global für das gesamte System zur Verfügung stehen. Private Sichten für einzelne Prozeßinstanzen oder Benutzer werden nicht gefordert.

Art der Organisationsstruktur

- Hierarchische Strukturierung der Organisation in Organisationseinheiten (Geschäftsbereich, Abteilungen)
- Hierarchische Strukturen bezüglich temporärer Zusammenschlüsse (Projekte)
- Festlegung von Vorgesetztenverhältnissen (fachlich und disziplinarisch)
- Stellenbesetzungen durch Mitarbeiter
- Beschreibung von Rollen und Mitarbeitern durch Fähigkeiten
- Vertreterregelungen
- Branchenspezifische Anpassung des Organisationsmodells durch dynamisch erweiterbare Attributmengen

Mit den Mitteln eines adäquaten Organisationsmetamodells sollte sich also jegliche formale Aufbauorganisation modellieren lassen, die sich in einem reellen Unternehmen findet. Das Metamodell sollte dabei aber so flexibel sein, dem Modellierer nicht die Nutzung aller Gestaltungsmöglichkeiten aufzudrängen. Der Modellierer sollte zu jedem Zeitpunkt frei wählen können, welche Strukturen und Beziehungen in sein konkretes Modell Eingang finden.

Dynamische Fähigkeiten der Organisationsstrukturverwaltung

- Operationen für die Manipulation des Organisationsmodells unter Einhaltung der Konsistenzbedingungen müssen verfügbar sein.

- Alle Aspekte des Organisationsmodells sollen durch Änderungsoperationen veränderbar sein. Das betrifft sowohl Änderungen der Ausprägungen von organisatorischen Entitäten (Bsp.: „Mitarbeiter ändern / hinzufügen / löschen“), als auch Änderungen auf Typebene (Bsp.: „Attribut 'Wohnort' von Mitarbeiter löschen“).

- Änderungen sollen zur Definitionszeit und zur Laufzeit des Systems möglich sein. (Als Laufzeit wird in diesem Zusammenhang der Systemzustand genannt, in dem sich eine Prozeßinstanz in der Ausführung befindet und sich auf dieses Organisationsmodell bezieht.)

Eigenständigkeit der Organisationsstrukturverwaltung

- Die Org.Modell-Komponente soll über ein API verfügen, mit dem die komplette Funktionalität der Komponente genutzt werden kann.

- Die Org.Modell-Komponente soll keine Abhängigkeiten zu anderen Teilen des Systems besitzen.

- Zugriff auf Daten des Organisationsmodells soll nur nach Autorisierung erfolgen. Die Festlegung der Zugriffsrechte soll für jeden Benutzer und jedes Datum feingranular möglich sein.

Dokumentation der Semantik

- Um Mißverständnisse bezüglich des Verhaltens des Systems auszuschließen, soll die Semantik des Modells möglichst exakt beschrieben sein. Das gilt für das Organisationsstrukturmodell und dessen Integritätsbedingungen, aber auch für die Operationen, die auf diesem Modell definiert sind, sowie für die Bestimmung von Aufgabenträgern.

Architektur

- Die Schnittstellen der Org.Modell-Komponente sollten so genau beschrieben sein, daß es möglich ist, die Komponente für ein beliebiges WfMS einzusetzen bzw. auch das Org.Modell an anderer Stelle nutzen zu können.

In [Spar01] ist ein Modell beschrieben, das als Grundlage für die Anforderungen dient. Die Konzepte aus [Spar01] bezüglich eines Organisationsmetamodells finden sich weitgehend in der Kategorie *„Art der Organisationsstruktur*“ wieder. Die Entität *Organisationstyp* wird aber nicht mit in die Anforderungen übernommen, da es sich bei dieser Entität um eine Typdefinition für eine andere im Modell befindliche Entität handelt (*Organisationseinheit*). Organisationstyp steht damit sprachlich auf einer anderen Ebene als die anderen Entitäten des Modells und sollte somit nicht im selben Modell untergebracht sein. *Organisationstyp* als Kategorisierung für *Organisationseinheit* hat zudem nicht die Wichtigkeit, als selbständige Entität modelliert werden zu müssen. Wird dies in einem Organisationsmodell gewünscht, kann die Beziehung *„Instanz von“*, die die beiden Entitäten verbindet, auch durch Attribute von *Organisationseinheit* abgebildet werden.

Weiterhin scheint auch die Beziehung „*nie über*“, welche für *Organisationstyp* definiert ist, zu restriktiv für ein Organisationsmodell. Diese Beziehung drückt aus, daß eine Instanz eines

Organisationstyps (Organisationseinheiten) sich hierarchisch nie über einer Ausprägung eines bestimmten anderen Organisationstyps befinden darf. Ähnliches gilt für die Einschränkung, daß sich eine Rolle nur in Ausprägungen eines bestimmten Organisationstyps finden kann.

Stellen gehören in diesem Organisationsmetamodell genau einer Organisationseinheit an. Es besteht keine Möglichkeit, Stellen auf andere Art zu gruppieren. Deshalb wird die Forderung nach Projektzusammenschlüssen mitaufgenommen. Stellen sollen sich zwar in genau einer Organisationseinheit befinden, aber an beliebig vielen Projekten teilnehmen können. Daher gilt auch die Aussage aus [Spar01] „Die disziplinarische Über- bzw. Unterordnung schließt die fachliche Über- bzw. Unterordnung mit ein“[4] nicht mehr.

Aufbauend auf diesen Anforderungen wird in den folgenden Kapiteln zuerst ein adäquates Organisationsmetamodell und die darin angewandten Konzepte hergeleitet und schließlich ein konkreter Entwurf für die Org.Modell-Komponente entwickelt.

4 [Spar01] S.78

3 Organisationsstrukturmetamodell

Die Überlegungen zum Organisationsmanagement im Kontext eines WfMS teilen sich grob in zwei Aufgabenteile. Zum einen muß in einem Modell die Struktur der Organisation in geeigneter Art und Weise abzulegen sein. Dadurch werden die Objekte der Organisation – also die organisatorische Struktur und die organisatorische Population – formal abgebildet. Das betrifft sowohl die Daten, die diese mit sich bringen als auch die Beziehungen, die diese untereinander aufweisen. Die zweite wichtige Aufgabe des Organisationsmanagements ist die Bestimmung von Aufgabenträgern aus diesem Modell. Dies wird in Kapitel 5 behandelt. Zunächst wird aber näher auf die Modellierung der Organisationsstruktur selbst eingegangen.

3.1 Modellierung des Metamodells

Da alle Organisationsmetamodelle letztendlich denselben Zweck haben, ist es naheliegend, daß diese sich in gewisser Weise ähneln. Das in dieser Arbeit vorgestellte Metametamodell zeigt an einigen Stellen Ähnlichkeiten insbesondere zu den in [Wied02] und [Kubi98] gezeigten Modellen. Unterschiede zu diesen werden in Abschnitt 3.2 erläutert und begründet.

Das im folgenden beschriebene Metamodell ist dargestellt mit den Mitteln eines erweiterten Entity-Relationship Diagramms (EER-Diagramm). Damit können zwar nicht alle Konsistenzbedingungen beschrieben werden, die an das Metamodell gestellt werden, dafür ist das EER-Diagramm aber sehr gut lesbar und einfach zu verstehen. Die Notwendigkeit, zusätzliche textuelle Anmerkungen am Diagramm oder im Text anzubringen, wird deshalb bewußt in Kauf genommen.

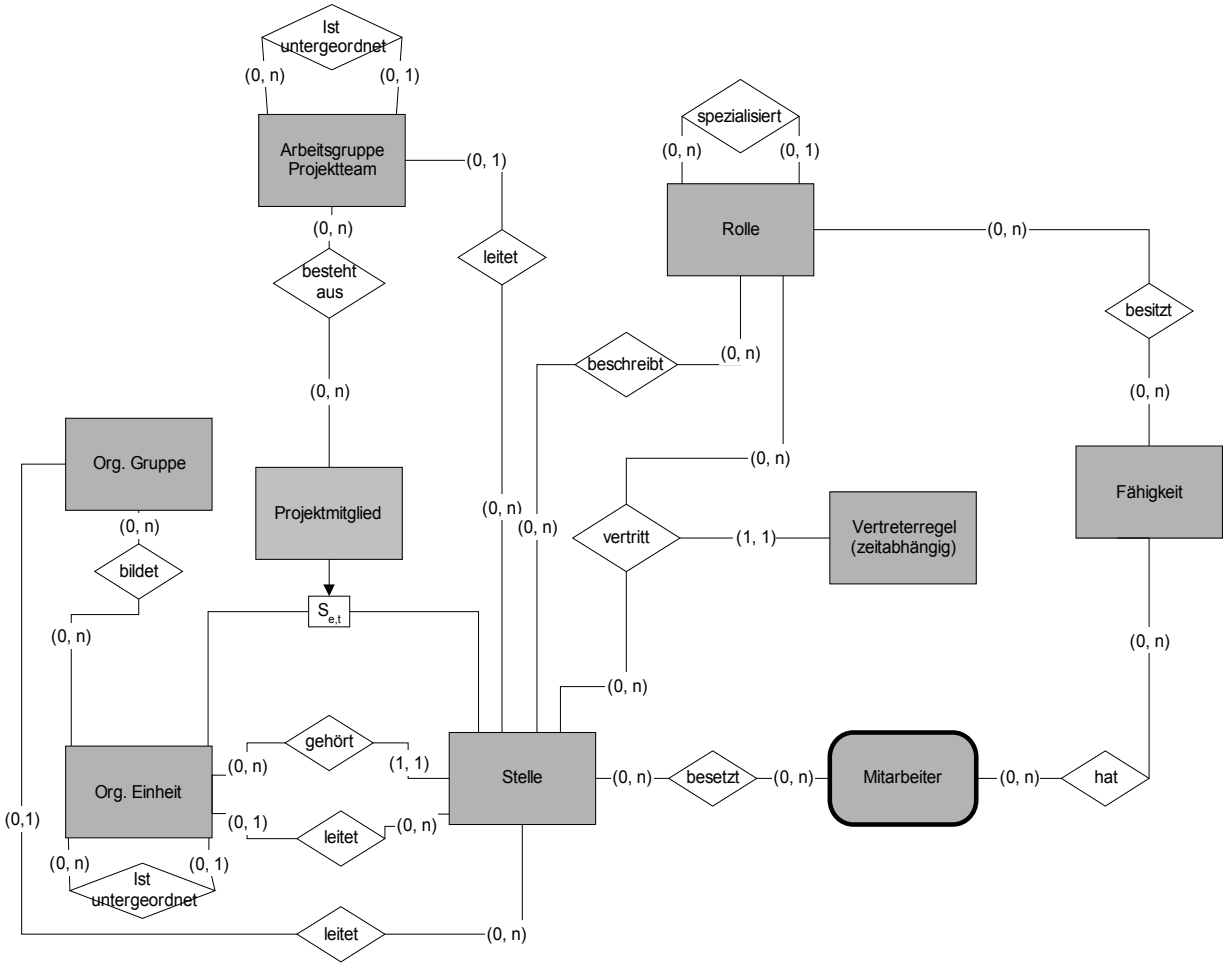


Abbildung 3-1: *Organisationsmetamodell*

Dreh- und Angelpunkt des Metamodells bildet die Entität *Mitarbeiter*. Um sie herum sind die Entitäten angeordnet, mit denen sich die Aufbaustruktur der Organisation beschreiben läßt.

Denkbar ist, daß dieses Metamodell nur den Kern eines umfassenderen Metamodells darstellt. In diesem Metamodell könnte außer der Organisationsstruktur noch das Ressourcenmanagement (Betriebsmittel, Raum, Merkmal) und die Rechteverwaltung (Berechtigung, Besitzer, Berechtigte) angesiedelt sein. Wie diese Aspekte in das Metamodell eingegliedert werden können, ist der Abbildung in Anhang B zu entnehmen. Eine nähere Betrachtung dazu würde allerdings den Rahmen dieser Arbeit sprengen.

Wie schon angedeutet hat die Entität *Mitarbeiter* eine zentrale Stellung in diesem Metamodell. Bei der Auflösung jeglicher Bearbeiterformeln wird letztendlich eine Teilmengenbildung auf der Menge der Mitarbeiter ausgeführt.

Da Mitarbeiter Stellen besetzten und die Stelle jeweils genau einer Organisationseinheit angehören, ist die Existenz mindestens einer Organisationseinheit im Modell zwingend. Die drei genannten Entitäten (*Organisationseinheit*, *Stelle* und *Mitarbeiter*) bilden den obligatorischen Teil des Modells.

Im einfachsten Fall kann allein mit diesen drei Entitäten eine Organisationsstruktur aufgebaut werden, mit der das Workflow-Management-System in Betrieb genommen werden kann. Für

kleine und überschaubare Organisationen kann dies durchaus schon ausreichen. Werden weitere Möglichkeiten gewünscht, die Modellierung der Organisationsstruktur um weitere Aspekte zu erweitern, können nach Bedarf die anderen optionalen Entitäten zur Modellierung mit hinzugezogen werden. Mit der Einbeziehung einer oder mehrerer optionaler Entitäten wächst natürlich auch die Möglichkeit, die durch die Bearbeiterformeln definierten Bearbeitermengen nach anderen Kriterien als nur nach der Besetzung von Stellen festzulegen.

Nachfolgend werden die im Ausgangsmodell vorhandenen Entitäten mit den ihnen zugehörigen Relationen beschrieben. Die Attribute sind, sofern diese hier angegeben sind, nur am Rande berücksichtigt, und deren Aufzählung ist keinesfalls vollständig; auf diese wird an anderer Stelle detailliert eingegangen (siehe Abschnitt 3.5).

Mitarbeiter
Ein Mitarbeiter stellt aus Sicht des WfMS den elementarsten Teil der Organisationsstruktur dar. Ein Mitarbeiter ist die einzige Entität des Metamodells, welche in der realen Welt ein Pendant besitzt; eine natürliche Person. Auch eine Maschine oder ein Prozeß auf einem Rechner können als Mitarbeiter aufgefaßt werden, wenn man den Begriff *Mitarbeiter* erweitert auf „*potentieller Aufgabenträger*“. Alle anderen Entitäten sind abstrakte Begriffe, die dafür umso gründlicher definiert werden müssen, um Mißverständnissen vorzubeugen.

Definition 3-1: Mitarbeiter

Ein Mitarbeiter ist die Abbildung einer natürlichen Person, einer Maschine oder eines Rechnerprozesses in die Aufbaustruktur einer Organisation.
Er besetzt keine oder auch beliebig viele Stellen.
Ein Mitarbeiter besitzt keine oder auch beliebig viele Fähigkeiten.

Ein Mitarbeiter führt die ihm zugewiesenen Aktivitäten aus, führt Änderungsoperationen jeglicher Art aus und ist somit die einzige Entität, welche mit dem WfMS interagiert.

Stelle
Durch die Stelle wird vom Mitarbeiter als konkretem Aufgabenträger in der Organisation abstrahiert. Das bietet die Möglichkeit, Aufgaben durch Bearbeiterformeln nicht nur den Mitarbeitern direkt zuzuweisen, sondern den Stellen, welche durch sie besetzt sind. Vorteilhaft ist das insbesondere, da Stellen in ihrer Position in der Aufbauorganisation wesentlich stabiler sind als Mitarbeiter. Man denke an Beförderungen, Versetzungen usw.

Definition 3-2: Stelle

Die Stelle ist die Summe von Aufgabenkategorien und Rollen, die dauerhaft zusammengefasst sind und von einem dafür qualifizierten Mitarbeiter unter normalen Umständen ausgeführt werden können. Eine Stelle kann dabei von keinem oder auch von beliebig vielen Mitarbeitern besetzt sein, und das sowohl gleichzeitig (Job Sharing) als auch zeitlich nicht überschneidend (Schichtarbeit).

Eine Stelle beschreibt somit die Schnittstelle zwischen den abstrakten Entitäten der organisatorischen Struktur und der realen Entität der organisatorischen Population (Mitarbeiter)[5].

5 Vgl. [Jab95]

Folgende zusätzliche Merkmale charakterisieren eine Stelle. Eine Stelle:

- ...wird beschrieben durch keine oder beliebig viele Rollen und ist zuständig für keine oder beliebig viele Aufgabenkategorien.

- ...hat jeweils keine oder genau eine andere Stelle als disziplinarischen Vorgesetzten und keinen oder beliebig viele fachliche Vorgesetzte. Das erklärt sich dadurch, daß eine Stelle genau einer Org.Einheit angehört und an beliebig vielen Projekten beteiligt sein kann. Dadurch ergibt sich die Möglichkeit, Bäume zu erzeugen, welche die Hierarchie der disziplinarischen (ein Baum) und fachlichen (ein Wald) Vorgesetztenbeziehungen in der Organisation widerspiegeln.

- ...ist genau in eine organisatorische Einheit eingegliedert, und kann keine oder auch beliebig viele organisatorische Einheiten leiten.

- Ist eine Stelle vorübergehend nicht besetzt (z.B. Urlaub), so können die ihr zugetragenen Aufgaben von einer anderen Stelle in Vertretung übernommen werden. Die Vertretung ist jeweils für genau eine Rolle der Stelle definiert.

Organisatorische Einheit
Die *Organisatorische Einheit* (*Org.Einheit*) faßt eine Menge von Stellen zusammen. Sie stellen damit die üblicherweise vorhandene Zuordnung von Stellen zu Abteilungen oder Fachbereichen dar. Nicht zu verwechseln ist dies mit der Darstellung von Projektgruppen, die fälschlicherweise oft als temporäre Organisationseinheiten bezeichnet werden. Org.Einheiten unterscheiden sich von Projekten dahingehend, daß eine Stelle nur einer einzigen Org.Einheit angehören kann. Damit ist eine eindeutige Einordung in die disziplinarische Hierarchie gegeben. Dagegen kann eine Stelle an beliebig vielen Projekten teilnehmen, wofür für jedes einzelne Projekt einer Stelle eine Einordung in eine fachliche Vorgesetztenhierarchie definiert ist. Für Projekte stellt dieses Metamodell eigene Beschreibungsmittel bereit.

Definition 3-3: Organisatorische Einheit

Eine Organisatorische Einheit (Org.Einheit) ist das Mittel, um Stellen zu gruppieren. Eine Stelle gehört dabei genau einer Org.Einheit an.
Aufgrund der rekursiven Beziehung der Entität zu sich selbst kann eine Org.Einheit weitere Org.Einheiten oder Stellen beinhalten.
Eine Org.Einheit wird von genau einer Stelle geleitet.

Eine Org.Einheit kann Stellen oder wiederum Org.Einheiten beinhalten. Das ist eine Analogie einem Dateisystem, wo Verzeichnisse Dateien oder Verzeichnissen beinhalten können.

Org.Gruppe
Eine *Organisatorische Gruppe* (*Org.Gruppe*) bietet die Möglichkeit, Org.Einheiten außerhalb gemeinsamer Projektaktivitäten zu gruppieren. Damit wird also z.B. eine projektübergreifende Kooperation von Abteilungen modellierbar, die sich nicht nur auf einzelne gemeinsame Projekte bezieht.

Definition 3-4: Organisatorische Gruppe

Die Organisatorische Gruppe (Org.Gruppe) faßt Org.Einheiten projektübergreifend zusammen. Eine Org.Gruppe kann aus beliebig vielen Org.Einheiten bestehen. Eine Org.Einheit kann dabei an beliebig vielen Org.Gruppen teilhaben.

Beispiel:
Ein konkretes Beispiel für eine Org.Gruppe ist die Zusammenarbeit zwischen der Abteilung Psychiatrie des Universitätsklinikums Ulm und der Abteilung Neuroinformatik der Universität Ulm.

In einer Org.Gruppe finden sich also Org.Einheiten zusammen, welche sich an Randbereichen ihrer Tätigkeit (Forschungsbereich oder Aufgabenbereich) tangieren oder überschneiden. Damit wird eine projektübergreifende interdisziplinäre Kooperation modellierbar.

Projektgruppe
Die Möglichkeit, Stellen unter Projektgruppen zusammenzufassen, erweitert die Ausdrucksmächtigkeit des Metamodells um die Möglichkeit, beliebig viele, voneinander unabhängige Hierarchien zu verwalten. Genau wie Organisationseinheiten können Projektgruppen sowohl Stellen als auch wiederum Projektgruppen beinhalten. Dennoch unterscheiden sich Projektgruppen von Org.Einheiten. Eine Stelle nicht nur einer, sondern beliebig vielen Projektgruppen angehören kann.

Definition 3-5: Projektgruppe

Projektgruppen bieten die Möglichkeit, Stellen oder ganze Org.Einheiten nach beliebigen anderen Kriterien als die des disziplinarischen Vorgesetztenverhältnis in Hierarchien einzuordnen. Diese Hierarchien sind unabhängig von der Hierarchie der Organisationsstruktur, die die disziplinarische Ordnung beschreibt.

Projektgruppen können aufgrund der rekursiven Beziehung zu sich selbst, aus beliebig vielen Teilprojektgruppen bestehen.

Gehört ein Projektmitglied einer Projektgruppe PG_a an, dann gehört sie auch allen Projektgruppen PG_b an, für die gilt, daß PG_a eine Teilprojektgruppe von PG_b ist. Das gilt auch transitiv.

Bei der Beschreibung der Org.Einheiten wurde die Analogie zu Dateisystemen angesprochen. Auch hier ist das nicht falsch. Das mehrfache Vorkommen von Stellen in verschiedenen Projekten ähnelt Dateien, die mittels HardLink in mehreren Verzeichnissen zugleich stehen (vgl. Ext2 – Extended File System[6]).

Projektmitglied
Die Entität Projektmitglied ist keine echte Entität. Sie dient lediglich als abstrakte Entität für die Subentitätstypen Org.Einheit und Stelle. Die Zerlegung in die Subentitätstypen ist exklusiv und total. Damit wird beschrieben, daß es keine anderen Projektmitglieder als Org. Einheiten und Stellen gibt und daß deren Schnittmenge leer ist.

6 [Berr00], [WPE00]

Die Tatsache, daß eine Org.Einheit als Projektmitglied an einem Projekt teilnimmt bedeutet, daß alle Org.Einheiten und Stellen, die sich in der disziplinarischen Ordnung unterhalb dieser Org. Einheit befinden, ebenfalls an diesem Projekt teilnehmen.

Rolle
Durch die Rolle wird eine Ansammlung von Fähigkeiten beschrieben. Dies können einfache aber auch komplexe Rollenbeschreibungen sein. Um nicht für jede Rolle sämtliche Fähigkeiten, die sie besitzt, modellieren zu müssen, sind die Rollenbeschreibungen hierarchisch angeordnet.
Ein Wurzelelement der Rollenbeschreibungen kann dabei die Rolle „Person“ sein, von der alle weiteren Rollen abgeleitet sind. Ob allerdings die Hierarchie der Rolle auf einer einzigen Wurzel aufbaut, oder auf beliebig vielen, bleibt der Entscheidung des Modellierers überlassen. Eine Rolle besitzt sowohl sämtliche ihr direkt zugewiesenen Fähigkeiten als auch die Fähigkeiten ihrer Ahnen.

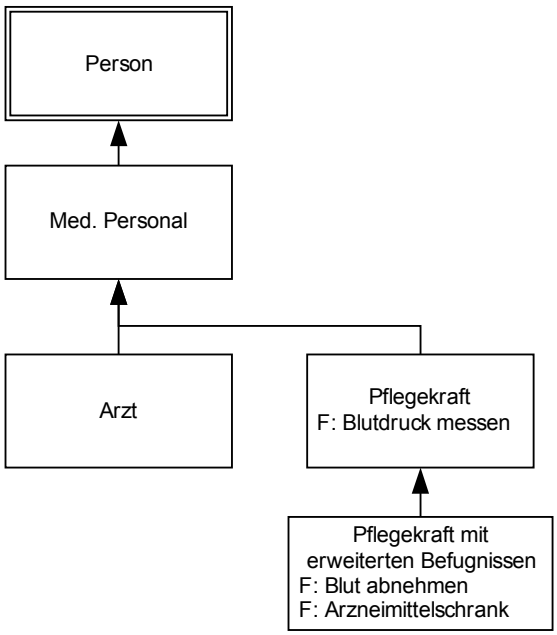


Abbildung 3-2: *Beispiel einer Rollenspezialisierung*

Beispiel:

Da die Rolle „Pflegekraft mit erweiterten Befugnissen“ von der Rolle „Pflegekraft“ abgeleitet ist, besitzt diese nicht nur die ihr direkt zugeordneten Fähigkeiten „Blut abnehmen“ und „Zugriff auf Arzneimittelschrank“, sondern auch die Fähigkeit „Blutdruck messen“, die sie von der Rolle Pflegekraft erbt. (Quelle [Kubi98])

Ein Mitarbeiter führt eine bestimmte Rolle aus, indem er eine Stelle besetzt, welche unter anderem oder auch allein durch diese Rolle beschrieben wird.

Definition 3-6: Rolle

Eine Rolle ist das Mittel zur Gruppierung einzelner Fähigkeiten, zur teilweisen oder auch kompletten Beschreibung einer Stelle. Eine Rolle umfaßt alle Fähigkeiten, die nötig sind, um einen bestimmten Typ von Aufgaben zu bearbeiten. Als besonderes Strukturierungsmerkmal bietet sich bei Rollen die Einordnung in eine Vererbungshierarchie an.

Stellen lassen sich also nicht allein durch das Vorhandensein einzelner Fähigkeiten, sondern auch durch komplexe Rollen beschreiben.

Vertreterregelung
Ist eine Stelle vorrübergehend nicht besetzt (z.B bei Urlaub oder Entlassung) oder der Mitarbeiter, der diese Stelle besetzt, augenblicklich nicht zur Bearbeitung einer anstehenden und ihm zugewiesenen Aufgabe im Stande (z.B. außer Haus), so kann die Aufgabe von einer anderen Stelle in Vertretung bearbeitet werden. Dazu muß für die Stelle eine Vertreterstelle definiert sein, welche diese Stelle in genau der angegebenen Rolle vertritt.

Definition 3-7: Vertreterregelung

Eine Stelle kann für je eine Rolle durch genau eine andere Stelle vertreten werden. Dabei ist die Vertreterregelung üblicherweise zeitlich beschränkt.

Die Möglichkeit einer transitiven Vertreterregelung ergibt offensichtlich keinen Sinn. Eine Stelle soll die Aufgaben, die sie in Vertretung wahrnimmt, nicht an eine weitere Stelle als Vertretung abgeben können. Dies entspräche auch nicht der Intention einer Vertretung. Eine Vertretung sollte im kleinen überschaubaren Rahmen eingerichtet werden. Für umfangreichere Möglichkeiten zur Zuordnung einer zweiten Menge von möglichen Bearbeitern mit niedrigerer Priorität stellt dieses Modell andere Mittel zur Verfügung

Ein weiterer Grund, warum diese Möglichkeit einer transitiven Vertreterregelung keinen Eingang in das Modell findet, ist die Vermeidung von Zyklen.

Fähigkeit
Eine besondere Position im EER-Diagramm hat die Entität *Fähigkeit*. Sie steht sowohl mit der Entität *Mitarbeiter* als auch mit der Entität *Rolle* in Beziehung. Mit Hilfe der *Fähigkeit* kann demnach eine Qualifikation modelliert werden, die eine *Rolle* beschreibt. Genauso kann damit die Qualifikation eines *Mitarbeiters* selbst beschrieben werden.

Bei der Auflösung von Bearbeiterformeln ist dann allerdings darauf zu achten, daß, falls zur Bearbeitung einer Aufgabe eine bestimmte Fähigkeit erforderlich ist, zwei Möglichkeiten existieren, die Menge der potentiellen Bearbeiter ausfindig zu machen:

- Ein Mitarbeiter besitzt eine Fähigkeit (beispielsweise Fähigkeit „*Rasenmähen*“)

- Ein Mitarbeiter besetzt eine Stelle (beispielsweise Stelle „*Hausmeister_1*“). Diese Stelle wird definiert durch eine oder mehrere Rollen (beispielsweise „*Hausmeister*“ oder „*Rasenpfleger*“). Diese Rolle wiederum faßt einige Fähigkeiten zusammen (Darunter auch die Fähigkeit „*Rasenmähen*“)

Für die Auflösung von Bearbeiterformeln ist dann zu beachten, daß die Semantik einer Fähigkeit bedeutet, daß eine von beiden oder auch beide Forderungen zusammen erfüllt sein müssen. Beispiel: Ein Mitarbeiter besetzt die Stelle *Hausmeister_1*, weiß aber nicht wie ein Rasenmäher zu bedienen ist. (Er besitzt die Fähigkeit nicht und kann die Aufgabe nicht übernehmen)

Definition 3-8: Fähigkeit

Eine Fähigkeit beschreibt sowohl eine Qualifikation als auch die Kompetenz eines Mitarbeiters oder einer Rolle.

Hier wird implizit davon ausgegangen, daß mit der Fähigkeit für die Bearbeitung einer Aufgabe auch die entsprechenden Zugriffsberechtigungen auf die zur Ausführung einer Aufgabe benötigte Daten oder Ressourcen vorhanden sind.

3.2 Kritik an dem vorgestellten Organisationsmetamodell

Das oben besprochene Metamodell für die Darstellung einer Organisationsstruktur dient als Ausgangspunkt für die weiteren Überlegungen. Aufbauend auf dem vorgestellten Metamodell wird nun evaluiert, ob dieses über alle Entitäten verfügt, um eine Organisation hinreichend genau zu modellieren. Dazu wird das Metamodell in einigen Aspekten noch einmal genauer betrachtet, die auch in [Wied02] und [Kubi98] erwähnt sind. Außerdem wird geprüft, ob etwaige Aspekte der Anforderungen an das Modell mehrfach beschrieben sind, wobei Inkonsistenzen auftreten können.

Im Unterschied zu anderen publizierten oder in anderen WfMS implementierten Organisationsmetamodellen bietet das oben vorgestellte Metamodell nicht die Möglichkeit, Aufgabenkategorien zu definieren.

Eine Aufgabenkategorie ist eine Ansammlung von Fähigkeiten, die für die Erfüllung eines Aufgabentyps erforderlich sind. Aus dieser Aufgabenkategorie heraus besteht eine Kompetenz für eine Stelle. D.h., eine Stelle ist für die durch einen Aufgabentyp beschriebene Aufgabe vorgesehen. Genauso verhält es sich aber auch mit der Entität Rolle. Auch durch eine Rolle – ebenso eine Zusammenfassung von Fähigkeiten, die für die Bearbeitung einer bestimmten Aufgabe vorliegen müssen – werden Aufgaben eines bestimmten Typs für eine Stelle vorgesehen. Offenbar würde hier durch die Hinzunahme der Entität Aufgabenkategorie hier ein Aspekt der Anforderungen doppelt modelliert werden. Beide Entitäten (Aufgabenkategorie und Rolle) bieten die Möglichkeit zur Spezialisierung und Vererbung der zugeordneten Fähigkeiten auf die spezielleren Ausprägungen. Hinzu kommt, daß die von den beiden Entitäten *Aufgabenkategorie* und *Rolle* ausgehenden Relationen zu *Fähigkeit* und *Stelle* dieselben Kardinalitäten besitzen.

Im Modell beibehalten wird die Entität *Rolle*, die in diesem Metamodell auch für das Beschreibungsmittel der Aufgabenkategorie steht. Dabei bildet *Vertreterregelung* zusammen mit *Stelle* und *Rolle* eine ternäre Relation. *Aufgabenkategorie* dagegen wird nicht mit in das Metamodell mitaufgenommen.

Nicht direkt aus dem Modell ersichtlich ist die Hierarchie von Vorgesetztenbeziehungen. Es würde sich anbieten, daß die Entität *Stelle* eine rekursive Beziehung zu sich selbst unterhalten könnte, mit der das Vorgesetztenverhältnis beschrieben werden könnte.

Zum einen ist es aber nicht einsichtig, aus welchem Grund eine beliebige Stelle beliebigen anderen Stellen vorgesetzt sein soll. Schlimmstenfalls könnten durch eine solche Konstruktionsmöglichkeit Zyklen in der disziplinarischen/fachlichen Ordnung entstehen. Wäre die Beschreibung der Vorgesetztenverhältnisse mittels einer rekursiven Beziehung der Entität Stelle auf sich selbst beschrieben, so ließe sich außerdem nur jeweils ein disziplinarischer und ein fachlicher Vorgesetzter für eine Stelle definieren. Dies wäre aber nicht ausreichend für den Fall, daß eine Stelle in mehreren Projekten beteiligt ist. Für jede Projektbeteiligung hat eine Stelle einen anderen fachlichen Vorgesetzten.

Eingängiger ist die Vorstellung, daß die einer Stelle S_a disziplinarisch vorgesetzte Stelle S_b diejenige sein sollte, die die Organisationseinheit OE_a leitet, der die Stelle S_a angehört. Mit den Mitteln des EER-Diagramms ist das nicht auszudrücken. Das Modell bietet also ausreichende Strukturierungsmöglichkeiten, so daß Vorgesetztenverhältnisse hierarchisch beschrieben werden können.

Eine weitere offene Frage ist, ob sich Vertraulichkeitsstufen für Mitarbeiter mit diesem Modell beschreiben lassen. Möglich wäre die Angabe einer Vertraulichkeitsstufe als Attribut eines Mitarbeiters. Hier allerdings soll die Vertraulichkeit implizit gegeben sein durch die Besetzung einer Stelle, die durch eine oder mehrere Rollen beschrieben ist. Eine Person, die eine Aufgabe dieser Stelle ausführt, auch wenn es eine Vertretung für diese Aufgabe ist, hat damit auch implizit die erforderliche Vertraulichkeit. Die Rolle ist schließlich das Mittel, um den Stellen und den sie besetzenden Personen die Aufgaben zuzuweisen.

Die letzte Anmerkung zu Vollständigkeit und Konsistenz des Modells bezieht sich auf die Leitungsfunktion der Org.Gruppen. Da es sich bei Org.Gruppen eher um lose Verbände in der interdisziplinären Kooperation handelt, kann man davon ausgehen, daß die Leiter der an der Org.Gruppe beteiligten Org.Einheiten die Leitungsfunktion gleichberechtigt ausüben. Für den Fall, daß trotzdem ein Leiter für die Org.Gruppe modelliert werden soll, wird noch eine zusätzliche (1:n)-Beziehung von Stelle zu Org.Gruppe eingerichtet.

Allein die Angabe von Kardinalitäten ist eine Einschränkung einer Beziehung zwischen Entitäten und schränkt die Ausdrucksmächtigkeit des Metamodells ein. Mit der Aufgabe jeglicher Kardinalitätseinschränkungen (alle Beziehungen mit einer (n:m)-Kardinalität versehen), wäre auch eine Mehrlinien- oder Matrixorganisation auf Ebene der Aufbauorganisation (als Ausprägung des beschriebenen Metamodells) modellierbar.

Eine Mehrlinienorganisation ist zwar mit den Mitteln des vorgestellten Metamodells nicht direkt darstellbar, jedoch ist es dem Modellierer freigestellt, durch Projekte beliebig viele parallele Hierarchien zu beschreiben, so daß dies auch als Mehrlinienorganisation aufgefaßt werden kann. Das liegt allerdings in der Interpretation des Modells und ist nicht durch das Metamodell ausdrückbar.

In diesem Abschnitt wurde detailliert auf das Organisationsmetamodell in seiner Darstellung als EER-Diagramm eingegangen. Beschrieben wurden alle Forderungen, die sich mit den Mitteln eines EER-Diagramms darstellen lassen. Zusätzliche Einschränkungen wurden in Form von Text zum Ausdruck gebracht. Bis zu diesem Punkt wurde ausgehend von einer ersten Betrachtung der Anforderungen ein Metamodell vorgeschlagen und in einem weiteren Schritt in einigen Punkten kritisch hinterfragt.

3.3 Forderung nach einem Organisationsmetametamodell

Die Frage, ob nicht schon die Einschränkung von Kardinalitäten eine Einschränkung des Metamodells bedeutet, wurde bereits angesprochen. Genau diese Ansicht wird von [Buss98] vertreten. Hier wird sogar behauptet, daß allein schon die Angabe von Entitäten für ein Metamodell die Ausdrucksmächtigkeit des Modells beeinträchtigt, da die Benennung von Entitäten und deren Beziehung einem speziellen Anwendungsbereich des Systems genügen könnten aber für andere Anwendungen zu stark eingeschränkt sein könnten.
Als Lösung dieser Anforderung beschreibt [Buss98] ein Metametamodell anstelle eines Metamodells[7]. In ihm befinden sich als beschreibende Elemente keine konkreten Entitäten, sondern im wesentlichen die beiden Typen *Organisationsobjekttyp* und *Organisationsbeziehungstyp,* aus denen die Entitäten und Beziehungen eines Metamodells instantiiert werden können.

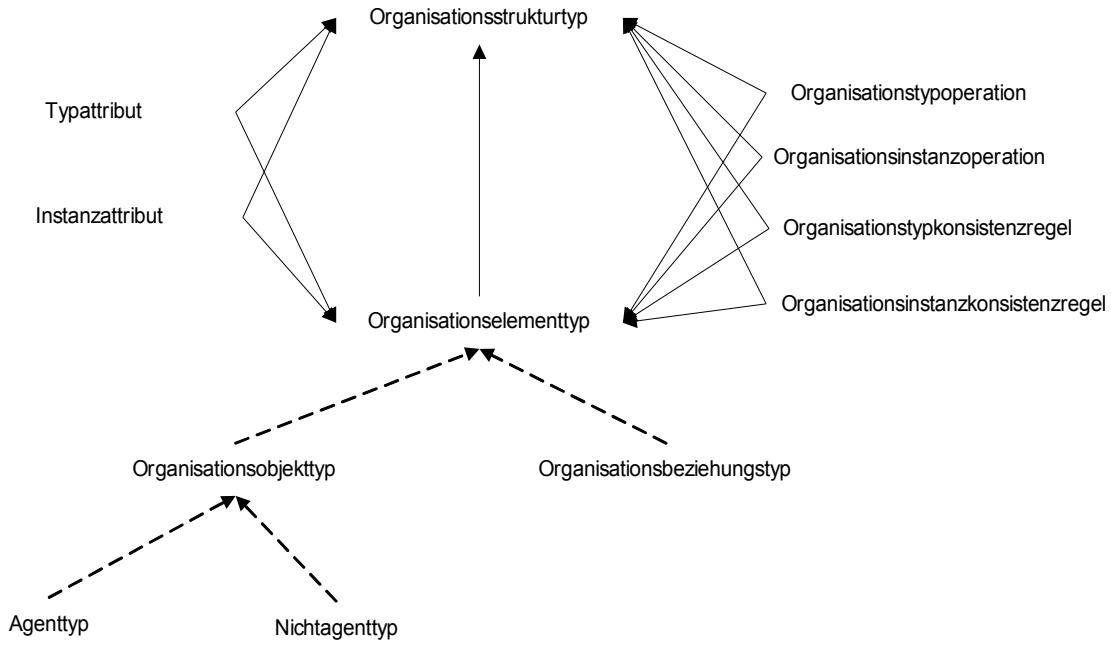


Abbildung 3-3: *Elemente des Metametamodells (aus [Buss98])*

Beide erben dabei von *Organisationselementtyp*, welcher Attributdefinitionen sowohl für die Typen selbst als auch für die Instanzen der Typen definiert. *Organisationselementtyp* definiert nicht nur Attribute, sondern beschreibt auch Operationen, die auf den Organisationselementtypen definiert sind, und Konsistenzregeln, die für diese gelten. Das gilt sowohl für die Typen als auch für die Instanzen (*Organisationstypoperation, Organisationsinstanzoperation, Organisationstypkonsistenzregel, Organisationsinstanzkonsistenzregel*). *Organisationselementtyp* gehört dabei genau einem *Organisationsstrukturtyp* an. Diese Entität wird eingeführt, um Organisationselementtypen gruppieren zu können. Damit lassen sich z.B bei der Modellierung eines Metamodells vom Organisationsstrukturtyp „hierarchische Organisation“ Organisationselementtypen aus diesem Repertoire instantiieren.

7 Im Kontext von [Buss98] wird allerdings dieses Metametamodell als Metamodell bezeichnet. Die Bezeichnung Metametamodell wurde nur für diese Arbeit eingeführt, um die Eigenschaften des Modells aus gegenüber denen dieser Arbeit abgrenzen zu können.

Organisationsstrukturtyp (Name)

Agenttyp (Name, Organisationsstrukturtyp)

Nichtagenttyp (Name, Organisationsstrukturtyp)

Organisationsbeziehungstyp (Name, [Agenttyp_1, Agenttyp_2] |
[Nichtagenttyp_1, Nichtagenttyp_2] | [Agenttyp, Nichtagenttyp] |
[Agenttyp, Organisationsbeziehungstyp] |
[Organisationsbeziehungstyp_1, Organisationsbeziehungstyp_1])

Typattribut (Name. Datentyp, Wert, Organisationsstrukturtyp |
Organisationsbeziehungstyp | Agenttyp | Nichtagenttyp |
Organisationstypoperation | Organisationsinstanzoperation |
Organisationstypkonsistenzregel | Organisationsinstanzkonsistenzregel)

Instanzattribut (Name, Datentyp, Organisationsstrukturtyp | Organisationsbeziehungstyp |
Agenttyp | Nichtagenttyp)

Organisationstypoperation (Name, {Parameter}, Implementierung,
Organisationsstrukturtyp | Organisationsbeziehungstyp | Agenttyp | Nichtagenttyp)

Organisationsinstanzoperation (Name, {Parameter}, Implementierung,
Organisationsstrukturtyp | Organisationsbeziehungstyp | Agenttyp | Nichtagenttyp)

Organisationstypkonsistenzregel (Name, {Parameter}, Implementierung,
Organisationsstrukturtyp | Organisationsbeziehungstyp | Agenttyp | Nichtagenttyp)

Organisationsinstanzkonsistenzregel (Name, {Parameter}, Implementierung,
Organisationsstrukturtyp | Organisationsbeziehungstyp | Agenttyp | Nichtagenttyp)

Abbildung 3-4: Relationenschema für das Metametamodells (aus [Buss98])

Offensichtlich müssen, um die Konsistenzregeln auf Metaebene definieren zu können, diese Mittel bereits auf Metametaebene vorhanden sein.

Abbildung 3-4 zeigt für dieses Vorhaben ein Relationenschema. Hier wird ersichtlich, daß für die Regeln und Operationen bei der Modellierung eines Metamodells noch Implementierungen explizit zur Verfügung gestellt werden müssen. Diese sind natürlich abhängig von dem Metamodell, das auf der Basis des Metametamodells erstellt wird.

Aufgabe des Workflow-Modellierers ist es, aus den Elementen des Metametamodells ein Metamodell herzustellen. D.h., er muß Entitäten wie *Person*, *Stelle* und *Fähigkeit* definieren, die Beziehungen zwischen diesen Entitäten herstellen, Attribute benennen, Konsistenzregeln festlegen und die Operationen programmieren, die dieses Metamodell unterstützen soll. Erst auf diesem Metamodell kann nun die Struktur der Organisation abgebildet werden. Ein Metamodell, wie das im vorangegangenen Abschnitt vorgestellte, ist in diesem Kontext nur

eine mögliche Ausprägung des Metametamodells. Da dieses Metametamodell kein Metamodell definiert sondern nur vorgibt, in welcher Art und Weise ein solches aufgebaut werden kann, hat dieser Ansatz eindeutig eine höhere Ausdrucksmächtigkeit.

Zumindest theoretisch ist dieses Konzept also die bessere Wahl. Im Hinblick auf eine Umsetzung müssen aber folgende Punkte bedacht werden:

- Selbst wenn für die Instantiierung eines Metamodells aus diesem Metametamodell geeignete Werkzeuge zur Verfügung stehen, sind dafür Überlegungen anzustellen, die fast dem Umfang der vorliegenden Arbeit entsprechen. Offensichtlich genügt es nicht, nur Typen festzulegen, sondern diese müssen auch noch formal zueinander in Beziehung gesetzt und mit Attributen versehen werden. Dazu müssen auch noch Konsistenzregeln in geeigneter Form definiert werden und für gewünschte Operationen Implementierungen bereitgestellt werden. Ob diese Aufgaben dem Anwender/Modellierer zugemutet werden sollen ist fraglich.

- Wird es dem Anwender/Modellierer selbst überlassen, Regeln und Operationen zu definieren, ist auch die Gefahr hoch, daß diese nicht der von ihm gewünschten Semantik entsprechen. Immerhin ist zur Instantiierung eines Metamodells ein sehr tiefgreifendes Wissen erforderlich. Die vom Anwender gewünschte Stabilität und Konsistenz kann somit nicht gewährleistet werden.

- Die Einbeziehung einer weiteren Ebene (Metametaebene), die eine Implementierung zur Bereitstellung der Flexibilität benötigt, wirkt sich auch auf die Performanz zur Laufzeit aus. Insbesondere ist der Grad der Auswirkung stark abhängig von den vom Anwender/Modellierer codierten Teilen für Konsistenzregeln und Operationen. Ein Faktor ist dabei sowohl die Effizienz des Codes selbst, als auch die Art und Weise, wie dieser in das System eingebunden ist. Möglichkeiten sind dabei die Integration als Skript, das zur Laufzeit interpretiert wird oder als Programmcode, der zuvor in ausführbaren Code übersetzt wird und zur Laufzeit dynamisch geladen wird.

Aus den genannten Gründen wird daher im folgenden das Konzept eines Metametamodells nicht weiter verfolgt. Mittelpunkt der weiteren Betrachtungen ist das in Abschnitt 3.1 vorgestellte Metamodell. Dieses besitzt zwar nicht die Ausdrucksmächtigkeit beliebiger Entitäten und Konsistenzregeln. Der Forderung nach Flexibilität des Metamodells wird aber zumindest dadurch Rechnung getragen, daß die Verwendung einiger Entitäten freigestellt ist und diese nach Belieben zum Metamodell hinzugenommen werden können, falls die höhere Ausdrucksmächtigkeit gewünscht ist. So besteht ein minimales Organisationsmodell als Ausprägung des vorgestellten Metamodells aus den Entitäten *Org.Einheit*, *Stelle* und *Mitarbeiter*. Dieser Aspekt wird im Rahmen des Entwurfs näher beschrieben.

Natürlich gibt es noch weitere Forderungen, welche Eingang in das Modell hätten finden können. Nachfolgend einige Beispiele:

- eine Relation zwischen Rolle und Mitarbeiter: Ein Mitarbeiter könnte so zur Ausübung einer Rolle fähig sein, ohne daß er eine Stelle besetzt, die durch diese Rolle beschrieben ist.

- eine Relation zwischen Org.Gruppe und Stelle: Ein Stelleninhaber könnte Mitglied einer Org.Gruppe sein, ohne daß die Org.Einheit, der diese Stelle angehört, in der Org.Gruppe involviert ist.

- ...

Diese „Unzulänglichkeiten“ werden an dieser Stelle bewußt in Kauf genommen, da die Hinzunahme von weiteren Entitäten und/oder Relationen die Ausdrucksmächtigkeit des Metamodells nur unwesentlich erhöhen würde. Außerdem ziehen diese Erweiterungen eine höhere Komplexität des Entwurfs und der Implementierung nach sich, die in keinem vernünftigen Verhältnis zu den geringfügig erweiterten Ausdrucksmöglichkeiten des Metametamodells stehen.

3.4 Formale Definition

Das in Abschnitt 3.1 besprochene Organisationsmetamodell erfährt nun an dieser Stelle eine formale Definition. Dabei wird ein mengenbasierter Ansatz gewählt. Das bietet die Möglichkeit, Operationen des Metamodells (siehe Kapitel 4) in Form von Mengenoperationen zu beschreiben, was der Präzision der zu beschreibenden Semantik zugute kommt.

Anmerkung:

Wie in Abschnitt 2.1.5 schon erwähnt, beschreiben die wenigsten verfügbaren Systeme die Semantik ihrer Modelle formal. Das führt dazu, daß Aspekte mißverstanden und falsch interpretiert und dadurch fälschlicherweise für einen Fehler in der Implementierung gehalten werden können. Durch eine unpräzise Beschreibung kann zudem das Verhalten des Systems nicht zweifelsfrei geklärt werden, was auch dazu führen kann, daß manche Modellierungselemente falsch eingesetzt werden. Deshalb wird im folgenden die Definition der Semantik mit besonderer Sorgfalt vorgenommen. Dieses Bestreben wird auch an anderen Stellen dieser Arbeit noch sichtbar.

Definition 3-9
Das Organisationsmetamodell *OMM* wird beschrieben durch folgende 3 Mengen:
OET, *ORT*, *GR*:

OET ist die Menge der Organisationsentitätstypen
OET *= {Fähigkeit, Mitarbeiter, OrgEinheit, OrgGruppe, Projektgruppe, Rolle, Stelle, Vertreterregelung}*

ORT ist die Menge der Organisationsrelationstypen
ORT *= {beschreibt, besetzt, besitzt, besteht_aus, bildet, gehört_zu, hat, ist_untergeordnet_OrgEinheit, ist_untergeordnet_Projektgruppe, leitet_OrgEinheit, leitet_OrgGruppe, leitet_Projektgruppe, spezialisiert, vertritt}*

GR2 ist die Menge der gültigen 2-stelligen Relationen
GR2 *= {* *(Mitarbeiter, Fähigkeit, besitzt),*
(Mitarbeiter, Stelle, besetzt),
(OrgEinheit, OrgEinheit, untergeordnet_OrgEinheit),
(OrgEinheit, OrgGruppe, bildet),
(Projektgruppe, OrgEinheit, besteht_aus),
(Projektgruppe, Projektgruppe, untergeordnet_Projektgruppe),
(Projektgruppe, Stelle, besteht_aus),
(Projektgruppe, Stelle, leitet),
(Rolle, Fähigkeit, besitzt),
(Rolle, Rolle, spezialisiert),
(Rolle, Stelle, beschreibt),
(Stelle, OrgEinheit, gehört_zu),
(Stelle, OrgEinheit, leitet_OrgEinheit),
(Stelle, OrgGruppe, leitet_OrgGruppe),
(Stelle, Projektgruppe, leitet_Projektgruppe) *}*

Es gilt:

$$\forall (oet_1, oet_2, ort) \in GR2 : (oet_1 \in OET \wedge oet_2 \in OET \wedge ort \in ORT)$$

GR3 ist die Menge der gültigen 3-stelligen Relationen
GR3 *= {(Stelle, Stelle, Rolle, vertritt)}*

Es gilt:

$$\forall (oet_1, oet_2, oet_3, ort) \in GR3 : (oet_1 \in OET \wedge oet_2 \in OET \wedge oet_3 \in OET \wedge ort \in ORT)$$

GR ist die Menge aller gültigen Relationen
GR *= GR2 ∪ GR3*

Ein Organisationsmetamodell OrgMM ist definiert durch:
OrgMM = (OET, ORT, GR)

Zusätzliche Einschränkungen erfährt das Metamodell durch Festlegung von Kardinalitäten für die gültigen Relationen *GR*.

Definition 3-10: Kardinalitäten

Die Operation ***card*** beschreibt die jeweils minimale und maximale Zahl an Beziehungen, die die Ausprägungen von Entitäten miteinander unterhalten können. Es gelten

$$card\ (gr) \rightarrow (\mathbb{N}_0, \mathbb{N}_0, \mathbb{N}_0, \mathbb{N}_0) : \forall\ gr \in GR2$$

und

$$card\ (gr) \rightarrow (\mathbb{N}_0, \mathbb{N}_0, \mathbb{N}_0, \mathbb{N}_0, \mathbb{N}_0, \mathbb{N}_0) : \forall\ gr \in GR3$$

für 2- und 3-stellige gültige Relationen. Die resultierenden 4- und 6-Tupel beschreiben je in Zweiergruppen die minimale und maximale Anzahl von Beziehungen der *OET*. Die Reihenfolge entspricht derselben, mit denen die *OET* in den Elementen von *GR* angegeben sind.

Obwohl die Funktionswerte der Operation ***card*** auch aus dem EER-Diagramm ersichtlich sind, werden sie in folgender Tabelle explizit aufgeführt.

Gültige Relation GR	min_{OET1}	max_{OET1}	min_{OET2}	max_{OET2}
(Mitarbeiter, Fähigkeit, besitzt)	0	∞	0	∞
(Mitarbeiter, Stelle, besetzt)	0	∞	0	∞
(OrgEinheit, OrgEinheit, untergeordnet_OrgEinheit)	0	1	0	∞
(OrgEinheit, OrgGruppe, bildet)	0	∞	0	∞
(Projektgruppe, OrgEinheit, besteht_aus)	0	∞	0	∞
(Projektgruppe, Projektgruppe, untergeordnet_Projektgruppe)	0	1	0	∞
(Projektgruppe, Stelle, besteht_aus)	0	∞	0	∞
(Projektgruppe, Stelle, leitet)	0	1	0	∞
(Rolle, Fähigkeit, besitzt)	0	∞	0	∞
(Rolle, Rolle, spezialisiert)	0	1	0	∞
(Rolle, Stelle, beschreibt)	0	∞	0	∞
(Stelle, OrgEinheit, gehört_zu)	1	1	0	∞
(Stelle, OrgEinheit, leitet_OrgEinheit)	0	∞	0	1
(Stelle, OrgGruppe, leitet_OrgGruppe)	0	∞	0	1
(Stelle, Projektgruppe, leitet_Projektgruppe)	0	∞	0	1

Tabelle 3-1: *Operation card für 2-stellige Relationstypen*

Für die Menge GR3, die nur aus einem Element besteht, stellt sich ***card*** folgendermaßen dar:

Gültige Relation GR	min_{OET1}	max_{OET1}	min_{OET2}	max_{OET2}	min_{OET3}	max_{OET3}
(Stelle, Stelle, Rolle, vertritt)	0	∞	0	∞	0	∞

Tabelle 3-2: *Operation card für 3-stellige Relationstypen*

Eine weitere Einschränkung des Modells ist, daß Entitäten, welche Beziehungen zu sich selbst unterhalten, keinen Zyklus bilden dürfen. Dies gilt für die Entitäten *Org.Einheit*, *Projektgruppe* und *Rolle*.

Für die Ausprägungen dieser 3 Entitäten seien dazu folgende Prädikate definiert:

- $untergeordnet\_OrgEinheit(oe_1, oe_2)$ für $oe_1, oe_2 \in OrgEinheit \wedge oe_1 \neq oe_2$
  („ oe_1 ist oe_2 untergeordnet“)

- $untergeordnet\_Projektgruppe(pg_1, pg_2)$ für
  $pg_1, pg_2 \in Projektgruppe \wedge pg_1 \neq pg_2$ („ pg_1 ist pg_2 untergeordnet“)

- $spezialisiert(r_1, r_2)$ für $r_1, r_2 \in Rolle \wedge r_1 \neq r_2$
  („ r_1 spezialisiert r_2 „)

Zur besseren Lesbarkeit werden diese Prädikate im folgenden mit *untergOE*, *untergProj* und *spez* abgekürzt. Auf diesen Prädikaten seien nun folgende Klauseln[8] mit transitiven Eigenschaften wie folgt definiert:

- $untergOE+(oe_1, oe_2) \leftarrow$
  $untergOE(oe_1, oe_2) \vee (untergOE(oe_1, oe_3) \wedge untergOE+(oe_3, oe_2))$
  für $oe_1, oe_2, oe_3 \in OrgEinheit \wedge oe_1 \neq oe_2 \neq oe_3$
  („ oe_1 ist oe_2 transitiv untergeordnet“)

- $untergProj+(pg_1, pg_2) \leftarrow$
  $untergProj(pg_1, pg_2) \vee (untergProj(pg_1, pg_3) \wedge untergProj+(pg_3, pg_2))$
  für $pg_1, pg_2, pg_3 \in Projektgruppe \wedge pg_1 \neq pg_2 \neq pg_3$
  („ pg_1 ist pg_2 transitiv untergeordnet“)

- $spez+(r_1, r_2) \leftarrow$
  $spez(r_1, r_2) \vee (spez(r_1, r_3) \wedge spez+(r_3, r_2))$
  für $r_1, r_2, r_3 \in Rolle \wedge r_1 \neq r_2 \neq r_3$
  („ r_1 spezialisiert r_2 transitiv„)

Diese Klauseln und Prädikate wurden an dieser Stelle eingeführt, um die Einschränkung auf eine zyklenfreie Struktur definieren zu können. Dazu werden die Klauseln nun wie folgt benutzt:

Definition 3-11: Zyklenfreie Organisationsstruktur

Für ein Organisationsmodell gilt:

$\neg\exists\, oe \in OrgEinheit : untergOE+(oe, oe)$
$\neg\exists\, pg \in Projektgruppe : untergProj+(pg, pg)$
$\neg\exists\, r \in Rolle : spez+(r, r)$

8 Die durch die Klauseln mit angehängtem „+“ qualifizierten Ausprägungen der Entitäten stellen also eine Obermenge der durch die Ausgangsprädikate qualifizierten Ausprägungen dar.

Genauso gelten auch die Umkehrschlüsse

$untergOE + (oe_1, oe_2) \rightarrow oe_1 \neq oe_2 \quad \forall\, oe_1, oe_2 \in OrgEinheit$

$untergProj + (pg_1, pg_2) \rightarrow pg_1 \neq pg_2 \quad \forall\, pg_1, pg_2 \in Projektgruppe$

$spez + (r_1, r_2) \rightarrow r_1 \neq r_2 \quad \forall\, r_1, r_2 \in Rolle$

Mit folgendem Prädikat wird das Faktum einer Vertreterregelung beschrieben:

$vertritt\,(s_1, s_2, r)$ für $s_1, s_2 \in Stelle \wedge r \in Rolle$ („ s_1 vertritt s_2 bezüglich der Rolle r “)

Für Vorgesetztenverhältnisse gilt, daß die einer Stelle S_a disziplinarisch vorgesetzte Stelle S_b diejenige ist, die die Organisationseinheit OE_a leitet, der die Stelle S_a angehört[9].

Es muß also gelten:

$$GehörtZu(S_a, OE_a) \wedge LeitetOE(S_b, OE_a) \rightarrow DiszVorgesetzt(S_b, S_a)$$

Dasselbe gilt auch für das fachliche Vorgesetztenverhältnis:

$$ProjMitarbeit(S_a, Proj_a) \wedge LeitetProj(S_b, Proj_a) \rightarrow FachlVorgesetzt(S_b, S_a)$$

Damit ist zwar die Zyklenfreiheit immer noch nicht sichergestellt. Dies ließe sich aber durch die Einführung folgender beider Einschränkungen erreichen:

$$LeitetOE(S_b, OE_a) \rightarrow GehörtZu(S_b, OE_a)$$

und

$$LeitetProj(S_b, Proj_a) \rightarrow ProjMitarbeit(S_a, Proj_a)$$

Das würde aber bedeuten, daß eine Stelle nur dann Leitungsfunktionen über eine Gruppierung wahrnehmen kann, wenn sie dieser selbst angehört. Das ist aber zu restriktiv für das Modell. Somit ist die Gewährleistung der Zyklenfreiheit eine Anforderung an die Komponente, welcher der Zugriff auf das Modell obliegt. Das gilt auch für alle anderen Entitäten, die rekursive Beziehungen zu sich selbst unterhalten. (Spezialisierung/Vererbung von Fähigkeiten auf Rollen, Enthaltenseinsbeziehung von Org.Einheiten und Projektgruppen).

Mit Hilfe eines letzten Prädikats sind alle Forderung an das Metamodell formal beschrieben:

$$\neg\exists\, s \in Stelle : vertritt\,(s, s, r) \quad r \in Rolle$$

Diese Forderung ist trivial. Es ergibt keinen Sinn, wenn sich eine Stelle selbst vertreten könnte.

9 siehe Abschnitt 3.2 (Seite 31)

3.5 Attribute

Ein Attribut beschreibt eine Eigenschaft einer Entität oder einer Relation. Beispiele hierfür sind:

- Name (Namensangabe für eine Ausprägung der Entität Person)
- Startdatum (als Attribut einer Relation. Beschreibt den Zeitpunkt, an dem die spezifizierten Datensätze miteinander in Beziehung gesetzt wurden.)

Definition 3-12: Attributtyp / Attribut

Ein Attributtyp ist ein 3-Tupel (Name, Datentyp, Domäne). Er beschreibt für die Entität/Relation, für die er definiert ist, eine Eigenschaft für jede Ausprägung dieser Entität/Relation. Innerhalb einer Entität/Relation ist der Name eines Attributtyps eindeutig. Die Domäne beschreibt eine Einschränkung des Wertebereichs, den ein Attribut annehmen kann.

Für jede Ausprägung einer Entität/Relation sind die Attributtypen instantiiert, die für diese Entität/Relation definiert sind. Ein solche Instantiierung eines Attributtyps wird Attribut genannt. Ein Attribut ist ein 2-Tupel (Name, Wert). Durch ein Attribut wird jedem Datensatz ein Wert für die beschriebene Eigenschaft zugewiesen.

Offensichtlich geschieht die Deklaration der Attributtypen auf Ebene des Schemas und nicht für jede Instanz einer Entität einzeln. Existiert beispielsweise ein Attributtyp *Telefonnummer* für die Entität Mitarbeiter, so hat jeder Mitarbeiter ein Attribut *Telefonnummer*, unabhängig davon, ob es mit einem Wert belegt ist, oder nicht.

Diese Festlegung auf Schemaebene erweist sich im Hinblick auf eine Implementation als nützlich, da durch sie die Attribute typsicher sind. Anders würde es sich verhalten, wenn für jede Instanz einer Entität ein Attribut *Telefonnummer* angelegt werden könnte. Es wäre möglich, daß ein Mitarbeiter eine Telefonnummer vom Typ einer Zeichenkette und ein anderer als große Ganzzahl besitzen würde. Das würde den Umgang mit Attributen ohne Zweifel erschweren.

Definition 3-13: Schema einer Entität/Relation

Für eine Organisationsentität/Relation x gilt:
$schema(x) = \{attribType_{1,} \ldots, attribType_n\}$
Das Schema einer Entität/Relation ist die Menge ihrer Attributtypen.

Definition 3-14: Instanz einer Entität/Relation

Eine Ausprägung einer Entität/Relation wird Instanz genannt. Eine Instanz *inst* einer Organisationsentität/Relation x enthält Werte für alle Attribute, die durch das Schema von x definiert sind.
$inst\ instanceOf\ (x) \Leftrightarrow inst = \{attrib_{1,} \ldots, attrib_n\}:$
$attrib_i\ instanceOf\ (attribType_i)\ für\ i = 1 \ldots n\ \forall\ attribType \in schema(x)$

Bei der bisherigen Besprechung des Metamodells wurde noch nicht auf die Attributtypen eingegangen. Ganz offensichtlich gehören Attributtypen wie *Name*, *Geschlecht*, *Wohnort* oder *Telefonnummer* zur Entität Mitarbeiter, da sie kategorisierte Eigenschaften von Mitarbeitern beschreiben. Dennoch ist es schwierig, wenn nicht gar unmöglich, eine vollständige Menge von Attributtypen für die Organisationsentitäten anzugeben.

Für die Entität Mitarbeiter sei ein Attributtyp *Gehalt* definiert. Ist es für die Funktion eines Workflow-Management-Systems erforderlich, Zugriff auf die Informationen bzgl. des Einkommens eines Mitarbeiters zu haben? Auszuschließen ist das nicht. Aber die Tatsache, ob ein Attributtyp *Gehalt* definiert ist, oder nicht, entscheidet darüber, ob in der Zuweisung durch eine Bearbeiterformel darauf Bezug genommen werden kann. Dennoch ist es nicht möglich die gesamte Menge der Attributtypen einer Entität anzugeben. Schließlich wären auch Eigenschaften wie Haarfarbe oder Schuhgröße Kandidaten für Attribute eines Mitarbeiters.

Um eine größtmögliche Ausdrucksmächtigkeit des Metamodells zu erreichen, existiert die Möglichkeit, die Entitäten und Relationen um branchenspezifische Attributtypen zu erweitern. Jeder erweiterte Attributtyp vergrößert die Ausdrucksmöglichkeit dahingehend, daß Bearbeiter nach einem weiteren Kriterium in die Menge der potentiellen Bearbeiter mit aufgenommen oder aus ihr ausgeschlossen werden können.

Die nächste Schwierigkeit besteht nun darin, festzulegen, welche Attributtypen eine Entität oder Relation grundsätzlich hat, und welche Attributtypen zu den erweiterten Attributtypen zählen. Das Kriterium für diese Einteilung lautet wie folgt:

Definition 3-15: Unterscheidung von Attributtypen

Attributtypen, die nicht unbedingt notwendig sind, um den Zusammenhalt des Metamodells zu sichern, gehören zu den erweiterten Attributtypen (*optional*) . Alle anderen Attributtypen sind *obligatorisch*.

Die obligatorischen Attributtypen einer Entität oder Relation sind die, die eine Ausprägung einer Entität oder Relation eindeutig bestimmen. Eine Organisationsentität $oe \in OET$ enthält als einzigen obligatorischen Attributtyp eine Identifikationsbezeichnung (ID). Eine gültige Relation $gr \in GR$ enthält überhaupt keinen obligatorischen Attributtyp.

Das bedeutet, daß selbst elementare Attribute wie der Name einer Person zu den erweiterten Attributtypen gehören. Das mag auf den ersten Blick merkwürdig erscheinen, aber für das Workflow-Management-System sind solche Attribute nicht nötig, da man sich bei der Konstruktion von Bearbeiterzuordnungen allein auf die Angabe von IDs beschränken könnte.

Ein konkretes System wird sich natürlich nicht mit dieser Beschränktheit zufrieden geben und bereits standardmäßig ein reichhaltiges Repertoire an Attributtypen mit ausliefern, was aber letztlich niemand daran hindern kann, ein Organisationsmodell ohne elementare Attribute wie den Namen einer Person zu betreiben. Für die Konzeption werden aber, wie in Definition 3-15 beschrieben, die allermeisten Attributtypen als erweitert behandelt.

3.6 Instantiierung des Organisationsmetamodells

Das in Abschnitt 3.4 formal definierte Organisationsmetamodell *OrgMM* bildet nun gewissermaßen eine Vorlage für die Instantiierung eines Organisationsmodells *OM*, das die Struktur einer konkreten Organisation beschreibt.
Der Sachverhalt, daß es sich bei *OM* um eine Instanz von *OrgMM* handelt wird beschrieben durch: $OM \in OrgMM$

Inhalt des Organisationsmodells sind die Instanzen der beschrieben Entitäten und deren Beziehungen zueinander.

Bei der Modellierung eines konkreten Organisationsmodells ist bei den Beziehungen zwischen den Entitäten Rollen, Fähigkeit und Stelle die Vorstellung einer Analogie zum Konzept der Objektorientierung (OO) hilfreich.

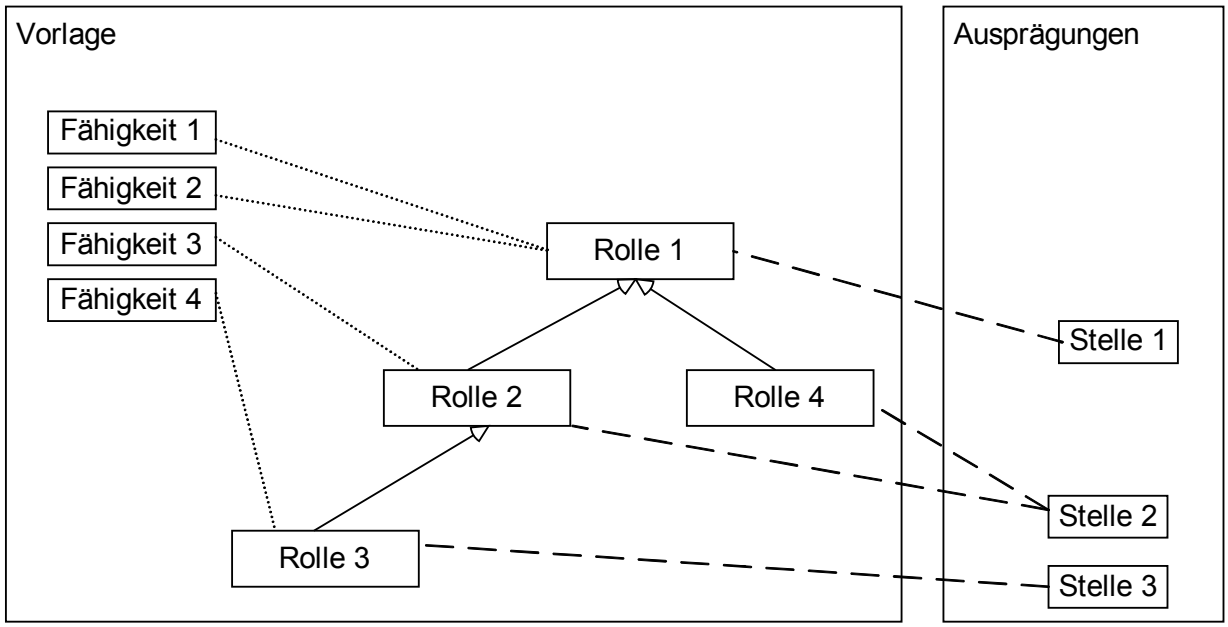


Abbildung 3-5: *Rollenhierarchie und Analogie zur OO-Technik*

Beispiele für Fähigkeiten in diesem Sinn sind „*Blut abnehmen*“, „*Medikamente verabreichen“* und „*Essen austeilen*“. Diese Fähigkeiten könnten zur Beschreibung der Rolle „*Krankenschwester*“ herangezogen werden. Stellen, die diese Rolle haben, könnten „*Krankenschwester_01*“, ...„*Krankenschwester_27*“ sein.

Abbildung 3-5 verdeutlicht diese Entsprechung. Die Rollen bilden gewissermaßen eine „Klassenhierarchie“ von Vorlagen. Jede Rolle ist beschrieben durch eine Menge von Fähigkeiten (-> „Klassenattribute“). Durch Vererbung lassen sich Rollen spezialisieren. Jede Stelle ist nun eine Ausprägung einer Rolle, wie ein Objekt die Ausprägung einer Klasse darstellt.
Allerdings gibt es einen Unterschied zur üblichen OO-Technik. In der Klassenhierarchie findet sich die vielfach umstrittene Mehrfachvererbung nicht, dafür aber auf Ebene der Ausprägungen. Eine Stelle kann nicht nur die Ausprägung einer einzelnen Rolle sein, sondern zugleich die Ausprägung mehrerer Rollen. Jede Stelle kann also nach dem Baukastenprinzip aus verschiedenen Rollen zusammengebaut werden. So kann in einem Betrieb eine Sekretärinnenstelle mit den Rollen „Poststelle“ und „Schriftverkehr“ existieren aber auch zumindest theoretisch eine Stelle, die sowohl „Generaldirektor“ als auch „Hausmeister“ ist.

3.7 Zusammenfassung

In diesem Kapitel wurde ein detailliertes Organisationsmetamodell entwickelt, mit dessen Ausdrucksmächtigkeit es möglich ist, eine umfangreiche Menge von verschiedenen Organisationsmodellen zu beschreiben.

Die Betrachtung eines existierenden Konzeptes von C. Bussler (Abschnitt 3.3) endete mit der Erkenntnis, daß der „Rückzug“ auf ein Metametamodell zwar in vielerlei Hinsicht eine theoretisch maximale Ausdrucksmächtigkeit für das Metamodell ergibt, sich letztendlich aber weniger gut dazu eignet, ohne hohen Aufwand ein Organisationsmetamodell damit zu entwerfen. Letztendlich müßte der Modellierer/Administrator einer Organisation dazu erst Entitäten wie Stellen, Rollen usw. entwerfen, auf deren Basis dann wiederum das Modell selbst aufzubauen wäre. Hinzu kommt, daß dieses Konzept nicht so performant zu implementieren ist wie das Konzept eines Organisationsmetamodells.

In Abschnitt 3.1 wurde ein Organisationsmetamodell vorgestellt, das nicht alle Modellierungsmöglichkeiten eines Metametamodells bietet, aber dennoch für die Beschreibung der allermeisten Organisationsmodelle ausreicht. Nachdem dieses Metamodell in einigen Punkten angepaßt wurde (Abschnitt 3.2), wurde die formale Definition des Organisationsmetamodells beschrieben.

Je umfangreicher das Repertoire des Metamodells ausgelegt ist, desto schwieriger ist es für den Modellierer einer Organisation, die richtigen Beschreibungsmittel für die Abbildung seiner existierenden Organisation auszuwählen. Die Modellierung der Organisationsstruktur ist dabei noch nicht einmal das Aufwendigste. Einen hohen Aufwand zieht dabei auch noch die Modellierung der Bearbeiterzuordnungen nach sich, die in Kapitel 5 besprochen werden.

4 Operationen des Metamodells

Da es sich bei einem Organisationsmodell, wie in den allgemeinen Anforderungen beschrieben, um ein Gebilde handelt, welches eine „lebendige“ Struktur abbildet, ist es einer Reihe von dynamischen Änderungen unterworfen.
Um dieser Tatsache Rechnung zu tragen, sollte ein Organisationsmetamodell Operationen anbieten, mit denen es möglich ist, Änderungen in der realen Welt auf der Datenbasis des Modells nachzuvollziehen. Dabei sollen natürlich die Konsistenzbedingungen, die an das Modell gestellt werden, berücksichtigt werden. Auch die Integrität der Daten soll dabei gewährleistet sein.

Eine Änderungsoperation stellt eine Überführungsfunktion des Organisationsmetamodells dar.

Definition 4-1: Änderungsoperation

$$\ddot{A}OP : OrgMM \rightarrow OrgMM \quad \text{d.h.}$$
$$\ddot{a}op(OM) = OM^* \quad \forall\, \ddot{a}op \in \ddot{A}OP ; OM , OM^* \in OrgMM$$

Eine Änderungsoperation *äop* aus der Menge aller auf *OrgMM* definierten Änderungsoperationen *ÄOP* bildet also ein Organisationsmodell *OM* auf ein anderes Organisationsmodell *OM** ab. Da ebenfalls $OM^* \in OrgMM$ gilt, befindet sich ein Organisationsmodell nach der Ausführung einer Änderungsoperation wieder in einem konsistenten Zustand, der allen Forderungen genügt, die in Abschnitt 3.4 dargelegt werden.

Es bietet sich an, Änderungsoperationen nach dem Maß ihrer Komplexität in einfache und komplexe Änderungsoperationen zu gruppieren. Einfache Änderungsoperationen sind in diesem Sinne Manipulationen, deren Wirkungskreis sich auf eine Ausprägung einer einzigen Entität oder Relation beschränkt (Bsp: „Definition einer neuen Fähigkeit“, „Änderung der Zuordnung Stelle – Organisationseinheit“).
Komplexe Änderungsoperationen beeinflussen mehrere Ausprägungen einer oder mehrerer Entitäten (Bsp: „Alle Mitarbeiter, die eine bestimmte Stelle besetzen, bekommen eine neue Fähigkeit“).

4.1 Änderungsprimitiven

Offensichtlich nutzen mehrere Änderungsoperationen bei ihrer Durchführung dieselben "primitiven" Manipulationen am Datenmodell (beispielsweise „Einfügen eines Datensatzes einer Entität“, „Herstellen einer Beziehung zwischen Datensätzen“). Um diese nicht für jede Änderungsoperation einzeln beschreiben, bzw. im Rahmen einer Implementation nicht redundant erstellen zu müssen, wird an dieser Stelle eine neue Klasse von Primitiven am Modell definiert.

Definition 4-2: Änderungsprimitive

$$\ddot{A}PRIM : OM \rightarrow Q$$
$$\ddot{a}Prim(OM) = q \text{ für } \ddot{a}Prim \in \ddot{A}PRIM ; OM \in OrgMM ; q \in Q$$

Dabei ist *Q* eine unbestimmte Menge. Im konkreten Fall einer Änderungsprimitive *äPrim* ist *Q* natürlich definiert, aber nur durch die Semantik der Änderungsprimitive. Damit soll ausgedrückt werden, daß eine Änderungsprimitive jegliche Manipulation am Datenmodell

vornehmen darf. Die Definition des Zustandes des Resultats und die Bedingungen, denen der Zustand genügt, ist dabei einzig und allein durch die Semantik der Primitive gegeben.

Interessant sind dabei nur die Primitiven, durch deren konsekutive Anwendung wieder ein Organisationsmodell $OM \in OrgMM$ entsteht.

d.h.:

$$\ddot{a}Prim_1(OM) = q_1$$
$$\ddot{a}Prim_2(q_1) = q_2$$
$$\dots$$
$$\ddot{a}Prim_n(q_{n-1}) = OM^* \quad \text{für } q_1, q_2, \dots, q_{n-1} \in Q;\ OM, OM^* \in OrgMM$$

Durch eine solche Anwendung von Änderungsprimitiven ist eine Änderungsoperation gegeben. Einfach ausgedrückt definieren Änderungsprimitiven Manipulationen am Datenmodell, deren Resultat nicht zwingend die Integrität des Datenmodells wahrt. Dies ist auch der Grund, warum den Änderungsoperationen eine transaktionsbasierte Ausführungslogik zugrunde liegt. Außerhalb des Kontextes einer Änderungsoperation sind die aus den Primitiven jeweils resultierenden Zustände nicht existent. Eine genaue Beschreibung der Änderungsprimitiven liefert [Wied02].

Da die vorliegende Arbeit unter anderem auch die Aspekte eine Implementierung untersuchen soll, wird an dieser Stelle eine Vorschau auf den Entwurf der Software gewagt. Beim Einsatz eines Datenhaltungssystems macht man sich unter vielen anderen Vorteilen die Tatsache zu Nutze, daß Daten nicht beliebig, sondern einem Schema entsprechend abgelegt sind. Durch die Definition eines Schemas wird dem Datenbestand eine Struktur auferlegt, die es dem Datenhaltungssystem erlaubt, bis zu einem gewissen Grad selbst über die Konsistenz und Integrität des verwalteten Datenbestandes zu wachen.

Die Möglichkeiten reichen von passivem Schutz vor Manipulationen, die diesen Zustand gefährden (*beispielsweise Kardinalitätsbeschränkungen von Beziehungstypen*), bis hin zu aktivem Wiederherstellen des konsistenten, integeren Zustandes nach einem manipulativen Eingriff (*beispielsweise referentielle Integrität*) durch die Verfolgung einer Kaskade.

Obwohl die genannten Beispiele insbesondere im Bereich der RDBMS Geltung haben, gilt allgemein, daß ein Datenhaltungssystem durch geeignete Datenmodellierung Einsicht in die Semantik der Daten hat und diese nutzen kann. Würde man sich im konkreten Fall einer Implementierung für den Einsatz eines RDBMS entscheiden, wären fast alle einschränkenden Bedingungen des Modell aus Kapitel 3 direkt in das Schema abbildbar.

Diese Möglichkeit nicht zu nutzen, wäre zum einen ineffizient zur Laufzeit und zum anderen auch umständlich zu implementieren. Ineffizient deshalb, weil man davon ausgehen kann, daß die im Datenhaltungssystem integrierten Mechanismen stark optimiert sind. Umständlich deshalb, weil die schon im Datenhaltungssystem implementierten Mechanismen selbst noch einmal redundant erstellt werden müßten.

Um nochmals auf das Beispiel des RDBMS zurückzukommen: Wollte man auf jegliche Unterstützung zur Datenintegrität verzichten – und das muß man, wenn man Änderungsprimitiven implementieren will – müßte man selbst (1:n)-Beziehungen mit einer Zwischentabelle realisieren. Nur dann könnte man auch eine Änderungsprimitive wie *addOrgRel* (siehe [Wied02]) zur Verfügung stellen, die ohne Rücksicht auf Kardinalitätsbeschränkungen Datensätze zueinander in Beziehung setzt.

Zusammenfassend ist also festzustellen, daß bei der konkreten Implementierung einer Org.Modell-Komponente die Fähigkeiten zur Integritäts- und Konsistenzsicherung des Datenhaltungssystems so weit wie möglich genutzt werden sollten. Auf die explizite Erstellung von Änderungsprimitiven in der Implementation sollte also verzichtet werden. Die Möglichkeiten des Datenhaltungssystems sollten mit einer auf das jeweilige Modell zugeschnittenen Schicht erweitert werden, so daß oberhalb dieser Schicht eine Schnittstelle für einfache Änderungsoperationen zu Verfügung steht, die vom zugrundeliegenden Datenhaltungssystem abstrahiert.

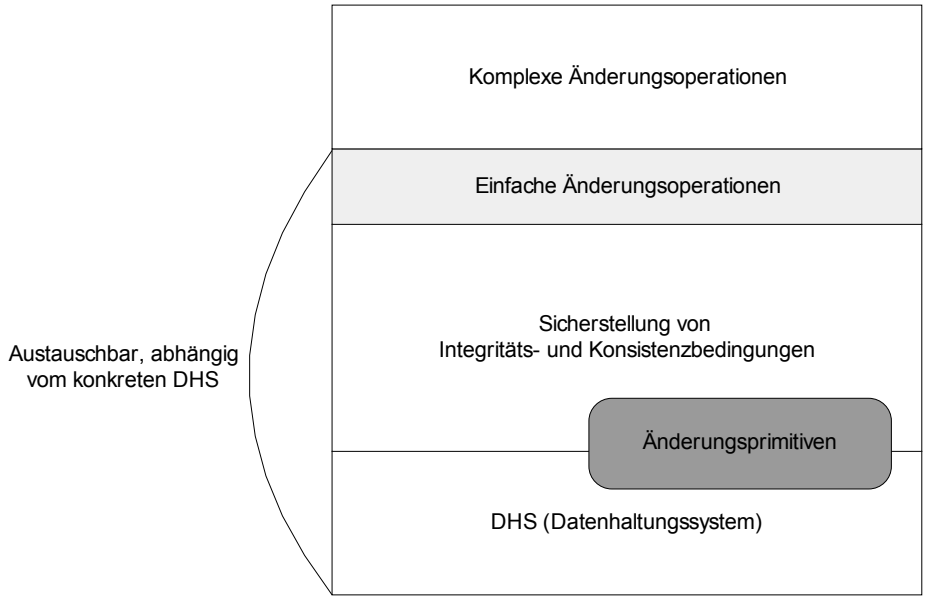


Abbildung 4-1: *Operationen des Metamodells*

Abbildung 4-1 zeigt, wie die einzelnen Gruppen voneinander abhängen. Die Schicht „*Sicherung von Integritäts- und Konsistenzbedingungen*“ sorgt zusammen mit dem DHS für die integere und konsistente Darstellung des Modells nach oben. Dazu werden implizit oder explizit Änderungsprimitiven zur Verfügung gestellt. Die Schicht „*einfache Änderungsoperationen*“ implementiert einen Satz von Änderungsoperationen und stellt somit eine Schnittstelle dar, die vom konkreten DHS abstrahiert. Diese drei Schichten zusammen bilden also ein austauschbares „Backend“.

Natürlich haben Änderungsprimitiven in der konzeptuellen Sicht ihre Berechtigung. Der Vorteil, daß durch ihre Anwendung die Komplexität leichter beherrschbar wird, verschwindet aber, wenn man bedenkt, daß bei einer Implementierung ein Datenhaltungssystem bereits einen großen Teil dieser Aufgaben abnehmen kann. Ob und inwieweit Änderungsprimitiven doch explizit erstellt werden, hängt dabei vom konkret eingesetzten Datenhaltungssystem ab. Daher wird an dieser Stelle nicht weiter auf Änderungsprimitiven eingegangen. [Wied02] liefert dazu eine ausführliche Beschreibung für die Änderungsprimitiven eines ähnlichen Organisationsmetamodells. Aufgabe der Konzeption ist es nun, einen geeigneten Satz von Änderungsoperationen zu definieren, die ein Backend für die Org.Modell-Komponente implementieren muß.

4.2 Änderungsoperationen

Änderungsoperationen müssen der Forderung nach transaktionsbasierter Ausführungslogik genügen. Die Definition einer Transaktion in diesem Kontext ist identisch mit der aus dem Bereich der Datenbankmanagementsysteme.

Eine Transaktion ist definiert durch die Eigenschaften, die durch das Akronym ACID zusammengefaßt sind[10]:

- **(A)tomarity**: (=Atomarität) Eine Transaktion wird ganz oder gar nicht durchgeführt. Kommt es bei der Durchführung zu Fehlern, so wird die Datenbasis auf den konsistenten Zustand zurückgesetzt, der vor der Durchführung der Transaktion Gültigkeit hatte.
- **(C)onsistency**: (=Konsistenz) Nach jeder Transaktion befindet sich die Datenbasis in einem konsistenten Zustand. Allerdings kann während der Ausführung der Transaktion die Konsistenz aufgehoben worden sein.
- **(I)solation**: (=Isolation) Jede Transaktion läuft abgeschlossen von anderen Transaktionen ab. Gleichzeitig ausgeführte Transaktionen beeinflussen sich gegenseitig nicht.
- **(D)urability**: (=Beständigkeit) Eine Änderung der Datenbasis, die durch eine erfolgreiche Transaktion vorgenommen wurde, ist beständig.

Eine Änderungsoperation besteht dabei aus der Anwendung einer oder der konsekutiven Anwendung mehrerer Änderungsprimitiven

$$äop(OM) = äprim_n(\ldots äprim_2(äprim_1(OM))\ldots)$$
$$für\ äop \in ÄOP\ ;\ äprim_1 \ldots äprim_n \in ÄPrim\ ;\ OM \in OrgMM$$

Die Änderungsoperationen werden nach dem Maß ihrer Komplexität in einfache und komplexe Änderungsoperationen gruppiert. In den folgenden Abschnitten wird ein Satz von Änderungsoperationen für das Modell aus Kapitel 3 hergeleitet, mit dem jegliche Dynamik einer Organisation auf dem Modell vollzogen werden kann.

4.2.1 Einfache Änderungsoperationen

Einfache Änderungsoperationen haben nur direkte Auswirkungen auf:

- eine einzelne Ausprägung einer Entität (Instanz) und deren Beziehung zu anderen Datensätzen
- das Schema einer Entität.

Im Falle von Beziehungen mit (n:m)-Kardinalität kann sich eine Änderungsoperation auch auf eine Ausprägung einer Beziehung auswirken.

Grundsätzlich stehen Operationen zur Verfügung, um den Entitäten Instanzen (Ausprägungen von Entitäten) hinzuzufügen und zu löschen. Dazu kommen Operationen, um den Entitäten Attributtypen hinzuzufügen und zu löschen. Die Attribute müssen mit Werten versehen werden können. Die für Entitäten zur Verfügung stehenden einfachen Operationen sind somit: *addInstance, delInstance, addAttributetype, delAttributetype, setAttributeVal.*

10 siehe [HäRa01]

Für Relationen stehen folgende Operationen zur Verfügung: *addRelation, delRelation*.[11]
Das gilt für Beziehungen jeglicher Kardinalität. Man kann die Beziehungen mit (1:n)-Kardinalitäten auch davon ausnehmen. Beziehungen, welche eine (1:n)-Kardinalität aufweisen, würden dann manipuliert mit der Operation *setAttributeVal* der Entität, die nur eine Beziehung dieses Typs unterhält. Die Beziehung dieser Instanz würde also als eines ihrer Attribute behandelt. Das mag zwar für den konkreten Entwurf mit einer zugrundeliegenden relationalen Speicherung zutreffen (relationale Datenbank), für die konzeptionelle Betrachtung sind aber auch diese Beziehungen wie alle anderen zu behandeln.

addInstance(oe, inst)
fügt der Organisationsentität *oe* die Instanz *inst* hinzu. Falls diese schon existiert, bleibt die Ausführung dieser Operation ohne Folgen für das Modell.

Vorbedingung: $oe \in OET\,;\, inst = instanceOf(oe)$
Nachbedingung: $inst \in oe$

delInstance(oe, inst)
entfernt aus der Organisationsentität *oe* die Instanz *inst*, falls diese existiert *(1)*. Dabei werden auch alle Beziehungen zu anderen Instanzen in *oe* oder anderen Organisationsentitäten aus dem Modell entfernt *(2)*. Von *inst* komplett abhängige Instanzen werden dabei ebenfalls gelöscht *(3)*, und zwar wiederum auch mit den Beziehungen, an denen sie beteiligt sind. Diese Operation kann eine Kaskade nach sich ziehen.
Für Entitäten, die eine rekursive Beziehung zu sich selbst unterhalten, bedeutet das, daß beim Löschen auch alle in der Hierarchie unter dieser Instanz liegenden Instanzen gelöscht werden, jeweils mit allen Instanzen, die von diesen abhängig sind[12].

beispielsweise Löschen einer Org.Einheit:

- Entfernen der Beziehungen zu Org.Gruppe, Arbeitsgruppe.
- Untergeordnete Org.Einheiten von *oe* werden gelöscht. Für diese Instanzen wird wiederum die gerade beschriebene Regel angewandt.
- Entfernen der abhängigen Instanzen in anderen Entitäten und deren Relationen (Stellen, Rollenbeschreibung, Vertreterregelungen, Stellenbesetzungen, Leitungsfunktionen in Arbeitsgruppe, Org.Gruppe)

Vorbedingung: $oe \in OET\,;\, inst = instanceOf(oe)$
Nachbedingung:

$(1)\ inst \notin oe$
$(2)\ \forall\, gr \in GR : \neg \exists\, rel \in gr : participates(inst, rel)$
$(3)\ \neg \exists\, inst_2 \in oe_2 : depends(inst_2, inst)$
$Anwendung\ von (2)\ auch\ auf\ inst_2$

Offensichtlich können durch eine unbedachte Anwendung dieser Operation mehr Instanzen gelöscht werden, als ursprünglich geplant. Wird z.B. eine Org.Einheit gelöscht, so sind davon

11 Die Unterteilung von Operationen in solche, die zur Manipulation von Entitäten dienen und solche, die die Manipulation von Beziehungen unterstützen, ist nur eine Möglichkeit. Ebenso könnten die Operationen unterteilt sein in solche, die die Instanzebene betreffen (*addInstance*, *delInstance*, *setAttributeVal*, *addRelation*, *delRelation*) und solche, die das Schema des Modells manipulieren, sich also auf der Metaebene befinden (*addAttributetype*, *delAttributetype*). Diese Unterscheidung ist an dieser Stelle nicht von Bedeutung. Sie ist erst von Interesse, wenn die Operationen im Zuge des Entwurfs eventuell an verschiedenen Stellen implementiert oder in verschiedenen Interfaces zur Verfügung gestellt werden sollen. Aus diesem Grund ist die Reihenfolge, in der die Operationen an dieser Stelle vorgestellt werden, nicht von Bedeutung.

12 Gilt für Org.Einheit, Projektgruppe und Rolle

auch alle untergeordneten Org.Einheiten betroffen, inklusive deren Stellen und die Beziehung der Stellen zu den Mitarbeitern. Ist dies nicht beabsichtigt, so müssen die betroffenen Stellen vorher mittels *addRelation* an eine andere Stelle in der Organisationsstruktur verschoben werden. Dies kann durch eine komplexe Änderungsoperation ausgeführt werden, die diese Operationen gruppiert.

addAttribute(x, attribType)
fügt der Entität/Relation *x* den Attributtyp *attribType* hinzu, falls nicht schon ein gleichnamiger Attributtyp für diese Entität/Relation existiert. Sollte ein solcher schon existieren, hat die Ausführung dieser Änderungsoperation keine Auswirkung auf das Modell.

Vorbedingung: $x \in (OET \cup ORT)$
Nachbedingung: $attribType \in schema(x)$

delAttribute(x, attribType)
entfernt den Attributtyp *attribType* aus dem Schema der Organisationsentität/Relation *x*, falls das Schema von *x attribType* enthält.

Vorbedingung: $x \in (OET \cup ORT)$
Nachbedingung: $attribType \notin schema(x)$

setAttributeVal(inst, attribType, value)
setzt den Wert des angegebenen Attributes der Instanz *inst* auf *value*, falls *value* aus der Domäne der Werte ist, die das Attribut annehmen kann.

Vorbedingung: $inst = instanceOf(x) \wedge attribType \in schema(x) \forall x \in (OET \cup ORT$
Nachbedingung:

$$val(attrib) = value$$
$$für\ attrib = instanceOf(attribType) \wedge attrib \in inst$$
$$falls\ value \in domain(attribType)$$

addRelation(rel, gr)
fügt dem Modell eine Relationsinstanz vom Typ gr hinzu; da *rel* eine Instanz von *gr* ist, finden sich in *rel* auch die Entitätsinstanzen, die miteinander in Beziehung gebracht werden sollen.

Vorbedingung: $gr \in GR, rel = instanceOf(gr)$
Nachbedingung: $rel \in gr$

Die Operation addRelation muß bei ihrer Anwendung insbesondere sicherstellen, daß keine Zyklen in Hierarchien entstehen. Das gilt für die Entitäten *Org.Einheit*, *Projektgruppe* und *Rolle*

delRelation(rel, gr)
entfernt eine Relationsinstanz vom Typ *gr* aus dem Modell. Die Relationsinstanz wird spezifiziert durch die Entitätsinstanzen, die sie in Beziehung setzt. Ist eine solche Relationsinstanz nicht vorhanden, hat diese Operation keine Auswirkung auf das Modell .

Vorbedingung: $gr \in GR, rel = instanceOf(gr)$
Nachbedingung: $rel \notin gr$

Bei der Operation *delInstance* handelt es sich scheinbar um eine komplexe Änderungsoperation. Tatsächlich ist es aber eine einfache Änderungsoperation. Die Komplexität entsteht hier durch die folgende notwendige Anpassung des Modells, um die Konsistenz des Modells aufrecht zu erhalten.

4.2.2 Komplexe Änderungsoperationen

Komplexe Änderungsoperationen sind solche Operationen, die semantisch höherwertige Manipulationen am Organisationsmodell vornehmen. Dabei handelt es sich um Operationen wie Verschieben, Teilen und Vereinigen von Entitätsinstanzen. Diese komplexen Operationen lassen sich jeweils mit einer Kette von einfachen Änderungsoperationen bewerkstelligen.

Da für die Anwendung einer einzelnen einfachen Änderungsoperation schon die Einhaltung der Korrektheitsbedingungen des Metamodells zugesichert wird, gilt dies folglich auch für die Anwendung der komplexen Änderungsoperationen. Die oben beschriebene Transaktionslogik (ACID) gilt allgemein für alle Änderungsoperationen, also auch für die komplexen Änderungsoperationen. Insbesondere die Eigenschaft der Atomarität ist hier nochmals erwähnenswert. Auch eine komplexe Änderungsoperation wird entweder ganz oder gar nicht durchgeführt, auch wenn bei einem Abbruch durch die Zusicherung der einfachen Änderungsoperationen ein konsistenter Zustand des Organisationsmodells gegeben wäre.

Die Menge der komplexen Änderungsoperationen beliebig groß. Bei einer geeigneten Umsetzung im Entwurf lassen sich Änderungsoperationen beliebig zur Org.Modell-Komponente hinzufügen oder aus ihr entfernen. [Wied02] zeigt eine Reihe von vordefinierten Operationen, die auch auf diesem Modell anwendbar sind. Weshalb auf diese hier nicht weiter eingegangen wird.

4.2.3 Einschränkung der Anwendbarkeit

Für die Änderungsoperation *delInstance* muß eine zusätzliche Einschränkung definiert werden. Es existiert ein Fall, in dem diese Operation nicht ausgeführt werden darf.

Wie in vielen anderen Systemen werden auch durch die Org.Modell-Komponente den Benutzern Beschränkungen auferlegt, die definieren, welche Berechtigungen der Benutzer besitzt und welche nicht. Die Funktion eines Systems teilt sich also in die Anwendung des Systems in der eigentlichen Aufgabe und in die Verwaltung des Systems auf. Die Verwaltung ist die Aufgabe eines Systemverwalters[13]. Obwohl die Aufgaben eines Systemverwalters an andere Benutzer delegiert werden können, existiert doch ein ausgezeichneter Benutzer, der diese Aufgaben übernimmt (Administrator, root, ...). Neben der Verwaltung des Systems als Hauptzweck ist dieser Benutzer für das System auch noch die Anlaufstelle für die Meldung jeglicher Fehlerzustände und Probleme, die während des Betriebs des Systems auftreten und nicht anderweitig gelöst werden können.

Im Falle einer Org.Modell-Komponente sind solche Fehlerzustände zum Beispiel:

- Es kann für einen Arbeitsschritt kein Bearbeiter gefunden werden.
- Beim Starten einer Prozeßinstanz wurde kein Prozeßverantwortlicher angegeben.
- ...

Um also die korrekte Funktion der Org.Modell-Komponente sicherstellen zu können und um Fehler erkennen zu können, muß ein solcher Systemverwalter existieren. Um dessen Existenz zu gewährleisten, darf die Operation *delInstance* nicht auf diesen Benutzer angewendet werden.

13 Analog zu Systemverwalter im Kontext von Betriebssystemen oder Datenbanken

4.3 Ausführungsmodus von Änderungsoperationen

Wie schon erwähnt, ist es durch Anwendung von komplexen Änderungsoperationen möglich, beliebig viele einfache und/oder komplexe Änderungsoperationen zu einer Transaktion zusammenzufassen. Das bedeutet, daß eine Änderungsoperation auch die Fusion von zwei oder mehr Organisationen bedeuten kann. Bei der Ausführung einer dermaßen komplexen Transaktion ist damit zu rechnen, daß sie einige Zeit in Anspruch nehmen wird. Um solche Operationen zeitlich planbar in das System einzubringen, besteht die Möglichkeit, eine Änderung nicht sofort, sondern zu einem späteren Zeitpunkt ausführen zu lassen.

Änderungsoperationen werden zu diesem Zweck immer im Kontext einer umfassenden Transaktion ausgeführt, auch wenn nur eine einzelne Änderungsoperation ausgeführt werden soll. Eine Transaktion wird auf dem Organisationsmodell erst dann ausgeführt, wenn dies ausdrücklich durch ein externes Ereignis initiiert wird (*Commit*). Bis zu diesem Zeitpunkt können weitere Änderungsoperationen in die Transaktion hinzugefügt werden. Die Transaktion selbst kann vor *Commit* auch abgebrochen werden (*Abort* → *Rollback*) oder beliebig oft pausiert (*Suspend*) und durch Hinzufügen von weiteren Änderungsoperationen wieder fortgesetzt werden. Somit können aufwendige Änderung am Organisationsmodell vorbereitet und zu einem festgelegten Zeitpunkt vollzogen werden. Das ist erforderlich, wenn eine bestimmte Änderung zu einem bestimmten Stichtag erfolgen soll, oder wenn die Änderung zu einer Tageszeit ausgeführt werden soll, zu der die Belastung des System durch die Prozeßausführung geringer ist als zu normalen Betriebszeiten.

Für eine Änderungsoperation muß dazu angegeben werden, ob sie den Beginn einer neuen Transaktion darstellt oder im Rahmen welcher bereits bestehenden Transaktion sie zur Ausführung kommt.

Die Einführung dieses Konzeptes zieht die komplette Problematik der Transaktionslogik nach sich wie:

- parallele Ausführung von Transaktionen
- Überschneidungen von Transaktionen
- Ausführung von Änderungsoperationen auf veraltetem Datenbestand
- ...

Aufgrund ihres Umfangs wurden diese Aspekte hier nur kurz erwähnt. Durch die ansatzweise Erwähnung an dieser Stelle soll eine spätere Einbindung der Transaktionslogik für Änderungsoperationen vorbereitet werden. Weitere Informationen zu dieser Problematik und bekannten Lösungen finden sich in der Literatur zu Datenbanksystemen[14].

14 beispielsweise in [HäRa01]

5 Bestimmung von Aufgabenträgern

Der Sinn einer Org.Modell-Komponente im Kontext eines WfMS besteht darin, dem WfMS die Zuordnung von Aufgaben zu konkreten Aufgabenträgern zu ermöglichen.

Teil der Modellierung einer Prozeßvorlage ist unter anderem, anzugeben, welche Mitarbeiter sich als Aufgabenträger für die den Prozeßschritten zugeordneten Aktivitäten qualifizieren. Bei der Instantiierung einer Prozeßvorlage werden diese Angaben für die Aktivitäten mit in die Prozeßinstanz übernommen und bei deren Ausführung als Standardvorgabe verwendet. Abweichend von diesen Standardvorgaben soll die Zuordnungsanweisung auf Instanzebene überschrieben werden können.

Kommt es zur Laufzeit des WfMS bei der Ausführung einer Prozeßinstanz zur Aktivierung eines Prozeßschrittes, so wird das WfMS die Zuordnungsanweisung der zugehörigen Aktivität ermitteln. Unter Zuhilfenahme der Org.Modell-Komponente wird die Zuordnungsanweisung auf konkrete Aufgabenträger aufgelöst und die Aktivität in deren Arbeitslisten zur Bearbeitung angeboten. Abbildung 5-1 veranschaulicht diesen Ablauf.

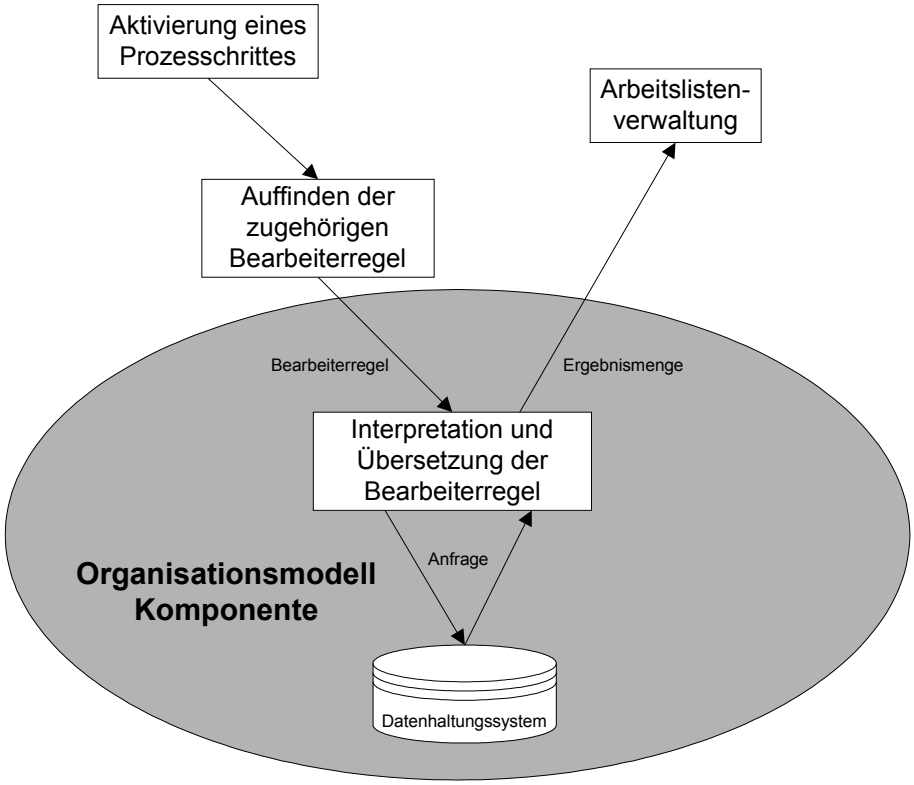


Abbildung 5-1: *Bestimmung von Aufgabenträgern*

[Kubi98] beschreibt, daß selbst bei einer semantisch korrekten Bearbeiterformel (Bestandteil einer Zuordnungsanweisung), die auf einem existierenden Modell aufgelöst wird, die Menge der potentiellen Bearbeiter leer sein kann. Aus diesem Grund ist bei jeder Instantiierung einer Prozeßvorlage ein Prozeßverantwortlicher konkret zu benennen. Liefert die Auflösung einer Zuordnungsanweisung die leere Menge, wird das WfMS einen besonders gekennzeichneten Eintrag in der Arbeitsliste des Prozeßverantwortlichen hinterlassen.

Die Anforderungen an die Zuordnungsanweisungen sind wie folgt:

- Aufgabenträger sollen sich nicht nur durch direkte Angabe des Namens in der Zuordnungsanweisung für eine Aufgabe qualifizieren können, sondern auch implizit z.B. durch die Forderung von Fähigkeiten, die ein Mitarbeiter besitzen muß,und/oder der Zugehörigkeit zu einer Org.Einheit. Da die Stellenbesetzung durch die Mitarbeiter vielleicht nur über eine relativ kurze Zeitspanne Gültigkeit besitzt, wären die direkten Mitarbeiterzuordnungen einer stetigen Anpassung unterworfen.

- Die Zuordnung der Bearbeitermenge zu einer Aktivität sollte auch in Abhängigkeit von der Ausführung einer Prozeßinstanz möglich sein. (Bsp. „Vorgesetzter des Bearbeiters von Aktivität X“). Man spricht in diesem Kontext von abhängigen Bearbeiterzuordnungen.

- Die Zuordnungsanweisungen sollen der Ausdrucksmächtigkeit des Organisationsstrukturmodells aus Abschnitt 3.1 gerecht werden. Insbesondere sollen durch die Zuordnungsanweisungen jegliche Attribute des Modells referenzierbar sein.[15]

- Da die Zuordnungen von Aufgaben zu potentiellen Aufgabenträgern beliebig komplex sein können, soll der Modellierer bei der Formulierung der Zuordnungsanweisungen unterstützt werden. Ein geeignetes Modellierungswerkzeug sollte fehlerhafte Anweisungen erst gar nicht zulassen und den Modellierer bei der manuellen und halbautomatischen Anpassung der Bearbeiterformeln nach Änderungen am Organisationsmodell (siehe Kapitel 6) führen.

- Einer Aktivität sollen Bearbeiterformeln in verschiedenen Prioritäten hinterlegt werden können. Damit ist es möglich, verschiedene Gruppen von Aufgabenträgern nur dann mit einer Aufgabe zu betrauen, wenn die höherpriorisierten Bearbeiterformeln für die Menge der potentiellen Bearbeiter die leere Menge liefern.

Das Resultat der Auflösung einer Zuordnung ist immer eine Menge von Mitarbeitern oder die leere Menge. Die Menge der potentiellen Bearbeiter ist dabei in der Form von Tupeln aus Identifikatoren von Mitarbeitern und Stellen. Somit kann für einen Mitarbeiter bestimmt werden, im Kontext welcher Stelle er eine Aufgabe zu erledigen hat, für den Fall, daß er mehrere Stellen besetzt.

Ein Benutzer meldet sich mit seinem eindeutigen Benutzernamen und dem dazugehörigen Passwort am System an. Da die Beziehung zwischen den Entitäten *Mitarbeiter* und *Stelle* eine (n:m)-Kardinalität besitzt, hat der Benutzer anschließend die Möglichkeit, aus einer Liste von Stellen, die er besetzt, diejenige auszuwählen, deren Funktionen er momentan ausübt. Für jede Stelle eines Mitarbeiters existiert eine separate Arbeitsliste und eine zusätzliche, in der die Aufgaben gesammelt werden, die einem Mitarbeiter direkt oder aufgrund spezieller Fähigkeiten zugewiesen werden. Meldet sich der Mitarbeiter am System an, so werden ihm die Liste für die ausgewählte Stelle und diese allgemeine Liste präsentiert. In die allgemeine Liste werden dann alle Aufgaben eingetragen, die zwar diesem Mitarbeiter, aber keiner seiner konkreten Stellen zugeordnet werden können.

15 Dieser Aspekt wurde in bisherigen Veröffentlichungen meist vernachlässigt. Doch gerade die Möglichkeit, jegliche Attribute referenzieren zu können, birgt einen großen Teil der Ausdrucksmächtigkeit des Modells und der darauf aufsetzenden Sprache.

Bei der Bearbeitung einer Aktivität wird zusätzlich zum Bearbeiter festgehalten, im Kontext welcher Stelle die Aktivität bearbeitet wurde. Dadurch werden die Möglichkeiten der Formulierung von abhängigen Zuordnungen erweitert. Ist nach Bearbeitung einer Aktivität auch die Stelle des Bearbeiters bekannt, können auch Rückschlüsse gezogen werden auf die Org.Einheit in der die Aktivität bearbeitet wurde. Ohne die Information über die Stelle wäre das nicht möglich, da Rückschlüsse nur über (1:1)-, oder (1:n)-Beziehungen eindeutig möglich sind, nicht aber über (n:m)-Beziehungen. Aus diesem Grund wird gefordert, daß das Ergebnis einer Bearbeiterzuordnung aus Tupeln der Form (MitarbeiterID, StelleID) besteht.

Eine Zuordnung selbst ist dabei eine textuelle Darstellung eines logischen Ausdrucks, der angewendet auf ein konkretes Org.Modell für jeden Mitarbeiter zu *wahr* oder *falsch* evaluiert, je nachdem, ob der Mitarbeiter in der Menge der potentiellen Aufgabenträger enthalten ist oder nicht. Um dies zu realisieren, werden im folgenden drei Ansätze näher betrachtet.

Namentlich handelt es sich um die Bestimmung von Aufgabenträgern mit Hilfe von:

- Statischer Selektoren (Abschnitt 5.1)
- eines funktionalen Ansatzes (Abschnitt 5.2)
- eines prädikatenlogischen Ansatzes (Abschnitt 5.3)

Dabei werden jeweils auch die Vor- und Nachteile der einzelnen Ansätze diskutiert.

5.1 Statische Selektoren

5.1.1 Definition der Zuordnungsanweisungen

Der folgende Abschnitt erläutert den Aufbau dieser logischen Ausdrücke.

Obwohl die Syntax der Zuordnungsanweisungen der in [Wied02] verwendeten ähnlich ist, unterscheidet sie sich teilweise in der Benennung der Komponenten, aus denen sie aufgebaut ist.

Definition 5-1: Bearbeiterzuordnung

Eine Bearbeiterzuordnung stellt den elementarsten Teil einer Zuordnungsanweisung dar.
Es sind zwei Arten von Bearbeiterzuordnungen definiert: abhängige und unabhängige. Mit unabhängigen lassen sich Mengen von Bearbeitern durch die Einschränkung auf den Wert eines einzelnen Attributes beschreiben.

<Selektor>.<Attributname> <Vgl.Operator> '<Konstante>'

Abhängige Bearbeiterzuordnungen beschreiben Mengen von Bearbeitern in Abhängigkeit der bisherigen Ausführung der behandelten Prozeßinstanz. Da die Verwaltung der Ausführungshistorie nicht Aufgabe der Org.Modell-Komponente ist, wird die Auflösung der abhängigen Bearbeiterzuordnungen nicht von der Org.Modell-Komponente durchgeführt. Abhängige Bearbeiterzuordnungen haben folgende Form.

<Selektor>.ID = $[OU|OP|A](ActivityID)

Tabelle 5-1 gibt einen Überblick über die als sinnvoll erachteten Selektoren, die auf ein Organisationsmodell anwendbar sind.

In der Tabelle befinden sich jeweils die Benennung des Selektors, die Entität, auf die er angewendet wird, eine Beschreibung der Selektionswirkung, ein Beispiel für die Syntax und ein optionaler Kommentar.

In den Beispielen wird immer auf das Attribut *Name* der angewandten Entität Bezug genommen. Tatsächlich können aber alle Attribute der Entität referenziert werden. Die Verwendung von „=“ als Vergleichsoperator ist ebenfalls keine allgemeine Einschränkung. Die Auswahl der möglichen Vergleichsoperatoren hängt vom Datentyp der referenzierten Attribute ab.

Für Zeichenketten sind außer „=“ beispielsweise die Operatoren „like“ und „caseIgnoreMatch“ denkbar, für numerische Werte „<“, „>“, „>=“ usw.

Bsp.:	*A.Name like 'Meier'*	(Bearbeiter heißt ähnlich wie 'Meier')
	OU.Standort caseIgnoreMatch 'UlM'	(Bearbeiter in einer Org.Einheit in Ulm)
	PGL.Budget > '1000000'	(Leiter einer Projektgruppe mit Budget > 1Mio)

Selektor	Anwendung auf	Name Wirkung	Beispiel	Kommentar
A	Mitarbeiter	“Agent” Direkte Zuordnung eines Mitarbeiter	A.Name = 'Müller'	
RA	Fähigkeit	“RoleAbility” Bearbeiter, die eine Stelle besetzen, die von einer Rolle mit einer Fähigkeit beschrieben ist.	RA.Name = 'Staplerfahren'	Mitarbeiter, die als Staplerfahrer arbeiten
AA	Fähigkeit	“AgentAbility” Bearbeiter, die eine bestimmte Fähigkleit besitzen	AA.Name = 'Staplerfahren'	Mitarbeiter, die zwar nicht zwingendermaßen als Staplerfahrer arbeiten, diese Fähigkeit aber trotzdem besitzen
R	Rolle	“Role” Bearbeiter, die eine Rolle ausführen durch eine Stelle, die sie besetzen	R.Name = 'Krankenschwester'	
R+	Rolle	“Role+” Wie R, nur werden hier abgeleitete Rollen ebenfalls miteinbezogen (auch transitiv)	R+.Name = 'Krankenschwester'	Gilt hier bspw. auch für 'Krankenschwester mit erweiterten Befugnissen'
OP	Stelle	“Org.Position” Bearbeiter, der eine angegebene Stelle besetzt	OP.Name = 'Krankenschwester1'	Die Person, die die Stelle Krankenschwester1 besetzt
OP+	Stelle	“Org.Position+” Vorgesetzter der Stelle OP	OP+.Name = 'Krankenschwester1'	Vorgesetzter der Stelle Krankenschwester1
OP++	Stelle	“Org.Position++” Wie OP+, zusätzlich transitiv	OP++.Name = 'Krankenschwester1'	Alle Vorgesetzten von Krankenschwester1. z.B. auch “Ärztlicher Direktor”
OP-	Stelle	“Org.Position-” Untergebene der Stelle OP	OP-.Name = 'Abteilungsleiter1'	Untergebene der Stelle Abteilungsleiter1
OP--	Stelle	“Org.Position-” Wie OP-, zusätzlich transitiv	OP--.Name = 'Generaldirektor'	Alle untergebenen Stellen von Generaldirektor z.B. auch “Sachbearbeiter1”
OU	Org.Einheit	“Org.Unit” Bearbeiter ist in der angegebenen Org.Einheit	OU.Name = 'Buchhaltung'	Bearbeiter besetzt eine Stelle in der Buchhaltung

Tabelle 5-1: *Selektoren für Bearbeiterzuordnungen (1/3)*

Selektor	Anwendung auf	Name Wirkung	Beispiel	Kommentar
OU+	Org.Einheit	“Org.Unit+” Bearbeiter ist in der übergeordneten Org.Einheit	OU+.Name = 'Buchhaltung'	Bearbeiter ist in der Org.Einheit, die der Buchhaltung übergeordnet ist.
OU++	Org.Einheit	“Org.Unit++” Wie OU+, zusätzlich transitiv	OU++.Name = 'Buchhaltung'	Wie OU+, einschließlich Bearbeiter der Org.Einheiten, die der Buchhaltung transitiv übergeordnet sind.
OU-	Org.Einheit	“Org.Unit-” Bearbeiter ist in der untergeordneten Org.Einheit.	OU-.Name = 'Buchhaltung'	Bearbeiter ist in einer Org.Einheit, die der Buchhaltung untergeordnet ist.
OU--	Org.Einheit	“Org.Unit--” Wie OP-, zusätzlich transitiv	OU--.Name = 'Buchhaltung'	Wie OU-, einschließlich Bearbeiter der Org.Einheiten, die der Buchhaltung transitiv untergeordnet sind.
OUL	Org.Einheit	“Org.UnitLeader” Leiter der Org.Einheit	OUL.Name = 'Buchhaltung'	Bearbeiter ist der Leiter der Buchhaltung.
OGL	Org.Gruppe	“Org.GroupLeader” Leiter der Org.Guppe	OGL.Name = 'Ultimate Cooperation'	Leiter der Kooperationsgruppe “Ultimate Cooperation”
OG	Org.Gruppe	“Org.Group” Bearbeiter besetzt eine Stelle, die zu einer Org.Einheit gehört, welche zu einer Org.Gruppe gehört.	OG.Name = 'Ultimate Cooperation'	Bearbeiter ist an der Org.Gruppe “Ultimate Cooperation” beteiligt
PG	Projektgruppe	“ProjektGroup” Bearbeiter ist Mitglied der Projektgruppe (d.h. er besetzt eine Stelle, die direkt der Projektgruppe angehört, oder die Stelle gehört zu einer Org.Einheit, die Mitglied der Projektgruppe ist)	PG.Name = “Antriebseinheit”	Bearbeiter ist am Projekt “Antriebseinheit” beteiligt.
PG+	Projektgruppe	“ProjektGroup+” Bearbeiter ist Mitglied der übergeordneten Projektgruppe	PG+.Name = “Antriebseinheit”	Bearbeiter ist dem Projekt “Flugzeug” zugeordnet

Tabelle 5-1: *Selektoren für Bearbeiterzuordnungen (2/3)*

Selektor	Anwendung auf	Name Wirkung	Beispiel	Kommentar
PG++	Projektgruppe	“ProjektGroup++” Wie PG+, zusätzlich transitiv	PG++.Name = “Antriebseinheit”	Wie PG+, einschließlich Bearbeiter der Projektgruppen, die der PG “Antriebseinheit” transitiv übergeordnet sind.
PG-	Projektgruppe	“ProjektGroup-” Bearbeiter ist Mitglied der unterrgeordneten Projektgruppe	PG-.Name = “Antriebseinheit”	Bearbeiter ist in einer Projektgruppe, die der PG “Antriebseinheit” untergeordnet ist.
PG--	Projektgruppe	“ProjektGroup--” Wie PG-, zusätzlich transitiv	PG--.Name = “Antriebseinheit”	Wie PG-, einschließlich Bearbeiter der Projektgruppen, die der PG “Antriebseinheit” transitiv untergeordnet sind.
PGL	Projektgruppe	“Proj.GroupLeader” Leiter der Projektgruppe	PGL.Name = “Antriebseinheit”	Leiter der Projektgruppe “Antriebseinheit”
TS	Stelle	“Tech.Superior” Alle fachlichen Vorgesetzten einer Stelle. Für jedes Projekt, an dem die Stelle teilnimmt, gibt es einen Vorgesetzten.	TS.Name = “Ingenieur1”	Alle fachlichen Vorgesetzen von “Ingenieur1”.
TI	Stelle	“Tech.Inferior” Alle fachlich Untergebenen einer Stelle. Für jedes Projekt, das die Stelle leitet, gibt es eine Menge von Untergebenen	TI.Name = “Abteilungsleiter1”	Alle fachlichen Untergebenen von “Abteilungsleiter1”.
SUBST	Stelle	“Substitution” Bearbeiter ist Vertreter der angegebenen Stelle	SUBST.Name = ““Ingenieur1”	Alle Vertreter von “Ingenieur1”. Potentiell für jede Rolle einer.

Tabelle 5-1: *Selektoren für Bearbeiterzuordnungen (3/3)*

Die Auflistung in Tabelle 5-1 ist weder minimal noch vollständig. Nicht minimal deshalb, weil manche Mengen von potentiellen Bearbeitern durch die Kombination von mehreren Bearbeiterzuordnungen dargestellt werden können. Nicht vollständig deshalb, weil es schon aufgrund der Vielzahl von Entitäten und Beziehungen eine viel größere Menge von Selektoren gibt. Die Menge der angegebenen Selektoren stellt lediglich eine für sinnvoll erachtete Auswahl dar.

Definition 5-2: Bearbeiterausdruck

Ein Bearbeiterausdruck ist eine einzelne Bearbeiterzuordnung oder sie setzt sich aus mehreren Bearbeiterzuordnungen zusammen, die mit dem logischen Operator 'AND' verknüpft sind.

$$BA = BZ \wedge BZ \wedge ... \wedge BZ$$

Die Auswahl, die auf Ebene der Bearbeiterzuordnungen mit Hilfe eines Selektors getroffen wurde, läßt sich also durch die Kombination mehrerer Bearbeiterzuordnungen in einem Bearbeiterausdruck weiter einschränken.

Beispiel für einen Bearbeiterausdruck:

$$(R+.Name='Lagerarbeiter') \wedge (AA.Name='Staplerfahren')$$

(Der Bearbeiter besetzt eine Stelle, die durch eine Rolle mit Namen „*Lagerarbeiter*“ oder eine von Lagerarbeiter abgeleitete Rolle beschrieben ist und der Bearbeiter selbst hat die Fähigkeit „*Staplerfahren*“.)

Definition 5-3: Bearbeiterformel

Durch eine Bearbeiterformel läßt sich ausdrücken, welche durch Bearbeiterausdrücke spezifizierten Mengen von potentiellen Aufgabenträgern in die Bearbeitermenge mit eingeschlossen werden sollen.

$$BF = BA \vee BA \vee ... \vee BA$$

Dafür steht in einer Bearbeiterformel der Operator 'OR' für den Einschluss weiterer Mengen zur Verfügung.

Beispiel für eine Bearbeiterformel:

$$((R+.Name='Lagerarbeiter') \wedge (AA.Name='Staplerfahren')) \\ \vee (R+.Name='Staplerfahrer')$$

(Der Mitarbeiter muß die selben Anforderungen wie oben erfüllen oder er muß eine Stelle besetzten, die von einer Rolle mit Namen „*Staplerfahrer*“ oder einer von dieser Rolle abgeleiteten Rolle beschrieben ist.)

Definition 5-4: Zuordnungsanweisung

für jede Aktivität existiert genau eine Zuordnungsanweisung. Diese Zuordnungsanweisung besteht aus einer Liste von Bearbeiterformeln.

$$ZAW = BF \sim BF \sim \ldots \sim BF$$

Die Reihenfolge, in der sich die Bearbeiterformeln in der Zuordnungsanweisung befinden, beschreibt die Reihenfolge in der sie nacheinander ausgewertet werden. Die Auswertung einer weiteren Bearbeiterformel erfolgt jeweils nur dann, wenn aus der Auflösung der vorherigen die leere Menge resultiert.
Damit läßt sich eine Priorität von Mengen potentieller Bearbeiter festlegen.

Beispiel für eine Zuordnungsanweisung:

$$\begin{aligned}&((R+.Name = 'Lagerarbeiter') \wedge (AA.Name = 'Staplerfahren')\\ &\vee (R+.Name = 'Staplerfahrer')\\ &\sim (OUL.Name = 'Lager') \wedge (AA.Name = 'Staplerfahren')\end{aligned}$$

Das angegebene Beispiel besteht aus zwei Bearbeiterformeln. Die anstehende Aufgabe soll von einem Mitarbeiter bearbeitet werden, der eine durch die Rolle '*Lagerarbeiter*' (oder spezieller) beschriebene Stelle besetzt. Zusätzlich soll der Mitarbeiter die Fähigkeit '*Staplerfahren*' besitzen. Alternativ kann die Aufgabe aber auch von einer Stelle mit der Rolle '*Staplerfahrer*' (oder spezieller) ausgeführt werden. Ergibt die Auflösung der ersten Bearbeiterformel die leere Menge, wird die zweite Formel ausgewertet. Diese Formel beschreibt den Bearbeiter als Leiter der Org.Einheit '*Lager*'. Allerdings soll er die Aufgabe nur dann zugewiesen bekommen, wenn er die Fähigkeit '*Staplerfahren*' besitzt. Resultiert auch hier die leere Menge, ermittelt das WfMS den zuständigen Prozeßverantwortlichen. Der Prozeßverantwortliche kann entweder explizit durch Anhängen einer zusätzlichen Bearbeiterformel einer Aktivität referenziert werden oder implizit durch die Implementierung des WfMS geschehen, so daß bei der leeren Menge als Ergebnis automatisch der Prozeßverantwortliche adressiert wird, auch ohne daß dies in der Zuordnungsanweisung ausdrücklich festgehalten ist.

Wenn der Prozeßverantwortliche ausdrücklich als letzter möglicher Bearbeiter festgelegt ist, bedeutet das anschaulich in Abbildung 5-1, daß er schon links oben mit in der Zuordnungsanweisung integriert ist. Wird aber implizit vom System verlangt, den Prozeßverantwortlichen als letzten möglichen Bearbeiter zu adressieren, wird die Org.Modell-Komponente zunächst bei der Auflösung der Zuordnungsanweisung möglicherweise eine leere Menge zurückliefern. Das WfMS muß darauf den Prozeßverantwortlichen selbst bestimmen.

Anmerkung:

Der Ausdruck $(R+.Name = 'Lagerarbeiter') \wedge (AA.Name = 'Staplerfahren')$ ließe sich auch mit einer von 'Lagerarbeiter' abgeleiteten Rolle 'Lagerarbeiter mit Staplerführerschein' modellieren.

Die Bearbeiterformeln weisen also eine Form ähnlich der einer disjunktiven Normalform auf. Eine solche Formel wird gebildet durch die Disjunktion von Konjunktionstermen. Ein Konjunktionsterm wird ausschließlich durch die konjunktive Verknüpfung von Literalen gebildet. Literale sind dabei nichtnegierte oder negierte Variablen. In vollständiger, disjunktiver

Normalform ist eine Formel, wenn jeder Konjunktionsterm alle Literale jeweils entweder in negierter oder nichtnegierter Form enthält[16].

Angewandt auf die beschriebenen Bearbeiterformeln entsprechen die Bearbeiterausdrücke den Konjunktionstermen und die Bearbeiterzuordnungen den Literalen. Bezüglich der DNF weisen die Bearbeiterformeln zwei Einschränkungen auf:

- Bearbeiterformeln haben im Allgemeinen nicht die Form einer vollständigen DNF. Bearbeiterzuordnungen (Literale) werden nur dann in die Bearbeiterausdrücke (Konjunktionsterme) aufgenommen, wenn deren Wertebelegung wesentlich für das Ergebnis der Formel ist. Dadurch wird eine abgekürzte Schreibweise erreicht. Die Angabe jedes Literals (alle möglichen Bearbeiterzuordnungen mit jeweils allen referenzierten Attributen, Vergleichsoperatoren und Konstanten, oder Abhängigkeiten) scheitert in der praktischen Anwendung schon an deren großer Anzahl.

- Bearbeiterausdrücke (Konjunktionsterme) enthalten Literale nur in nichtnegierter Form. Das ist nur scheinbar eine Einschränkung der Ausdrucksmächtigkeit. Durch den Einsatz des Operators „≠“ lassen sich aber alle explizit auszuschließenden Bearbeiterausdrücke auch als positive Literale darstellen.

  Bsp.: $BA = BZ \wedge (A.Name \neq 'Müller')$
  schließt „Müller“ aus dem angegebenen Bearbeiterausdruck *BA* aus.

  Gegebenenfalls ist diese Zuordnung dann in jedem Bearbeiterausdruck ausdrücklich anzugeben, um den tatsächlichen Ausschluss aus der gesamten Bearbeiterformel zu erreichen.

  $$BF = (BZ_1 \wedge (A.Name \neq 'Müller')) \vee (BZ_2 \wedge BZ_3 \wedge (A.Name \neq 'Müller'))$$

  Würde in diesem Beispiel der Mitarbeiter mit Namen „*Müller*“ nicht explizit aus dem zweiten Bearbeiterausdruck der Formel ausgeschlossen, wäre es möglich, daß er sich trotz des Ausschlusses aus dem ersten Bearbeiterausdruck für die Menge der potentiellen Bearbeiter qualifiziert.

Im folgenden wird diese eingeschränkte Form der disjunktiven Normalform als „angepaßte disjunktive Normalform“ oder „angepaßte DNF“ bezeichnet.

5.1.2 Lösen von Abhängigkeiten in Bearbeiterzuordnungen

Bei der Ausführung einer Prozeßinstanz wird für jede ausgeführte Aktivität die ID des Mitarbeiters, und die ID der Stelle, die er zu dem Augenblick besetzte als er die Aktivität ausführte, in der Historie festgehalten. Damit ist es z.B. möglich für die Bearbeitung einer im Ablauf der Prozeßinstanz weiter hinten liegenden Aktivität einen Vorgesetzten des Bearbeiters der referenzierten Aktivität zu beauftragen.

Die Speicherung anderer Attribute, die einen Mitarbeiter für die Ausführung einer Aktivität qualifiziert haben, ist nicht möglich, weil im Allgemeinen nach Auswertung einer

16 Weitere Informationen dazu bieten [Dud93], [Schö00] oder beliebige andere Literatur zu Boolescher Algebra.

Bearbeiterformel nicht mehr nachvollziehbar ist, ob ein Mitarbeiter beispielsweise aufgrund einer bestimmten Fähigkeit, oder der Zugehörigkeit zu einer bestimmten Projektgruppe für die Ausführung dieser Aktivität ausgewählt wurde. Schon bei einer Verknüpfung von Bearbeiterausdrücken mittels „AND“ ist nicht mehr klar, durch welche (von mehreren) Bedingungen sich ein Bearbeiter für die Menge der potentiellen Bearbeiter qualifiziert hat – oder im Falle einer „OR“-Verknüpfung, durch welche der mindestens geforderten Bedingungen. Die einzigen Informationen über die Ausführung eine Aktivität, die in der Ausführungshistorie bezüglich des Org.Modells abgelegt wird, ist die Benutzeridentifikation (MitarbeiterID) des Bearbeiters und die ID der Stelle, in deren Besetzung er die Aktivität bearbeitet hat. Zusätzlich läßt sich aus der Identität der Stelle auch noch die Org.Einheit ermitteln, in welcher die Aktivität bearbeitet wurde. Es besteht keine Möglichkeit auf Basis dieser Daten über eine in Bearbeitung befindliche oder abgeschlossene Aktivität weitere Schlüsse über die Ausführung zu ziehen. Das liegt daran, daß von den drei genannten Entitäten *Org.Einheit*, *Stelle*, und *Mitarbeiter* nur noch (n:m)-Beziehungen zu anderen Entitäten führen. Abhängigkeiten wie „gleiche Rolle wie...“ oder „gleiches Projekt wie...“ sind aufgrund des eingesetzten Metamodells nicht formulierbar, da aus den Daten von *Org.Einheit* / *Stelle* / *Mitarbeiter* nicht eindeutig auf ein Projekt, oder eine Rolle geschlossen werden kann.

In der Syntax der Zuordnungsanweisung ist die Möglichkeit, abhängige Bearbeiterzuordnung zu formulieren zwar vorgesehen, die Org.Modell-Komponente erwartet aber für die Bestimmung von Bearbeitermengen Zuordnungsanweisungen, in denen bereits alle Abhängigkeiten gelöst sind. Lassen sich Abhängigkeiten auf einfache Weise schon vor der Übergabe an die Org.Modell-Komponente auflösen, wird dadurch auch der Kommunikationsaufwand zwischen den einzelnen Komponenten gering gehalten.

Im folgenden wird erläutert, wie das WfMS mit Hilfe der Org.Modell-Komponente abhängige Bearbeiterzuordnungen in unabhängige Bearbeiterzuordnungen verwandeln kann. Die beiden einzigen Informationen zum Bearbeiter einer aktivierten, laufenden oder beendeten Aktivität, die die Ausführungshistorie liefern kann, sind die Angaben der ID des tatsächlichen Bearbeiters und die ID der Stelle, in deren Kontext der Bearbeiter die Aktivität ausführt oder ausgeführt hat.

Eine abhängige Bearbeiterzuordnung hat die allgemeine Form:

<Selektor>.ID = $[OU|OP|A](ActivityID)

Als Beispiel sei genannt:

$OP+.ID = \$OP(ActivityX)$ „Vorgesetzter der Stelle, die Aktivität X ausgeführt hat“

Die Semantik dieser abhängigen Bearbeiterzuordnung erschließt sich folgendermaßen:
„$OP(ActivityX)“ liefert die ID der Stelle, von welcher die Aktivität X ausgeführt wurde. Die Angabe von „OU“, „OP“ oder „A“ bewirkt an dieser Stelle den Bezug auf die ID der Org.Einheit (Organisational Unit), in der sich die Stelle befindet, die ID der Stelle (Organisational Position) von Aktivität X, oder die ID des Mitarbeiters (Agent), der diese Aktivität bearbeitet hat. Wird dieser Wert in obige Bearbeiterzuordnung eingefügt, ergibt sich die unabhängige Bearbeiterzuordnung:

$OP+.ID = '0815'$ „Vorgesetzter der Stelle 0815“

Bevor das WfMS eine Zuordnungsanweisung zur Auflösung an die Org.Modell-Komponente übergibt, ist im WfMS zu prüfen, ob die enthaltenen Formeln abhängige Bearbeiterzuordnungen beinhalten. Unter Zuhilfenahme der Ausführungshistorie kann das WfMS die abhängigen in unabhängige Bearbeiterzuordnungen verwandeln.

Folgende Abhängigkeiten sind dabei möglich:
Als Selektoren kommen all diejenigen in Frage, deren Attributangabe sich auf die Entitäten Org.Einheit, Stelle oder Mitarbeiter beziehen. Das sind die Informationen, die entweder direkt von der Ausführungshistorie bezogen , oder indirekt rekonstruiert werden können.
Laut Tabelle 5-1 beziehen sich folgende Selektoren auf die angegebenen Entitäten:

Org.Einheit	{OU, OU-, OU--, OU+, OU++, OUL}
Stelle	{OP, OP-, OP--, OP+, OP++, TS, TI, P}
Mitarbeiter	{A}

Damit ergeben sich vielfältige Möglichkeiten abhängiger Bearbeiterzuordnungen. Die im folgenden angegebenen sollen hier nur als beispielhafte Auswahl dienen:

- $OP+.ID = \$OP(ActivityX)$
  „disziplinarischer Vorgesetzter des Bearbeiters von Aktivität X“

- $OU.ID = \$OU(ActivityX)$
  „Bearbeiter ist in derselben Org.Einheit wie der Bearbeiter von Aktivität X.“

- $A.ID = \$A(ActivityX)$
  „Bearbeiter ist Bearbeiter von Aktivität X.“

- $OP.ID = \$OP(ActivityX)$
  „Die Stelle, in der der Bearbeiter Aktivität X ausführte.“

- $TI.ID = \$OP(ActivityX)$
  „Die fachlich Untergeordneten des Bearbeiters von Aktivität X“

Die Semantik der nicht angeführten abhängigen Bearbeiterzuordnungen läßt sich leicht erschließen.

Abbildung 5-2 zeigt die Interaktion zwischen den Komponenten, die notwendig ist, um eine abhängigen Bearbeiterzuordnung aufzulösen.

Das Beispiel zeigt, wie das WfMS beim Parsen einer Bearbeiterformel auf eine abhängige Bearbeiterzuordnung stößt. Aus der Historie der betroffenen Prozeßinstanz wird ausfindig gemacht, wer den referenzierten Prozeßschritt bearbeitet hat. Mit dieser Information kann das WfMS die abhängige Bearbeiterzuordnung durch eine unabhängige Bearbeiterzuordnung substituieren.

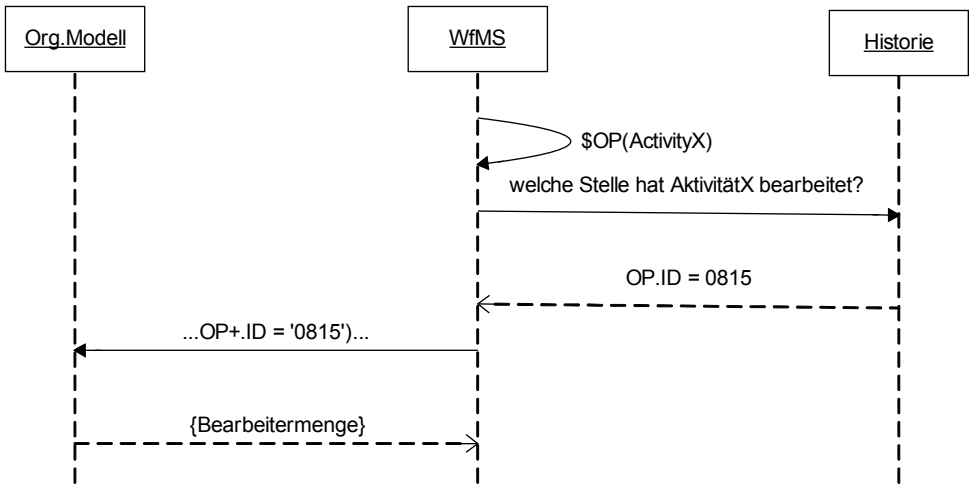


Abbildung 5-2: *Interaktion zur Auflösung abhängiger Zuordnungen*

Einen Sonderfall stellen die Selektoren dar, die sich auf die Entität Org.Einheit beziehen. Die Information, in welcher Org.Einheit eine Aktivität bearbeitet wurde, läßt sich nicht aus der Ausführungshistorie direkt gewinnen. Wohl aber aus der Org.Modell-Komponente, wenn bekannt ist, um welche Stelle es sich handelt. Um diese Information zu gewinnen, existieren zwei unterschiedliche Konzepte:

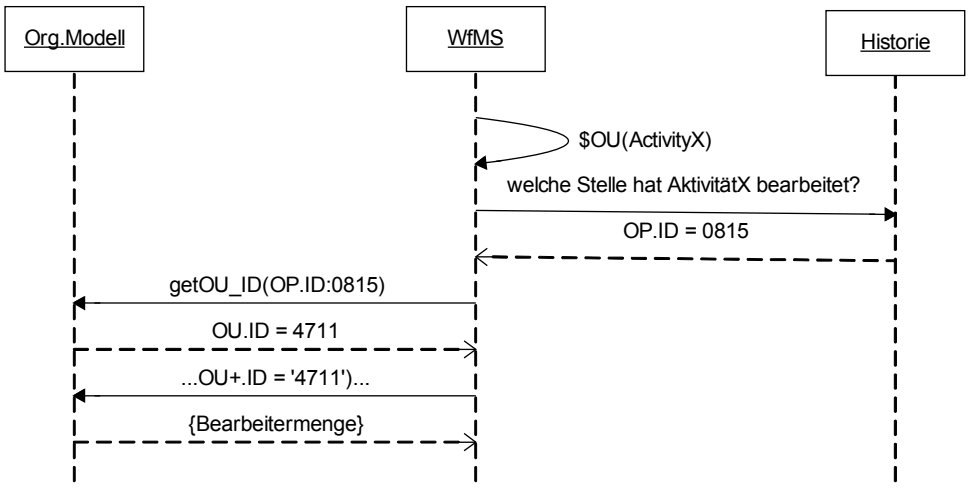


Abbildung 5-3: *Interaktion zur Auflösung abhängiger Zuordnungen (spezieller Fall)*

Der erste Ansatz besteht darin, daß das WfMS zuerst explizit das Org.Modell befragt, um welche Org.Einheit es sich konkret handelt. Daraufhin wird das Ergebnis dieser Anfrage in die Bearbeiterzuordnung geschrieben und im Rahmen einer kompletten Bearbeiterformel nach Auflösung eventuell weiterer vorhandener Abhängigkeiten zur Bestimmung der Aufgabenträger an das Org.Modell geschickt. Abbildung 5-3 veranschaulicht diesen zusätzlichen Kommunikationsaufwand.

Die zweite Möglichkeit, die Information über die zugehörige Org.Einheit zu gewinnen, ist die Hinzunahme weiterer Selektoren für

Org.Einheit {OU, OU-, OU--, OU+, OU++, OUL},

die sich nicht auf Attribute der Entität Org.Einheit, sondern auf Attribute der Entität Stelle beziehen.

Es sind dies:

Selektor	Anwendung auf	Name Wirkung	Beispiel	Kommentar
OUOP	Stelle	"Org.Unit by OP" Bearbeiter ist in derselben Org.Einheit wie die angegebene Stelle	OUOP.ID = '0815'	
OUOP+	Stelle	"Org.Unit+ by OP" Bearbeiter ist in der Org.Einheit, die der Org.Einheit der angegebenen Stelle übergeordnet ist	OUOP+.ID = '0815'	
OUOP++	Stelle	"Org.Unit++ by OP" Wie OUOP+, zusätzlich transitiv	OUOP++.ID = '0815'	Wie OU+, einschließlich Bearbeiter der Org.Einheiten, die der Org.Einheit der angegebenen Stelle transitiv übergeordnet sind.
OUOP-	Stelle	"Org.Unit+ by OP" Bearbeiter ist in einer Org.Einheit, die der Org.Einheit der angegebenen Stelle untergeordnet ist	OUOP-.ID = '0815'	
OUOP--	Stelle	"Org.Unit-- by OP" Wie OUOP-, zusätzlich transitiv	OUOP--.ID = '0815'	Wie OU-, einschließlich Bearbeiter der Org.Einheiten, die der Org.Einheit der angegebenen Stelle transitiv untergeordnet sind.

Tabelle 5-2: *Selektoren für die Bestimmung von Org.Einheiten durch Stellen*

Egal, welche der beiden Möglichkeiten letztendlich implementiert wird, die Org.Modell-Komponente kommt bei der Auflösung von Zuordnungsanweisungen nur mit unabhängigen Bearbeiterzuordnungen in Kontakt.

Ein Entwurfsaspekt ist dabei auch, daß der Kommunikationsaufwand gering gehalten wird. Anstatt ständig Rückfragen von der Org.Modell-Komponente an die Ausführungshistorie weiterreichen zu müssen, muß das WfMS für jede entdeckte Abhängigkeit nur einen Funktionsaufruf an die Ausführungshistorie initiieren (und gegebenenfalls an die Org.Modell-Komponente) und kann danach sämtliche Substitutionen auf einmal vornehmen. Dieses Vorgehen spricht für die erste Möglichkeit. Doch auch dieser Kommunikationsschritt kann noch entfallen, wenn die oben genannten zusätzlichen Selektoren implementiert werden. Allerdings ist anzunehmen, daß die erstgenannte Möglichkeit dem Modellierer eingängiger ist, da die Syntax der Bearbeiterzuordnung auch ohne die zusätzlichen Selektoren schon sehr komplex ist.

Allgemein gilt für beide Konzepte, daß durch die Auflösung der Abhängigkeiten im WfMS selbst jegliche Abhängigkeiten beschrieben werden können, die auf unabhängige Zuordnungen abgebildet werden können. Welche das sind, bestimmt die konkrete Implementierung des WfMS.

5.1.3 Bewertung

Das Konzept der statischen Selektoren bietet nur eine eingeschränkte Ausdrucksmächtigkeit. Diese ist gegeben durch die tatsächlich in einer Implementierung unterstützten Selektoren. Ein Selektor beschreibt dabei eine vorgegebene Beziehung zwischen Elementen des Organisationsmetamodells. Abweichungen von diesen vordefinierten Beziehungen sind nicht möglich. Um die Ausdrucksmächtigkeit dieses Konzeptes zu steigern, muß die Anzahl der verfügbaren Selektoren erhöht werden, was aber der Lesbarkeit der Zuordnungsanweisungen nicht zuträglich ist. Hinzu kommt, daß mit diesem Konzept zwar Verknüpfungen mittels UND und ODER möglich sind, nicht aber Verkettungen von Selektionen.
Ein Beispiel soll dies verdeutlichen:

„Mitarbeiter einer Org.Einheit, die geleitet wird von der Stelle, die auch eine Projektgruppe mit einem Budget von > 1.000.000 leitet.“

Obige Anforderung an die Menge der potentiellen Bearbeiter kann durch das Konzept der statischen Selektoren nicht beschrieben werden.

5.2 Funktionaler Ansatz

Hauptkritikpunkt am oben vorgestellten Konzept der statischen Selektoren ist die fehlende Möglichkeit, Ausdrücke zu verketten . Im folgenden wird ein funktionales Konzept entwickelt, welches die Verkettung durch Navigation mittels Funktionen ermöglicht.

5.2.1 Funktionsunterstützte Bearbeiterzuordnungen

Navigierend bedeutet hier, daß eine Funktion immer entlang einer Beziehung von einer auf eine andere Entität führt (oder auch auf dieselbe, falls die Entität eine Beziehung zu sich selbst unterhält). Um navigierenden Zugriff im Modell zu realisieren, benötigt man für jede Beziehung zwischen den Entitäten jeweils für beide Richtungen eine Funktion. Die Menge der in diesem Ansatz verwendeten Funktionen ist vollständig im Gegensatz zur Menge der Selektoren aus dem vorherigen Abschnitt.

Bei der im folgenden beschriebenen Sprache sind die Bezeichnungen der Entitäten Teil des Wortschatzes. Da bei der Definition von Sprachen und Quellcode zur besseren Verständlichkeit und Internationalisierung nur die englische Sprache zum Einsatz kommt, werden nun für die Entitäten die Entsprechungen im Englischen eingeführt.

Die Entitäten des Metamodells haben folgende Entsprechungen im Englischen:

Deutsch	Englisch
Projektgruppe	ProjectGroup
Org.Gruppe	OrgGroup
Org.Einheit	OrgUnit
Stelle	OrgPosition
Mitarbeiter	Agent
Vertreterregelung	SubstitutionRule
Rolle	Role
Fähigkeit	Ability

Tabelle 5-3: *Entsprechungen der Entitäten im Englischen*

Bei der Übersetzung von *Mitarbeiter* wurde nicht der Begriff *Person* gewählt, sondern der etwas allgemeinere, umfassendere Begriff *Agent*. Damit lassen sich nicht nur natürliche Personen beschreiben, sondern auch andere mögliche Aufgabenträger wie Fertigungsmaschinen oder Rechnerprozesse. Die Annahme, daß ein Agent in Form eines Rechnerprozeßes auftreten kann, widerspricht dabei nicht einmal der Definition von *Mitarbeiter* in Abschnitt 3.1, in der ein Mitarbeiter als Abbildung eines real existierenden mit dem System interagierenden Aufgabenträgers eingeführt wird. Auch wenn die Vorstellung gewöhnungsbedürftig ist, so ist auch ein auf einem Rechner laufender Prozeß in gewisser Weise in der Realität existent, zumindest aber ist er realer als z.B. der abstrakte Ausdruck *Stelle*.

Auf die Punkte für die abkürzende Schreibweise wird verzichtet, um bei der unten angegebenen Syntax Mehrdeutigkeiten zu verhindern.

Tabelle 5-4 gibt einen Überblick über die zur Verfügung stehenden Funktionen. Die Funktionen sind gruppiert nach der Entität, auf die sie angewendet werden. Die zweite Spalte legt den Typ der Funktionswerte fest. Desweiteren ist in der dritten Spalte auch die Semantik der Funktionen angegeben.

Funktion	Funktionswert vom Typ	Semantik
OrgUnit.getSubOrgUnit	OrgUnit	liefert die untergeordneten Org.Einheit
OrgUnit.getSupOrgUnit	OrgUnit	liefert die übergeordneten Org.Einheit
OrgUnit.getSubOrgUnit+	OrgUnit	liefert alle untergeordneten Org.Einheiten (auch transitiv)
OrgUnit.getSupOrgUnit+	OrgUnit	liefert alle übergeordneten Org.Einheiten (auch transitiv)
OrgUnit.getOrgGroup	OrgGroup	liefert die Org.Gruppe, an denen die Org.Einheit teilnimmt
OrgUnit.getProjectGroup	ProjectGroup	liefert die Projekte, an denen die Org.Einheit teilnimmt
OrgUnit.getOrgPosition	OrgPosition	liefert die Stellen der Org.Einheit
OrgUnit.getManager	OrgPosition	liefert den Leiter der Org.Einheit
OrgGroup.getOrgUnit	OrgUnit	liefert die Org.Einheiten, aus denen die Org.Gruppe besteht
OrgGroup.getManager	OrgPosition	liefert den Leiter der Org.Gruppe
OrgPosition.getOrgUnit	OrgUnit	liefert die Org.Einheit, zu der die Stelle gehört
OrgPosition.getManagedOrgUnit	OrgUnit	liefert die Org.Einheiten, die die Stelle leitet
OrgPosition.getManagedOrgGroup	OrgGroup	liefert die Org.Gruppen, die die Stelle leitet
OrgPosition.getProjectGroup	ProjectGroup	liefert die Projektgruppen, an der die Stelle teilnimmt
OrgPosition.getManagedProjectGroup	ProjectGroup	liefert die Projektgruppen, die die Stelle leitet
OrgPosition.getRole	Role	liefert die Rollen, mit denen die Stelle beschrieben ist
OrgPosition.getAgent	Agent	liefert die Mitarbeiter, die diese Stelle besetzen
ProjectGroup.getSubProjectGroup	ProjectGroup	liefert die übergeordnete Projektgruppe
ProjectGroup.getSupProjectGroup	ProjectGroup	liefert die untergeordneten Projektgruppen
ProjectGroup.getSubProjectGroup+	ProjectGroup	liefert alle übergeordnete Projektgruppe (auch transitiv)
ProjectGroup.getSupProjectGroup+	ProjectGroup	liefert alle untergeordneten Projektgruppen (auch transitiv)
ProjectGroup.getManager	OrgPosition	liefert den Leiter der Projektgruppe
ProjectGroup.getOrgUnit	OrgUnit	liefert die beteiligten Org.Einheiten
ProjectGroup.getOrgPosition	OrgPosition	liefert die beteiligten Stellen
Role.getOrgPosition	OrgPosition	liefert die Stellen, die die Rolle beschreibt
Role.getAbility	Ability	liefert die Fähigkeiten, die die Rolle beschreiben
Role.getSubRole	Role	liefert die abgeleiteten, spezialisierten Rollen
Role.getSupRole	Role	liefert die abstraktere, übergeordnete Rolle
Role.getSubRole+	Role	liefert alle abgeleiteten, spezialisierten Rollen (auch transitiv)

Tabelle 5-4: *Funktionen zum navigierenden Zugriff (1/2)*

Funktion	Funktionswert vom Typ	Semantik
Agent.getOrgPosition	OrgPosition	liefert die Stellen des Mitarbeiters
Agent.getAbility	Ability	liefert die Fähigkeiten des Mitarbeiters
Ability.getAgent	Agent	liefert die Mitarbeiter, die diese Fähigkeit besitzen
Ability.getRole	Role	liefert die Rollen, die durch diese Fähigkeiten beschrieben sind

Tabelle 5-4: *Funktionen zum navigierenden Zugriff (2/2)*

Der erste Parameter einer Funktion ist die Entität, auf die sie angewendet wird. In der vorgeschlagenen Punktnotation (siehe unten) ist dieser Parameter nur implizit, da durch die Position der Funktion innerhalb der Bearbeiterzuordnung schon festgelegt ist, auf welche Entität sie sich bezieht. Der zweite Parameter bietet die Möglichkeit einer Selektion auf der Zielmenge der jeweiligen Funktion. Die Angabe einer Selektion ist optional.

Definition 5-5: Bearbeiterzuordnungen unter Verwendung von Funktionen

Der folgende reguläre Ausdruck definiert die Anwendung dieser Funktionen als Bearbeiterzuordnung:

$$<Selektion>.[<Funktionsname>([<Selektion>])]^*$$

Eine Selektion hat dabei folgende Form:

$$<Entitätstyp>.<Attributname><Vgl.Operator>'<Konstante>'$$

An folgenden Beispielen wird die schrittweise Entwicklung einer Bearbeiterzuordnung mit Hilfe von navigierenden Funktionen demonstriert:

Schritt 1: ***(OrgPosition.Name = 'Hausmeister1')***
selektiert die Stelle 'Hausmeister1'

Schritt 2: *(OrgPosition.Name = 'Hausmeister1')****.getAgent()***
ermittelt die Mitarbeiter, die die Stelle 'Hausmeister1' besetzen.

Ein komplexeres Beispiel beschreibt folgende Bearbeitermenge:
„Leiter einer Unterabteilung in 'Ulm' von Abteilung 'Entwicklung'“

Schritt 1: ***(OrgUnit.Name = 'Entwicklung')***

selektiert die Org.Einheiten mit Namen 'Entwicklung' (Funktionswert vom Typ OrgUnit).

Schritt 2: *(OrgUnit.Name = 'Entwicklung')****.getSubOrgUnit(OrgUnit.Standort = 'Ulm')***

davon die untergeordnete Org.Einheiten (Funktionswert vom Typ OrgUnit)
Auf dieser Zielmenge wird eine Selektion vorgenommen, so daß sie nur noch die Org.Einheit am Standort Ulm beinhaltet.

Schritt 3: *(OrgUnit.Name = 'Entwicklung').getSubOrgUnit(OrgUnit.Standort = 'Ulm')* ***.getManager()***

davon die Leiter (Funktionswert vom Typ OrgPosition)
Ohne zusätzliche Selektion.

Schritt 4: *(OrgUnit.Name = 'Entwicklung').getSubOrgUnit(OrgUnit.Standort = 'Ulm') .getManager****.getAgent()***

davon die Mitarbeiter, die diese Stellen besetzen.
Ohne zusätzliche Selektion.

Anmerkungen:

- Die Funktionen können jeweils miteinander verkettet werden, allerdings nur so, daß der Resultattyp der eingeschlossenen Funktion dem Typ der Eingabeparameter der einschließenden Funktion entspricht. Diese Einschränkung läßt sich relativ einfach sicherstellen und dem Modellierer in geeigneter Form nahebringen[17]. Eine solche Verkettung von Funktionen bildet gewissermaßen einen Pfad durch das Metamodell.

- Offensichtlich sind die Funktionswerte von Funktionen potentiell Mengen von Funktionswerten. Das hängt davon ab, ob die Selektionen einzel- oder mengenwertige Ergebnisse liefern.
  Es muß also damit gerechnet werden, daß jede Funktion als Eingabeparameter eine Menge von Parametern erhält.

Abbildung 5-4 zeigt die Entitäten des Metamodells und die Funktionen mittels derer zwischen ihnen navigiert werden kann in Form eines gerichteten Graphen. Bei Kanten, die in beide Richtungen verlaufen, handelt es sich eigentlich um zwei Kanten (jeweils eine in jede Richtung, für eine der Funktionen). Aus Gründen der Übersichtlichkeit wird auf die Darstellung dieser „doppelten“ Kanten verzichtet.

17 Ein Beispiel hierfür sind die modernen Editoren für Quelltexte, die bei Angabe eines Ausdrucks automatisch Vervollständigung im Kontext diese Ausdrucks vorschlagen.

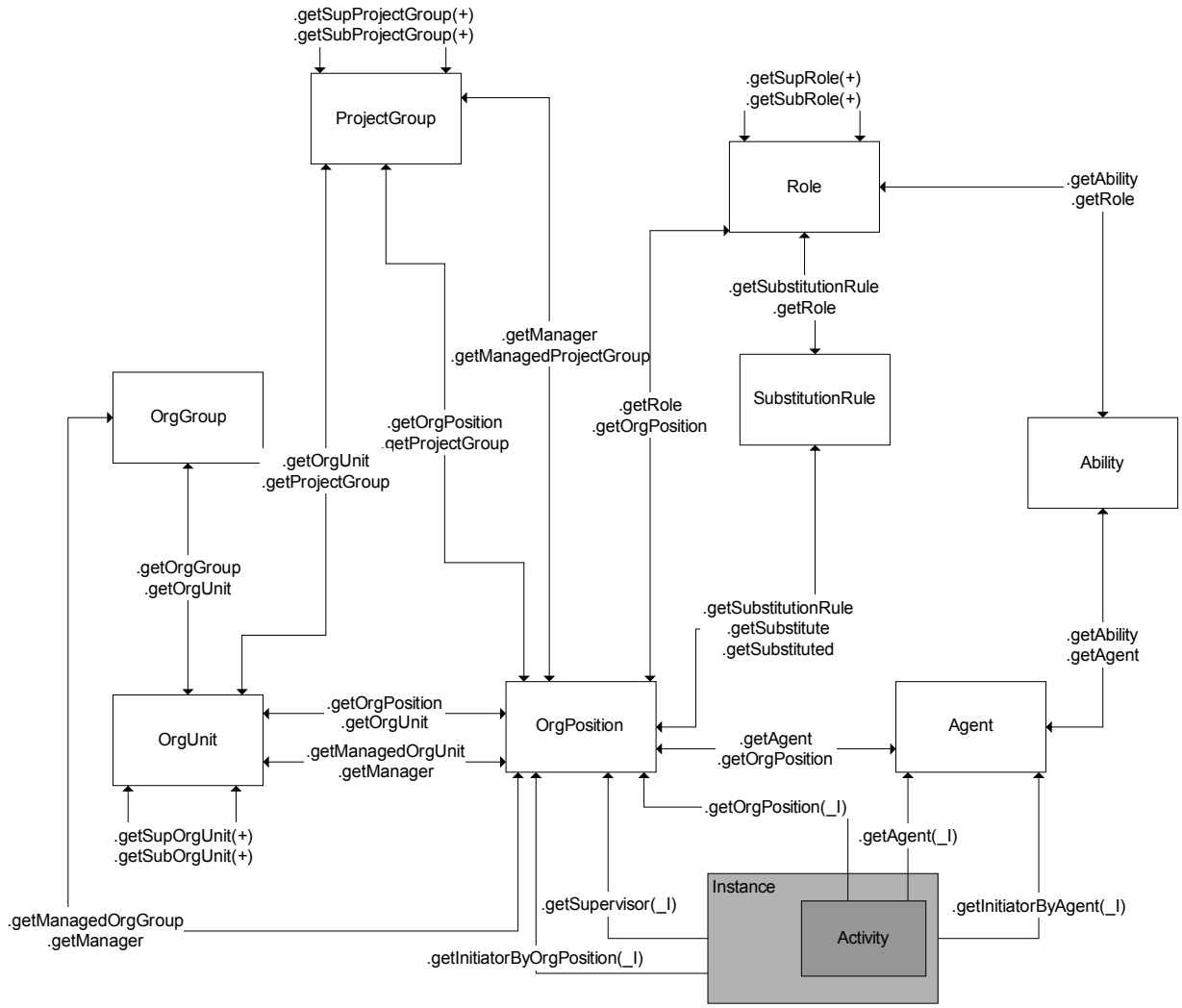


Abbildung 5-4: *Navigation im Modell mittels Funktionen*

Die Richtung der Funktionen ergibt sich aus Tabelle 5-4 und nicht zuletzt auch aus den selbstsprechenden Namen der Funktionen. Die Besonderheiten der Vertreterregelung werden in Abschnitt 5.2.2 und die Modellierung von abhängigen Bearbeiterzuordnungen mittels *„Instance“* und *„Activity“* in Abschnitt 5.2.3 behandelt.

Die oben gezeigten Beispiele führen explizit zu einer Menge von Mitarbeitern. Um konkrete Aufgabenträger ermitteln zu können, muß die letzte Funktion einer Verkettung als Ergebnis zumindest implizit eine Menge von Mitarbeitern liefern. Das ist aber nur dann möglich, wenn für jedes mögliche Ende eines Verkettungspfades eine Semantik hinterlegt ist, die die Bearbeiterzuordnung auf eine Menge von Mitarbeitern zurückführt. Aus Tabelle 5-4 ist ersichtlich, daß jede Entität am Ende eines Verkettungspfades liegen kann. Also muß auch für jede Entität eine „Pfadvervollständigung“ definiert sein.

Diese hat folgende Standardvorgaben:

Nr.	Pfadende	Vervollständigung zu
(1)	OrgPosition	.getAgent
(2)	Role	.getOrgPosition
(3)	OrgUnit	.getOrgPosition
(4)	OrgGroup	.getOrgUnit
(5)	ProjectGroup	.getOrgUnit + .getOrgPosition
(6)	Ability	.getRole + .getAgent

Tabelle 5-5: *Standardsemantiken für die Entitäten des Modells*

Als Beispiel für die Anwendung dient folgende Bearbeiterzuordnung:

(OrgPosition.Name = 'Hausmeister1')

Es handelt sich um das erste Beispiel von Seite 70. Der Pfad endet bei der Entität OrgPosition. Nach Regel (1) aus Tabelle 5-5 wird diese Bearbeiterzuordnung implizit bei der Auflösung durch „ .getAgent()“ zu

*(OrgPosition.Name = 'Hausmeister1')**.getAgent()***

ergänzt.

Ein komplexeres Beispiel ist die Bearbeiterzuordnung

(OrgUnit.Name = 'Entwicklung').getProjectGroup(ProjectGroup.Ort = 'Europa')

Es handelt sich um alle Projekte in Europa, an denen die Org.Einheit 'Entwicklung' teilnimmt. Das sind also alle Mitarbeiter und Org.Einheiten, die an den selben europäischen Projekten arbeiten wie die Org.Einheit 'Entwicklung'. Egal, ob sie direkt an den Projekten beteiligt sind oder dadurch, daß sie eine Stelle in einer Org.Einheit besetzen, die an diesen Projekten beteiligt ist.

Diese Bearbeiterzuordnung wird gemäß der Standardsemantik für ProjectGroup nach den Regeln (5), (3) und (1) ergänzt zu

(OrgUnit.Name = 'Entwicklung').getProjectGroup(ProjectGroup.Ort = 'Europa')
.getOrgUnit().getOrgPosition().getAgent()

und nach (5) und (1) ergänzt zu

(OrgUnit.Name = 'Entwicklung').getProjectGroup(ProjectGroup.Ort = 'Europa')
.getOrgPosition().getAgent()

Bei dieser Pfadvervollständigung handelt es sich um Standardsemantiken für die Entitäten des Modells. Sicher entsprechen diese Vorgaben nicht den Erwartungen aller Anwender. Betrachtet man Regel (6) so werden alle auf der Entität *Ability* endenden Pfade sowohl durch *.getRole()* über *Role* als auch direkt über *.getAgent()* zu *Agent* aufgelöst. Durch das „+“ in Tabelle 5-5 wird angedeutet, daß sich ein Pfad bei der Vervollständigung auch aufspalten kann. Das

bedeutet, daß sich für eine zu erledigende Aufgabe nicht nur Mitarbeiter qualifizieren, die eine Stelle besetzen, die durch eine Rolle mit der entsprechenden Fähigkeit beschrieben ist, sondern auch Mitarbeiter, die ohne eine solche Stelle zu besetzen diese Fähigkeit besitzen. Die Semantik ist also, daß durch „+“ verbundene Vervollständigungsanweisungen den Ausdruck duplizieren, diese dann mit den jeweiligen Vervollständigungen versehen und mit „AND“ verknüpfen.

Um eine größtmögliche Flexibilität des Org.Modells zu erreichen, lassen sich bei der Modellierung einer Organisation diese Vorgaben durch andere, selbstdefinierte Semantiken ersetzen. Diese lassen sich für jede Entität außer *Agent* durch eine beliebige Verkettung der vorhandenen Funktionen beschreiben. Auch dieser Pfad muß nicht zwingend bei der Entität *Agent* enden. Endet der Pfad bei einer anderen Entität als *Agent,* ist der Pfad dieser Entität an die Bearbeiterzuordnung anzuhängen. Allerdings ist bei der Änderung der Semantik einer Entität sicherzustellen, daß die Zyklenfreiheit innerhalb der Pfadvervollständigung erhalten bleibt, da sonst eine Auflösung der Bearbeiterzuordnungen nicht mehr möglich ist.

Beispielsweise ist hier ein Mitarbeiter mit einer Hausmeisterstelle genannt, der zwar einen Staplerführerschein besitzt, in dessen Stellenprofil „Staplerfahren“ aber nicht auftaucht. Daß dieser Mitarbeiter ebenfalls für die Bearbeitung dieser Aufgabe in Frage kommt, muß also nicht unbedingt in der Absicht des Modellierers sein. Ein Modellierer kann also beispielsweise mit der Regel (6) nicht einverstanden sein. Der Modellierer würde also die Regel (6) für die Entität *Ability* von

<.getRole + .getAgent> zu *<.getRole>*

ändern.

5.2.2 Vertreterregelung

Im folgenden wird darauf eingegangen, wie die bereits erwähnte Möglichkeit zur Beschreibung von Vertreterregelungen mit dem funktionalen Ansatz umgesetzt wird.
Eine Vertreterregelung kann betrachtet werden als Beziehung zwischen zwei Stellen und einer Rolle. Die Semantik dabei ist, daß eine Stelle in genau einer Rolle von einer anderen Stelle vertreten werden kann.

Funktion	Funktionswert vom Typ	Semantik
Role.getSubstitutionRule	SubstitutionRule	liefert alle Vertreterregeln, die für diese Rolle gelten
SubstitutionRule.getSubstitute	OrgPosition	liefert die Stelle des Vertreters dieser Regel
SubstitutionRule.getSubstituted	OrgPosition	liefert die Stelle des Vertretenen
SubstitutionRule.getRole	Role	liefert die Rolle, für die diese Vertreterregel gilt
OrgPosition.getSubstitutionRule	SubstitutionRule	liefert die Vertreterregeln, die für eine Stelle eingerichtet sind

Tabelle 5-6: *Funktionen im Zusammenhang mit SubstitutionRule*

In Tabelle 5-6 sind die Funktionen im Zusammenhang mit *SubstitutionRule* aufgelistet. Auch diese Funktionen erlauben nach ihrer Anwendung eine Selektion auf der Zielmenge. Eine

Besonderheit ist dabei die Funktion *OrgPosition.getSubstitutionRule*. Eine Selektion, die dieser Funktion folgt, kann sich nicht nur auf Attribute der Entität *SubstitutionRule,* sondern auch auf Attribute der Entität *Role* beziehen.

Einige Beispiele veranschaulichen die Anwendung der Vertreterregelungen.

- *(OrgPosition.Name = 'Hausmeister1').getSubstitutionRule(Role.Name ='Rasenpfleger').getSubstitute()*

  liefert die Stellen, die die Stelle '*Hausmeister1*' in der Rolle '*Rasenpfleger*' vertreten.

- *(Role.Name = 'Krankenschwester').getSubstitutionRule().getSubstitute()*

  beschreibt alle Stellen, die eine Rolle '*Krankenschwester*' vertreten.

- *(Role.Name = 'Krankenschwester').getSubstitutionRule().getSubstituted()*

  beschreibt alle Stellen, die in der Rolle '*Krankenschwester'* vertreten werden.

- *(OrgPosition.Name = 'Generaldirektor') .getSubstitutionRule(SubstitutionRule.Zeitraum = '2005').getSubstitute()*

  Alle Stellen, die die Stelle '*Generaldirektor*' im Jahr 2005 vertreten (egal in welcher Rolle).

Als Standardsemantik für die Pfadvervollständigung ist für die Entität *SubstitutionRule* <*.getSubstitute*> definiert. Unter Berücksichtigung dieser Regel lassen sich obige Bearbeiterzuordnungen (1, 2 und 4) noch weiter verkürzt darstellen als:

- *(OrgPosition.Name = 'Hausmeister1').getSubstitutionRule(Role.Name ='Rasenpfleger')*

- *(Role.Name = 'Krankenschwester').getSubstitutionRule()*

- *(OrgPosition.Name = 'Generaldirektor').getSubstitutionRule(SubstitutionRule.Zeitraum = '2005')*

5.2.3 Abhängige Bearbeiterzuordnungen

Auch die Einbeziehung von Abhängigkeiten in die Bearbeiterzuordnungen ermöglicht der funktionale Ansatz. Zu diesem Zweck werden für die Betrachtung des funktionalen Ansatzes zwei neue fiktive Entitäten *Activity* und *Instance* mit folgenden Funktionen eingeführt. Wie sich diese in das Metamodell integrieren zeigt Abbildung 5-4 auf Seite 72.

Funktion	Funktionswert vom Typ	Semantik
Activity.getAgent	Agent	liefert den Mitarbeiter, der die Aktivität bearbeitet hat
Activity.getOrgPosition	OrgPosition	liefert die Stelle, in deren Besetzung der Mitarbeiter die Aktivität bearbeitet hat
Instance.getInitiatorAsOrgPosition	OrgPosition	liefert die Stelle des Initiators der Prozessinstanz
Instance.getInitiatorAsAgent	Agent	liefert die MitarbeiterID des Initiators der Prozessinstanz
Instance.getSupervisor	OrgPosition	liefert die Stelle des Prozessverantwortlichen der Prozessinstanz

Tabelle 5-7: *Funktionen für abhängige Bearbeiterzuordnungen*

Anmerkungen:

- Aus Tabelle 5-4, 5-6 und 5-7 ist ersichtlich, daß keine Funktionen existieren, die zu den Entitäten *Activity* und *Instance* führen. Folglich können diese Entitäten nur am Anfang einer Bearbeiterzuordnung referenziert werden.

- Zur Selektion bietet die Entität *Activity* nur das Attribut *ID*. Als Vergleichsoperator ist nur der Gleichheitsoperator erlaubt („=“) .

- Um in den Kontext der Prozeßinstanz zu gelangen, ist keine Selektionsangabe nötig/möglich, da es sich um einen Bezug auf die Metaebene der Aktivität handelt.

- Die Funktion *getInitiator()* steht in zwei Versionen zur Verfügung - als *getInitiatorAsAgent()* und *getInitiatorAsOrgPosition()*. Damit läßt sich für den Initiator eines Prozeßes sowohl der Mitarbeiter selbst, als auch die Stelle in deren Ausübung er den Prozeß gestartet hat, ermitteln. Für die Funktion *getSupervisor()* ist dies nicht vorgesehen, da der Prozeßverantwortliche nur durch eine Stelle festgelegt ist. Ein Prozeßverantwortlicher wird entweder durch die Prozeßvorlage oder davon abweichend durch den Initiator der Instanz bestimmt.

- Die Möglichkeit, die Org.Einheit zu bestimmen, in der die Aktivität bearbeitet wurde, wird nicht explizit durch eigene Funktionen bereitgestellt. Um an diese zu gelangen, kann der Modellierer mittels der Funktion *.getOrgUnit()* zur Org.Einheit navigieren. Folgendes Beispiel verdeutlicht dies:

  *(Activity.ID = '0815').getOrgPosition()****.getOrgUnit()***

Weitere Beispiele demonstrieren die Anwendung der neu eingeführten fiktiven Entitäten:

(Activity.ID = '0815').getOrgPosition()
Die Stelle, die die Aktivität 0815 bearbeitet hat.

(Activity.ID = '0815').getOrgPosition().getOrgUnit().getManager()
Der Vorgesetzte des Bearbeiters der Aktivität 0815

Instance.getInitiatorAsOrgPosition()
Die Stelle des Initiators der Prozeßinstanz

Instance.getInitiatorAsAgent(Agent.Wohnort = 'Ulm')
Der Initiator der Prozeßinstanz, falls dieser in Ulm wohnt

Instance.getSupervisor.getOrgUnit()
Die Org.Einheit, zu der der Prozeßverantwortliche der Instanz gehört

Wie beim Konzept der statischen Selektoren wird auch hier eine minimale Intelligenz im WfMS vorausgesetzt, welche die Org.Modell-Komponente bei der Auflösung von Bearbeiterformeln unterstützt. Bevor das WfMS eine Bearbeiterformel zur Auflösung an die Org.Modell-Komponente übergibt, stellt sie durch einen einfachen Parser fest, ob sich abhängige Bearbeiterzuordnungen in ihr befinden. Diese können vom WfMS nach einer Interaktion mit der Historienkomponente auf einfache Weise durch unabhängige Terme ersetzt werden. Somit sind, wenn die Bearbeiterformel an die Org.Modell-Komponente übergeben wird, nur noch unabhängige Bearbeiterzuordnungen aufzulösen.
Tabelle 5-8 zeigt, wie die Abhängigkeiten vom WfMS ersetzt werden.

Ursprüngliche Abhängigkeit	Ersetzung durch
(Activity.ID = '0815').getAgent()	(Agent.ID = '007')
(Activity.ID = '0815').getOrgPosition()	(OrgPosition.ID = '4711')
Instance.getInitiatorAsOrgPosition()	(OrgPosition.ID = '4711')
Instance.getInitiatorAsAgent()	(Agent.ID = '007')
Instance.getSupervisor()	(OrgPosition.ID = '4711')

Tabelle 5-8: *Substitution der Abhängigkeiten*

Die in der Tabelle genannten Werte für die Konstanten sind nur als Beispiel zu verstehen. Trifft das WfMS beim Parsen auf eine konkrete ID einer Aktivität, so wird es je nach angewandter Funktion die ID des Bearbeiters oder die ID der entsprechenden Stelle von der Historien-komponente ermitteln und diese in die Substitution einsetzen.

Auch für diese Entitäten sind Standardsemantiken für eine Pfadvervollständigung definiert.

Nr.	Pfadende	Vervollständigung zu
(7)	Activity	.getAgent
(8)	Instance	.getSupervisor

Tabelle 5-9: *Standardsemantiken für die fiktiven Entitäten*

5.2.4 Zusammenfassung und Bewertung

Diese von Funktionen unterstützten Bearbeiterzuordnungen können wie die in Abschnitt 5.1 vorgestellten Bearbeiterzuordnungen mit den logischen Operatoren *„AND“*und *„OR“* zu Bearbeiterformeln verknüpft werden.

Die Entscheidung, ob für die Verknüpfung die Beschränkung auf die Form einer angepaßten disjunktiven Normalform (siehe Seite 61) aufrechterhalten werden soll, wird von zwei Faktoren beeinflußt:

- Zum einen ist es für den Parser, der die Formeln erkennen soll, einfacher mit einer solchen restriktiven Form zurechtzukommen, da hierbei nur die Bindungsprioritäten der Operatoren berücksichtigt werden müssen und eine beliebige (aber dennoch korrekte) Klammerung nicht erlaubt ist. In der Theorie ist das aber keine Einschränkung der Ausdrucksmächtigkeit der Formeln, da nachweisbar ist, daß auch durch eine disjunktive Normalform alle Bedingungen ausgedrückt werden können, die mit beliebig geklammerten Ausdrücken beschrieben werden können.

- Andererseits stellt sich die Frage, ob die Komplexität, die diese Formeln in disjunktiver Normalform annehmen, dem Modellierer zufallen soll, wenn technisch die Möglichkeit besteht, die gewünschte Bedingung durch eine beliebig korrekt geklammerte Formel auch einfacher auszudrücken.

Diese Frage zu beantworten ist zwar eher eine Aufgabe eines konkreten Entwurfs, soll aber an dieser Stelle vorweggenommen werden. Es lohnt sich tatsächlich, den höheren Aufwand bei der Implementierung in Kauf zu nehmen und dem Modellieren die Möglichkeit zu bieten, die Bearbeiterzuordnungen bei Verknüpfungen mittels *„AND“* und *„OR“* beliebig zu klammern. *„AND“* hat dabei die Wirkung einer Schnittoperation auf der Menge der potentiellen Bearbeiter, *„OR“* bewirkt eine Vereinigung der Mengen.

Auch explizite Ausschlüsse aus der Menge der potentiellen Bearbeiter lassen sich damit formulieren:

```
....AND (OrgPosition.Name != 'Hausmeister1')
```

Um aber die Herangehensweise an die Formulierung zu erleichtern, wird zusätzlich der Operator „EXCEPT“ für explizite Ausschlüsse eingeführt. Obiges Beispiel würde mit diesem Operator folgendermaßen aussehen:

```
...EXCEPT (OrgPosition.Name = 'Hausmeister1')
```

Die Bearbeiterformeln wiederum lassen sich durch priorisierte Aneinanderreihung zu Zuordnungsanweisungen gruppieren. Die jeweils nächste Bearbeiterformel einer Zuordnungsanweisung wird nur dann aufgelöst, wenn die vorherige bei der Auflösung zur leeren Menge evaluiert. Neben der schon beschriebenen Möglichkeit, explizite Vertreterregelungen zu formulieren, bietet sich hiermit die Möglichkeit, einen Vertreter implizit durch Benennung in einer Zuordnungsanweisung niedrigerer Priorität zu definieren.

Gegenüber dem Konzept der statischen Selektoren hat der funktionsorientierte Ansatz den Vorteil, daß er beliebig verkettete Bearbeiterzuordnungen ermöglicht. Die Selektoren beschreiben gewissermaßen einen festen Pfad innerhalb des Metamodells. Die Funktionen sind

dahingehend flexibler, als daß der Modellierer beim Festlegen der Bearbeiterzuordnungen selbst im Metamodell navigieren kann. Damit sind beliebig geschachtelte Terme möglich. Zusätzlich ist es möglich, bei jeder Funktion eine Selektion auf deren Zielmenge mit anzugeben, und nicht nur auf der ersten Entität des Pfades, wie bei den Selektoren.

5.3 Prädikatenlogischer Ansatz

Ein komplett anderes Konzept ist, die Menge der potentiellen Bearbeiter einer Aktivität nicht konstruktiv durch Funktionen oder Selektoren zu beschreiben, sondern deskriptiv durch prädikatenlogische Ausdrücke. Diese Betrachtung lehnt sich sehr an *Prolog*[18] als wohl bekannteste logische Programmiersprache an.

5.3.1 Prädikatenlogische Bearbeiterformeln

Das gesamte Wissen über die Struktur der Organisation wird bei diesem Konzept als Faktenbasis betrachtet. Die Fakten dieser Wissensbasis leiten sich direkt aus dem Vorhandensein eines Datensatzes (Atom) in einer Entität oder einer Beziehung eines bestimmten Typs zwischen Datensätzen ab.

Eine Zuordnungsanweisung kann nun direkt aus Fakten formuliert werden oder aus vorher definierten Regeln, die sich wiederum auf diese Fakten stützen. Bei der Auflösung der Zuordnungsanweisungen werden freie Variablen mit Atomen belegt (Unifikation) und anschließend wird systematisch versucht, die Anweisung zu falsifizieren. Passende Belegungen (matches) sind die Ergebnisse, die in die Menge der potentiellen Bearbeiter mit aufgenommen werden.

Das Paradigma der Prädikatenlogik und deren Umsetzung in Prolog wird in der einschlägigen Literatur zu diesem Thema ausführlich behandelt. Exemplarisch sei hier [CloMe03] genannt. Deshalb wird an dieser Stelle auf genauere Ausführungen verzichtet.

Die Semantik der verwendeten Prädikate und Klauseln dürfte ausreichend durch die sprechende Namenswahl erkennbar sein. Ein Beispiel verdeutlicht das prädikatenlogische Konzept:

Attribut(A1), hatName(A1, 'Bezeichnung'), hatWert(A1, 'Krankenschwester'), Rolle(_R),
hatAttribut(_R,A1), Stelle(_S), RolleBeschreibtStelle(_R, _S), Attribut(A2),
hatName(A2, 'Bezeichnung'), hatWert(A2, 'Station5'),
Org.Einheit(_O), hatAttribut(_O, A2),
not(StelleInOrg.Einheit(_S,_O)), Mitarbeiter(M), MitarbeiterBesetztStelle(M, _S)

Dieser Ausdruck ist zu lesen als:

„Es gibt ein Attribut A1“
und „A1 hat den Namen 'Bezeichnung'“
und „A1 hat den Wert 'Krankenschwester'“
und „Es gibt eine Rolle _R“
und „_R hat das Attribut A1“
und „Es gibt eine Stelle _S“
und „Die Rolle _R beschreibt die Stelle _S“

18 Prolog ist eine Programmiersprache mit prädikatenlogischem Paradigma (siehe [CloMe03])

und „Es gibt ein Attribut A2“
und „A2 hat den Namen 'Bezeichnung'“
und „A2 hat den Wert 'Station5'“
und „Es gibt eine Org.Einheit _O“
und „Die Org.Einheit _O hat das Attribut A2“
und „Die Stelle _S befindet sich nicht in der Org.Einheit _O“
und „Es gibt einen Mitarbeiter M“
und „Mitarbeiter M besetzt die Stelle _S“

Die angegebene Zuordnungsanweisung beschreibt also die Menge der potentiellen Bearbeiter als Mitarbeiter, die eine Stelle besetzen, die durch eine Rolle Krankenschwester beschrieben ist, aber nicht zur Org.Einheit Station5 gehört.

Bei der Auflösung wird das Attribut A1 mit dem Wert *Krankenschwester* und das Attribut A2 mit dem Wert *Station5* unifiziert. Danach wird systematisch versucht, mögliche Belegungen für die Variablen_R, _S, _O und M zu finden. Der Unterstrich vor den Variablen R, S und O bewirkt, daß deren konkrete Belegungen für die Ergebnisse nicht von Bedeutung sind.

Durch

Krankenschwester(M) :- Attribut(A), hatName(A, 'Bezeichnung'), hatWert(A, 'Krankenschwester'), Rolle(_R), hatAttribut(_R,A1), Stelle(_S), RolleBeschreibtStelle(_R, _S), Mitarbeiter(M), MitarbeiterBesetztStelle(M, _S)

läßt sich eine Klausel als neue Regel definieren, und in der Form

Krankenschwester(M), Attribut(A), hatName(A, 'Name'), hatWert(A, 'Maier'), Mitarbeiter(M), not(hatAttribut(M, A))

(Alle Krankenschwestern außer 'Maier')

wiederum in anderen Klauseln oder Regeln einsetzen.

5.3.2 Abhängigkeiten

Auch die Unterstützung von abhängigen Bearbeiterzuordnungen läßt sich nahtlos in das prädikatenlogische Konzept integrieren. Wie bei den zuvor vorgestellten Ansätzen übernimmt auch hier das WfMS die einfache Aufgabe der Ersetzung von Abhängigkeiten.

Abhängigkeit	Substitution	Semantik
bearbeitetVonStelle(AktivitätX, S)	Attribut(A), hatName(A, 'ID'), hatWert(A, '0815'), hatAttribut(S, A)	S ist die Stelle, die AktivitätX bearbeitet hat.
bearbeitetVonMitarbeiter(AktivitätX, M)	Attribut(A), hatName(A, 'ID'), hatWert(A, '007'), hatAttribut(M, A)	M ist der Mitarbeiter, der AktivitätX bearbeitet hat.

Tabelle 5-10: *Prädikate für abhängige Zuordnungen*

Die Prädikate der abhängigen Zuordnungen zeigt Tabelle 5-10. Wie beim funktionalen Konzept ist auch hier ein besonderes Prädikat für die Org.Einheit, in der die Aktivität

bearbeitet wurde, nicht nötig, da der Modellierer von einer Stelle S selbständig zur Entität Org.Einheit navigieren kann.

Die in der Tabelle genannten Werte für die Konstanten sind auch hier nur als Beispiel zu verstehen. Trifft das WfMS beim Parsen auf eine konkrete ID einer Aktivität, so wird es je nach angewandtem Prädikat die ID des Bearbeiters, oder die ID der entsprechenden Stelle von der Historienkomponente ermitteln und diese in Substitution einsetzen. Das ist auch der Grund, warum an dieser Stelle keine namensgleichen Prädikate *bearbeitetVon(X, Y)* eingesetzt werden können, die sich nur in den Parametertypen unterscheiden. Das WfMS muß erkennen können, ob nach der ID der Stelle, oder der ID des Mitarbeiters zu suchen ist.

5.3.3 Bewertung

Mit der Prädikatenlogik erster Stufe, die Prolog zugrunde liegt, läßt sich die gesamte Mengentheorie formalisieren. Somit kann beim Einsatz dieses Konzeptes die gesamte Ausdrucksmächtigkeit des Organisationsmetamodells genutzt werden, insbesondere ist dabei auch die implizite Unterstützung rekursiver Beziehungen interessant.
Negativ fällt allerdings auf, daß schon einfache Bearbeiterformeln schnell unübersichtlich werden.

5.4 Zusammenfassung und Vergleich der Konzepte

Positiv bei allen drei vorgestellten Ansätzen fällt auf, daß die in Abschnitt 3.1 auf Seite 27 beschriebene Semantik zur transitiven Mitgliedschaft an Projektgruppen nicht einmal starr Anwendung finden muß. Alle drei Konzepte bieten die Möglichkeit genau eine Projektgruppe auszuwählen oder auch alle über- oder untergeordneten. Das selbe gilt auch für die Hierarchien der Org.Einheiten und der Rollen.

Abschließend werden die grundlegenden Eigenschaften der drei vorgestellten Konzepte sowie deren Vor- und Nachteile gegeneinander abgewägt.

Offensichtlich handelt es sich beim ersten Ansatz (Selektoren) um eine vereinfachte Herangehensweise des funktionalen Konzepts. Ein Selektor ist letztendlich nichts anderes als ein vordefinierter Pfad durch das Metamodell, welcher bei der Entität *Agent* endet. Die Angabe einer Selektion ist dabei aber nur für eine Entität möglich: für die, die den Anfang des Pfades markiert. Da sich die Selektoren nicht verketten oder auf eine andere Art und Weise miteinander in Beziehung setzen lassen, ist die Ausdrucksmächtigkeit der Anfragesprache sehr eingeschränkt und hängt direkt ab von der Auswahl der tatsächlich zur Verfügung gestellten Selektoren. Als vollständig ist die Menge der Selektoren anzusehen, wenn für jeden möglichen Pfad ein solcher angegeben wird. Aber schon die relativ eingeschränkte Menge aus Tabelle 5-1 wirkt unübersichtlich und nicht besonders intuitiv. Insbesondere die Umsetzung zur Auflösung von Abhängigkeiten in Bearbeiterzuordnungen wirkt alles andere als ästhetisch.

Abhilfe gegenüber diesem starren Konzept schafft die Verwendung von Funktionen zur freien Navigation im Organisationsmetamodell. Für jede Beziehung zwischen zwei Entitäten existiert für jede Richtung eine Funktion, um die Beziehung in einer Bearbeiterzuordnung referenzieren zu können. Durch konsekutive Anwendung dieser Funktionen läßt sich ein Pfad im Metamodell frei beschreiben. Außerdem bietet dieses Konzept bei jeder Anwendung einer Funktion die Möglichkeit einer zusätzlichen Selektion auf ihrer Zielmenge. Die Angabe von

(Standard-/Benutzer-)Semantiken zur impliziten Pfadvervollständigung vereinfacht die Formulierung der Bearbeiterzuordnungen. Die Einbeziehung von Abhängigkeiten wirkt hier deutlich eleganter als beim ersten Ansatz.

Keine konstruktive, sondern eine deskriptive Herangehensweise weist das dritte vorgestellte Konzept auf. Mit den Mitteln der Prädikatenlogik erster Stufe werden Mengen von Mitarbeitern beschrieben, die sich für die Bearbeitung einer Aufgabe qualifizieren. Durch die Anlehnung an die logische Sprache *Prolog* und die präzise Definition der Prädikatenlogik hat dieses Konzept einen besonderen Charme. Auch die Integration der Abhängigkeiten läßt sich gut bewerkstelligen. Allerdings sind die prädikatenlogischen Formulierungen bereits bei vergleichsweise einfachen Bearbeiterformeln kaum noch zu überschauen.
Zudem wird auch deutlich, daß die Verbindung von Attributen zu Entitäten jedesmal explizit hergestellt werden muß, was die Ausdrücke nicht gerade einfach hält.

Außerdem ist die ungewohnte Herangehensweise bei der Formulierung prädikatenlogischer Ausdrücke vielen Anwendern nicht geläufig und wird diesen auch nur schwer nahezubringen sein. Die unausgesprochene Antipathie, die viele Anwender den logischen Sprachen entgegenbringen und die auch hier zum Tragen kommen würde, spricht gegen den Einsatz dieses Konzeptes zur Ermittlung von Aufgabenträgern. Die genannten Argumente werden in Tabelle 5-11 noch einmal übersichtlich gegenübergestellt.

	Selektoren	**Funktionsorientiert**	**Prädikatenlogisch**
Verständlichkeit	0	+	0
Ausdrucksmächtigkeit	0	+	+
Vollständigkeit	-	+	+
Verknüpfung*(AND / OR)*	+	+	+
Verkettung	-	+	0
Formulierung von Abhängigkeiten	0	0	+

Tabelle 5-11: *Vergleich der verschiedenen Konzepte*

Die wichtigsten Eigenschaften sind Verständlichkeit und Ausdrucksmächtigkeit. Die vier genannten Eigenschaften *Vollständigkeit*, *Verknüpfung*, *Verkettung* und *Formulierung von Abhängigkeiten* sind Teilaspekte der Ausdrucksmächtigkeit. Für die Symbole in der Tabelle gilt folgende Legende:

- „-“: Eigenschaft nicht vorhanden oder schlecht in das Konzept integrierbar
- „0“: Eigenschaft vorhanden
- „+“: Eigenschaft vorhanden und gut in das Konzept integriert

So ist das funktionsorientierte Konzept die beste Wahl für die geforderte Ausdrucksmächtigkeit und Verständlichkeit. In den folgenden Betrachtungen zu einem konkreten Entwurf wird deshalb nur noch auf den funktionsorientierten Ansatz Bezug genommen.

6 Anpassung von Bearbeiterformeln nach Änderungsoperationen

Bearbeiterformeln referenzieren immer ein konkretes Organisationsmodell. Werden Änderungen durch die in Abschnitt 4.2.1 vorgestellten Änderungsoperationen am Organisationsmodell vorgenommen, so muß sichergestellt werden, daß die Bearbeiterformeln auch nach diesen Änderungen konsistent bleiben. Da sich die Bearbeiterformeln aus Bearbeiterzuordnungen zusammensetzen und die Abhängigkeiten zum Organisationsmodell letztendlich in ihnen zu finden sind, wird im folgenden auf den Einfluß von Änderungsoperationen auf die Bearbeiterzuordnungen eingegangen. Dieses Kapitel beschreibt Situationen, in denen eine Anpassung von Bearbeiterzuordnungen grundsätzlich auszuschließen ist, und solche in denen mit Anpassungen zu rechnen ist. Des weiteren wird diskutiert, wie und wann diese vorzunehmen sind.

Nicht besprochen werden hier die direkten Auswirkungen von Änderungsoperationen auf die Menge potentieller Bearbeiter. Diese ändert sich möglicherweise bei jeder beliebigen Änderungsoperation. Wird beispielsweise einem Mitarbeiter mit der Änderungsoperation *addRelation* eine neue Fähigkeit zugeordnet, kann er sich dadurch für die Bearbeitung einer Aktivität qualifizieren, deren zugehörige Bearbeiterformel schon ausgewertet wurde. Das hat aber keine direkte Auswirkung auf die Bearbeiterformeln sondern auf die Arbeitslisteneinträge. Für die Aufrechterhaltung der Gültigkeit von Arbeitslisteneinträgen müssen eigene Strategien entwickelt werden. Nach Änderungsoperationen am Org.Modell müssen gegebenenfalls Bearbeiterformeln auf dem Org.Modell neu ausgewertet werden. Weiterführende Überlegungen dazu finden sich in [Wied02].

6.1 Eingrenzung auf kritische Operationen

Nicht alle Änderungsoperationen haben tatsächlich Einfluß auf Bearbeiterzuordnungen und die Mengen potentieller Bearbeiter, die diese beschreiben. [Wied02] listet die Änderungsoperationen auf, die Anpassungen von Bearbeiterzuordnungen nach sich ziehen können. Von den einfachen Änderungsoperationen ist nur die Operation *deleteInstance* betroffen. Wird ein Datensatz in eine Entität mit aufgenommen (*addInstance*), kann dieser noch durch keine vorhandene Bearbeiterzuordnung referenziert werden. Auch *addRelation* und *deleteRelation* erfordern keine Anpassungen, da Beziehungen nicht von Bearbeiterzuordnungen referenziert werden können.

Im Gegensatz zu [Wied02], wo Attribute nicht in Bearbeiterzuordnungen genannt werden können, werden im Rahmen dieser Arbeit die Bearbeiterzuordnungen ausschließlich auf Basis der Attribute aufgebaut. Folglich müssen auch Anpassungen nach Änderungen der Attributmengen oder der Attributwerte in Betracht gezogen werden. Hier stehen die Operationen *addAttribute* und *deleteAttribute* und *setAttributeValue* zur Verfügung. Für *addAttribute* gilt dasselbe wie für *addInstance:* Ein neu hinzugefügtes Attribut kann noch nicht durch vorhandene Bearbeiterzuordnungen referenziert werden. Bei *deleteAttribute* verhält es sich ähnlich wie bei der Operation *deleteInstance.* Mit einem Unterschied: Wird ein Datensatz, der durch eine Bearbeiterzuordnung direkt referenziert wird, gelöscht, liefert die Auflösung der Bearbeiterzuordnung im schlimmsten Fall eine leere Menge.

Bsp.: Der Mitarbeiter Norbert Niemand wird gelöscht und es existiert eine Zuordnung *(Mitarbeiter.Name = 'Norbert Niemand').getOrgPosition()...*

Wird aber ein Attribut gelöscht, welches in einer Bearbeiterzuordnung verwendet wird, so wird die Bearbeiterzuordnung dadurch in ihrer syntaktischen Korrektheit beeinflußt.

Bsp.: Das Attribut Schuhgröße wird gelöscht und es existiert eine Zuordnung *(Mitarbeiter.Schuhröße = '42').getAbility()...*

Auch nach Ausführung der Operation *setAttributeValue* kann eine Situation entstehen, in der Bearbeiterzuordnungen angepaßt werden müssen. Hier bleibt eine Bearbeiterzuordnung zwar weiterhin syntaktisch korrekt, kann aber trotzdem ihre ursprüngliche Bedeutung verlieren, wenn sie die leere Menge liefert.

Bsp.: Mittels *setAttributeValue* wird die Abteilung '*Entwicklung*' in '*Development*' umbenannt, und es existiert eine Bearbeiterzuordnung *(OrgUnit.Name= 'Entwicklung').getOrgPosition()...*

Unabhängig von diesen Unterschieden sollte in allen drei Fällen nach Ausführung der Operationen eine Anpassung an den betroffenen Bearbeiterzuordnungen vorgenommen werden, da deren Semantik nun nicht mehr der ursprünglichen Intention des Modellierers entspricht. So entsteht also nach Ausführung der Operationen *deleteInstance*, *deleteAttribute* und *setAttributeValue* potentiell eine Situation, in der sich Bearbeiterformeln in einem inkonsistenten Zustand befinden können. Alle anderen einfachen Änderungsoperationen sind diesbezüglich unkritisch.

6.2 Ausführung von Anpassungen

Änderungsoperationen sind auf einem konsistenten Organisationsmodell zu jedem Zeitpunkt ausführbar, unabhängig davon, ob in deren Folge Anpassungen an Bearbeiterformeln notwendig sind oder nicht. Darauf gründet sich auch die Verantwortlichkeit der Änderungsoperation, notwendige Anpassungen an Bearbeiterformeln nach ihrer Ausführung zu initiieren.

6.2.1 Anpassungen nach deleteInstance

Nach Ausführung der Operation *deleteInstance* muß also folgendes festgestellt werden:
Existiert eine Bearbeiterzuordnung, die ein Attribut der Entität, aus der die Instanz gelöscht wurde, mit dem Gleichheitsoperator auf den Wert vergleicht, den auch die gelöschte Instanz für dieses Attribut hatte?

Als Beispiel steht hier ein mögliches Schema für die Entität Agent:

Agent (ID, Name, Vorname, Geburtstag, Wohnort)

Durch die Operation *deleteInstance* wurde folgender Datensatz gelöscht

(0815, 'Niemand', 'Norbert', 29.02.42, 'Ulm')

Für jedes der Attribute *ID*, *Name*, *Vorname*, *Geburtstag* und *Wohnort* muß nun geprüft werden, ob eine Bearbeiterzuordnung existiert, die diese Attribute auf denselben Wert vergleicht, den

auch der gelöschte Datensatz für dieses Attribut hatte. D.h. in Bearbeiterzuordnungen muß nach folgenden Teilen gesucht werden:

- Agent.ID = 0815
- Agent.Name = 'Niemand'
- Agent.Vorname = 'Norbert'
- Agent.Geburtstag =29.02.42
- Agent.Wohnort = 'Ulm'

Alle gefundenen Bearbeiterzuordnungen müssen nun daraufhin geprüft werden, ob sie die leere Menge ergeben, oder ob sie Trefferergebnisse liefern. Ergibt sich eine Bearbeiterzuordnung dabei zur leeren Menge, ist es wahrscheinlich, daß sie nicht mehr dieselbe semantische Bedeutung hat wie zum Zeitpunkt ihrer Modellierung. Diese Bearbeiterzuordnungen, bzw. die Bearbeiterformeln, in denen diese enthalten sind, müssen nun dem Modellierer zur Überprüfung vorgelegt werden. Der Modellierer entscheidet dann, ob er sie anpassen will, oder ob die Bearbeiterzuordnung beibehalten werden soll.

Anmerkung:

Nur allein aus der Tatsache, daß eine Bearbeiterzuordnung die leere Menge ergibt, kann nicht geschlossen werden, daß sie unkorrekt ist. Beispielsweise kann eine Bearbeiterzuordnung lauten:

(Agent.Tauglichkeitsstufe = 'T7')

Diese wurde vielleicht definiert, um Agenten mit Tauglichkeitsstufe 7 explizit mittels (*EXCEPT (Agent.Tauglichkeitsstufe = 'T7')*) von einer Aufgabe auszuschließen. Dabei ist es unerheblich, ob diese Bearbeiterzuordnung auf ein konkretes Organisationsmodell angewandt die leere Menge liefert. Das würde letztendlich nur bedeuten, daß zum Zeitpunkt der Auflösung der Formel kein Mitarbeiter diese Tauglichkeitsstufe hat. Dennoch ist die Bearbeiterzuordnung sinnvoll.

Beim Auffinden der betroffenen Bearbeiterzuordnung wird nur nach solchen gesucht, die den gegebenen Attributwert auf Gleichheit vergleichen. Andere Vergleichsoperatoren werden dabei nicht berücksichtigt. Das hat den Grund, daß beim Einsatz eines unschärferen Vergleichsoperators als dem Gleichheitsoperator durchaus eine leere Treffermenge in Kauf genommen wird, wohingegen das bei einem Vergleich auf Gleichheit im Allgemeinen nicht der Falls ist.

6.2.2 Anpassungen nach deleteAttribute

Die Anpassungen nach der Operation deleteAttribute gestalten sich gegenüber der Operation *deleteInstance* dahingehend einfacher, als daß nicht nach Ausprägungen verschiedener Attribute gesucht werden muß, sondern nur nach der Verwendung des gelöschten Attributs. Beispiel:

Das Attribut *Schuhgröße* der Entität Agent wird gelöscht. Danach sind alle Bearbeiterzuordnungen ausfindig zu machen, die folgendes enthalten:

Agent.Schuhgröße <Vgl.Op> <Value>

Im Gegensatz zu den Bearbeiterzuordnungen, die nach der Operation *deleteInstance* nur mögliche Kandidaten für eine Anpassung sind, sind die Bearbeiterformeln, die solche Teilausdrücke beinhalten, auf jeden Fall anzupassen, da sie syntaktisch nicht mehr korrekt sind. Die gefundenen Formeln werden dem Modellierer zur Bearbeitung vorgelegt.

6.2.3 Anpassungen nach setAttributeValue

Wird die Operation *setAttributeValue* ausgeführt, befinden sich weiterhin alle Bearbeiterformeln in syntaktisch korrekter Form, dennoch kann eine Anpassung der Bearbeiterformeln notwendig sein. Dabei ist eine Überprüfung ähnlich der nach der Operation *deleteInstance* erforderlich, allerdings in vereinfachter Form. An dieser Stelle müssen die Zuordnungen nur auf die Verwendung eines Attributs überprüft werden: auf das Attribut, dessen Wert verändert wurde.

Bsp.: Mittels *setAttributeValue* wird die Abteilung '*Entwicklung*' in '*Development*' umbenannt, und es existiert eine Bearbeiterzuordnung

(OrgUnit.Name = 'Entwicklung').getOrgPosition()...

Nach Ausführung der Operation müssen alle Zuordnungen überprüft werden, ob sie folgenden Term beinhalten:

(OrgUnit.Name = 'Entwicklung')

Zuordnungen, die diesen Term beinhalten, sind potentielle Kandidaten für eine Anpassung und müssen dem Modellierer zur Entscheidung vorgelegt werden, falls diese bei der Auflösung eine leere Menge zur Folge haben. Diese Bearbeiterzuordnungen müssen nicht zwingendermaßen angepaßt werden (siehe Anmerkung in Abschnitt 6.2.1). Auch für die Beschränkung in der Suche nach der Anwendung des Gleichheitsoperators gilt, was oben schon erwähnt wurde.

6.2.4 Zusammenfassung der Vorgehensweisen bei Anpassungen

Abschließend werden Abläufe der nötigen Anpassungen noch einmal in einer Übersicht miteinander verglichen. Abbildung 6-1 zeigt diese in Form von Entscheidungsbäumen.

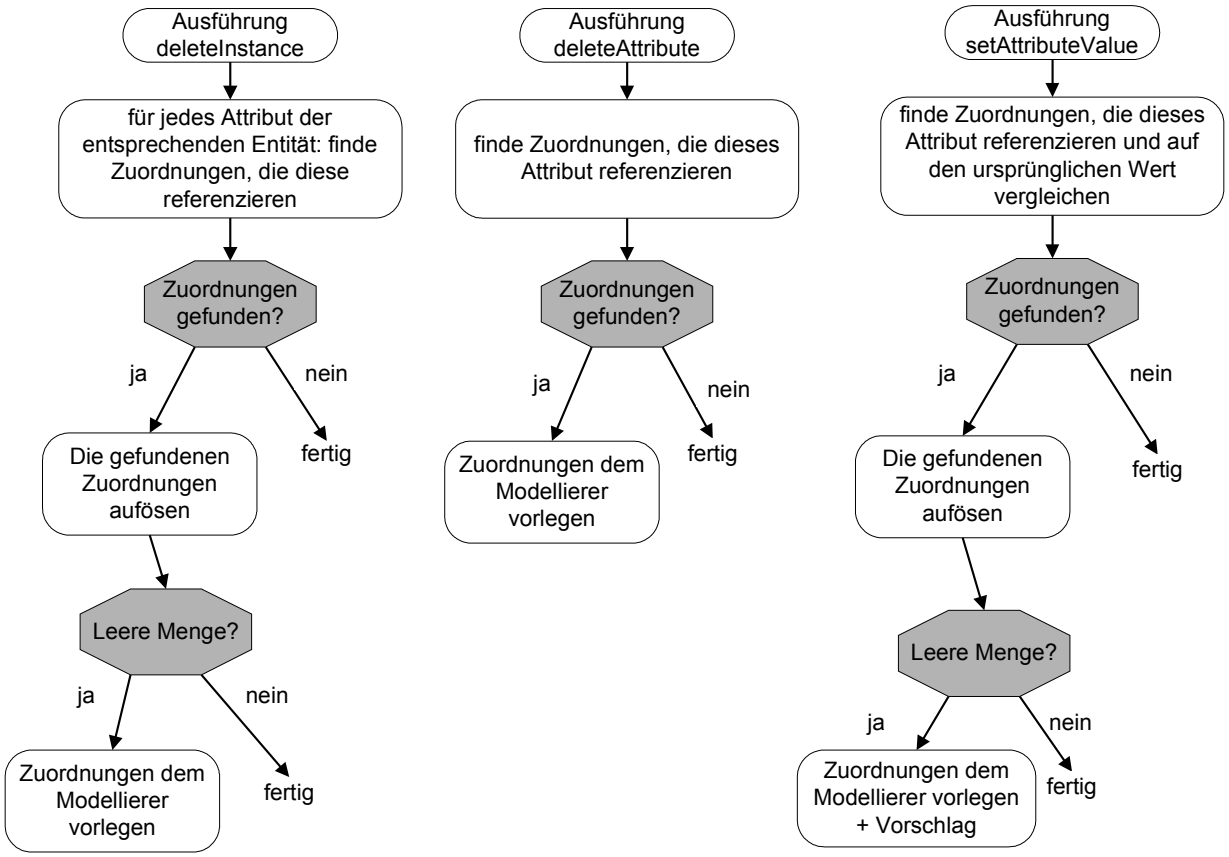


Abbildung 6-1: *Entscheidungsbäume für Anpassungen nach Änderungsoperationen*

Offensichtlich unterscheiden sich die Verfahren für die Operationen *deleteInstance* und *setAttributeValue* nur dadurch, daß nach *setAttributeValue* die Bearbeiterzuordnungen nur nach dem Vorhandensein eines einzigen Attributs zu durchsuchen sind, bei *deleteInstance* dagegen nach allen Attributen der gelöschten Entität. Außerdem kann nach der *setAttributeValue* vom System der Vorschlag gemacht werden, den neuen Wert des Attributes direkt in die Bearbeiterformeln zu propagieren.

Die Operation *deleteAttribute* unterscheidet sich von den beiden anderen dadurch, daß eine Anpassung der Bearbeiterformeln, die von der Änderung betroffenen sind, zwingend vorgenommen werden muß, da diese syntaktisch nicht mehr korrekt sind.

6.3 Verzögerte Ausführung von Anpassungen

[Wied02] schlägt vor, für Anpassungen von Bearbeiterformeln nach Änderungsoperationen den Zeitpunkt der tatsächlichen Anpassungen frei bestimmen zu können. Damit bietet sich die Möglichkeit, aufwendige zeitraubende Anpassungen nicht direkt nach der Änderungsoperation, sondern erst zu einem späteren Zeitpunkt, zu dem das System mehr freie Ressourcen hat, durchzuführen. Läßt man das zu, ist dies nicht problematisch, solange nur Anpassungen nach *deleteInstance* und *setAttributeValue* ausgelassen wurden, da hier schlimmstenfalls eine leere Menge als Ergebnismenge entstehen kann.

Werden allerdings die notwendigen Anpassungen nach deleteAttribute ausgelassen und sollen diese zu einem späteren Zeitpunkt vollzogen werden, müssen alle Bearbeiterformeln vor deren

Auflösung auf syntaktische Korrektheit geprüft werden, so daß sichergestellt werden kann, daß sie keine Attribute referenzieren, die nicht mehr vorhanden sind.

Trifft das WfMS im laufenden Betrieb auf eine solche nicht korrekte Bearbeiterzuordnung, ist unter Umständen der Modellierer, der die ursprüngliche Änderungsoperation durchgeführt hat, nicht verfügbar, und die Ausführung der Prozeßinstanz kommt zum Erliegen. Selbst wenn der Modellierer, der die Änderungsoperation angestoßen hat, anwesend ist, kann dieser Probleme haben, den Kontext in dem eine Anpassung des Modells stattgefunden hat, nachzuvollziehen, wenn diese schon länger zurückliegt.

Aus diesem Grund wird davon abgesehen, die verzögerte Ausführung von Anpassungen zuzulassen. Da die Anpassungen von Bearbeiterformeln nach Änderungsoperationen aufwendig ist und dieser Aufwand von der Anzahl der Bearbeiterformeln im System abhängt, sind Möglichkeiten, effektiv auf die Bearbeiterformeln zugreifen zu können, umso wichtiger.

6.4 Anpassungen nach komplexen Änderungsoperationen

Um die Bearbeiterformeln zu finden, die nach Änderungen am Organisationsmodell angepaßt werden müssen, muß unter Umständen ein nicht geringer Aufwand in Kauf genommen werden. Das hängt in erster Linie davon ab, wieviele Bearbeiterformeln im System abgelegt sind.

Wie in Kapitel 4 gezeigt, werden komplexe Änderungsoperationen auf Basis der einfachen Änderungsoperationen erstellt. Dabei wird die Logik der einfachen Änderungsoperationen zu mächtigeren Änderungstransaktionen zusammengefaßt. Die Ausführung einer einfachen Änderungsoperation beinhaltet aber immer auch die folgende, erforderliche Anpassung von Bearbeiterformeln. Im Rahmen von komplexen Änderungsoperationen können sich aber notwendige Anpassungen durch eine Folge von einfachen Änderungsoperationen wieder aufheben. In diesem Fall wäre die erzwungene Anpassung von Bearbeiterformeln nach jeder einfachen Änderungsoperationen äußerst ineffizient.

Es ist also sinnvoll, für die komplexen Änderungsoperationen eine Möglichkeit zu schaffen, um die automatische Anpassung von Bearbeiterformeln nach den Operationen *deleteInstance*, *deleteAttribute* und *setAttributeValue* außer Kraft setzen zu können. Komplexe Änderungsoperationen müssen dabei allerdings nach ihrer Ausführung in Eigenverantwortung sicherstellen, daß alle notwendigen Anpassungen an vorhandenen Bearbeiterformeln vorgenommen werden.

6.5 Zusammenfassung

Im vorangegangenen Kapitel wurde auf Anpassungen von Bearbeiterformeln eingegangen, die durch Änderungen am Organisationsmodell notwendig werden können. Dabei wurden zuerst die Änderungsoperationen identifiziert, die Situationen erzeugen, in denen Anpassungen an Bearbeiterformeln überhaupt erst notwendig werden.

Für die drei Änderungsoperationen *deleteInstance*, *deleteAttribute* und *setAttributeValue* wurden Vorgehensweisen gezeigt, in welcher Form Anpassungen an Bearbeiterformeln notwendig sind und wie diese durchgeführt werden können.

Von verzögerten Anpassungen an Bearbeiterformeln wurde Abstand genommen, da dies nur schwer zu bewerkstelligen ist und keine signifikanten Vorteile gegenüber der sofortigen Anpassung zeigt. Es ist nur scheinbar von Vorteil, die Anpassungen auf einen späteren Zeitpunkt zu v, um Aufwand zu sparen, da sich im Gegenzug potentiell syntaktisch inkorrekte Bearbeiterformeln im System befinden können und somit bei jedem Prozeßschritt, für den eine Menge von Aufgabenträgern festgestellt werden muß, die entsprechende Bearbeiterformel zuerst auf Fehlerfreiheit geprüft werden muß.

Zuletzt wurde die Möglichkeit diskutiert, im Kontext von komplexen Änderungsoperationen die Anpassungen von Bearbeiterformel auszusetzen und diese Verantwortung den komplexen Änderungsoperationen selbst zu übertragen. Das ist von Vorteil, da sich im Rahmen von komplexen Änderungsoperationen Anpassungen gegenseitig aufheben können und somit einiger Aufwand eingespart werden kann. Die Informationen über diese Begebenheiten finden sich in der Ausführungslogik der semantisch höheren komplexen Änderungsoperationen.

7 Entwurf

In diesem Kapitel werden Aspekte erläutert, die ausgehend von der bisher entwickelten Konzeption zu einem konkreten Entwurf führen. Im Gegensatz zur Konzeption, die sich als Modell des Problembereichs versteht, zeigt der Entwurf als Modell des Lösungsbereichs konkrete Wege zur Implementierung der Lösung.

Die wohl wichtigste Designentscheidung für die Implementierung ist die Frage nach den Datenhaltungssystemen, welche die Org.Modell-Komponente unterstützen soll. In vielen Organisationen wird für die Verwaltung der Organisationsstruktur ein Verzeichnisdienst eingesetzt. Aus diesem Grund widmet sich Abschnitt 7.1 der Frage, ob ein Verzeichnisdienst den Anforderungen der Org.Modell-Komponente genügt.

Ein Datenmodell für die zu verwaltenden Daten wird in Abschnitt 7.2 entworfen. Abschnitt 7.3 zeigt, wie die funktionalen Anforderungen, die an das Org.Modell gestellt werden, in Schnittstellen gruppiert werden. In Abschnitt 7.4 wird dem Workflow-Modellierer eine intelligente Benutzerschnittstelle zum Entwurf von Bearbeiterformeln zur Seite gestellt. Ziel dieser Überlegung ist es, dem Modellierer im Zweifelsfall auch ohne Kenntnis der Syntax der Bearbeiterformeln die Modellierung und Bearbeitung von Bearbeiterformeln zu ermöglichen. Abschnitt 7.5 legt die Architektur der Komponente fest und geht darauf ein, wie diese in das Gesamtkonzept des WfMS eingegliedert wird. Außerdem behandelt dieser Teil die Frage, an welcher Stelle im System die Zuordnungsanweisungen verwaltet werden.

Abschnitt 7.2 zeigt das Datenmodell der Org.Modell-Komponente und Abschnitt 7.6 beschreibt, wie der modulare Aufbau des Strukturmodells implementiert wird. Die Bearbeiterformeln können dabei nicht direkt auf das zugrundeliegende Datenhaltungssystem angewendet werden. Aus diesem Grund müssen die Bearbeiterformeln in dessen Anfragesprache umgewandelt werden. Abschnitt 7.7 zeigt die Übersetzung von Bearbeiterformeln in SQL.

7.1 Backend (Verzeichnisdienst versus Relationale Datenbank)

Im folgenden Abschnitt soll geklärt werden, welches Datenmodell dazu geeignet ist, die Daten des Organisationsmodells abzulegen. Dabei soll insbesondere auf die Fähigkeiten eines Verzeichnisdienstes eingegangen werden.
X.500 basierte Verzeichnisdienste werden im nächsten Abschnitt 7.1.1 hinreichend genau besprochen, um eine solche Entscheidung fällen zu können.

Auf die Fähigkeiten einer relationalen Datenbank wird im folgenden nicht näher eingegangen, da deren Kenntnis vorausgesetzt wird. So werden ab Abschnitt 7.1.2 Vergleiche zwischen Verzeichnisdiensten und RDBMS[19] angestellt, wobei die Eigenschaften der RDBMS, auf die der Text verweist, als bekannt angenommen werden. Informationen über Eigenschaften und Funktionsweise von relationalen Datenbanken liefert z.B. [HäRa01].

19 RDBMS – Relationales Datenbankmanagementsystem

7.1.1 LDAP / X.500

LDAP (Lightweight Directory Access Protocol) beschreibt ein Protokoll zur Client/Server – Kommunikation, welches es dem Client gestattet, auf einen vom Server angebotenen Verzeichnisdienst Zugriff zu erlangen[20]. Sämtliche Angaben über LDAP, für die nicht ausdrücklich eine andere Quelle angegeben ist, lassen sich in beliebiger Literatur zum Thema *Verzeichnisdienste* nachvollziehen. Stellvertretend ist hier [KlüLa03] zu nennen.

Einsatzgebiete für Verzeichnisdienste finden sich überall dort, wo Daten zueinander in hierarchischen Beziehungen stehen, da ein Verzeichnis selbst eine Hierarchie darstellt. So werden Verzeichnisse genutzt, um die Daten von Organisationen zu verwalten: In Form eines Telefonbuches oder als Unterstützung für den Mailclient zur schnellen Recherche nach E-Mail Adressen. Die Möglichkeiten eines Verzeichnisdienstes reichen bis hin zur netzwerkumfassenden Verwaltung von Benutzerdaten und deren Organisation in Gruppen samt zugehöriger Rechteverwaltung. Mittels eines Verzeichnisses kann damit eine netzweite Benutzerauthentifizierung realisiert werden (Single Sign On). Auch die Implementierung eines DNS-Servers ist denkbar.

Interessanter als das Protokoll selbst ist dabei das Datenmodell, welches nicht nur LDAP, sondern auch den meisten anderen Verzeichnisdiensten zugrunde liegt. Dabei handelt es sich um das Datenmodell X.500, ein Standard der ITU (International Telecommunication Union)[21]. Dieser Standard beschreibt zusammen mit anderen Standards aus der X.500-Serie das Schema, nach welchem Daten in einem Verzeichnis angeordnet sind, und welche Operationen auf diesem Datenmodell definiert sind. LDAP verhält sich zu X.500 also so wie ein RDBMS zum Relationenmodell.

7.1.1.1 X.500-Datenmodell

Die X.500 Standards beschreiben eine Welt von Objekten. Dabei spielen die wesentlichen Konzepte der objektorientierten Softwareentwicklung eine bedeutende Rolle. Der Verzeichnisbaum – in der X.500-Terminologie Directory Information Tree (DIT) genannt – besteht dabei aus Objekten. Diese Objekte wiederum werden gebildet aus einer Auflistung optionaler und obligatorischer Attribute, die sie beschreiben.

Attribute bestehen aus einem Namen und einem oder einer ganzen Liste von Werten. Ein besonderes Attribut eines jeden Objektes ist dabei der Distinguished Name (DN), der das Objekt mit Hilfe dieses voll qualifizierenden Namens, ähnlich der Pfadangabe einer Datei in einem Dateisystem, im DIT anordnet. Durch diesen DN entsteht im wesentlichen die Hierarchie des DIT.

20 In diesem Zusammenhang wird auch von einem DUA (Directory User Agent = Client) und einem DSA (Directory Service Agent = Server) gesprochen.

21 Siehe [ITU01_1]

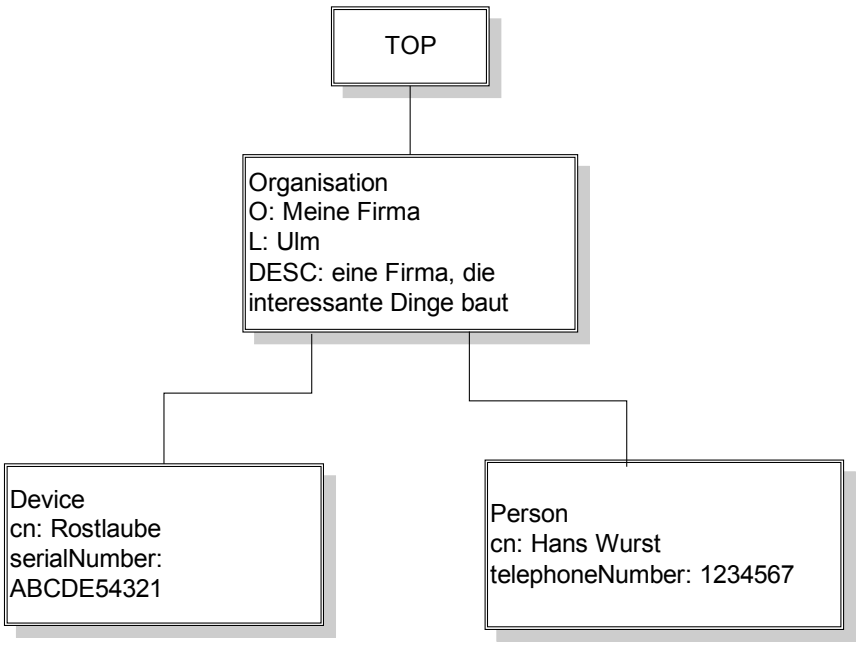


Abbildung 7-1: *Ein einfacher Verzeichnisbaum*

Die Objekte sind dabei, wie aus der OO-Technologie bekannt, Instanzen von Klassen. Der einzige Unterschied zwischen den Objekten der objektorientierten Softwareentwicklung und den in einem Verzeichnis genutzten Objekten besteht darin, daß die Objekte in einem Verzeichnis nur Attribute beinhalten, während die Objekte in der objektorientierten Softwareentwicklung sowohl Attribute als auch die auf ihnen deklarierten Operationen implementieren. Die Klassen werden in der X.500 Terminologie als Objektklassen bezeichnet.

Abbildung 7-1 zeigt einen einfachen Verzeichnisbaum. Unterhalb des namenlosen Wurzelobjektes „*top*“, in dem jeder Verzeichnisbaum seinen Ursprung hat, befindet sich ein Objekt der Objektklasse Organisation mit Namen „*Meine Firma*“ mit einigen Attributen. Hierarchisch unterhalb dieser Organisation angeordnet befinden sich zwei weitere Objekte - ein Firmenwagen und eine Person - auch jeweils mit Angabe der Attribute, und deren Werten.

7.1.1.2 Schema von Objektklassen und Attributtypen

Um die Verwendung von „anonymen Clients“ zu unterstützen, sind sowohl die Objektklassen als auch die Attributtypen, die als Vorlage für die Instantiierung eines Attributes dienen, standardisiert. Der Ausdruck „anonymer Client“ soll hier bedeuten, daß ein beliebiger, dem Server nicht bekannter Client problemlos mit dem Verzeichnis auf dem Server arbeiten kann, sofern er dessen Schemata kennt.

Genau wie in der objektorientierten Softwareentwicklung lassen sich hier Objektklassen durch Ableitung von anderen Objektklassen durch die Hinzunahme von zusätzlichen Attributen erweitern. Für die Klassifizierung von Objektklassen stehen drei Arten zur Verfügung:

- *structural*: Eine als structural definierte Objektklasse legt grundlegende Attribute eines Objekts fest. Die Typisierung einer Objektklasse als structural ist eigentlich nur dann gerechtfertigt, wenn auf ihr eine komplett neue Klassenhierarchie aufbaut. Ein Beispiel für eine als structural definierte Objektklasse ist die Objektklasse Person.

- *auxiliary*: Als auxiliary sind Objektklassen definiert, die eine andere Objektklasse um zusätzliche Attribute erweitern. Als Beispiel sei hier die Objektklasse InetOrgPerson genannt.

- *abstract*: Die als abstract typisierten Objektklassen stellen das exakte Pendant zur abstrakten Klasse in der objektorientierten Softwareentwicklung dar. Abstrakte Objektklassen können nicht instantiiert werden. Sie stellen nur eine Basis für weitere davon abgeleitete Objektklassen dar.

Abbildung 7-2 zeigt eine Klassenhierarchie von häufig genutzten Objektklassen. In einem Dokument der ITU (Selected Object Classes)[22] sind die dargestellten Klassen beschrieben. Es ist die Klassenhierarchie der Objektklasse Person. In dieser Hierarchie befindet sich auch die wohl am meisten genutzte Objektklasse InetOrgPerson.

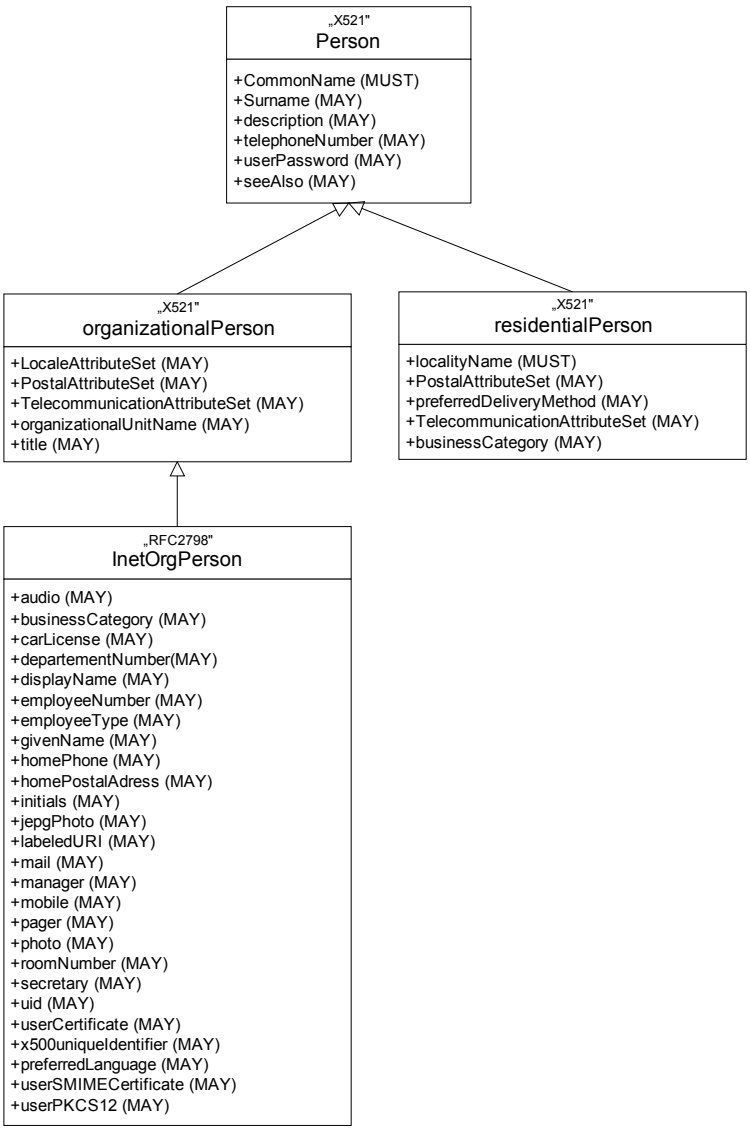


Abbildung 7-2: *Beispiel einer Vererbungshierarchie in X.500*

22 Siehe [ITU01_2]

Jeder Verzeichniseintrag besitzt noch ein spezielles mengenwertiges Attribut zur Angabe der Klassenzugehörigkeit. Da es sich bei dem Attribut *objectClass* um ein mengenwertiges Attribut handelt, ergibt sich hier die Möglichkeit, polymorphe Objekte zu kreieren.
Wie in der objektorientierten Softwareentwicklung läßt sich durch diese „Vielgestaltigkeit“ der Objekte ausdrücken, daß sie nicht nur die Instanz einer Klasse sind, sondern gleichzeitig Instanz mehrerer verschiedener Klassen.

Bei dem Attribut *objectClass* handelt es sich um ein Attribut des Objektes. Somit unterscheidet sich diese Art der Polymorphie von der aus der objektorientierten Softwareentwicklung darin, daß hier ein einzelnes Objekt Instanz von beliebigen Klassen sein kann. Es handelt sich also gewissermaßen um eine Polymorphie auf Objektebene und nicht wie gewohnt auf Klassenebene. Dadurch läßt sich unter anderem auch die vielfach umstrittene Mehrfachvererbung realisieren.

Die Liste der implementierten Objektklassen, welche jedes Objekt führt (Attribut *objectClass*), beinhaltet unter anderen Einträgen auch immer entweder *top*, wobei es sich um einen echten Verzeichniseintrag handelt, oder *alias*. Objekte, die von *alias* instantiiert sind, stellen keine echten Einträge dar, sondern bilden nur jeweils einen Verweis auf ein anderes Objekt. Durch das Konzept der Aliase wird also die strikte Baumstruktur des Verzeichnisses aufgebrochen.

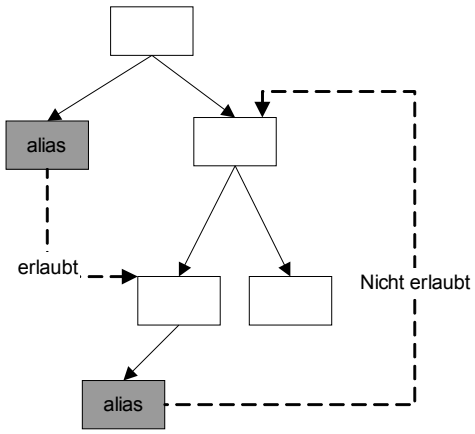


Abbildung 7-3: *Aliase*

Aliase dürfen allerdings nicht beliebig im Verzeichnis gelegt werden. Bei ihrer Verwendung muß stets darauf geachtet werden, daß kein Zyklus entsteht (siehe Abbildung 7-3).

Wie schon angedeutet, handelt es sich bei den Definitionen von Objektklassen und Attributtypen um standardisierte Schemata. Als Beispiele für die Schemadefinition befinden sich die Schemata der Objektklassen *Person*, *organisationalPerson*, *residentialPerson* und *InetOrgPerson* in Anhang C. Bei der Betrachtung dieser durch eine ebenfalls standardisierte Syntax beschriebene Definition fällt die Angabe einer punktseparierten Zahl auf. Hierbei handelt es sich um den *ObjectIdentifier* (*OID*), welcher einen weltweit eindeutigen Bezeichner für die jeweilige Objektklasse darstellt. Diese OIDs sind eingegliedert in einen hierarchischen Namensraum, ähnlich dem DNS. Wie bei der Vergabe von Domänen im Internet wird die Vergabe von OID-Domänen von den oberen Hierarchieebenen an verschiedene Organisationen delegiert.

Die Domäne 1.3.6.1 wurde 1988 an vom ANSI (American National Standards Institute) an die IANA (Internet Assigned Numbers Authority) übergeben.

Unter *http://www.iana.org/cgi-bin/enterprise.pl* steht ein Formular bereit, mit dessen Hilfe man innerhalb weniger Tage eine Domäne unterhalb von 1.3.6.1.4.1 für private Zwecke kostenlos zugewiesen bekommt. Dies stellt somit den kostengünstigsten Weg zur eigenen OID dar. Eine Registrierung beim ANSI schlägt dagegen mit über 1000 US-Dollar zu Buche.

Die Anzahl der registrierten OIDs ist immens. Daher lohnt sich vor der Definition eigener Schemata ein Blick auf bereits Vorhandenes auf jeden Fall. Einen Überblick über registrierte X.500-Schemadefinitionen bietet [Alv97].

Zweck einer OID ist die eindeutige Identifikation von Objektklassen und Attributtypen, die ebenfalls eine solche OID tragen. Dadurch können sich Server und Client sicher sein, daß sie über dieselben Typen sprechen. Das ermöglicht den Einsatz anonymer Clients (siehe oben).

Attributtypen, die wie Vorlagen für Attribute zu verstehen sind, werden zum Zwecke der Wiederverwendbarkeit von den Objektklassen getrennt definiert und tragen zur eindeutigen Identifikation ebenfalls eine OID. In ihrer Schemadefinition verweisen Attributtypen auf Regeln, mit denen die Werte der von ihnen beschriebenen Attribute auf Gleicheit, Ordnungs- und Enthaltenseinsbeziehungen hin verglichen werden können. Beispiele für die Schemadefinition von Attributtypen gibt Anhang D.

7.1.1.3 Aspekte der Verteilung

Aufgrund der Baumstruktur eines Verzeichnisses bietet das X.500-Datenmodell die Möglichkeit, einen DIT auf mehrere DSAs (Directory Service Agent) zu verteilen. Ein solcher Schnitt im Verzeichnisbaum liegt dabei zwischen zwei miteinander verbundenen Einträgen und separiert somit einen Teilbaum vom DIT. Die Verbindung der Teilbäume auf verteilten Servern erfolgt durch Referenzen, die als Referrals bezeichnet werden. Für die Verbindung existieren zwei Arten von Referrals: Subordinate und Superior Referrals, die jeweils von einem Eintrag auf den Sohn auf einem anderen DSA (subordinate) oder von einem Eintrag auf den Vaterknoten (superior) verweisen. Die Auflösung von Referrals geschieht für den Client üblicherweise transparent auf dem Server, so daß der Client nicht weiß, daß es sich um ein verteiltes Verzeichnis handelt.

Die Vorteile einer solchen Partitionierung sind:

- Bessere Möglichkeiten zur Lastverteilung
- Verteilung der Administration

Um eine Verteilung der Administration zu erreichen, muß ein Teilbaum nicht notwendigerweise auf einem entfernten DSA liegen. Das Modell läßt generell zu, administrative Tätigkeiten auf Teilbäume beschränkt zu verteilen.

beispielsweise kann in einer Organisation eine Abteilung die Mitarbeiter, die bis dahin als InetOrgPerson modelliert waren, auf eine andere Objektklasse migrieren, welche von InetOrgPerson abgeleitet ist. Da die Objekte der neuen Objektklasse implizit auch Instanzen der Oberklasse InetOrgPerson sind (aufgrund von Polymorphie), wird vorhandene Software, die die neue Objektklasse nicht kennt, davon nicht beeinträchtigt. Die Änderungen bleiben ihr sogar verborgen.

7.1.1.4 Funktionales Modell von LDAP

Bei LDAP handelt es sich um ein sitzungsbasiertes Kommunikationsprotokoll. Das bedeutet, daß vor dem Datenaustausch zwischen Server und Client, eine Verbindung initiiert werden muß. Zu diesem Zweck stellt LDAP die Operationen *bind* und *unbind* zur Verfügung, um eine Sitzung zu initiieren und zu beenden. In der Operation *bind* wird auch die Authentifizierung des Benutzers vorgenommen und gegebenenfalls die Art der Verschlüsselung ausgehandelt, falls es sich nicht um einen anonymen *bind* handelt.

Ist die Verbindung aufgebaut, stehen dem Client folgende Operationen auf dem Server zur Verfügung:

- *search*: mittels eines angegebenen Filters kann nach Objekten gesucht werden, die in ihren Attributen mit den im Filter angegebenen übereinstimmen. Die Suche kann durch eine Vielzahl von Parametern angepaßt werden.
- *modify*: ändert den Attributwert eines spezifizierten Eintrags.
- *add*: fügt einen neuen Eintrag in das Verzeichnis ein.
- *delete*: löscht einen Eintrag aus dem Verzeichnis.
- *modifyRDN*: ändert den Relative Distinguished Name (ein Teil des Distinguished Name - vergleichbar mit einer relativen Pfadangabe einer Datei) eines Eintrages und bewirkt somit eine Verschiebeoperation eines ganzen Teilbaumes.
- *compare*: Prüft für einen Eintrag ein Attribut/Wert-Paar.
- *abandon*: bricht eine bereits initiierte Operation ab.

Da ein Verzeichnis prinzipiell beliebig verteilt ist, muß mit nicht unerheblichen Latenzzeiten gerechnet werden bis der Server, mit dem der Client kommuniziert, Antworten liefern kann. Das liegt daran, daß der Server bis zur vollständigen Beantwortung der Anfrage sämtliche auftauchende Referrals verfolgen muß. Aus diesem Grund ist LDAP als asynchrones Protokoll konzipiert. Das ermöglicht dem Client im Gegenzug auch zu beliebigen Zeitpunkten Operationen auf dem Server anzustoßen, ohne auf die Beendigung vorangegangener Operationsanforderungen warten zu müssen. Mit der Operation *abandon*, die als Parameter die MessageID der entsprechenden Operation beinhalten muß, kann die Ausführung einer bereits gestarteten Operation abgebrochen werden.

Für die Integration von LDAP in Clientsoftware existieren für viele Plattformen und Sprachen LDAP-Bibliotheken und SDKs. Eine Übersicht bietet Anhang E.

7.1.1.5 LDAP Implementierungen

Für die LDAP Spezifikationen findet sich eine ganze Reihe freier und kommerzieller Serverimplementierungen (DSA):

- Open LDAP (OpenSource)
- Novell eDirectory
- Microsoft Active Directory
- Siemens DirX

um nur einige Beispiele zu nennen.
Offenbar wird gerade beim Microsoft Active Directory die LDAP Kompatibilität häufig angezweifelt, weshalb Microsoft eine Richtigstellung veröffentlicht hat, um diese üblichen Mißverständnisse auszuräumen [MS03].
Der Microsoft Directory Service „Active Directory“ ist eine Implementierung sowohl des LDAP Protokolls zum Service-Frontend als auch eine Implementierung eines X.500 kompatiblen Service Backends (d.h. das Schema der Datenhaltung). Windows benutzt die im Active Directory abgelegten Daten zur Rechteverwaltung in Rechnernetzen und ermöglicht damit ein Single Sign On zur Authentifizierung gegenüber allen Ressourcen im Netz, welche die Autorisierungsdaten vom Active Directory beziehen.

Das Active Directory wurde erstmals in Windows 2000 integriert und erfährt durch die Integration in die neueren Server-Betriebssysteme sowohl herstellerspezifische Erweiterungen als auch weitere Anpassungen an das LDAP Protokoll.

Aufbauend auf der Funktionalität eines X.500 basierten Verzeichnisdienstes bietet das Active Directory eine Reihe höherer Funktionen, die durch das LDAP Protokoll nicht abgedeckt sind. Hierunter fällt auch die Möglichkeit des Einsatzes eines „distributed locator service“, wodurch ein Client nicht mehr die direkte Adresse des Servers kennen muß.

Nach Aussage von Microsoft ist das Active Directory zum größten Teil LDAP-kompatibel. Ein White Paper von Microsoft [MS03] listet die dabei implementierten RFCs auf.

7.1.2 Eignung eines Verzeichnisdienstes für die Org.Modell-Komponente

In den vorangegangenen Abschnitten wurden die Grundlagen von X.500 basierten Verzeichnisdiensten beschrieben. Mit diesem Wissen über Verzeichnisdienste ist es nun möglich, zu evaluieren, inwieweit ein Verzeichnisdienst für den Einsatz in der Org.Modell-Komponente eines WfMS geeignet ist.

Auch wenn in der Literatur Organisationsmetamodelle grundsätzlich als ER-Diagramm dargestellt werden, bedeutet das beim Design der Software noch nicht, daß nur eine relationale Datenbank dafür eingesetzt werden kann. Konkret soll in der folgenden Betrachtung geprüft werden, ob der Einsatz einer LDAP-Implementierung möglich ist, oder vielleicht sogar gegenüber dem Einsatz eines RDBMS (relational database management system) Vorteile bringt.

Ein Verzeichnisdienst stellt die in ihm gespeicherten Daten durch eine Baumstruktur zueinander in Beziehung. Der Sachverhalt, daß eine Stelle einer Organisationseinheit angehört, wird beispielsweise dadurch manifestiert, daß sich die entsprechende Stelle im Verzeichnis unterhalb der Organisationseinheit befindet. Somit ist von vorne herein schon eine

hierarchische Struktur vorgegeben. Beim Einsatz eines Modells, welches durch ein RDBMS dargestellt ist, obliegt es dem Anwender, ob er eine hierarchische oder eine beliebige andere Struktur definieren möchte.

Beim Einsatz von Aliasen ist die Einschränkung zu beachten, daß ein Alias immer nur auf einen Eintrag in einem anderen Teilbaum verweisen darf, niemals aber auf einen Eintrag, der einen Vorgänger (auch transitiv) des verweisenden Eintrags darstellt. Damit ist in einem Verzeichnis die Zyklenfreiheit garantiert. Verzeichnisse unterstützen also Daten, deren Struktur jeweils genau eine Vater-Sohn Beziehung beinhaltet. Steht ein Eintrag mit mehr als einem anderen Eintrag einer jeweils anderen Klasse in einer (1:n)-Beziehung, so ist man auf die Anwendung von Aliasen angewiesen. Modellierung von (n:m)-Beziehungen läßt das X.500-Modell nicht zu.

Bei einem RDBMS sind die Beziehungen zwischen den Entitäten zwar auch von Beginn an festgelegt, allerdings gibt es eine Anzahl verschiedener Beziehungstypen (bezüglich der Kardinalitäten). Auch ist beim Zugriff auf die Daten nicht von vorne herein ein Pfad (Verzeichnisstruktur) gegeben, sondern der Pfad wird durch jede Anfrage durch Joins von an sich „flachen“ Relationen aufs Neue gebildet. Durch Joins lassen sich sogar Beziehungen zwischen Entitäten herstellen, die beim Modellieren der DB noch gar nicht in Erwägung gezogen worden sind.

Die stärkere Flexibilität des Relationenmodells im Vergleich zu der eines verzeichnisbasierten Modells (in diesem Beispiel das X.500 Modell) ergibt sich daraus, daß bei ersterem schon die Relationen – also die Konstrukte des Schemas – zueinander in Beziehung gesetzt werden, wodurch auch Typsicherheit gewährleistet ist. Bei Verzeichnissen werden jedoch die konkreten Einträge – also die Ausprägungen der zu verwaltenden Daten – zueinander in Beziehung gesetzt. Da sich ein Objekt in einem Verzeichnis unabhängig von den dazu instantiierten Objektklassen an einer beliebigen Position im Verzeichnisbaum befinden kann, besteht in einem Verzeichnis keine Typsicherheit.

Aus oben genannten Gründen läßt sich schließen, daß sich mit einem RDBMS eine größere Anzahl von verschiedenen Modellen darstellen läßt, d.h., die Art der Datenhaltung in einem Verzeichnisdienst wie LDAP besitzt nicht die Ausdrucksmächtigkeit, die das in Kapitel 3 vorgestellte Metamodell fordert. Das Modell einer relationalen Datenbank stellt in dieser Hinsicht eine viel breitere Basis dar, selbst wenn es noch Anforderungen gibt, die selbst das Relationenmodell nicht auszudrücken vermag. (Bsp. „Organisationseinheit darf nicht sich selbst übergeordnet sein“). Solche Einschränkungen zu überwachen, ist dann aber eine Anforderung an die benutzende Anwendung – also die DB-Clients. Gerade diese Freiheitsgrade sind es aber, die dem zu entwerfenden Metamodell seine Flexibilität verleihen sollen. Weitere Gründe, die gegen den Einsatz von LDAP sprechen:

- LDAP hat keinen transaktionsorientierten Ansatz. Es wäre zwar nicht unmöglich, aber zumindest problematisch, transaktionssichere Änderungsoperationen auf dem Organisationsmodell anzubieten, wenn dem zugrundeliegenden Datenhaltungssystem dieses Konzept fremd ist. Die Datenhaltung eines Verzeichnisdienstes ist eher statisch, d.h., eine Aktualisierung des Datenbestandes in kurzen Intervallen kann zu Inkonsistenzen führen. Ein Verzeichnisdienst ist bei der Beantwortung von Anfragen, die auf lesenden Zugriffen beruhen, zwar effizient ([Wang00]), ist aber nicht geeignet für einen stark dynamischen Datenbestand ([SUSE], [MLinx])[23].

23 Eine Suche nach „Schreibzugriff LDAP“ mit einer gängigen Web-Suchmaschine liefert dazu weitere Quellen.

- LDAP bietet keine Überwachung von referenzieller Integrität. Diese Eigenschaft gestaltet gerade den Umgang mit einem RDBMS bequem, da bestimmte zu definierende Integritätsbedingungen von der Anwendungslogik in das Datenhaltungssystem ausgelagert werden. Außerdem sind damit auch die komfortablen Funktionen wie kaskadierendes Löschen oder Aktualisieren verbunden.

Anstoß für die Betrachtung der Verzeichnisdienste gibt die Tatsache, daß viele Organisationen eben diese Art der Datenhaltung für ihre Organisationsstruktur gewählt haben. Daher ist eine mögliche Integration dieser bestehenden Verzeichnisse in ein Organisationsmodell für ein Workflow-Management-System nicht zu vernachlässigen.

Offensichtlich wurden diese Verzeichnisse aber unter der Annahme eingeführt, daß nur eine einzige Hierarchie in der Organisation zu berücksichtigen sei. Daß in dieser Hierarchie sowohl Organisationseinheiten als auch Personen und andere Ressourcen beinhaltet sein können, verschleiert allerdings die Tatsache, daß in einer Organisation Hierarchien parallel zueinander und unabhängig voneinander existieren können. Die Rede ist hier von der Existenz von Projektorganisationen. Bereits diese Parallelität wäre mit einem Verzeichnisdienst nur umständlich zu modellieren. Tatsächlich werden diese Hierarchien in real existierenden Verzeichnissen auch selten zu finden sein.

Werden dann (n:m)-Beziehungen benötigt (z.B. zur Modellierung der Besetzung einer Stelle durch Personen, die wiederum mehrere Stellen besetzen können) überschreitet dies die Möglichkeiten eines Verzeichnisses.

Da aber bekanntermaßen in Windowsnetzen gerade eine solche (n:m)-Beziehung zwischen Benutzern und Gruppen besteht, diese Daten aber in einem Verzeichnisdienst -namentlich Active Directory- vorgehalten werden, stellt sich die Frage, wie diese Daten im Verzeichnis zueinander in Beziehung gesetzt sind[24].

Tatsächlich sind die im Verzeichnis gespeicherten Objekte überhaupt nicht mittels ihrer Position (im Verzeichnis) zueinander in Beziehung gesetzt. Dafür tragen Objekte, die mit anderen Objekten in Beziehung stehen, einzelwertige oder mengenwertige Attribute[25], welche die Beziehungen zu den Zielobjekten unter Angabe des Distinguished Names herstellen. Als Beispiel wird die Realisierung der (n:m)-Beziehung zwischen Benutzern und Gruppen beschrieben.

Beispiel: Microsoft Active Directory – (n:m)-Beziehung

Benutzer: Objekte vom Typ *User* enthalten ein mengenwertiges Attribut „*memberOf*“, welches die Distinguished Names der Gruppen bezeichnet, in denen der Benutzer eine Mitgliedschaft inne hat.

Gruppen: Objekte vom Typ *Group* enthalten ein mengenwertiges Attribut „*members*“, welches die Distinguished Names der Benutzer auflistet, die Mitglieder dieser Gruppe sind.

24 Detaillierten Einblick in das Active Directory erhält man z.B durch einen Browser, der die Schnittstelle ADSI (Active Directory Service Interface) nutzt (ADSI-Browser).

25 Je nachdem, ob es sich um eine (1:n)- oder um eine (n:m)-Beziehung handelt

Wie schon angedeutet, wird die Modellierungsmächtigkeit des Verzeichnisses, Objekte in ihrer hierarchischen Ordnung darzustellen, dabei gar nicht genutzt. Auch bei Hierarchien bedient man sich derselben Methode wie bei gezeigtem Beispiel: es werden ebenfalls mengenwertige Attribute benutzt, um die Beziehungen herzustellen.

Die Möglichkeiten, die die Struktur eines Verzeichnisses bietet, werden dabei offensichtlich nicht ausgeschöpft. Diese werden lediglich dazu benutzt, um Objekte nach anderen Aspekten als deren organisatorischen Beziehungen zueinander anzuordnen. Bei der Modellierung von Beziehungen durch Attribute ist die Position eines Eintrages im Verzeichnis ja tatsächlich unerheblich.

Trotzdem soll der weiten Verbreitung von Verzeichnisdiensten in bestehenden Organisationen dadurch Rechnung getragen werden, daß im folgenden evaluiert wird, inwieweit die Darstellung einer Organisation in Form eines Verzeichnisses vom Organisationsmodell eines Workflow-Management-Systems gemeinsam mit anderen Softwarekomponenten genutzt werden kann. Sollte dies möglich sein, dann wäre eine Neuabbildung der Struktur des Unternehmens auf das Organisationsmodell nicht mehr nötig.

7.1.2.1 Gemeinsame Nutzung des Datenhaltungssystems

Die gemeinsame Nutzung eines Datenhaltungssystems durch mehrere Systeme ist im allgemeinen problematisch. Wird ein vorhandenes Verzeichnis/Datenbank, welches von einer anderen Softwarekomponente verwaltet wird, auch durch das WfMS benutzt, so müßte sich das WfMS darauf verlassen können, daß kein anderes Programm Änderungen am Datenbestand vornimmt, ohne die Konsistenzregeln, die das WfMS an das Modell stellt, zu verletzen.

Das gilt, sobald an die Korrektheit des Datenbestandes Bedingungen geknüpft sind, die sich nicht allein mit den Mitteln des Datenhaltungssystems (Konsistenz- und Integritätsbedingungen) sicherstellen lassen.

Sich darauf zu verlassen, daß andere Softwarekomponenten, zusätzlich zu den vom Datenhaltungssystem garantierten Beschränkungen weitere Einschränkungen beachten, ist nicht ratsam. Vor allem bei schon existierender Fremdsoftware wird es schwierig sein, nachträglich zusätzliche Constraints zu integrieren, deren Sinn sich ihr nicht erschließen würde.

So darf z.B. eine Projektgruppe oder Organisationseinheit zwar gelöscht werden, ohne die Konsistenz des Org.Modells zu beeinträchtigen. Allerdings müssen dabei im Falle der Projektgruppe sämtliche Mitgliedschaften dieser Gruppe aufgelöst und untergeordnete Projektgruppen ebenfalls gelöscht werden. Im Falle der Organisationseinheit müssen ebenfalls die untergeordneten Einheiten gelöscht, die Projektmitgliedschaften aufgelöst und die zugeordneten Stellen gelöscht oder verschoben werden, was von der Semantik der Operation abhängt. Sollen dabei auch die entsprechenden Stellen gelöscht werden, müssen auch die Stellenbesetzungen durch Mitarbeiter, Stellenbeschreibung durch Rollen und die tangierten Vertreterregelungen gelöscht werden.

Selbst wenn angenommen wird, daß die schreibenden Zugriffe verschiedener Komponenten auf das gemeinsame Repository derart koordiniert sind, daß sie jegliche Beschränkungen einhalten würden, die das WfMS fordert, ergeben sich noch weitere Hürden.

Wie in [Wied02] Kapitel 5 ausführlich diskutiert wird, sind nach Änderungen an der Aufbauorganisation in vielen Fällen auch Anpassungen an den referenzierenden

Bearbeiterformeln notwendig. Werden diese Anpassungen nicht vorgenommen, entstehen verwaiste Referenzen. Im einfachsten Fall bedeutet dies, daß sich die Menge der möglichen Bearbeiter nur unwesentlich verändert. Genausogut kann es aber auch sein, daß die Auflösung der Bearbeiterformeln die leere Menge ergibt, obwohl potentielle Bearbeiter nach einer Verschiebe-Operation nur anderweitig hätten referenziert werden müssen.

Beispielsweise muß nach dem Entfernen oder Ändern von Einträgen aus dem Org.Modell die Komponente zur Verwaltung der Bearbeiterformeln benachrichtigt werden, damit diese einfache Anpassungen der Bearbeiterformeln selbst vornehmen oder komplexere Anpassungen mit Unterstützung des Modellierers veranlassen kann.

Offensichtlich muß also die Komponente, die Änderungen am Organisationsmodell vornimmt, die Komponente, welche für die Verwaltung und Anpassung der Bearbeiterformeln zuständig ist, über die Änderungen in Kenntnis setzen. Ist die Org.Modell-Komponente nicht selbst Auslöser der Änderung, so muß zumindest die Möglichkeit bestehen, daß diese über eine erfolgte Änderung benachrichtigt wird. Eine zweite, zumindest theoretische Möglichkeit wäre, daß die Org.Modell-Komponente aktiv Änderungen an der Datenbasis verfolgen kann. Dies scheidet aber aufgrund des zu hohen Aufwandes aus.

Folglich führt kein Weg daran vorbei, daß die Org.Modell-Komponente die einzige schreibende Komponente für die Daten des Organisationsmodells sein sollte.

7.1.2.2 Import von Verzeichnisdiensten

Wie oben diskutiert ist eine gemeinsame Nutzung der Datenbasis durch mehrere schreibende Komponenten nicht empfehlenswert oder sogar unmöglich. Trotzdem sollte die Möglichkeit bestehen, Daten, welche die Struktur der Organisation beschreiben, in das Organisationsmodell des WfMS zu importieren. Hoffnungsvolle „Kandidaten“ für dieses Vorgehen sind zumindest Daten für die Entitäten „*Org.Einheit*“ und „*Stelle*“, die zueinander in einer 1:n Beziehung stehen sowie die Übernahme von personenbezogenen Daten.

Nach intensiverer Beschäftigung mit diesem Thema ist es zumindest möglich, einen Mechanismus zur Synchronisation zwischen einem existierenden Verzeichnis und dem Datenhaltungssystem der Org.Modell-Komponente zu entwickeln. Dies würde den Rahmen dieser Arbeit sprengen, vor allem unter der Annahme, daß die Strukturen von bestehenden Verzeichnissen keinesfalls einheitlich sind. Leider gibt es zur Modellierung von Organisationen in Verzeichnissen keine allgemeingültigen Vorgehensweisen. Hinzu kommt, daß es schwierig sein dürfte, von Organisationen die tatsächliche Struktur existierender Verzeichnisse in Erfahrung zu bringen.

7.1.3 Existierende Systeme mit LDAP-Unterstützung

Verschiedene verfügbare Workflow-Management-Systeme unterstützen die Zusammenarbeit mit X.500 kompatiblen Verzeichnisdiensten. Bei näherer Betrachtung der Dokumentationen drängt sich allerdings der Verdacht auf, daß es sich bei dieser Unterstützung eher darum handelt, den Wünschen der Anwender nachzukommen, die bereits einen Verzeichnisdienst für die Verwaltung ihrer Organisationsdaten nutzen. Dagegen ist nichts einzuwenden, da ein Import von Daten aus einem Verzeichnis tatsächlich eine überflüssige redundante Speicherung und Verwaltung der Daten vermeidet. Aber der Grund, daß die Systeme LDAP-Verzeichnisse nutzen können ist sicher nicht der, daß diese Vorgehensweise technische Vorteile verspricht.

Das geht aus der Umsetzung der Lösungen hervor. Unter diesem Gesichtspunkt wurden die Systeme *Staffware 2000* [Staff99] und *IBM Websphere MQ Workflow* [MQWF04] näher betrachtet.

Offenbar sind sich die Hersteller dieser Systeme der Problematik bewußt, die eine gemeinsame Nutzung eines Datenhaltungssystems durch mehrere Systeme mit sich bringt. Keines dieser Systeme nutzt einen Verzeichnisdienst als primäre Ablage für die Organisationsdaten. Dafür stellen beide die Möglichkeit zur Verfügung, Daten aus vorhandenen Verzeichnissen in das systeminterne Datenhaltungssystem zu importieren.

Unter *MQ Workflow* existiert dazu seit Version 3.4 die Komponente „LDAP Bridge“. Mit dieser lassen sich Daten aus einem Verzeichnis (auch Microsoft ActiveDirectory) unter Berücksichtigung von Mapping-Regeln in das Datenformat von *MQ Workflow* exportieren. Diese FDL-Datei kann nun sowohl in die Runtime- als auch in die Buildtime-Umgebung geladen werden, die *MQ Workflow* getrennt voneinander verwaltet. Auch die Übertragung von Daten aus *MQ Workflow* in ein Verzeichnis ist vorgesehen. Mit der LDAP-Bridge lassen sich die Datenbestände von *MQ Workflow* und einem Verzeichnisdienst sogar synchronisieren.

Staffware 2000 nutzt dagegen im Zusammenspiel mit einem X.500 kompatiblen Verzeichnis das Verzeichnis an Stelle seines eigenen Datenhaltungssystems. Dazu muß das Schema des Verzeichnisses in einigen Punkten an die Vorgaben von Staffware angepaßt werden. Allerdings kann Staffware nur lesend auf den Verzeichnisdienst zugreifen. Mit den Administrationswerkzeugen von Staffware 2000 lassen sich die Organisationsdaten dann zwar noch betrachten, aber nicht mehr ändern. Änderungen am Datenbestand der Organisationsstruktur müssen dabei unter Verwendung von LDAP-Administrationswerkzeugen oder anderen Anwendungen, die in ein Verzeichnis schreiben können, direkt am Verzeichnis getätigt werden. Ob und wie diese Änderungen ohne Kontrolle von Staffware die Konsistenz des Datenbestandes oder die Ausführung laufender Prozeßinstanzen beeinträchtigen, ist nicht erwähnt.

7.1.4 Zusammenfassung

Aus genannten Gründen sollte nur die Org.Modell-Komponente schreibend auf den Datenbestand des Organisationsmodells zugreifen können. Damit kann die Entscheidung für ein konkretes Datenhaltungssystem unabhängig von anderen Komponenten gefällt werden. Bei näherer Betrachtung stellt man fest, daß sich die Konsistenz- und Integritätsbedingungen des Metamodells aus Kapitel 3 fast komplett durch ein RDBMS abdecken lassen. Für den Entwurf und die Implementierung der Org.Modell-Komponente wird also im folgenden eine relationale Datenbank angenommen.
Dies schränkt in keiner Weise die Möglichkeit ein, zukünftig auch anderen Komponenten z.B. über eine LDAP Schnittstelle Zugriff auf das Organisationsmodell zu gewähren[26].

26 Tatsache ist, daß die meisten LDAP Implementierungen eine relationale Datenbank als Backend nutzen.

7.2 Datenmodell

Der nächste Schritt bei der Entwicklung des Metamodells, ist die Abbildung des EER-Diagramms auf ein Relationenschema. Das bedeutet nicht zwingend, daß zur Implementierung eine relationale Datenbank herangezogen wird. Das Relationenschema bietet die Möglichkeit, die Entitäten und ihre Beziehungen „flach“ zu sehen. Außerdem finden hier die Attribute der Entitäten Platz, die die Darstellung des EER-Diagramms überladen hätten.

Das in Abschnitt 3.1 entwickelte Modell wird folgendermaßen in ein Relationenschema abgebildet. Grundlage dafür ist das EER-Diagramm in Abbildung 3-1. Abbildung 7-4 veranschaulicht, wie die Tabellen im Datenmodell angeordnet sind, und zeigt dabei auch die Beziehungen der Entitäten zueinander.

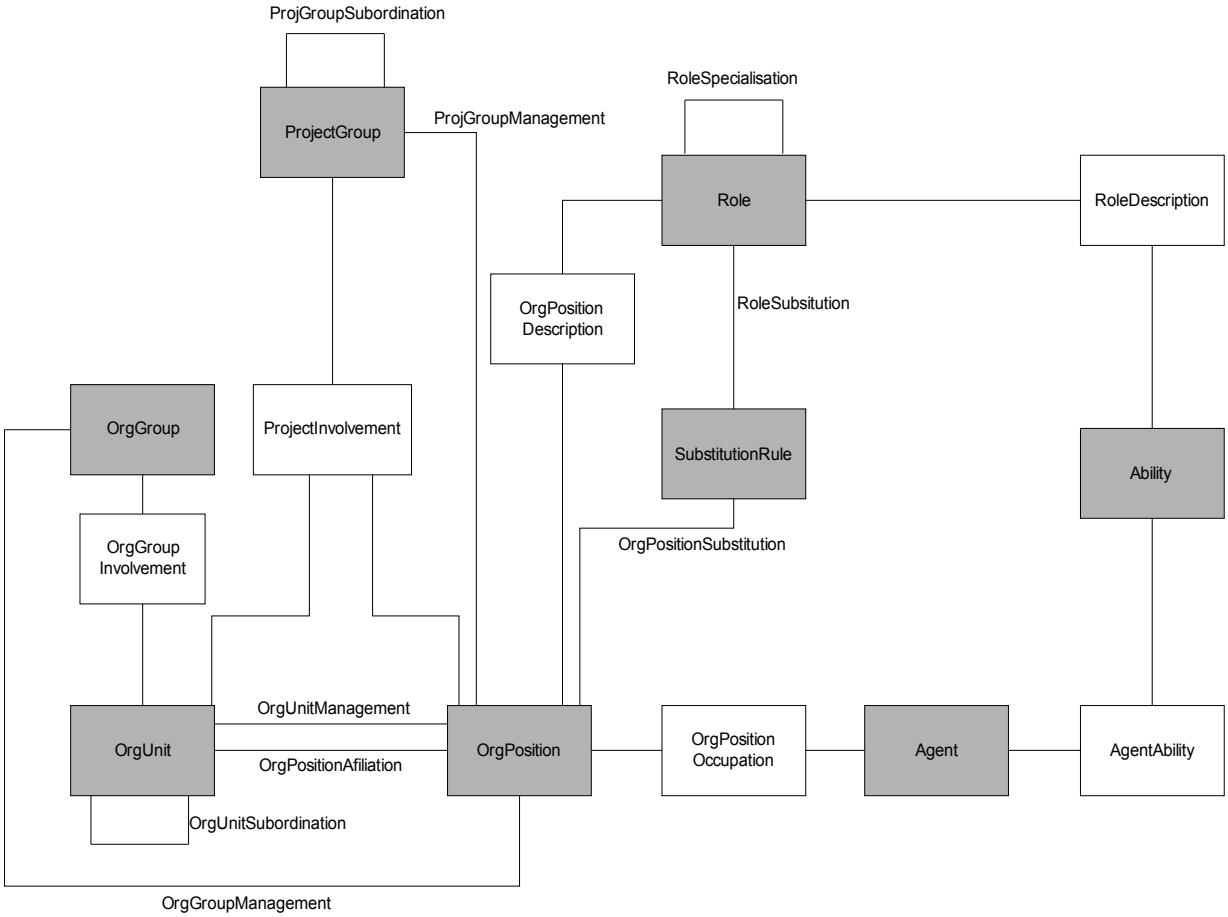


Abbildung 7-4: *Tabellen des Datenmodells und die Beziehungen der Entitäten zueinander*

Im Rahmen der Beschreibung des Datenmodells wird noch noch einmal auf die Unterscheidung von Attributtypen[27] in optionale und obligatorische Attribute eingegangen. Entsprechend Definition 3-15 sind alle Attribute optional, die keine Schlüsselattribute sind. Das genügt für die konzeptionelle Sichtweise. Aber für den Entwurf wird ein pragmatischerer Ansatz gewählt. Zusätzlich zu den Schlüsselattributen sind die angegebenen Attribute obligatorisch:

- Agent.UserName *(eindeutig – unique)* – eindeutige Identifikation des Benutzers
- OrgPosition.Name – beschreibender Name der Stelle

27 Definition 3-15 auf Seite 41

Das hat folgende Gründe:

- Ein Benutzer muß sich nicht einen numerischen Wert als Benutzerkennung merken, um sich am System anmelden zu können, sondern einen eindeutigen Namen, der aus einer Zeichenkette besteht.

- Bei der Anmeldung am System wird einem Benutzer eine Liste von Stellen vorgelegt, in deren Kontext er sich am System anmelden kann. Diese Auswahl wird ihm leichter fallen, wenn er aufgrund der beschreibenden Namen der Stellen entscheiden kann und nicht nur die IDs der Stellen angeboten bekommt.

Um sicherzustellen, daß diese Attribute immer vorhanden sind, und nicht durch Änderungsoperationen gelöscht werden können, sind diese als obligatorisch zu behandeln.

Im folgenden sind Schlüsselattribute unterstrichen. Attribute, die zwingend in der Entität vorhanden sein müssen und auch nicht gelöscht werden können, sind mit *(+)* markiert *(obligatorisch),* optionale Attribute mit *(-)*. Fremdschlüssel sind zusätzlich durch „*F*“ gekennzeichnet. Wie in Abschnitt 3.5 beschrieben sind alle Attribute außer den Identifikatoren und Fremdschlüsseln optional. Die Attributmenge der Entitäten kann nach Belieben mittels Änderungsoperationen (siehe Kapitel 4) erweitert und auch in Zuordnungsanweisungen referenziert werden. Hier aufgeführt sind nur die Attribute, die in der Standardkonfiguration enthalten sind. Obwohl die Erklärung mancher Attribute trivial ist, sind sie der Vollständigkeit halber trotzdem erwähnt.

Relationen für die „echten“ Entitäten:

OrgUnit *(ID, Name, Description, ManagerID, SupID)*

- (+) ID = eindeutige Identifizierung dieser Org.Einheit
- (-) Name = Name der Org.Einheit
- (-) Description = Beschreibung der Org.Einheit
- (+)*F* ManagerID = ID des Mitarbeiters, der die Org.Einheit leitet
- (+)*F* SupID = ID der übergeordneten Org.Einheit, falls eine solche existiert

OrgPosition *(ID, OrgUnitID, Name, Description)*

- (+) ID = eindeutige Identifizierung dieser Stelle
- (+) Name = Name der Org.Einheit
- (+)*F* OrgUnitID = ID der Org.Einheit, der diese Stelle angehört
- (-) Description = Beschreibung der Stelle

Agent *(ID, UserName, Password, Firstname, Lastname, Address, Phone, Email)*

- (+) ID = eindeutige Identifizierung dieses Mitarbeiters (numerischer Wert)
- (+) UserName = der eindeutige Anmeldename eines Benutzers
- (+) Password = verschlüsseltes Passwort oder Hashwert des Passworts
- (-) Firstname = Vorname des Mitarbeiters
- (-) Lastname = Nachname des Mitarbeiters
- (-) Address = Anschrift des Mitarbeiters
- (-) Phone = Telefonnummer
- (-) Email = Email - Adresse

SubstitutionRole *(ID, OrgPositionID, SubstituteID, RoleID, Start, End)*

- (+) *ID* = eindeutige Identifizierung dieser Vertreterregelung
- (+) OrgPositionID = ID der Stelle, deren Vertretung beschrieben wird
- (+)*F* SubstituteID = ID der vertretenden Stelle
- (+)*F* *RoleID* = ID der Rolle, für die die Vertreterregelung gilt
- *(-)* Start = Datums- / Zeitwert für den Beginn der Gültigkeit der Vertreterregelung
- *(-)* End = Datums- / Zeitwert für das Ende der Gültigkeit der Vertreterregelung

Anmerkung:
Die Informationen der Relation *Substitute* könnten auch in der Relation *OrgPositionDescription* untergebracht werden. Da allerdings davon ausgegangen werden muß, daß vielleicht nur für wenige Stellenbeschreibungen Vertreterregelungen definiert sind, würde dies unnötig viele leere Felder verursachen, insbesondere, wenn die Anzahl der Attribute von *Substitute* groß ist.

Role *(ID, SupID, Name, Description)*

- (+) ID = eindeutige Identifizierung dieser Rolle
- (+)*F* SupID = ID der Rolle, von der diese Rolle abgeleitet ist
- (-) Name = Name der Rolle
- (-) Description = Beschreibung der Rolle

Ability *(ID, Name, Description)*

- (+) ID = eindeutige Identifizierung dieser Fähigkeit
- (-) Name = Name der Fähigkeit
- (-) Description = Beschreibung der Fähigkeit

ProjectGroup *(ID, Name, Description, ManagerID, SupID)*

- (+) ID = eindeutige Identifizierung dieser Projektgruppe
- (-) Name = Name dieser Projektgruppe
- (-) Description = Beschreibung dieser Projektgruppe
- (+)*F* ManagerID = ID des Mitarbeiters, der dieser Projektgruppe leitet
- (+)*F* SupID = ID der Projektgruppe, der diese Projektgruppe untergeordnet ist.

OrgGroup *(ID, Name, Description, ManagerID)*

- (+) ID = eindeutige Identifizierung dieser Org.Gruppe
- (-) Name = Name dieser Org.Gruppe
- (-) Description = Beschreibung dieser Org.Gruppe
- (+)*F* ManagerID = ID des Mitarbeiters, der dieser Org.Gruppe leitet

Die zusätzlichen Relationen für die Modellierung von (n:m)-Beziehungen sind:

OrgPositionOccupation *(OrgPositionID, AgentID)*

- (+)*F* OrgPositionID = ID der besetzten Stelle
- (+)*F* AgentID = ID des Mitarbeiters, der die Stelle besetzt

OrgPositionDescription *(OrgPositionID, RoleID)*

- (+)*F* OrgPositionID = ID der durch die Rolle beschriebenen Stelle
- (+)*F* RoleID = ID der Rolle, durch die die Stelle beschrieben wird

RoleDescription*(RoleID, AbilityID)*

- (+)*F* RoleID = ID der Rolle, die die Fähigkeit besitzt
- (+)*F* AbilityID = ID der Fähigkeit, die der Rolle zugeordnet wird

AgentAbility (AgentID,AbilityID)

- (+)*F* AgentID = ID des Mitarbeiters, der die Fähigkeit besitzt
- (+)*F* AbilityID = ID der Fähigkeit, die dem Mitarbeiter zugeordnet wird

ProjectInvolvement *(ProjectID, InvolvedID, Discrimination)*

- (+)*F* ProjectID = ID des Projekts, dem das Projektmitglied angehört
- (+)*F* InvolvedID = ID des Projektmitglieds, welches an dem Projekt teilnimmt
- (+)*F* Discrimination = entscheidet, ob es sich bei dem Schlüsselattribut InvolvedID um die ID einer Stelle oder einer Org.Einheit handelt

OrgGroupInvolvement *(OrgGroupID, OrgUnitID)*

- (+)*F* OrgGroupID = ID des Org.Gruppe, dem die Org.Einheit angehört
- (+)*F* OrgUnitID = ID der Org.Einheit, die an der Org.Gruppe teilnimmt

Anmerkung:
Die Relationen für die Modellierung von (n:m)-Beziehungen beinhalten nur feste Attribute. Es existiert keine Möglichkeit, die Attributmenge dieser Relationen zu manipulieren. Die Operationen *addAttributetype(x, attribType)* und *delAttributetype(x, attribType)* stehen nur für die Relationen zur Verfügung, die die Entitäten selbst darstellen. Zusätzliche Attribute in diesen Relationen ergeben auch keinen Sinn, da sie nicht durch Bearbeiterzuordnungen referenzierbar wären.

7.3 Schnittstellen

Grundsätzlich werden vier unterschiedliche Gruppen von funktionalen Anforderungen an die Org.Modell-Komponente gestellt. Diese Anforderungen wurden schon in Abschnitt 2.2 explizit beschrieben. Im Rahmen des Entwurfs werden diese Anforderungen nun unter Berücksichtigung der aus der Konzeption gewonnenen Aspekte umgesetzt. Von besonderer Bedeutung sind dabei die Schnittstellen, über die die Org.Modell-Komponente in die Menge der anderen Komponenten des Gesamtsystems integriert werden kann.

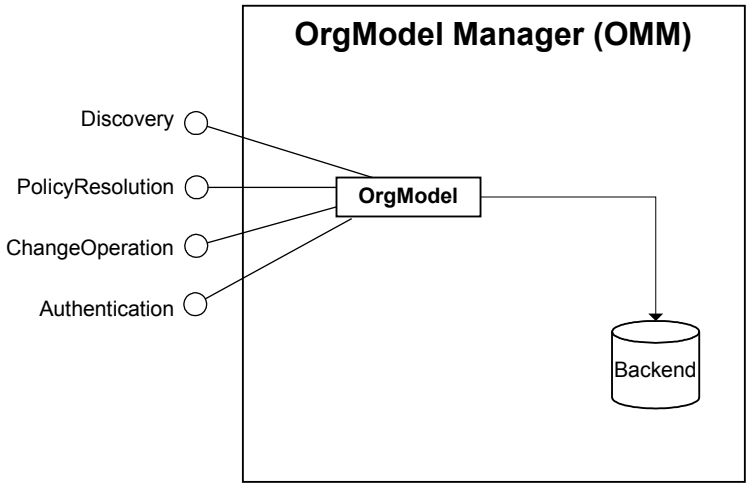


Abbildung 7-5: *Schnittstellen der Org.Modell-Komponente*

Abbildung 7-5 zeigt die in vier Gruppen aufgeteilten Anforderungen als Schnittstellen, die durch die Org.Modell-Komponente zu implementieren sind. Die wichtigste Schnittstelle ist dabei *PolicyResolution* zur Bestimmung von Aufgabenträgern. Fast genauso wichtig ist die Schnittstelle, durch die die manipulativen Zugriffe mittels Methoden für Änderungsoperationen angeboten werden (*ChangeOperation*). Schließlich handelt es sich bei dem vorgestellten Organisationsmodell nicht um ein statisches Modell, sondern um ein Modell, das beliebig vielen Änderungen unterworfen sein kann.

Die dritte Schnittstelle *Discovery* dient dazu, Clients einen Einblick in die Strukturdaten der Organisation zu gewähren. Dabei handelt es sich um Informationen, die sowohl die eigentlichen Daten („Welche Abteilungen sind vorhanden?“) als auch die Metadaten betreffen („Welche Attribute hat ein Projekt?“).
Letztendlich ist auch die Authentifizierung von Agenten eine Aufgabe der Org.Modell-Komponente. Diese Aufgabe wird durch die Schnittstelle *Authentication* beschrieben.

Im folgenden sind die vier Schnittstellen jeweils mit ihren Operationen beschrieben. Da der Adept-Prototyp [ADEPT] in Java[28] realisiert wird, sind die Methoden an die Syntax von Java angelehnt. Für die Einleitung von Ausnahmebehandlungen sind Exceptions vorgesehen. Angegeben ist jeweils der allgemeinste Typ `Exception`. Für die Implementierung sollen aber speziellere Typen verwendet werden.

7.3.1 Schnittstelle PolicyResolution

Diese Schnittstelle definiert Methoden, um mit Bearbeiterformeln umgehen zu können.

`List<AgentResult> resolvePolicy (String orgPolicy)`
: Löst eine Bearbeiterformel in Tupel von Bearbeitern und Stellen auf. Für diese Tupel wird eine eigene Klasse *AgentResult* bereitgestellt.

28 [Java5]

`String checkSyntax (String orgPolicy)`
prüft, ob die Syntax einer Bearbeiterformel korrekt ist, d.h., ob sie überhaupt aufgelöst werden kann.
Bei Erfolg ist der zurückgegebene String leer, ansonsten enthält er die Fehlermeldung des Parsers.

`boolean isMember (int OrgPositionID, String orgPolicy)`
überprüft, ob die Stelle, deren ID als Parameter angegeben ist, in der Menge der potentiellen Bearbeiter, die durch die Bearbeiterformel beschrieben ist, enthalten ist.

`boolean isMember (String UserName, String orgPolicy)`
überprüft, ob der Agent, dessen Benutzername als Parameter angegeben ist, in der Menge der potentiellen Bearbeiter, die durch die Bearbeiterformel beschrieben ist, enthalten ist.

```
boolean isMember (int OrgPositionID, String UserName,
                  String orgPolicy)
```
überprüft, ob in der Menge der potentiellen Bearbeiter, die durch die Bearbeiterformel beschrieben ist, das Tupel aus angegebener Stelle und Agent enthalten ist.

Die Funktionen `resolvePolicy` und `isMember` (mit drei unterschiedlichen Signaturen) sind einander sehr ähnlich. Der Unterschied besteht darin, daß `resolvePolicy` für eine gegebene Bearbeiterformel die Menge der sich qualifizierenden Tupel von Stellen und Agenten ermittelt. Die Funktion `isMember` beantwortet zusätzlich die Frage, ob ein gegebenes Tupel in der Ergebnismenge enthalten ist oder nicht. In diesem Sinne stellt die Funktion `isMember` eine Erweiterung von `resolvePolicy` dar und kann konzeptionell zur Auflösung der Bearbeiterformel auch die Funktion `resolvePolicy` heranziehen. Andererseits besteht auch die Möglichkeit die Funktion `isMember` unabhängig von `resolvePolicy` zu implementieren. Die Enthaltenseinsbeziehung läßt sich abhängig vom Datenhaltungssystem auf eine Schnittmengenoperation oder auf eine Konjunktion von Bedingungen im Selektionsteil der konkreten Anfragesprache des Datenhaltungssystems abbilden, was eine höhere Effizienz bedeuten kann.

Bei einer Implementierung mit einer relationalen Datenbank bedeutet das, daß bei einer *isMember*-Anfrage in SQL der gesuchte Datensatz in den Selektionsteil des SQL-Statement mit aufgenommen werden kann.

Beispiel: Funktionsaufruf: `isMember ('NNiemand', orgPolicy)`

Die Überstzung von `orgPolicy` in SQL lautet:

```
SELECT [Projektion]
FROM [JoinKette]
WHERE [Seletion]
```

Um die Frage zu beantworten ob sich der Agent `NNiemand` in der Menge der potentiellen Bearbeiter befindet, wird das SQL Statement um den Ausdruck „`AND Agent.UserName = NNiemand`“ erweitert:

```
SELECT [Projektion]
FROM [JoinKette]
WHERE [Seletion] AND Agent.UserName = NNiemand
```

Abhängig davon, ob die resultierende Antwort ein Tupel enthält, läßt sich entscheiden, ob der angegebene Mitarbeiter in der Menge enthalten ist oder nicht. So läßt sich die Enthaltenseinsfrage durch die Datenbank effizienter beantworten, als wenn die Menge der aus `orgPolicy` resultierenden Bearbeiter an den DBClient übergeben werden und dieser in einer Schleife nach dem angegebenen Mitarbeiter suchen muß.

7.3.2 Schnittstelle ChangeOperation

Die Schnittstelle ChangeOperation stellt Methoden zur Verfügung, mit denen Manipulationen am Organisationsmodell unterstützt werden. Berücksichtigt sind dabei Änderungen sowohl an den Daten selbst als auch an deren Schemata. Es handelt sich dabei um die einfachen Änderungsoperation aus Abschnitt 4.2.1.

Beim Entwurf dieser Schnittstelle stellt sich die Frage, ob für jede einzelne Entität und Relation ein eigener Satz an Methoden zur Verfügung gestellt werden soll, oder ob die betroffenen Entitäten und Relationen nur als Typparameter Eingang in die dadurch generischen Methoden finden sollen. Um diesen Weg zu beschreiten, werden die Entitäten und Relationen als Aufzählungstypen oder Konstanten definiert. Das hat auch den Vorteil, daß die Schnittstelle schmaler wird. Durch die gewonnene Generizität ist es auch möglich, das Org.Modell für zukünftige Anforderungen zu erweitern, ohne daß dazu aufwendige Anpassungen an der Schnittstelle notwendig werden.

Der Parameter *transactionNumber*, der bei jedem Aufruf einer Änderungsoperation anzugeben ist, bestimmt die Transaktion, in deren Kontext die Änderungsoperation durchgeführt werden soll. Siehe dazu Abschnitt 4.3 (Ausführungsmodus von Änderungsoperationen).

Zum Ausführen von Änderungen muß zuerst mittels `startTransaction()` eine Transaktion angelegt werden. Nach dem Hinzufügen von Änderungsoperationen zu dieser Transaktion kann diese abgebrochen, pausiert oder ausgeführt werden (`abortTransaction(), suspendTransaction(), commitTransaction()`).

enum Entity = {Agent, Ability, OrgPosition, SubstitutionRule, Role, OrgUnit, OrgGroup, ProjectGroup}

enum RelType = {AgentAbility, RoleDescription, OrgPositionOccupation,
OrgPositionDescription, ProjectInvolvement, OrgGroupInvolvement,
OrgUnitSubordination, ProjGroupSubordination, OrgGroupManagement,
OrgUnitManagement, ProjGroupManagement, OrgPositionAfiliation,
RoleSpecialisation, OrgPositionSubstitution, OrgPositionSubstitute,
RoleSubsitution}

AgentAbility	– die Beziehung von *Agent* zu *Ability*
RoleDescription	– die Beschreibung von *Role* durch *Ability*
OrgPositionOccupation	– die Besetzung von *OrgPosition* durch *Agent*
OrgPositionDescription	– die Beschreibung von *OrgPosition* durch *Role*
ProjectInvolvement	– legt die Projektbeteiligungen von *OrgPosition* und *OrgUnit* fest
OrgGroupInvolvement	– legt die Org.Gruppenbeteiligungen von *OrgUnit* fest
OrgUnitSubordination	– legt die übergeordnete *OrgUnit* fest
ProjGroupSubordination	– legt die übergeordnete *ProjectGroup* fest
OrgGroupManagement	– legt den Leiter der *OrgGroup* fest

OrgUnitManagement	– legt den Leiter der *OrgUnit* fest
ProjGroupManagement	– legt den Leiter der *ProjectGroup* fest
OrgPositionAfiliation	– legt die Zugehörigkeit der *OrgPosition* zur *OrgUnit* fest
RoleSpecialisation	– legt die übergeordnete (allgemeinere) *Role* fest
OrgPositionSubstitution	– legt für *SubstitutionRule* fest, welche *OrgPosition* vertreten wird.
OrgPositionSubstitute	– legt für *SubstitutionRule* fest, welche *OrgPosition* eine Vertretung darstellt
RoleSubsitution	– legt für eine *SubstitutionRule* fest, zu welcher *Role* sie gehört

Die Beziehungen und Relationstypen sind auch aus Abbildung 3-1 auf Seite 24 ersichtlich. Auch anhand der Darstellung des Datenmodells in Abbildung 7-4 auf Seite 104 können die Beziehungen nachvollzogen werden.

enum Datatype = {int, float, String, Date}

enum Discriminator = {OrgUnit, OrgPosition}

`int startTransaction ()`
legt eine neue Transaktion an.
Der Rückgabewert ist die Transaktionsnummer, die die neue Transaktion eindeutig bestimmt.

`void suspendTransaction (int transactionNumber)`
pausiert die angegebene Transaktion. Die Transaktion wird fortgesetzt durch Hinzufügen einer weiteren Änderungsoperation oder `Commit()` oder kann mittels `abort()` abgebrochen werden.

`void abortTransaction (int transactionNumber)`
bricht die angegeben Transaktion ab.

`void commitTransaction (int transactionNumber)`
führt die angegeben Transaktion aus.

`int addInstance (Entity entity, int transactionNumber)`
fügt der angegebenen Entität einen neuen Datensatz (Instance) hinzu. Alle Attribute bleiben leer bzw. werden auf Standardwerte gesetzt. Der Rückgabewert ist die ID des neuen Datensatzes. Beim Anlegen eines neuen Datensatzes für Agent muß beachtet werden, daß das Attribut UserName eindeutig sein muß. Zu diesem Zweck wird der UserName beim erzeugen des Datensatzes auf eine Stringrepräsentation des aktuellen Zeitstempels gesetzt.

```
void deleteInstance (Entity entity, int ID, boolean adjust,
                     int transactionNumber)
```
Löscht aus der angegebenen Entität den Datensatz mit der ebenfalls angegebenen ID. Auch alle abhängigen Datensätze in anderen Entitäten („Cascading Delete“) werden dabei gelöscht. Der Parameter *adjust* gibt an, ob nach Ausführung der Änderungsoperation die Anpassung von Bearbeiterformeln durchgeführt werden soll oder ob diese im Rahmen einer komplexen Änderungsoperation zu einem späteren Zeitpunkt getätigt wird.

```
void addAttributeType (Entity entity, String attributename,
                       Datatype dt, int transactionNumber)
```

fügt der angegebenen Entität einen neuen Attributtyp mit angegebenem Datentyp hinzu.

```
void deleteAttribute(Entity entity, String attributename,
                     boolean adjust,
                     int transactionNumber)
                     throws Exception
```

löscht den Attributtyp aus der Entität und damit auch alle Werte der Datensätze für diesen Attributtyp (aber nur, falls es sich nicht um ein festes Attribut handelt). Dabei muß beachtet werden, daß es sich auch bei *Agent.UserName* und *OrgPosition.Name* um obligatorische Attribute handelt, die mit dieser Operation nicht gelöscht werden können.

Der Parameter *adjust* gibt an, ob nach Ausführung der Änderungsoperation die Anpassung von Bearbeiterformeln durchgeführt werden soll oder ob diese im Rahmen einer komplexen Änderungsoperation zu einem späteren Zeitpunkt getätigt wird.

Wirft eine Exception, falls es sich bei dem angegebenen Attribut um ein festes Attribut handelt, welches nicht gelöscht werden kann, oder falls ein Attribut mit angegebenem Namen nicht existiert.

```
void setAttributeValue(Entity entity, int ID,
                       String attributename,
                       String value, boolean adjust,
                       int transactionNumber)
                       throws Exception
```

setzt für den angegebenen Datensatz einen Wert für ein Attribut.
Mit dieser Methode lassen sich nur die Werte „echter“ Attribute ändern. Schlüsselattribute, die die Beziehungen (Relationen) der Datensätze definieren, sind über *addRelation* und *deleteRelation* anzupassen.

Der Parameter *adjust* gibt an, ob nach Ausführung der Änderungsoperation die Anpassung von Bearbeiterformeln durchgeführt werden soll, oder ob diese im Rahmen einer komplexen Änderungsoperation zu einem späteren Zeitpunkt getätigt wird

Wirft eine Exception:

- falls der angegebene Wert (value) sich nicht auf den Datentyp des Attributs umsetzten läßt.
- falls sich das angegebene Attribut nicht durch diese Methode ändern läßt (feste Attribute)
- falls ein Attribut mit dem angegebenen Namen nicht existiert.

```
void addRelation(RelType rt, int param1, int param2,
                 int transactionNumber)
                 throws Exception

void addRelation(RelType rt, int param1, int param2,
                 Diskriminator dis, int transactionNumber)
                 throws Exception
```

fügt dem angegebenen Relationstyp einen neuen Datensatz hinzu. Die Parameter sind die IDs der beteiligten Datensätze in den benachbarten Tabellen.

Bemerkungen:

- Bei der Definition der Schnittstelle ist genau zu dokumentieren, in welcher Reihenfolge die Parameter erwartet werden.

- *Diskriminator dis* legt für die Beziehung *ProjectInvolvement* fest, ob es sich bei der Beziehung um eine *OrgUnit* oder eine *OrgPosition* handelt.

Verhalten im Fehlerfall:

- Wird die Methode mit vier Parametern für eine andere Relation als *ProjectInvolvement* aufgerufen, so wird der Parameter *dis* ignoriert und die Parameter zum Aufruf der Methode mit drei Parametern weitergeleitet.

- Wirft eine Exception:
  - ✗ falls für die angegebenen Parameter (IDs) keine zu verbindenden Datensätze existieren.
  - ✗ falls die dreistellige Methode mit RelType *ProjectInvolvement* aufgerufen wird.
  - ✗ falls durch Hinzufügen der Relation ein Zyklus in einer Hierarchie des Organisationsmodells entstehen würde. Das gilt für Rollen, Projektgruppen und Org.Einheiten.

```
void deleteRelation(RelType rt, int param1, int param2,
                    int transactionNumber)
                    throws Exception

void deleteRelation(RelType rt, int param1, int param2,
                    Diskriminator dis, int transactionNumber)
                    throws Exception
```

zum Löschen von Relationen. Die Parameter entsprechen denen von `addRelation`. Auch die Bemerkungen und das Verhalten im Fehlerfall entsprechen der Methode *`addRelation`*.

7.3.3 Schnittstelle Discovery

Die Schnittstelle Discovery bietet Operationen zur Informationsgewinnung über das Organisationsmodell. Dabei werden sowohl Informationen über die Daten selbst als auch über die Schemata der Entitäten berücksichtigt.

`boolean isActivated (Entity entity)`

prüft, ob die angegebene Entität aktiviert ist oder nicht. Mit Hilfe dieser Methode kann festgestelt werden, ob eine Entität in der aktuellen Konfiguration des Metamodells verwendet wird.

`List<String> getAttributes (Entity entity)`

Liefert die Attribute einer Entität als Liste von Strings.

```
Datatype getDataType (Entity entity, String attributename)
                      throws Exception
```

Liefert den Datentyp des angegebenen Attributes. Datatype ist als eigene Klasse definiert . Eine Beschränkung auf [Int, Float, String, Date] ist hier sinnvoll.
Dazu ist ein entsprechendes Mapping zwischen den Java-Datentypen und den Datentypen der Datenbanken erforderlich.

Wirft eine Exception, falls ein Attribut mit dem angegebenen Namen für diese Entität nicht existiert.

```
List getValues (Entity entity, String attributename)
                throws Exception
```

liefert alle verschiedenen Attributwerte eines angegebenen Attributes.

Wirft eine Exception falls ein Attribut mit dem angegebenen Namen für diese Entität nicht existiert.

```
List<String> getOrgPositions (String userName)
                               throws Exception
```

liefert die Stellen, die ein angegebener Benutzer (Agent) besetzt als Liste von Namen der Stellen.

Bei der Anmeldung am System soll dem Benutzer, falls er mehrere Stellen in der Organisation besetzt, die Auswahlmöglichkeit gegeben werden, zu entscheiden, für welche Stelle er sich anmelden will.

Wirft eine Exception, falls ein Agent mit dem angegebenen Namen nicht existiert.

Eine Anwendung dieser Schnittstelle ist die im Abschnitt 7.4 vorgeschlagene Werkzeug zur graphisch unterstützen Modellierung von Bearbeiterformeln.

7.3.4 Schnittstelle Authentication

Authentication stellt Methoden zur Verfügung, mit denen ein Agent seine Identität dem System gegenüber bestätigen kann.

```
boolean authenticate(String userID, String pwd)
```

prüft die Identität des Benutzers mit der angegebenen userID.

Aus Gründen der Sicherheit wird keine Exception geworfen, um im Falle eines Mißerfolgs keine weiteren Information über den Grund des fehlgeschlagenen Authentifizierungsversuchs preiszugeben.

```
boolean setPassword(String userID, String oldPwd,
                    String newPwd)
```

ändert für den angegebenen Benutzer (Agent) das Passwort.
Der Rückgabewert gibt Auskunft darüber, ob die Änderung erfolgreich war, oder nicht. Aus Gründen der Sicherheit wird keine Exception geworfen, um im Falle eines Mißerfolgs keine weiteren Information über den Grund des fehlgeschlagenen Änderungsversuchs preiszugeben.

Die implementierende Klasse kann zur Realisierung dieser Funktionen entweder das Backend der Org.Modell-Komponente heranziehen oder auch auf beliebige externe Quellen zurückgreifen (LDAP-Verzeichnis, Active Directory, ...). Dies festzulegen ist aber nicht Aufgabe der Schnittstelle.

Anmerkung:
Die Schnittstelle *Authentication* dient nur dazu, die Funktionalität zu demonstrieren, bietet aber nicht keinen Schutz vor unerlaubtem Datenzugriff. Um dies effektiv zu verhindern, müssen zusätzliche Verfahren wie z.B. Session-IDs und verschlüsselte Ende-zu-Ende Kommunikation eingesetzt werden.
Da die Problematik der Sicherheit im Allgemeinen als gelöst angenommen wird und dies auch nicht Schwerpunkt dieser Arbeit ist, wird auf den Einsatz von Sicherheitsmechanismen in diesem Entwurf bewußt verzichtet.

7.4 Intelligente Unterstützung bei der Modellierung von Bearbeiterformeln

Wie schon in den Anforderungen zu Beginn von Kapitel 5 erwähnt, soll die Handhabung von Zuordnungsanweisungen von einem Werkzeug geeignet unterstützt werden. Die folgenden Überlegungen und Verfahrensweisen beziehen sich auf das in Abschnitt 5.2 eingeführte Konzept funktionsorientierter Bearbeiterzuordnungen und deren Einbettung in Bearbeiterformeln. Die Unterstützung für den Modellierer soll dabei so weit gehen, daß dieser zur Bearbeitung von Bearbeiterzuordnungen deren Syntax nicht im Detail beherrschen muß.

Ziel dieser Bestrebungen ist eine intuitiv zu benutzende graphische Benutzerschnittstelle (GUI). Diese soll dem Benutzer ermöglichen, mit nur wenig mehr als ein paar Mausklicks Bearbeiterformeln zu erstellen und diese Aktivitäten zuzuordnen.

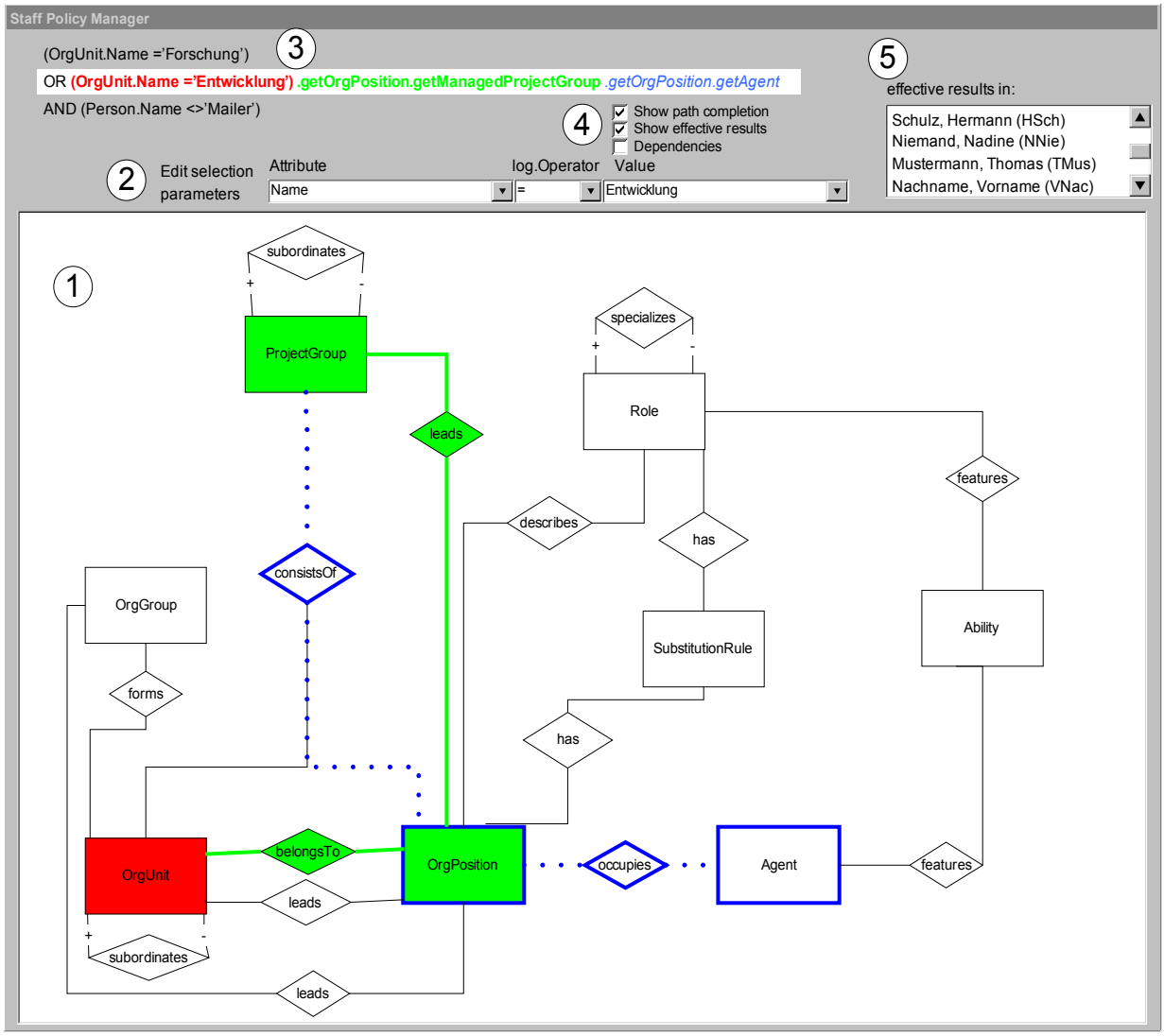


Abbildung 7-6: *GUI eines Werkzeuges zur intuitiven Modellierung von Bearbeiterformeln*

Abbildung 7-6 gibt einen Einblick, wie die Benutzerschnittstelle aussehen könnte. Die Interaktion mit dem Modellierer läuft dabei folgendermaßen ab:

Um eine Bearbeiterformel zu erstellen, wird dem Modellierer das Metamodell seiner Organisation präsentiert. Dabei werden nur die Entitäten und Beziehungen gezeigt, die tatsächlich im Modell Verwendung finden. Wie schon in Abschnitt 3.1 erwähnt, sind nicht alle Entitäten und Beziehungen für ein Modell notwendig. Ein konkretes Modell besteht zumindest aus den Entitäten Org.Einheit (OrgUnit), Stelle (OrgPosition) und Mitarbeiter (Agent) und deren Beziehungen untereinander. Alle anderen Teile des Metamodells sind als optional zu verstehen. Damit läßt sich ein modulares Modell realisieren, welches auch einfacheren Ansprüchen genügt und die Komplexität für den Anwender reduziert. Die Benutzerschnittstelle in Abbildung 7-6 zeigt das gesamte Spektrum des Metamodells (1).

Soll nun eine Bearbeiterformel modelliert werden, markiert der Benutzer die Entität, die den Anfang eines Pfades beschreibt. Im gezeigten Beispiel ist dies die Entität *OrgUnit*. Für die erste Entität eines Pfades ist eine Selektion anzugeben. Diese wird mit Hilfe der Dropdown Boxes (2) definiert. Die erste Dropdown Box zeigt die Attribute der ausgewählten Entität, die zweite die Vergleichsoperatoren, die für den Datentyp des ausgewählten Attributs zur

Verfügung stehen. In der dritten Dropdown Box werden die Werte der Attribute aufgelistet. Der tatsächlich zu bestimmende Wert kann aber auch ein anderer als der in der Liste vorgegebene sein. Beispielsweise die Angaben von *'M*ller'* in der Selektion *(Agent.Name like 'M*ller')*.

Während des Vorgangs der Modellierung / Bearbeitung zeigt das Werkzeug ständig die aktuelle Form der Bearbeiterformel an (3). Die jeweils gerade bearbeitete Bearbeiterzuordnung wird dabei besonders vorgehoben. Nach der oben beschriebenen Angabe einer Selektion hat der Benutzer die Möglichkeit, den Pfad der Bearbeiterzuordnung zu erweitern. Er tut dies, indem er eine von der aktuellen Entität ausgehende Beziehung markiert. Im Beispiel ist dies die Beziehung „belongsTo“. Die Beziehung und die Zielentität wird darauf hin farblich hervorgehoben. Im Bearbeitungsfeld (3) wird an die Bearbeiterzuordnung die Funktion „*getOrgPosition()*“ angehängt. Auch für diese Entität ist die Angabe einer Selektion möglich.

Die Bearbeiterzuordnung in Abbildung 7-6 beschreibt die Menge der Bearbeiter durch die Projekte, die von Stellen aus der Org.Einheit „*Entwicklung*“ geleitet werden. Der Pfad durch das Metamodell ist dabei grün gekennzeichnet. Die Semantik zur Pfadvervollständigung der Entität „*ProjectGroup*“ ist in diesem Beispiel „*getOrgPosition().getAgent()*“. Dieser ist durch die blau umrandeten Elemente und die in blauer Schrift ergänzte Bearbeiterzuordnung angedeutet, falls diese Option aktiviert ist (4). Dabei wird die Pfadvervollständigung in der Bearbeiterzuordnung und visuell im Metamodell zu jedem Zeitpunkt der Modellierung aktuell wiedergegeben.

Ebenso ist es möglich, während der Modellierung / Bearbeitung einer Bearbeiterformel bei jedem Schritt die effektiv betroffenen Bearbeiter mit Namen und Benutzerkennung aufzulisten (5). Dazu wird nach jedem Bearbeitungsschritt die aktuelle Bearbeiterformel neu aufgelöst. Da dies einen erhöhten Kommunikationsaufwand nach sich zieht und bei komplexen Formeln auch die Org.Modell-Komponente stark belasten kann, besteht die Möglichkeit, diese Aktualisierung zu deaktivieren (4). Die Option *„Show Dependencies*“ ermöglicht, die fikitven Entitäten *Instance* und *Activity* in das Werkzeug einzublenden.

7.5 Architektur

In diesem Abschnitt wird darauf eingegangen, in welcher konkreten Form die Org.Modell-Komponente realisiert und in welcher Weise sie in das Gesamtsystem des WfMS integriert werden soll.

7.5.1 Aspekt der Verteilung

Die Softwaretechnik bietet verschiedene Entwurfskonzepte, um eine Verteilung des Gesamtsystems zu ermöglichen, wie zum Beispiel ein Repositorymodell, eine zentral ausgerichtete Client-Server Struktur oder eine dezentralisierte Peer-to-Peer Architektur. Allgemein ist anzumerken, daß man sich durch die Entkopplung der einzelnen Teilsysteme über die Grenzen eines Prozesses[29] hinaus generell die Problematik der Zugangskontrolle einhandelt. Laufen alle Teilsysteme in einem einzigen Prozeß, muß die Autorisierung gegenüber dem System nur ein einziges Mal an zentraler Stelle vorgenommen werden. Geht man aber zu einem verteilten System über, so muß jede Komponente, die Dienste für das Gesamtsystem anbietet, selbst entscheiden, ob Zugriffe auf deren Funktionalität oder die

29 „Prozeß“ im Kontext von Betriebssystemen.

verwalteten Ressourcen berechtigt sind oder nicht.
Weitere Überlegungen in diese Richtung würden allerdings den Rahmen dieser Arbeit sprengen. Und so wird im folgenden bewußt auf die Betrachtung dieses Aspektes verzichtet.

Betrachtet man sich die Anforderungen an die Org.Modell-Komponente an, ist erkennbar, daß hier der Einsatz eines Dienstes gefordert wird. Das Client-Server Konzept entspricht also am ehesten den geforderten Rahmenbedingungen. In diesem Zusammenhang bildet die Org.Modell-Komponente den Server, der beliebigen Clients – also beispielsweise auch dem WfMS – den Dienst anbietet, Berechtigungsfragen auf einer konkreten Organisationsstruktur aufzulösen. In diesem Zusammenhang wird im folgenden vom Org.Modell-Server als gleichbedeutendem Ausdruck für Org.Modell-Komponente gesprochen. Implementiert wird der Org.Modell-Server in einer Form, in der er als Webservice auch auf einer entfernten Maschine angesprochen werden kann. Ebenso möglich ist auch die Verwendung im lokalen Betrieb, indem die Org.Modell-Komponente nicht als Server, sondern im selben Prozeß wie der „Client“[30] läuft. Durch Verwendung derselben Schnittstellen und den Einsatz des Patterns „abstract class factory“ ist es für einen Programmierer transparent, in welcher Form die Org.Modell-Komponente vorhanden ist.

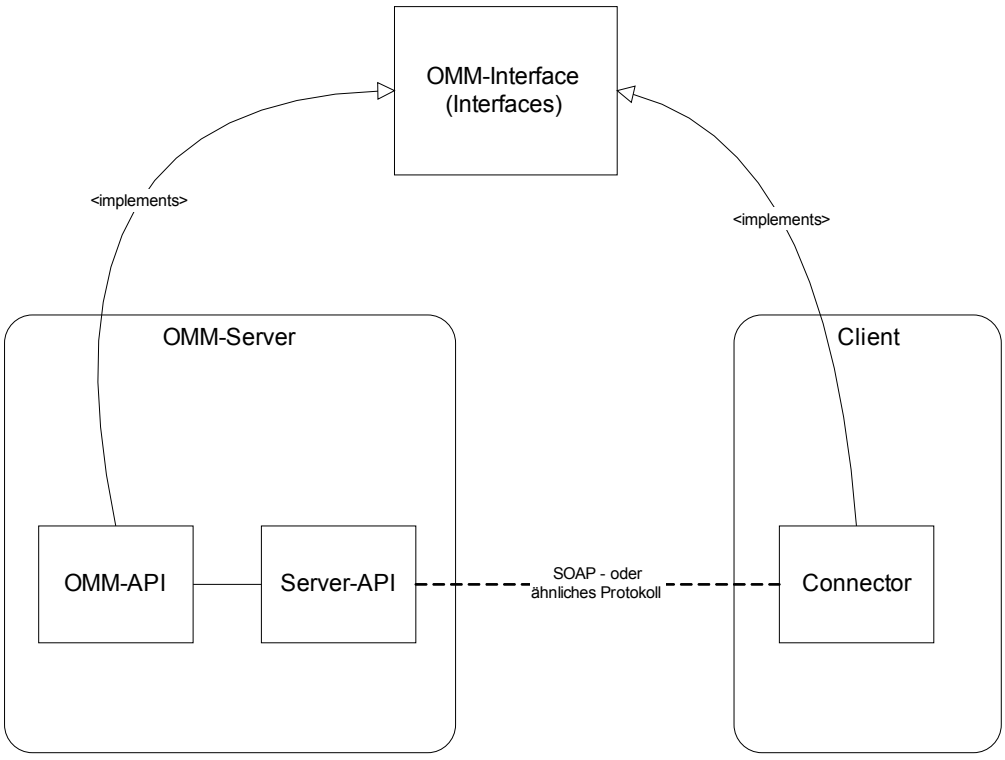


Abbildung 7-7: *Client - Server Architektur*

Diese Anordnung wird durch Abbildung 7-7 veranschaulicht. Die vier in Abschnitt 7.3 vorgestellten Schnittstellen *PolicyResolution*, *ChangeOperation*, *Discovery* und *Authentication* sind in dieser Darstellung unter dem Stichwort OMM-Interfaces zusammengefaßt. Die Transparenz der Kommunikation wird dadurch realisiert, daß sowohl die serverseitige OMM-API als auch der Connector auf Clientseite diese Interfaces implementieren. Um eine Vorstellung von der Funktionsweise des Connectors zu bekommen, bietet es sich an, eine

30 In diesem Fall handelt es sich eigentlich nicht um einen Client, da es in diesem Szenario auch keinen Server gibt.

Analogie zu einem JDBC-Treiber herzustellen. Auch dieser wird mit Hilfe des Classloaders zur Ausführung in das Programm geladen, so daß der Programmierer nur mit lokalen Komponenten arbeiten muß. Die Kommunikation zum Datenbankserver übernimmt der Treiber. Für den Programmierer ist somit die Kommunikation transparent.

In diesem Szenario bietet sich dem Programmierer die Möglichkeit, den Connector oder auch die OMM-API direkt anzusprechen. Um sich zur OMM-Komponente zu verbinden sind Aufrufe in der folgenden Art und Weise vorgesehen:

Für die vier OMM-Interfaces existiert eine Class Factory, die je nach Anforderung als Implementierung des Interfaces die OMM-API direkt oder den Connector zurückliefert.

Also etwa in der Art:

```
PolicyResolution polRes =
OMMFactory.getPolicyResolution(REMOTE)
```

oder

```
ChangeOperation changeOp = OMMFactory.getChangeOperation(LOCAL)
```

Der Connector benötigt dazu aber noch weitere Parameter wie z.B. die Adresse und Port der Remotemaschine. Diese werden aber nicht als Parameter des Methodenaufrufs an die Klasse übergeben. Die Klasse bezieht diese Daten bei ihrer Instantiierung aus einem separaten Property-File.

Ein Client-Programm kann offensichtlich die verschiedenen Schnittstellen auch von verschiedenen OMM-Serverinstanzen anfordern, d.h., es besteht die Möglichkeit im Produktiveinsatz, wo hohe Performanz gefordert wird, mehrere OMM-Server auf einem System von replizierten Datenbankservern zu betreiben. Schreibzugriffe (hier: *ChangeOperation*) erfolgen dabei auf dem Master, bedeutsame Lesezugriffe (hier: *PolicyResolution*) auf dem Master oder einem Slave. Weniger wichtige Lesezugriffe (hier: *Discovery*) lassen sich dabei ganz auf einen Replikationsserver auslagern.

Interessanterweise ist der Anwendungsbereich des Org.Modell-Servers nicht auf den Einsatz im Zusammenspiel mit einem WfMS beschränkt. Der Org.Modell-Server unterstützt jegliche Clients bei der Bestimmung von Berechtigten in einer konkreten Organisationsstruktur. Dabei ist es unerheblich, auf was sich diese Berechtigungen beziehen. Dies liegt komplett im Bereich des Clients. Für den Server spielt es keine Rolle, ob der Client den Zugang zu bestimmten Räumen in einem Gebäude regelt, ob er Zugriffe auf Dateien und Verzeichnisse in einem Dateisystem verwaltet, oder ob er für ein WfMS die Aufgabenträger einer anstehenden Aktivität bestimmen läßt.

Ein Client, der die Berechtigungen für ein Dateisystem mit Hilfe des Org.Modell-Servers verwaltet, kann dafür Abhängigkeiten definieren wie

```
(FileName =
'/versand/0815.odt').getOwner().getOrgUnit().getManager()
```

(Damit ist gemeint: der disziplinarisch Vorgesetzte des Besitzers der angegebenen Datei)

Solange diese Abhängigkeiten im Client vor der Auflösung durch den Server ersetzt werden zu

```
(Agent.ID = '007').getOrgUnit().getManager()
```

kann der Org.Modell-Server für beliebige Berechtigungsfragen dienlich sein, bei der das Organisationsmodell benötigt wird.

Um allerdings diese Unabhängigkeit vom Anwendungsbereich zu erreichen, ist es notwendig, daß die Org.Modell-Komponente in jeder Hinsicht unabhängig von Aspekten der Anwendung ist, in der sie benutzt wird. Das bedeutet, daß die Org.Modell-Komponente von Abhängigkeiten, die nicht komplett ihrer eigenen Verwaltung unterliegen, unbehelligt bleiben muß. Im konkreten Fall eines WfMS sind dies die fiktiven Entitäten *Instance* und *Activity*[31]. Erreichen kann man dies durch die externe Auflösung der Abhängigkeiten. Es bietet sich also an, wenn die Auflösung der Abhängigkeiten nicht innerhalb des Org.Modell-Servers erfolgt, die Zuordnungsanweisungen auch nicht dort zu speichern bzw. zu verwalten. Dies würde sonst eine höheres Maß an Kommunikation zwischen den einzelnen Komponenten des Systems hervorrufen.

7.5.2 Verantwortlichkeit für die Verwaltung von Zuordnungsanweisungen

Tatsächlich wurde bis jetzt immer davon ausgegangen, daß die Zuordnungsanweisungen nicht in der Org.Modell-Komponente verwaltet werden. Im folgenden werden die Konsequenzen erläutert, die eine Speicherung innerhalb oder außerhalb der Org.Modell-Komponente jeweils zur Folge haben.
Diese Entscheidung hat inbesondere bei den beiden Anforderungen „Auflösung von abhängigen Bearbeiterformeln“ und „Anpassung von Bearbeiterformeln nach Änderungsoperationen“ Einfluß auf die Faktoren:

- Kommunikationsaufwand
- Verteilung der Funktionalität im Gesamtsystem
- Performanz

Nimmt man zunächst an, daß die Zuordnungsanweisungen im Client (z.B WfMS) verwaltet und gespeichert werden, so erkennt man, daß es tatsächlich sinnvoll ist, Abhängigkeiten in den Formeln vor der Übergabe an den Org.Modell-Server aufzulösen, so daß dieser zur Auflösung nur noch Formeln aus unabhängigen Bearbeiterzuordnungen erhält. Somit ist der Aufwand für die Kommunikation auf ein Minimum reduziert. Der Client schickt eine Bearbeiterformel mit schon aufgelösten Abhängigkeiten und der Aufforderung der Bestimmung der möglichen Bearbeiter an den Server. Ohne weitere Kommunikation liefert dieser darauf das Ergebnis der Anfrage. Das hat natürlich zur Folge, daß im Client die Logik vorhanden sein muß, die die Abhängigkeiten erkennen, auflösen und durch unabhängige Teilausdrücke ersetzen kann. Das rudimentäre Parsen und Ersetzen kann dem Client durch entsprechende Escapesequenzen vor abhängigen Teilausdrücken erleichtert werden.

31 Siehe Abschnitt 5.2.3

Beispielsweise wird ein Abhängigkeit mit dem Zeichen '$' gekennzeichnet:

(Agent.Name = 'Norbert Niemand') OR (Activity.ID = '0815').getOrgPosition()
wird zu
*(Agent.Name = 'Norbert Niemand') OR **$**(Activity.ID = '0815').getOrgPosition()*

Für die Speicherung der Zuordnungsanweisungen außerhalb der Org.Modell-Komponente spricht auch die Überlegung, daß die Ausdrücke, die die Zuordnung von Aktivitäten zu potentiellen Bearbeitern beschreiben, der Logik nach eher den Aktivitäten selbst als dem Organisationsmodell zuzuordnen sind.
Problematisch ist bei diesem Szenario allerdings die Anforderung, Anpassungen an Bearbeiterformeln vorzunehmen, nachdem Änderungen am Org.Modell vorgenommen wurden. Um die Bearbeiterformeln ausfindig zu machen, die von Änderungen betroffen sind, ist ein nicht unerheblicher Kommunikationsaufwand erforderlich, wenn sich die Bearbeiterformel außerhalb des Org.Modell-Servers befinden. Unter Umständen müssen viele Bearbeiterformeln daraufhin geprüft werden, ob sie nach einer Änderung am Org.Modell angepaßt werden müssen.

Geht man allerdings davon aus, daß die Zuordnungsanweisungen in der Org.Modell-Komponente gespeichert sind, ist der Kommunikationsaufwand bei Anpassungen nach Änderungen am Organisationsmodell faktisch nicht vorhanden. Ein Client ruft eine Änderungsoperation auf, der Org.Modell-Server führt diese aus und kann ohne weitere Interaktion mit anderen Komponenten die Anpassungen der Bearbeiterformel vornehmen.

Als nachteilig erweist sich die Speicherung der Zuordnungsanweisungen in der Org.Modell-Komponente aber für die Anforderung „Auflösung von abhängigen Bearbeiterformeln“. Trifft der Org.Modell-Server beim Parsen einer Bearbeiterformel auf Abhängigkeiten, muß dieser für jede einzelne Abhängigkeit eine Interaktion mit dem Client starten.

Offensichtlich haben beide Szenarien, also die Speicherung der Zuordnungsanweisungen innerhalb oder außerhalb der Org.Modell-Komponente bei der einen Anforderung positive und bei der anderen Anforderung negative Effekte. Die Frage, deren Antwort die Entscheidung bestimmt, muß also lauten:

Welcher Anforderung („Auflösung von abhängigen Bearbeiterformeln“ / „Anpassung von Bearbeiterformeln nach Änderungsoperationen“) wird ein höherer Stellenwert eingeräumt?

Abhängig von dieser Antwort wird sein, in wie hoch die Performanz der beiden Funktionen sein wird.
Es ist davon auszugehen, daß der Auflösung von abhängigen Bearbeiterformeln eine höhere Priorität zukommen muß als der Anpassung von Bearbeiterformel. Aus diesem Grund werden die Zuordnungsanweisungen nicht in der Org.Modell-Komponente verwaltet und Abhängigkeiten außerhalb des Servers aufgelöst. Dadurch bleibt auch die oben erwähnte Möglichkeit bestehen, die Org.Modell-Komponente frei von Abhängigkeiten eines Anwendungsbereiches betreiben zu können.

Das heißt:
Die Komponente zur Verwaltung des Organisationsmodells für ein Workflow-Management-System ist nunmehr nicht nur auf die Anwendung im Rahmen eines Workflow-Management-

System beschränkt, sondern hat sich zu einer eigenständigen Komponente für Organisationsmodelle allgemein entwickelt.

Um allerdings die nach Änderungsoperationen am Org.Modell notwendigen Anpassungen von Zuordnungsanweisungen dennoch so effizient wie möglich unterstützen zu können, wird vorgeschlagen, im WfMS eine logische Datenstruktur[32] über die Bearbeiterformeln zu legen.
Wie die Erkenntnisse aus Kapitel 6 zeigen, müssen bestimmte Informationen über Bearbeiterformeln schnell verfügbar sein. Fragen wie

- welche Bearbeiterformeln referenzieren *Entität x*?
- welche Bearbeiterformeln referenzieren *Attribut y*?

können unter Verwendung eines Index beantwortet werden. Um allerdings die korrekte Funktion zu gewährleisten, muß der Index beim Neuanlegen und Bearbeiten von Bearbeiterformeln gepflegt werden.

7.5.3 Statische Struktur

Nachdem nun die Form der Org.Modell-Komponente geklärt ist, und auch die Entscheidung über den Ort der Speicherung von Zuordnungsanweisungen gefällt wurde, wird nun die Struktur, in die sich die Org.Modell-Komponente gliedert, im Detail vorgestellt.

32 beispielsweise ein Index

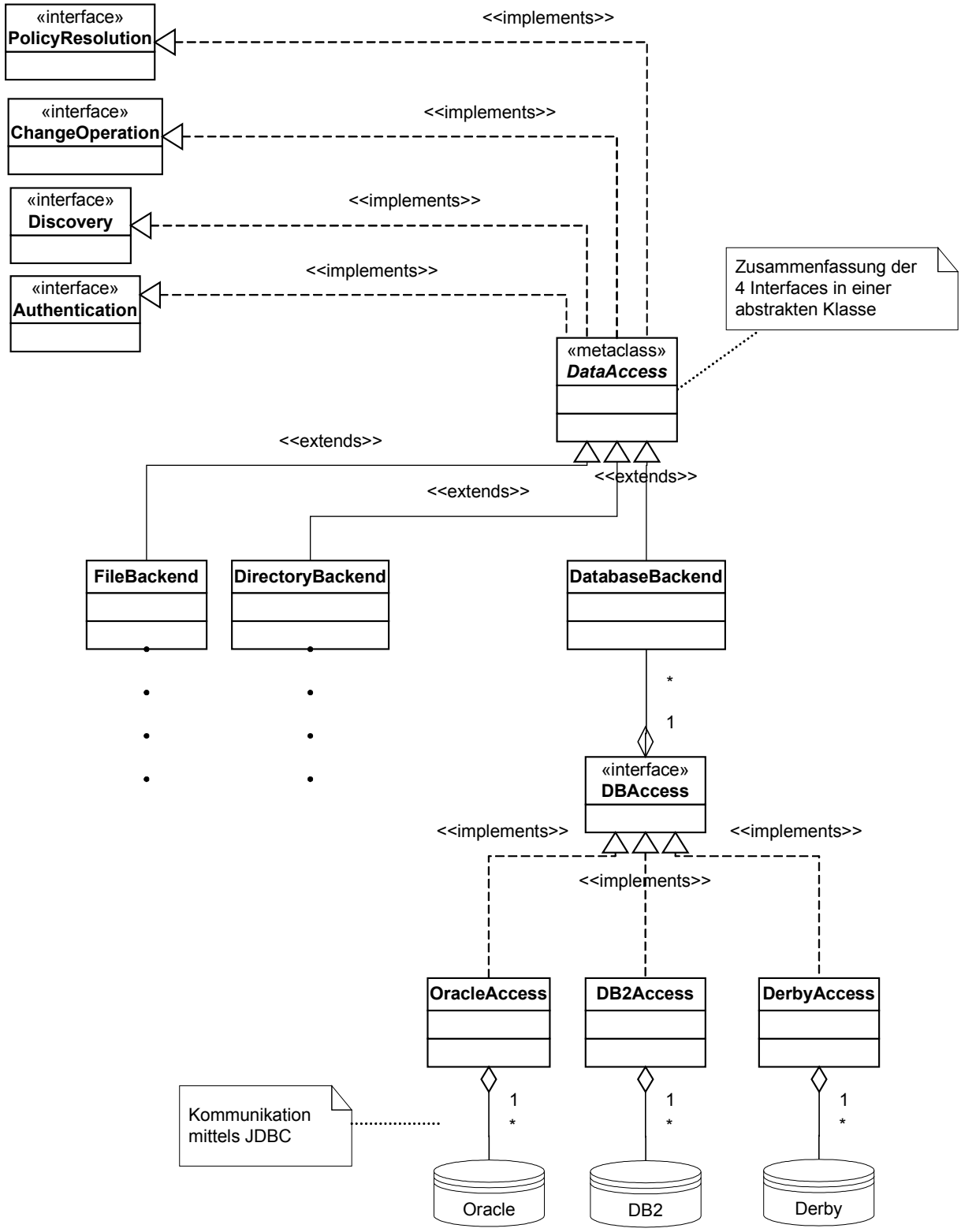


Abbildung 7-8: *Strukturelle Zerlegung der Org.Modell-Komponente*

Die vier im oberen Teil von Abbildung 7-8 abgebildeten Schnittstellen wurden schon in Abschnitt 7.3 ausführlich beschrieben. Diese Abbildung zeigt im Wesentlichen den Teil der Org.Modell-Komponente, die sich in Abbildung 7-7 hinter dem Stichwort OMM-API verbirgt. Die OMM-API bildet gewissermaßen den Zugangspunkt zur Funktionalität der Komponente.

Anmerkung:

> Aus der Abbildung, die die Client-Server-Architektur zeigt (Abbildung 7-7) geht zwar hervor, daß auch der *Connector* diese Schnittstellen implementiert, dies tut er aber nur zum Zwecke der Kommunikation mit dem Org.Modell-Server und somit nur zur indirekten Bereitstellung der Funktionalität.

Egal welche Schnittstelle man betrachtet, es fällt auf, daß die Aufgaben der Org.Modell-Komponente größtenteils eine Umsetzung auf das zugrundeliegende Datenhaltungssystem bedeuten. Da sich diese zum Teil grundlegend in der Art unterscheiden, in der sie Zugriff auf die gespeicherten Daten gewähren, ist die Umsetzung stark von der Wahl des Datenhaltungssystems abhängig. Das wiederum bedeutet, daß geeignete Schnittstellen, die einen Austausch ermöglichen, umso wichtiger sind. Die abstrakte Klasse *DataAccess* ist in diesem Zusammenhang eine Zusammenfassung der vier angebotenen Schnittstellen. Soweit es möglich ist, werden Lösungen schon in ihr implementiert. Ein Beispiel hierfür ist die Methode `checkSyntax` aus der Schnittstelle *PolicyResolution.* Ob eine Zuordnungsanweisung in syntaktisch korrekter Form vorliegt oder nicht, ist unabhängig vom zugrundeliegenden Datenhaltungssystem.

Das Auflösen von Zuordnungsanweisungen hingegen ist nicht unabhängig vom Datenhaltungssystem möglich. Für diese Aufgabe muß unter anderem eine Übersetzung der Zuordnungsanweisungen in die entsprechende Anfragesprache erfolgen[33]. Aus diesem Grund beerbt die abstrakte Klasse *DataAccess* nach Belieben instantiierbare Klassen für die Integration verschiedener Datenhaltungssystem. Beispielsweise ist die Integration von Verzeichnisdiensten, Datenbanken oder auch der direkte Zugriff auf ein Dateisystem denkbar. Für die beabsichtigte Implementierung genügt allerdings die Einbeziehung von relationalen Datenbanken.

Im konkreten Fall einer Realisierung auf der Grundlage von relationalen Datenbanken aggregiert die Klasse *DatabaseBackend* eine Implementierung der Schnittstelle *DBAccess*, um Zugriff auf eine Datenbank zu erlangen.

Die Notwendigkeit der Schnittstelle *DBAccess* ergibt sich aus der nicht Tatsache, daß für den Zugriff auf RDBMS zwar Standards existieren[34], die Hersteller von Datenbanken aber die Standards in ihren Produkten nicht vollständig, sondern nach eigenem Ermessen nur Teilmengen davon umsetzen. Das eigentliche Dilemma dabei ist aber, daß die Hersteller ihre Datenbanken trotz allem als standardkonform bewerben und verkaufen.

Diese Abweichungen vom Standard zu kompensieren ist die Aufgabe der Schnittstelle *DBAccess*. Allerdings kann keine beliebige Datenbank unterstützt werden, von der nicht schon vor der Definition der Schnittstelle bekannt ist, in welchem Umfang sie den SQL-Standard unterstützt, da *DBAccess* immer einen „kleinsten gemeinsamen Nenner“ der Implementierung annehmen muß. Auf jeden Fall werden durch die in Abschnitt 7.5.5 definierte Schnittstelle mindestens die Datenbanken Oracle[35], IBM DB2[36] und Derby[37] transparent verwendbar.

33 Der Umsetzung von Zuordnungsanweisungen in SQL ist ein eigenes Kapitel gewidmet (Kap. 7.7)
34 SQL-89, SQL-92, SQL-99 siehe [DaDa96], [MeSi01]
35 [Oracle]
36 [DB2]
37 [Derby]

7.5.4 Integration komplexer Änderungsoperationen

Änderungsoperation werden durch das Interface *ChangeOperation* beschrieben. Dabei handelt es sich allerdings nur um die in Abschnitt 4.2.1 vorgestellten einfachen Änderungsoperationen. Komplexe Änderungsoperationen dagegen finden sich nicht in der Schnittstellenbeschreibung. Diese setzen sich zusammen aus einfachen Änderungsoperationen oder wiederum komplexen Änderungsoperationen und klassifizieren sich somit als Anwender des Interface *ChangeOperation*.

Obwohl komplexe Änderungsoperationen selbst nicht Teil dieser Arbeit sind, wird im folgenden in groben Zügen beschrieben, wie sich diese in die Architektur des OMM-Servers eingliedern.

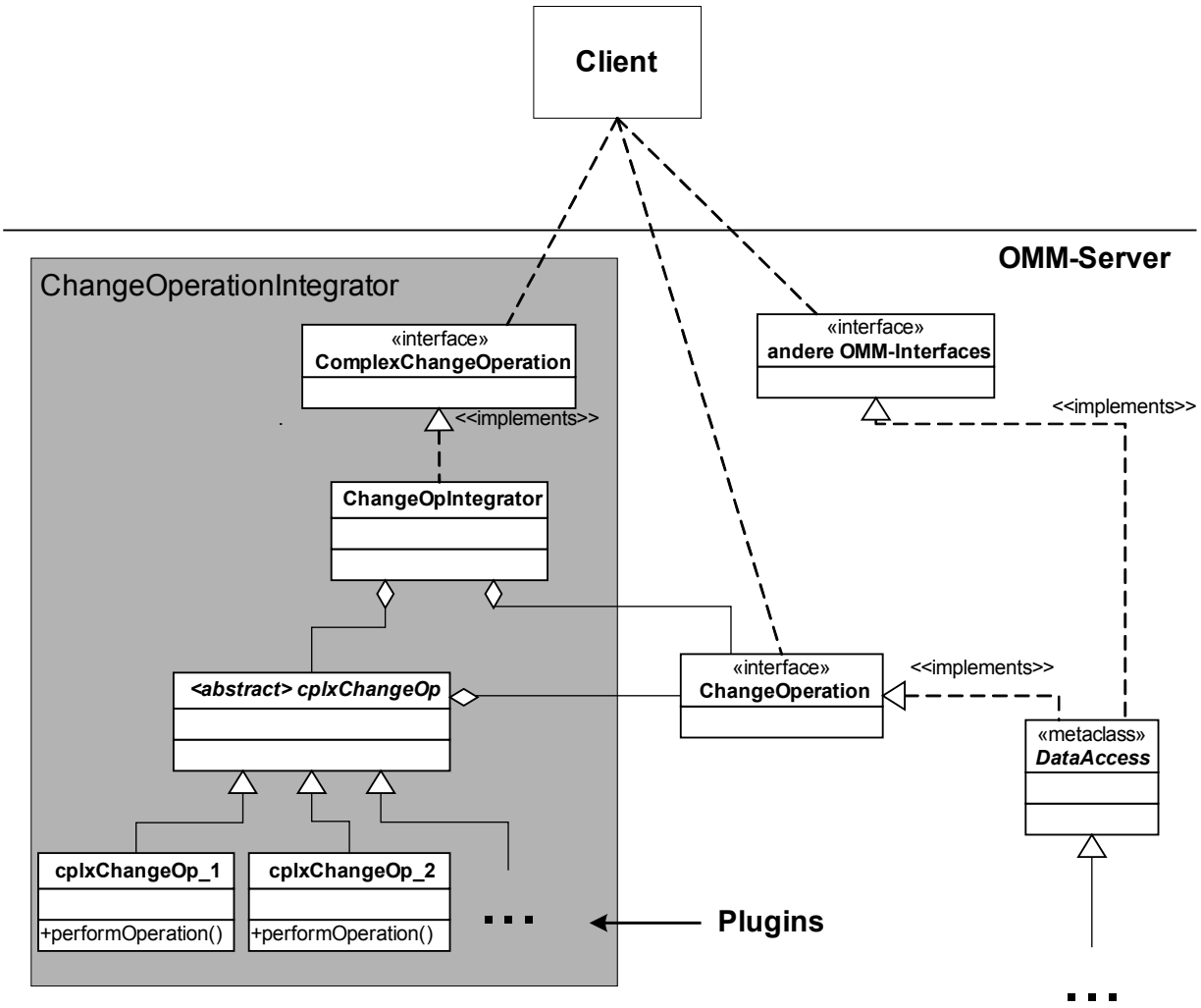


Abbildung 7-9: *Architektur - ChangeOperationIntegrator*

Wie aus Abbildung 7-9 ersichtlich ist, muß eine zusätzliche Schnittstelle Complex*ChangeOperation* in die API des OMM-Servers aufgenommen werden, um die komplexen Änderungsoperationen anbieten zu können. Da die Menge der komplexen Änderungsoperationen beliebig groß ist, sollen diese nicht fest in ein Interface integriert werden. Vielmehr soll die Möglichkeit bestehen, das Repertoire an komplexen Änderungsoperationen nach Belieben erweitern zu können, ohne dabei Schnittstellen ändern und vorhandenen Quellcode neu kompilieren zu müssen. Wenn komplexe

Änderungsoperationen in Form von Plugins erstellt werden, besteht sogar die Möglichkeit, diese während des OMM-Servers hinzuzufügen oder aus dem Repertoire des WfMS auch wieder zu entfernen. Dazu wird der Code zur Laufzeit dynamisch geladen.

Auch an anderer Stelle im Prototyp von ADEPT2 wurde dieses Konzept schon umgesetzt. In einem Werkzeug[38] zur Definition und Bearbeitung von Prozeßvorlagen als Graphen wurden komplexe Änderungsoperationen als dynamisch ladbare Plugins umgesetzt. Nähere Informationen dazu liefert [GWBTS05].

Für die Verwaltung und die Ausführung von komplexen Änderungsoperationen ist die in der Abbildung gezeigte Klasse *ChangeOpIntegrator* verantwortlich. Diese prüft den Inhalt eines Verzeichnisses auf das Vorhandensein von Java Class-Dateien (*.class). ChangeOpIntegrator lädt diese Klassen und nimmt sie, falls sie Instanzen von *cplxChangeOp* sind, in die Liste der vorhandenen Änderungsoperationen auf. Jede einzelne Änderungsoperation liegt dabei in Form einer eigenen Klasse vor, die die Operation beschreibt (in der Abbildung: *cplxChangeOp_1, cplxChangeOp_2, ...).*

Das Interface *ComplexChangeOperation* muß Methoden anbieten, mit denen ein Client Informationen über zur Verfügung stehende Änderungsoperationen anfordern kann. Dazu können die Operationen Auskunft über benötigte Parameter, Name und Art der Änderung, Semantik (als textuelle Beschreibung für den Anwender) usw. geben.
Außerdem müssen die Klassen, die eine Änderungsoperation beschreiben, eine Methode `performOperation(params)` besitzen, über welche sie überhaupt erst ausführbar werden.

Auch die einfachen Änderungsoperationen können über den *ChangeOpIntegrator* bereitgestellt werden. Die Bereitstellung der Schnittstelle *ChangeOperation* ist somit nicht mehr zwingend erforderlich.

Vorerst wird das Konzept der dynamisch ladbaren Änderungsoperationen allerdings noch nicht umgesetzt, da komplexe Änderungsoperationen bis zu diesem Zeitpunkt noch nicht spezifiziert sind. So soll die Architekturzeichnung in Abbildung 7-9 nur eine Idee vermitteln, wie komplexe Änderungsoperationen zukünftig in den OMM-Server eingebunden werden können.

7.5.5 Schnittstelle DBAccess

Da insbesondere Schnittstellen in der Softwareentwicklung von großer Bedeutung sind, wird im folgenden das Interface *DBAccess* im Detail vorgestellt.

Die Schnittstelle DBAccess ermöglicht es, den Zugriff auf eine konkrete DB transparent zu gestalten. Trotz SQL-Standards unterscheiden sich RDBMS in der Art und Weise, wie Angaben über die Metadaten einer DB gewonnen werden können.

38 PTEditor (Process Template Editor)

enum DataType = {int, float, String, Date}

`ResultSet executeQuery (String sqlStatement)throws Exception`

Zum Ausführen von SQL-Statements, die ohne Anpassungen von allen implementierenden DBs akzeptiert werden. Für SQL-Statements, die eine Ergebnismenge zurückliefern (*SELECT*).

Für die Implementierung wird vorgeschlagen, diesen Aufruf an die Methode `executeQuery` aus dem Package `java.sql.Statement` weiterzureichen. Rückgabewert und Exception sind in der JDK-Dokumentation beschrieben.

`int executeUpdate (String sqlStatement) throws Exception`

Zum Ausführen von SQL-Statements, die ohne Anpassungen von allen implementierenden DBs akzeptiert werden. Für SQL-Statements, die keine Ergebnismenge zurückliefern. (*INSERT, DELETE*)

Für die Implementierung wird vorgeschlagen, diesen Aufruf an die Methode `executeQuery` aus dem Package `java.sql.Statement` weiterzureichen. Rückgabewert und Exception sind in der JDK-Dokumentation beschrieben.

`boolean tableExists (String tableName)`

Um festzustellen, ob eine angegebene Tabelle in der DB existiert.

`List<String> getColumnNames (String tableName)`

Liefert die Spalten einer Tabelle als Liste von Strings

`boolean columnExists (String tableName, String columnName)`

Prüft, ob die angegebene Spalte in der angegebenen Tabelle vorhanden ist.

Folgender Anwendungsfall veranschaulicht die Notwendigkeit dieser Methode: Durch eine Änderungsoperationen wird ein Attribut für eine Entität neu angelegt. Dazu muß überprüft werden, ob es nicht schon existiert.

- Überläßt man diese Prüfung der Datenbank, so muß für jede Datenbank unter Umständen ein eigener Fehler interpretiert werden.
- Ein Benutzer legt die Attribute "Name", NaME" und "name" an. Die verwendete Datenbank unterstützt die unterschiedlichen Schreibweisen und legt neue Spalten an, welche durch die Bearbeiterformeln nicht referenziert werden können. Dieser Fehler kann dann nur mit großer Mühe aufgedeckt werden.

```
Datatype getDataType (String tableName, String columnName)
                      throws Exception
```

Liefert den Datentyp des angegebenen Attributes.
Dazu ist ein entsprechendes Mapping zwischen den Java-Datentypen und den Datentypen der Datenbanken erforderlich.
Wirft eine Exception falls *tableName* oder *columnName* nicht existieren.

```
void addColumn (String tableName, String columnName,
                Datatype datatype)
                throws Exception
```

Es ist anzunehmen, daß die verwendeten Datenbanken sich in der Syntax der „ALTER TABLE“ Anweisung unterscheiden. Deshalb wird diese Funktion schon durch das Interface angeboten. Es ist vorgesehen, diese Methode mittels „ALTER TABLE“ zu implementieren.
Die Exception wird geworfen, falls das Attribut schon existiert, egal mit welchem Datentyp.

```
void deleteColumn (String tableName, String columnName)
                   throws Exception
```

Es ist vorgesehen, diese Methode mittels „ALTER TABLE“ zu implementieren.
Die Exception wird geworfen, falls das Attribut nicht gelöscht werden darf (wenn es sich also nicht um ein optionales Attribut handelt).

Da in erster Linie die Effizienz der Interaktion mit dem Datenhaltungssystem für die Gesamtleistung der Org.Modell-Komponente verantwortlich ist, ist es von Bedutung, das Datenhaltungssystem möglichst in schnellster Betriebsart zu betreiben. Alle bekannten Datenbanken bieten dazu interne Cachingverfahren an. Ohne weitere Konfiguration werden oft benötigte Daten vom DB-Server im schnellen Primärspeicher gehalten, um nachfolgende Anfragen schneller beantworten zu können, als wenn die Daten erneut vom Sekundärspeicher gelesen werden müßten. Neben dieser recht unspezifischen Art der Performanzsteigerung existiert bei manchen Systemen die Möglichkeit, explizit zu bestimmen, welche Daten grundsätzlich im Primärspeicher gehalten werden sollen.

Als Beispiel wird hier *MySQL* (http://www.mysql.com) angeführt. Laut Dokumentation [MySQL] lassen sich bei MySQL Tabellen vom Typ '*Memory*' anlegen, welche sich zur Laufzeit des DB-Servers komplett im Hauptspeicher befinden. Damit lassen sich bestimmte, oft benötigte oder auch alle Tabellen der Org.Modell-Komponente (siehe nächster Abschnitt) gezielt im Hauptspeicher halten. Beim Beenden des DB-Servers gehen allerdings die Inhalte dieser Tabellen verloren, wenn sie nicht vorher in den Sekundärspeicher zurückgeschrieben werden. Übrig bleiben nur die Definitionen der '*Memory*'-Tabellen. So muß also vor dem ersten Zugriff auf eine solche Tabelle zuerst der Inhalt von einer Tabelle im Sekundärspeicher in diese eingelesen werden. Während des Betriebs der Org.Modell-Komponente können Lese- und Schreibzugriffe auf die Tabellen im Hauptspeicher erfolgen. Es muß nur sichergestellt werden, daß bei Beendigung der Laufzeit die eventuell veränderten Daten wieder in den Sekundärspeicher zurückgeschrieben werden.

Diese Logik sollte allerdings aus Sicht der Architektur unterhalb der Schnittstelle *DBAccess* implementiert werden. Es bietet sich an, für diese zweite Art des Zugriffs auf eine Datenbank eine weitere Implementierung der Schnittstelle *DBAccess* anzubieten.

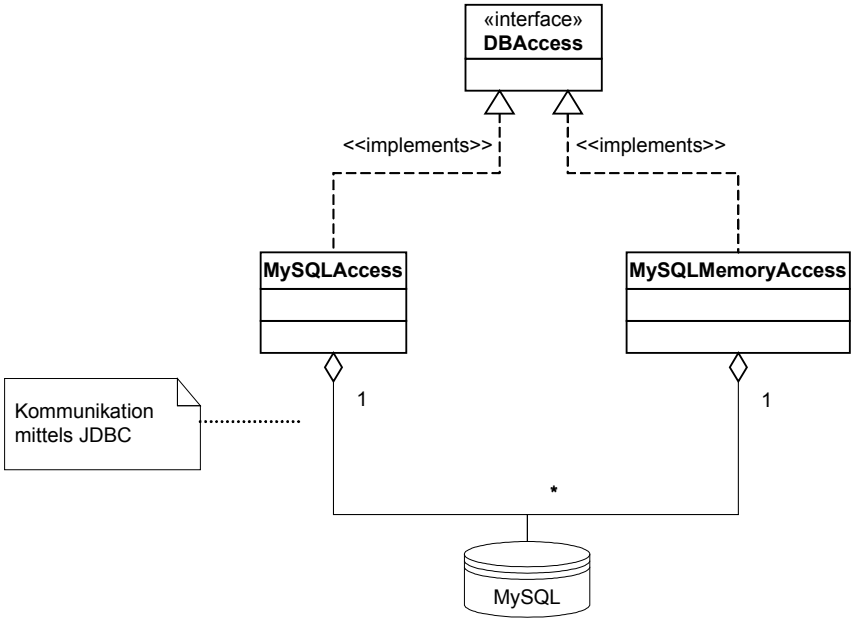


Abbildung 7-10: *Verschiedene Zugriffsarten für eine Datenbank*

Abbildung 7-10 zeigt wie diese zweite, implementierende Klasse unterhalb von *DBAccess* angeordnet ist. Dabei können durch beide Klassen ein und dieselbe Datenbank angesprochen werden. Allerdings müssen dabei eventuell noch Synchronisationsprobleme gelöst werden, die entstehen, wenn beide Klassen zugleich dieselbe Datenbank nutzen und die eine Klasse in Tabellen in Primärspeicher und die andere auf Tabellen im Sekundärspeicher schreibend zugreifen. Es ist aber nicht absehbar, daß ein zeitgleicher Zugriff auf eine Datenbank über zwei verschiedene Klassen benötigt wird. Somit ist die Synchronisationsproblematik nicht weiter ausschlaggebend. Um die Wiederverwendbarkeit des Codes voll auszuschöpfen ließe sich die Klasse *MySQLMemoryAccess* von der Klasse *MySQLAccess* ableiten.

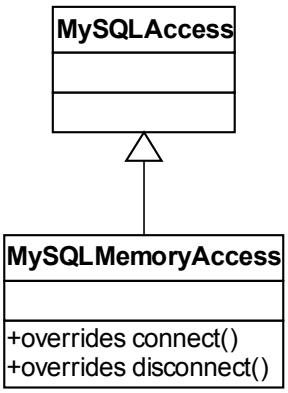


Abbildung 7-11: *Vererbung von Klassen für den Datenbankzugriff*

Dazu muß die Klasse *MySQLMemoryAccess* nur die Methoden *connect()* und *disconnect()* überschreiben, um darin die 'Memory'-Tabellen mit Daten zu füllen und nach Änderungen an diesen Daten in *disconnect()* wieder zurückzuschreiben (siehe Abbildung 7-11).

Allerdings ist nicht sicher, ob diese Verfahrensweise überhaupt Vorteile in der Zugriffsgeschwindigkeit bietet. Es ist anzunehmen, daß auch die internen Cachingmechanismen anderer Datenbanken einen effizienten Zugriff gestatten.

7.6 Modularer Aufbau des Metamodells

Wie in der konzeptionellen Betrachtung zu Beginn von Abschnitt 3.1 schon kurz erwähnt wurde, läßt sich das Metamodell an die Anforderungen einer Organisation dahingehend anpassen, daß nicht verwendete Entitäten ausgeblendet werden können. Damit läßt sich die Komplexität sowohl beim Modellieren der Organisationsstruktur als auch beim Erstellen der Bearbeiterformeln effektiv verringern. Eine Benutzerschnittstelle, wie die in Abschnitt 7.4 vorgestellte, hat damit die Möglichkeit, nicht benutzte bzw. nicht aktivierte Entitäten auszublenden. Trotzdem ist es dem Modellierer weiterhin freigestellt, diese Entitäten manuell in Bearbeiterformeln zu referenzieren. Hier offenbart sich der eigentliche Sinn der Modularisierung des Metamodells.

Eine Organisation kann sich zu Beginn des Einsatzes eines WfMS mit einem einfachen Organisationsmodell zufrieden geben und später zum Zweck der Modellierung komplexerer Bearbeiterformeln das Metamodell an die geänderten Bedürfnisse anpassen. Das bedeutet dann die Hinzunahme neuer Entitäten ins Metamodell. Die genauere Beschreibung der Organisation, die dadurch möglich wird, führt auch zu einer gesteigerten Ausdrucksmächtigkeit der Bearbeiterformeln, da nun mehrere Entitäten referenziert werden können.

Nicht vorgesehen ist das nachträgliche Entfernen oder Löschen von Entitäten aus dem Organisationsmetamodell. Es stellt sich die Frage, wie in diesem Fall Bearbeiterformeln zu interpretieren wären, die sich auf diese Entitäten beziehen. Wäre das nachträgliche Ausblenden von Entitäten erlaubt, würde die bedeuten, daß die ausgeblendeten Enitäten dem Modellierer nicht mehr zur graphisch unterstützen Modellierung von Bearbeiterformeln zur Verfügung stehen. Bestehende Bearbeiterformeln würden dadurch nicht beeinträchtigt.

Selbst nicht aktivierte, leere Entitäten lassen sich durch manuell erstellte Bearbeiterformeln referenzieren. Es gibt dabei keine Besonderheiten, die beachtet werden müßten. Eine Anfrage gegen eine leere Entität zu stellen fügt sich nahtlos in das mengentheoretische Konzept der Anfragesprachen (siehe Kapitel 5) ein. Die Semantik einer solchen Formel weist dabei keine Besonderheit auf.

Die Modularisierung des Organisationsmetamodells ist folglich kein Mechanismus der Org.Modell-Komponente. Das Nichtdarstellen von Teilen des Metamodells bedeutet für das Verhalten der Komponente keine Einschränkung. Damit wird letztendlich nur eine vereinfachte Darstellung gegenüber dem Benutzer ermöglicht.

Auch ausgeblendete Entitäten existieren in der Org.Modell-Komponente zu jedem Zeitpunkt.

Definition 7-1: Aktivierte / Nicht aktivierte Entitäten

Das minimale Metamodell besteht aus den Entitäten *OrgUnit, OrgPosition* und *Agent*. Diese Entitäten sind immer aktiviert, selbst wenn sie keine Datensätze beinhalten. Alle anderen Entitäten gelten als nicht aktiviert, wenn sie leer sind und als aktiviert, sobald sie mindestens einen Datensatz beinhalten.

Die Org.Modell-Komponente stellt Methoden zur Verfügung, mit denen ein Client für jede Entität feststellen kann, ob diese aktiviert ist.

Es bleibt hierbei noch offen, ob ein Client alle aktivierten Entitäten darstellt oder nicht. Zur vereinfachten Darstellung bleibt es letztendlich dem Benutzer überlassen, ob er sich alle aktivierten Entitäten anzeigen läßt.

Realisiert wird der Status einer Entität intern konkret dadurch, daß festgestellt wird, ob die zugehörige Tabelle der Datenbank Datensätze enthält oder nicht. Das ist an dieser Stelle keine Einschränkung auf eine relationale Datenbank. Ähnliches läßt sich auch mit anderen Datenhaltungssystemen bewerkstelligen.

7.7 Auflösung von Bearbeiterformeln

Dieser Abschnitt beschreibt die Vorgehensweise, wie die Bearbeiterformeln auf die Menge potentielle Bearbeiter aufgelöst werden. Beim konkreten Einsatz einer relationalen Datenbank bedeutet dies eine Umsetzung der Bearbeiterformeln in SQL-Select-Statements.

Das Ergebnis einer Bearbeiterformel besteht, wie zu Beginn von Kapitel 5 beschrieben, nicht nur aus einer Menge von Mitarbeiter-IDs, sondern aus einer Menge von Tupeln der Form (StelleID, MitarbeiterID). Da der Pfad einer Bearbeiterformel immer bei der Entität Agent endet[39], wird bei der Pfadverfolgung in den meisten realistischen Fällen auch die Information über die von den Agenten besetzten Stellen verfügbar sein. Das erleichtert die Übersetzung der häufigsten Formeln. Bei einem Pfad mit der Endung „Ability → Agent“ steht diese Information nicht zur Verfügung, so daß hier eine aufwendigere Verfahrensweisen notwendig ist.

7.7.1 Beispielhafte Umsetzung nach SQL

Jede Funktion liefert ihr Ergebnis als Menge von IDs der entsprechenden Zielentität. Die Navigation im Metamodell wird dabei auf Joins abgebildet. Dabei muß insbesondere auf die Attribute geachtet werden, über denen der Join gebildet wird. Um dies zu verdeutlichen, wird zuerst an einem Beispiel gezeigt, wie ein solches SQL-Statement zu entwickeln ist. Im Anschluß wird daraus die allgemeine Form erarbeitet.

Das folgende Beispiel zeigt die Umwandlung der Bearbeiterzuordnung:

(ProjectGroup.Budget > '1000000')
.getManager().getOrgUnit().getSupOrgUnit().getManager()

Diese Bearbeiterzuordnung beschreibt die Leiter von Organisationseinheiten, die untergeordnete Organisationseinheiten besitzen, in denen sich Projektleiter mit Projekten mit einem Budget von mehr als 1 Mio. befinden.

Der erste Teilausdruck „*(ProjectGroup.Budget > '1000000')*“ beschreibt die IDs der Projektgruppen, die ein Budget von mehr als 1 Mio. haben. Dieser Teilausdruck hat in SQL folgende Form:

```
SELECT PG.ID
FROM ProjectGroup AS PG
WHERE PG.Budget > 1000000
```

39 Der Pfad einer Bearbeiterformel endet zumindest implizit über die Pfadvervollständigung immer bei der Entität Agent.

Durch die Funktion *.getManager()* werden daraus die Projektleiter selektiert.

```
SELECT PG.ManagerID
FROM ProjectGroup AS PG
WHERE PG.Budget > 1000000
```

Durch die Anwendung der Funktion *.getOrgUnit()* werden die Org.Einheiten beschrieben, in denen sich diese Projektleiter befinden.

```
SELECT OP.OrgUnitID
FROM ProjectGroup AS PG
     JOIN OrgPosition AS OP ON PG.ManagerID = OP.ID
WHERE PG.Budget > 1000000
```

.getSupOrgUnit() ermittelt davon dann die IDs der Org.Einheiten, die diesen übergeordnet sind.

```
SELECT OU.SupID
FROM ProjectGroup AS PG
     JOIN OrgPosition AS OP ON PG.ManagerID = OP.ID
     JOIN OrgUnit AS OU ON OP.OrgUnitID = OU.ID
WHERE PG.Budget > 1000000
```

Mittels *.getManager()* erhält man dann die IDs der Leiter dieser Org.Einheiten

```
SELECT OU2.ManagerID
FROM ProjectGroup AS PG
     JOIN OrgPosition AS OP ON PG.ManagerID = OP.ID
     JOIN OrgUnit AS OU ON OP.OrgUnitID = OU.ID
     JOIN OrgUnit AS OU2 ON OU.SupID = OU2.ID
WHERE PG.Budget > 1000000
```

Wie in Abschnitt 5.2 ausführlich behandelt, muß vor der Übersetzung noch die implizite Pfadvervollständigung stattfinden. Folgt man den in Tabelle 5-5 angegebenen Standardsemantiken kommt man zur vollständigen Bearbeiterformel

(ProjectGroup.Budget > '1000000').getManager()
*.getOrgUnit().getSupOrgUnit().getManager()**.getAgent()***

Die Pfadvervollständigung ist in fetter Schrift angegeben. Die Entsprechung der kompletten Bearbeiterformel in SQL lautet dann:

```
SELECT OP2.ID, OPO.AgentID
FROM ProjectGroup AS PG
     JOIN OrgPosition AS OP ON PG.ManagerID = OP.ID
     JOIN OrgUnit AS OU ON OP.OrgUnitID = OU.ID
     JOIN OrgUnit AS OU2 ON OU.SupID = OU2.ID
     JOIN OrgPosition AS OP2 ON OU2.ManagerID = OP2.ID
     JOIN OrgPositionOccupation AS OPO ON OP2.ID =
                                          OPO.OrgPositionID
WHERE PG.Budget > 1000000
```

Beobachtungen:

- Das SQL-Statement läßt sich den Funktionen entlang schrittweise entwickeln.
- Mit jeder angewandten Funktion wird ein zusätzlicher Join in das SQL-Statement eingefügt. (Führt die Funktion über eine (n:m)-Beziehung werden wegen der zusätzlich notwendigen Zwischenrelation zwei Joins eingefügt.)
- Bei jedem Schritt „wandert“ das projizierte Attribut des vorhergehenden Schrittes als Join-Attribut in die Joinkette des nachfolgenden Schrittes und bildet mit der ID der neu hinzugejointen Tabelle das Joinkriterium.
- Auffällig ist auch, daß im endgültigen SQL-Statement nicht nur ein sondern zwei Attribute projiziert werden. Dies sind also sowohl die Mitarbeiter, als auch die Stellen, die sie besetzen.

Die an dem Beispiel gezeigte Vorgehensweise ist schon dahingehend optimiert, daß bei Funktionen, die eine (1:1)- oder (1:n)- Beziehung verfolgen, kein Join angewandt wird. Denn die Information der IDs der Datensätze der Zielentität ist bereits in der Entität zu finden, die dieser auf dem Pfad einen Schritt vorangeht (siehe Schritt 2: *.getManager()*). Für die systematische Betrachtung ist das allerdings nicht unbedingt von Vorteil. In obigem Beispiel wird dabei ein Join zur nächsten Entität immer erst einen Schritt nach der angewandten Funktion eingefügt. Im folgenden wird eine Vorgehensweise beschrieben, bei der durch die Anwendung einer Funktion immer im selben Schritt auch der Join bis hin zur entsprechenden Zielentität vollzogen wird. Das erscheint auf den ersten Blick ineffizient. Hält man sich aber vor Augen, daß die durch die Teilschritte sukzessive enstehenden SQL-Statements zwar syntaktisch korrekt sind, aber nicht konkret auf einer Datenbank ausgeführt werden müssen, sondern nur temporäre Zwischenschritte darstellen, verliert die Argumentation der scheinbar überflüssigen Joins an Gewicht.

Außerdem führt der Pfad einer jeden Bearbeiterformel letztendlich zur Entität Agent, somit ist es unerheblich, ob in einen Zwischenschritt ein an dieser Stelle nicht notwendiger Join vorweggenommen wird. Auf jeden Fall aber werden nicht benötigte Joins von der Anfrageoptimierung der Datenbank wieder aus der Anfrage entfernt. Wird bei Anwendung einer Funktion immer der Join-Pfad auf die Zielentität der Funktion gelenkt, erhält man außerdem einen sanfteren Übergang von der Konzeption zum Entwurf. Damit lassen sich die Einwirkungen jeder einzelnen Funktion auf die Zwischenergebnisse besser nachvollziehbar beschreiben.

Zur Verdeutlichung wird noch einmal die obige Formel mit verfollständigtem Pfad herangezogen.

(ProjectGroup.Budget > '1000000').getManager()
*.getOrgUnit().getSupOrgUnit().getManager()**.getAgent()***

Der erste Teilausdruck „*(ProjectGroup.Budget > '1000000')“* wird genau wie oben übersetzt zu:

```
SELECT PG.ID
FROM ProjectGroup AS PG
WHERE PG.Budget > 1000000
```

Anwendung der Funktion *.getManager()*

```
SELECT PG.ManagerID
FROM ProjectGroup AS PG
     JOIN OrgPosition AS OP ON PG.ManagerID = OP.ID
WHERE PG.Budget > 1000000
```

Anwendung der Funktion *.getOrgUnit()*

```
SELECT OP.OrgUnitID
FROM ProjectGroup AS PG
     JOIN OrgPosition AS OP ON PG.ManagerID = OP.ID
     JOIN OrgUnit AS OU ON OP.OrgUnitID = OU.ID
WHERE PG.Budget > 1000000
```

Anwendung der Funktion *.getSupOrgUnit()*

```
SELECT OU.SupID
FROM ProjectGroup AS PG
     JOIN OrgPosition AS OP ON PG.ManagerID = OP.ID
     JOIN OrgUnit AS OU ON OP.OrgUnitID = OU.ID
     JOIN OrgUnit AS OU2 ON OU.SupID = OU2.ID
WHERE PG.Budget > 1000000
```

Anwendung der Funktion *.getManager()*

```
SELECT OU2.ManagerID
FROM ProjectGroup AS PG
     JOIN OrgPosition AS OP ON PG.ManagerID = OP.ID
     JOIN OrgUnit AS OU ON OP.OrgUnitID = OU.ID
     JOIN OrgUnit AS OU2 ON OU.SupID = OU2.ID
     JOIN OrgPosition AS OP2 ON OU2.ManagerID = OP2.ID
WHERE PG.Budget > 1000000
```

Anwendung der Funktion *.getAgent()*

```
SELECT OPO.OrgPositionID, A.ID
FROM ProjectGroup AS PG
     JOIN OrgPosition AS OP ON PG.ManagerID = OP.ID
     JOIN OrgUnit AS OU ON OP.OrgUnitID = OU.ID
     JOIN OrgUnit AS OU2 ON OU.SupID = OU2.ID
     JOIN OrgPosition AS OP2 ON OU2.ManagerID = OP2.ID
     JOIN OrgPositionOccupation AS OPO ON OP2.ID =
                                          OPO.OrgPositionID
     JOIN Agent AS A ON OPO.AgentID = P.ID
WHERE PG.Budget > 1000000
```

7.7.2 Allgemeine Vorgehensweise

Anhand der beispielhaften Übersetzung nach SQL läßt sich die allgemeine Vorgehensweise nun einfacher nachvollziehen.

Eine zu übersetzende Bearbeiterzuordnung beginnt immer mit einer Selektion in folgender Form:

```
<EntityName>.<Attribut> <VglOP> 'Konstante'
```

diese Selektion wird übersetzt zu:

```
SELECT T$Count.ID
FROM <EntityName> AS T$Count
WHERE T$Count.<Attribut> <VglOP> 'Konstante'
```

Da der Pfad einer Bearbeiterzuordnung auch mehrmals über ein und dieselbe Entität führen kann, muß berücksichtigt werden, daß in diesem Fall die zugehörige Tabelle wiederholt in die Joinkette des Statements aufgenommen werden muß. Um die verschiedenen Inkarnationen der Tabellen dennoch eindeutig referenzieren zu können, werden Aliase über diesen definiert. Als Beispiel dafür steht die Tabelle *OrgUnit* in obigem Beispiel (*OrgUnit AS OU, OrgUnit AS OU2*). Um allerdings nicht für jede Tabelle einen eigenen Zähler mitführen zu müssen, wird eine zum Übersetzungsalgorithmus globale Zählervariable eingeführt. Deren Wert ist zu Beginn des Algorithmus *'0'* und wird für jedes neue Einbinden einer Tabelle um den Wert *'1'* inkrementiert. Dieses Verändern der Variable findet immer nach einer vollständig durchgeführten Änderung am SQL-Statement statt, und zwar jeweils um die Anzahl der neu hinzugekommenen Tabelleninkarnationen. Im oben dargestellten SQL-Ausdruck wird $Count jeweils durch den Wert dieser Variablen ersetzt. Die Aliase für die referenzierten Tabellen lauten somit „T0“ - „T*n-1*“.

Für jede Funktion, die in einer Bearbeiterzuordnung aufgerufen wird, muß am SQL-Statement eine Änderung vorgenommen werden. Zu diesem Zweck werden die drei Bestandteile des SQL-Statements (*SELECT*, *FROM* und *WHERE*) einzeln betrachtet.

Tabelle 7-1 zeigt dazu für jede Funktion des Modells[40] die erforderlichen Anpassungen an den einzelnen Teilen des zu konstruierenden SQL-Statements.

In den Zeilen der transitiven Funktionen steht nur der Vermerk *<REKURSIV>*. Die Übersetzung dieser Funktionen wird in Abschnitt 7.7.3 gesondert behandelt.

40 Die Funktionen sind schon aus Kapitel 5.2 bekannt. Die Tabellen 5-4 (S.69) und (S.) geben Auskunft über deren Anwendbarkeit und Semantik.

Funktion	SELECT	FROM
OrgUnit .getSubOrgUnit	**SET:** T$Count.ID	**ADD:** JOIN OrgUnit AS T$Count ON T($Count-1).ID = T$Count.SupID
OrgUnit .getSupOrgUnit	**SET:** T$Count.ID	**ADD:** JOIN OrgUnit AS T$Count ON T($Count-1).SupID = T$Count.ID
OrgUnit .getSubOrgUnit+	***<REKURSIV>***	
OrgUnit .getSupOrgUnit+	***<REKURSIV>***	
OrgUnit .getOrgGroup	**SET:** T($Count+1).ID	**ADD:** JOIN OrgGroupInvolvement AS T$Count ON T($Count-1).ID = T$Count.OrgUnitID JOIN OrgGroup AS T($Count+1) ON T$Count.OrgGroupID = T($Count+1).ID
OrgUnit .getProjectGroup	**SET:** T($Count+1).ID	**ADD:** JOIN ProjectInvolvement AS T$Count ON T($Count-1).ID = T$Count.InvolvedID JOIN ProjectGroup AS T($Count+1) ON T$Count.ProjectID = T($Count+1).ID **WHERE** **ADD:** AND T$Count.Discrimination = ORGUNIT
OrgUnit .getOrgPosition	**SET:** T$Count.ID	**ADD:** JOIN OrgPosition AS T$Count ON T($Count-1).ID = T$Count.OrgUnitID
OrgUnit .getManager	**SET:** T$Count.ID	**ADD:** JOIN OrgPosition AS T$Count ON T($Count-1).ManagerID = T$Count.ID
OrgGroup .getOrgUnit	**SET:** T($Count+1).ID	**ADD:** JOIN OrgGroupInvolvement AS T$Count ON T($Count-1).ID = T$Count.OrgGroupID JOIN OrgUnit AS T($Count+1) ON T$Count.OrgUnitID = T($Count+1).ID

Tabelle 7-1: *Übersetzung der Funktionen in SQL (1/4)*

Funktion	SELECT	FROM
OrgGroup .getManager	**SET:** T$Count.ID	**ADD:** JOIN OrgPosition AS T$Count ON T($Count-1).ManagerID = T$Count.ID
OrgPosition .getOrgUnit	**SET:** T$Count.ID	**ADD:** JOIN OrgUnit AS T$Count ON T($Count-1).OrgUnitID = T$Count.ID
OrgPosition .getManagedOrgUnit	**SET:** T$Count.ID	**ADD:** JOIN OrgUnit AS T$Count ON T($Count-1).ID = T$Count.ManagerID
OrgPosition .getManagedOrgGroup	**SET:** T$Count.ID	**ADD:** JOIN OrgGroup AS T$Count ON T($Count-1).ID = T$Count.ManagerID
OrgPosition .getProjectGroup	**SET:** T($Count+1).ID	**ADD:** JOIN ProjectInvolvement AS T$Count ON T($Count-1).ID = T$Count.InvolvedID JOIN ProjectGroup AS T($Count+1) ON T$Count.ProjectID = T($Count+1).ID **WHERE** **ADD:** AND T$Count.Discrimination = ORGPOSITION
OrgPosition .getManagedProjectGroup	**SET:** T$Count.ID	**ADD:** JOIN ProjectGroup AS T$Count ON T($Count-1).ID = T$Count.ManagerID
OrgPosition .getRole	**SET:** T($Count+1).ID	**ADD:** JOIN OrgPositionDescription AS T$Count ON T($Count-1).ID = T$Count.OrgPositionID JOIN Role AS T($Count+1) ON T$Count.RoleID = T($Count+1).ID
OrgPosition .getAgent	**SET:** T($Count+1).ID	**ADD:** JOIN OrgPositionOccupation AS T$Count ON T($Count-1).ID = T$Count.OrgPositionID JOIN Agent AS T($Count+1) ON T$Count.AgentID = T($Count+1).ID
ProjectGroup .getSubProjectGroup	**SET:** T$Count.ID	**ADD:** JOIN ProjectGroup AS T$Count ON T($Count-1).ID = T$Count.SupID
ProjectGroup .getSupProjectGroup	**SET:** T$Count.ID	**ADD:** JOIN ProjectGroup AS T$Count ON T($Count-1).SupID = T$Count.ID
ProjectGroup .getSubProjectGroup+	***<REKURSIV>***	
ProjectGroup .getSupProjectGroup+	***<REKURSIV>***	
ProjectGroup .getManager	**SET:** T$Count.ID	**ADD:** JOIN OrgPosition AS T$Count ON T($Count-1).ManagerID = T$Count.ID
ProjectGroup .getOrgUnit	**SET:** T($Count+1).ID	**ADD:** JOIN ProjectInvolvement AS T$Count ON T($Count-1).ID = T$Count.ProjectID JOIN OrgUnit AS T($Count+1) ON T$Count.InvolvedID = T($Count+1).ID **WHERE** **ADD:** AND T$Count.Discrimination = ORGUNIT

Tabelle 7-1: *Übersetzung der Funktionen in SQL (2/4)*

Funktion	SELECT	FROM
ProjectGroup .getOrgPosition	**SET:** T($Count+1).ID	**ADD:** JOIN ProjectInvolvement AS T$Count ON T($Count-1).ID = T$Count.ProjectID JOIN OrgPosition AS T($Count+1) ON T$Count.InvolvedID = T($Count+1).ID
		WHERE
		ADD: AND T$Count.Discrimination = ORGPOSITION
Role .getOrgPosition	**SET:** T($Count+1).ID	**ADD:** JOIN OrgPositionDescription AS T$Count ON T($Count-1).ID = T$Count.RoleID JOIN OrgPosition AS T($Count+1) ON T$Count.OrgPositionID = T($Count+1).ID
Role .getAbility	**SET:** T($Count+1).ID	**ADD:** JOIN RoleDescription AS T$Count ON T($Count-1).ID = T$Count.RoleID JOIN Ability AS T($Count+1) ON T$Count.AbilityID = T($Count+1).ID
Role .getSubRole	**SET:** T$Count.ID	**ADD:** JOIN Role AS T$Count ON T($Count-1).ID = T$Count.SupID
Role .getSupRole	**SET:** T$Count.ID	**ADD:** JOIN Role AS T$Count ON T($Count-1).SupID = T$Count.ID
Role .getSubRole+	***<REKURSIV>***	
Role .getSupRole+	***<REKURSIV>***	
Agent .getOrgPosition	**SET:** T($Count+1).ID	**ADD:** JOIN OrgPositionOccupation AS T$Count ON T($Count-1).ID = T$Count.AgentID JOIN OrgPosition AS T($Count+1) ON T$Count.OrgPositionID = T($Count+1).ID
Agent .getAbility	**SET:** T($Count+1).ID	**ADD:** JOIN AgentAbility AS T$Count ON T($Count-1).ID = T$Count.AgentID JOIN Ability AS T($Count+1) ON T$Count.AbilityID = T($Count+1).ID
Ability .getAgent	**SET:** T($Count+1).ID	**ADD:** JOIN AgentAbility AS T$Count ON T($Count-1).ID = T$Count.AbilityID JOIN Agent AS T($Count+1) ON T$Count.AgentID = T($Count+1).ID
Ability .getRole	**SET:** T($Count+1).ID	**ADD:** JOIN RoleDescription AS T$Count ON T($Count-1).ID = T$Count.AbilityID JOIN Role AS T($Count+1) ON T$Count.RoleID = T($Count+1).ID
Role .getSubstitutionRule	**SET:** T$Count.ID	**ADD:** JOIN SubstitutionRule AS T$Count ON T($Count-1).ID = T$Count.RoleID

Tabelle 7-1: *Übersetzung der Funktionen in SQL (3/4)*

Funktion	SELECT	FROM
SubstitutionRule .getSubstitute	**SET:** T$Count.ID	**ADD:** JOIN OrgPosition AS T$Count ON T($Count-1).SubstituteID = T$Count.ID
SubstitutionRule .getSubstituted	**SET:** T$Count.ID	**ADD:** JOIN OrgPosition AS T$Count ON T($Count-1).OrgPositionID = T$Count.ID
SubstitutionRule .getRole	**SET:** T$Count.ID	**ADD:** JOIN Role AS T$Count ON T($Count-1).RoleID = T$Count.ID
OrgPosition .getSubstitutionRule	**SET:** T$Count.ID	**ADD:** JOIN SubstitutionRule AS T$Count ON T($Count-1).ID = T$Count.OrgPositionID

Tabelle 7-1: *Übersetzung der Funktionen in SQL (4/4)*

Offensichtlich beeinflussen die Funktionen nur den Teil, in dem die Joinkette gebildet wird (*FROM*). Ausnahmen davon sind die Funktionen, die über die Projektbeteiligung führen. Da Projektmitglieder sowohl Org.Einheiten als auch Stellen sein können, müssen aus den zurückgelieferten Tupeln noch diejenigen selektiert werden (*WHERE*), die entweder eine Org.Einheit oder eine Stelle beschreiben. ORGEINHEIT und ORGPOSITION sind dabei globale Konstanten mit der Werten 0 und 1.

Änderungen an der Joinkette und der Selektion entsprechen jeweils nur Anfügen neuer Teile, während Änderungen der Projektion komplette Ersetzungen sind[41]. Projiziert wird jeweils der Primärschlüssel der zuletzt hinzugejointen Tabelle, also die ID der Zielentität. Streng genommen ist diese Ersetzung nicht notwendig, da der Pfad einer Bearbeiterzuordnung immer bei der Entität *Agent* endet. Das bedeutet, daß letztendlich immer die ID des Agenten projiziert wird. Allerdings hat diese Ersetzung zur Folge, daß nach Anwendung jeder Funktion ein syntaktisch und semantisch korrektes SQL-Statement als Zwischenergebnis entsteht.

Auch die Anwendung der Selektion[42] beeinflußt die Form des SQL-Statements. Allerdings sind diese Änderungen weit weniger komplex als bei der Übersetzung von Funktionen. Bei einer Selektion

```
<EntityName>.<Attribute> <VglOP> 'Konstante'
```

wird im Selektionsteil des SQL-Statements folgende Ergänzung gemacht:

```
AND T$Count.<Attribute> <VglOP> 'Konstante'
```

Eine Ausnahme davon bildet die Selektion, die auf die Funktion OrgPosition.getSubstitutionRole folgt. Wie schon erwähnt kann sich diese Selektion nicht nur auf Attribute der Entität *SubstitutionRole* sondern auch auf Attribute der Entität *Role* beziehen. Ist dies der Fall, so muß bei der Anwendung der Selektion auch noch ein Join auf die Entität *Role* vollzogen werden.

41 Im Sinne der Relationenalgebra ist der Teilausdruck hinter *SELECT* keine Selektion, sondern eine Projektion. In SQL wird die Selektion mit dem Schlüsselwort *WHERE* eingeleitet.

42 Hier: Eine Selektion im Kontext der Org.Modell Anfragesprache

Das bedeutet:

```
getSubstitutionRole(Role.<Attribute> <VglOP> 'Konstante')
```

bewirkt in der Joinkette :

```
ADD: JOIN Role AS T$Count
     ON T($Count-1).RoleID = T$Count.ID
```

und im Selektionsteil:

```
ADD: AND T$Count.<Attribute> <VglOP> 'Konstante'
```

Da hier ein neuer Alias für eine Tabelle eingeführt wurde, muß nach dieser Übersetzung auch die Zählervariable entsprechend inkrementiert werden. Tritt dieser spezielle Fall auf, so muß bedacht werden, daß eine Funktion, die nun angewendet wird, auf die Situation trifft, daß T$(Count-1) eben nicht die Entität *SubstitutionRole* sondern *Role* bezeichnet.
Für diesen Fall der drei von *SubstitutionRole* wegführenden Funktionen gelten folgende Übersetzungsregeln:

Funktion	SELECT	FROM
SubstitutionRule .getSubstitute	**SET:** T$Count.ID	**ADD:** JOIN OrgPosition AS T$Count ON T($Count-2).SubstituteID = T$Count.ID
SubstitutionRule .getSubstituted	**SET:** T$Count.ID	**ADD:** JOIN OrgPosition AS T$Count ON T($Count-2).OrgPositionID = T$Count.ID
SubstitutionRule .getRole	**SET:** T$Count.ID	**ADD:** JOIN Role AS T$Count ON T($Count-2).RoleID = T$Count.ID

Tabelle 7-2: *Übersetzung der Funktionen von SubstitutionRule (Spezialfall)*

Diese Besonderheit kommt aufgrund der ternären Beziehung von Stelle, Rolle und Vertreterregelung zustande. Wird nach *OrgPosition.getSubstitutionRole* eine Selektion auf Attribute von *Role* gefordert, muß ein zusätzlicher Join mit der entsprechenden Tabelle vollzogen werden. (Das wird in den meisten Fällen so sein, denn üblicherweise wird eine Vertretung bezüglich einer bestimmten Rolle gefordert und nicht irgendeine Vertretung der Stelle.) Allerdings wird dann der Pfad der Bearbeiterzuordnung trotzdem von *SubstitutionRole* aus fortgesetzt [T$(Count-2)]. Würde dies nicht beachtet werden, wäre die Übersetzung äquivalent zu :

```
OrgPosition.getSubstitutionRole()
.getRole(Role.Attribute ='Value').getSubstitutionRole()
.getSubstitute()
```

Für eine Menge von Stellen werden die Vertreterregeln ausgewählt. Für diese Vertreterregeln werden dann die Rollen bestimmt, für die diese Regeln gelten und mit einer Selektion eingeschränkt. Nun werden von diesen Rollen aus alle Vertreterregeln bestimmt und von diesen dann die Stellen die dort als Vertretung eingetragen sind.

und dieser Ausdruck hat eine andere als die geforderte Semantik:

```
OrgPosition
.getSubstitutionRole(Role.Attribute = 'Value')
.getSubstitute()...
```

> *Für eine Menge von Stellen werden die Vertreterregeln bestimmt, die bezüglich einer Auswahl an Rollen (Selektion) für diese Stellen definiert sind. Von diesen Vertreterregeln aus werden nun die Vertreterstellen bestimmt.*

Wie schon angedeutet ist die Ersetzung des Projektionsteils des SQL-Statements (SELECT) nicht unbedingt bei jedem Schritt notwendig. Die Ersetzung hat aber den Vorteil, daß nach jedem Zwischenschritt ein korrektes Statement als Zwischenschritt entsteht. Letztendlich ist aber bei jedem Statement die Projektion auf die ID der Stelle (*OrgPosition*) und die ID eines Benutzers (*Agent*) anzugeben. Bei der Übersetzung eines Ausdruckes muß also für jede Anwendung der Funktion *getAgent()* überprüft werden, ob es sich um die letzte Funktion im Pfad dieser Bearbeiterzuordnung handelt und ob der Pfad hierher über die Entität *OrgPosition* geführt hat. Ist dies der Fall, so werden die Attribute *T$(Count-1).OrgPositionID*[43] und *T$Count.ID* projiziert:

```
SELECT T$(Count-1).OrgPositionID, T$Count.ID
FROM ...
```

Ist die vorletzte Entität auf dem Pfad nicht *OrgPosition* bedeutet dies, daß für die Menge der potentiellen Bearbeiter keine Informationen über die besetzten Stellen vorliegen. In diesem Fall werden die Attribute *NULL* und *T$Count.ID* projiziert.

```
SELECT NULL, T$Count.ID
FROM ...
```

In der Liste der Tupel von möglichen Bearbeitern können also auch Tupel auftauchen, deren StellenID leer ist (NULL). Zu Beginn von Abschnitt 7.7 wurde darauf schon einmal eingegangen.

Für diese Bearbeiter wird die zu bearbeitende Aktivität nun nicht im Kontext einer Stelle angeboten. Diese Aktivität wird in der Arbeitsliste auflistet, die Aufgaben für den Bearbeiter beinhaltet, die dieser aufgrund von bestimmten Fähigkeiten zugewiesen bekommt und nicht auf Grund der Besetzung einer Stelle.

7.7.3 Übersetzung der transitiven Funktionen

Für die Übersetzung der transitiven Funktionen wird ein weiterer Bestandteil des SQL-Statements betrachtet. Mittels des Schlüsselwortes *WITH* wird die Definition temporärer Tabellen ermöglicht.
Der SQL-Standard erlaubt damit auch die rekursive Definition temporärer Tabellen. Die temporären Tabellen können wie alle anderen Tabellen im Statement referenziert werden. Dieses Vorgehen eignet sich, um die Funktionen, die Beziehungen transitiv verfolgen, in SQL umzusetzen. Tabelle 7-3 zeigt für die transitiven Funktionen den Aufbau der temporären Tabellen.

43 Bei T$(Count-1) handelt es sich dabei immer um die Tabelle OrgPositionOccupation

Funktion	WITH
OrgUnit.getSubOrgUnit+	**ADD:** T$($PARAMS) AS (SELECT $PARAMS FROM OrgUnit AS init WHERE init.ID = T($-1).ID UNION ALL SELECT $PARAMS FROM T$ AS parent JOIN OrgUnit AS child ON parent.ID = child.SupID)
OrgUnit.getSupOrgUnit+	**ADD:** T$($PARAMS) AS (SELECT $PARAMS FROM OrgUnit AS init WHERE init.ID = T($-1).ID UNION ALL SELECT $PARAMS FROM T$ AS parent JOIN OrgUnit AS child ON parent.SupID = child.ID)
ProjectGroup .getSubProjectGroup+	**ADD:** T$($PARAMS) AS (SELECT $PARAMS FROM ProjectGroup AS init WHERE init.ID = T($-1).ID UNION ALL SELECT $PARAMS FROM T$ AS parent JOIN ProjectGroup AS child ON parent.ID = child.SupID)
ProjectGroup .getSupProjectGroup+	**ADD:** T$($PARAMS) AS (SELECT $PARAMS FROM ProjectGroup AS init WHERE init.ID = T($-1).ID UNION ALL SELECT $PARAMS FROM T$ AS parent JOIN ProjectGroup AS child ON parent.SupID = child.ID)
Role.getSubRole+	**ADD:** T$($PARAMS) AS (SELECT $PARAMS FROM Role AS init WHERE init.ID = T($-1).ID UNION ALL SELECT $PARAMS FROM T$ AS parent JOIN Role AS child ON parent.ID = child.SupID)
Role.getSupRole+	**ADD:** T$($PARAMS) AS (SELECT $PARAMS FROM Role AS init WHERE init.ID = T($-1).ID UNION ALL SELECT $PARAMS FROM T$ AS parent JOIN Role AS child ON parent.SupID = child.ID)

Tabelle 7-3: *Rekursive Definition der temporären Tabellen für die transitiven Funktionen*

Bei der rekursiven Definition von temporären Tabellen muß darauf geachtet werden, daß nicht mittels einer Wildcard (*) alle Attribute als Spalten mit aufgenommen werden können. Deshalb muß vor der Übersetzung ermittelt werden, welche Attribute die entsprechende Entität besitzt. Diese ersetzen in Form einer kommaseparierten Aufzählung die Variable *$PARAMS*.

Wichtig ist, daß tatsächlich alle Attribute als Spalten in die definierte temporäre Tabelle aufgenommen werden, da zu diesem Zeitpunkt nicht bekannt ist, ob auf der Entität, auf der die transitive Funktion ausgeführt wird, eine Selektion folgt, und falls ja, auf welchem Attribut sie eine Bedingung beschreibt. Wird die Bearbeiterformel vor der Übersetzung schon komplett geparst, kann die Information über ein folgende Selektion bereits verfügbar sein. Dann müssen nur das Attribut, auf dem die Selektion angegeben ist, und die Schlüssel, über die die Entität betreten und wieder verlassen wird, in die temporäre Tabelle aufgenommen werden. Diese temporäre Tabelle wird nun wie die anderen Tabellen mit in den Join aufgenommen.

Für alle transitiven Funktionen aus Tabelle 7-3 gilt, daß dazu folgende Änderungen am SQL-Statement vollzogen werden müssen:

Funktion	SELECT	FROM
Alle transitiven Funktionen	**SET:** T$.ID	**ADD:** JOIN T$ ON T($-1).ID = T$.ID

Tabelle 7-4: *Übersetzung der transitiven Funktionen nach SQL*

Zur Demonstration wird noch einmal das Beispiel von Seite 131 herangezogen:

(ProjectGroup.Budget > '1000000').getManager()
.getOrgUnit().getSupOrgUnit+().getManager().getAgent()

Dabei wurde die Funktion *getSupOrgUnit()* durch die transitive Funktion *getSupOrgUnit+()* ersetzt wurde. Komplett in SQL übersetzt lautet diese Bearbeiterzuordnung nun:

```
WITH T3($PARAMS) AS
(
     SELECT $PARAMS
     FROM OrgUnit AS init
     WHERE init.ID = T2.ID
  UNION ALL
     SELECT $PARAMS
     FROM T3 AS parent
          JOIN OrgUnit AS child ON parent.SupID = child.ID
)
SELECT T5.OrgPositionID, T6.ID
FROM ProjectGroup AS T0
     JOIN OrgPosition AS T1 ON T0.ManagerID = T1.ID
     JOIN OrgUnit AS T2 ON T1.OrgUnitID = T2.ID
     JOIN T3 AS T3 ON T2.ID = T3.ID
     JOIN OrgPosition AS T4 ON T3.ManagerID = T4.ID
     JOIN OrgPositionOccupation AS T5 ON T4.ID =
                                         T5.OrgPositionID
     JOIN Agent AS T6 ON T5.AgentID = T6.ID
WHERE T0.Budget > 1000000
```

Hier müssen vor der Übergabe an die Datenbank noch die Attribute der Entität *OrgUnit* ermittelt werden und an Stelle von `$PARAMS` eingefügt werden. In diesem Beispiel ist es nicht möglich, konkrete Werte für die Attribute einzusetzen, da diese erst zur Laufzeit auf einem existierenden Organisationsmodell ermittelt werden können.

7.7.4 Umsetzung von Bearbeiterformeln

In den vorangegangenen Abschnitten wurde beschrieben, wie Bearbeiterzuordnungen in SQL übersetzt werden. Offen blieb dabei die Frage, wie mit kompletten Formeln umgegangen wird.

Bearbeiterformeln bestehen wie in Abschnitt 5.2.4 beschrieben, aus Bearbeiterzuordnungen, die mit den Operatoren AND, OR und EXCEPT verknüpft sind. Die Bindungsstärke dieser Operatoren läßt sich mit einer beliebigen aber korrekten Klammerung beeinflussen und somit auch die Semantik der Bearbeiterformel an die eigenen Bedürfnisse anpassen.
Die Übersetzung von kompletten Bearbeiterformeln in SQL geschieht folgendermaßen:

- Die Klammerung ist von der Bearbeiterformel ohne Änderungen direkt in SQL zu übernehmen.
- *AND* ist der Operator für die Schnittmengenbildung und wird in SQL zu *INTERSECT*.
- *OR* ist der Operator für die Mengenvereinigung und wird in SQL zu *UNION*.
- *EXCEPT* ist der Operator für die Bildung von Differenzen von Mengen und wird zum expliziten Ausschluss als *EXCEPT* in SQL übernommen.
- Die mittels den Operatoren und der Klammerung verknüpften Bearbeiterzuordnungen werden gemäß der vorangegangenen Abschnitte in SQL übersetzt.

Somit besteht die Möglichkeit, eine Bearbeiterformel in ein einziges – wenn auch komplexes – SQL-Statement zu übersetzen.

Bei näherer Betrachtung ist die Lösung zur Einbindung der Operatoren *AND*, *OR* und *EXCEPT* nicht ganz korrekt. Der Grund dafür ist, daß sich diese Operatoren auf Mengen von Tupeln der Form *(OrgPositionID, AgentID)* beziehen und nicht auf Mengen von einstelligen Attributen. Dieser Unterschied kann Folgen haben, die auf den ersten Blick nicht ersichtlich sind. Tabelle 7-5 zeigt für ein einfaches Beispiel von Ergebnisrelationen, die mittels der genannten Operatoren verknüpft werden, die tatsächliche Ergebnismenge und die eigentlich gewünschte Ergebnismenge. Diese stimmen für *UNION* überein. Bei *INTERSECT* und *EXCEPT* zeigen diese Unterschiede. Das liegt daran, daß zur Berechnung der Ergebnismengen die Werte aller Tupelbestandteile herangezogen werden und nicht nur die Werte, die nicht NULL sind. Es handelt sich dabei eigentlich um einen Grenzfall der Logik. Es ist fraglich ob gilt:

NULL ***AND*** *TRUE = TRUE* oder

NULL ***AND*** *TRUE = FALSE*

In diesem Fall gilt letztgenannte Aussage und das führt zur Abweichung von der gewünschten Semantik. Dies läßt sich umgehen, wenn für die Operatoren *INTERSECT* und *EXCEPT* noch Spezialfälle für die Übersetzung definiert werden.

Wird aus der einer Menge von Tupeln mittels EXCEPT eine Tupel entfernt, so ist die gewünschte Semantik die, daß alle Tupel entfernt werden, die im Wert von AgentID übereinstimmen so daß also eventuell auftretende NULL-Werte für OrgPositionID nicht berücksichtigt werden. Ähnlich verhält es sich auch mit dem Schnitt von Tupelmengen. Hier ist erwünscht, daß alle Tupel, die im Wert von AgentID übereinstimmen, mit in die Ergebnismenge aufgenommen werden.

OrgPositionID	AgentID		tatsächliches Ergebnis OrgPositionID	AgentID		gewünschtes Ergebnis OrgPositionID	AgentID
1015	250		1015	250		1015	250
1016	281		1016	281		1016	281
1017	289		1017	289		1017	289
UNION			NULL	281		NULL	281
NULL	281		NULL	290		NULL	290
NULL	290						
1015	250		1015	250		1016	281
1016	281		1016	281			
1017	289		1017	289			
EXCEPT							
NULL	281						
NULL	290						
1015	250		leer			1015	250
1016	281					1017	289
1017	289						
INTERSECT							
NULL	281						
NULL	290						

Tabelle 7-5: *Tatsächliche und gewünschte Ergebnisse von Mengenoperationen*

Grundsätzlich können dabei vier verschiedene Situationen entstehen. X, Y und N (N1, N2) seien Mengen von Tupeln *(OrgPositionID, AgentID)*, die mit den Operatoren *INTERSECT* oder *EXCEPT* verknüpft werden sollen. X und Y enthalten nur Tupel, in denen *OrgPositionID* keinen NULL-Wert annimmt. In N (N1, N2) können für *OrgPositionID* NULL-Werte auftauchen. Die vier Situationen sind wie folgt:

(1) X [**INTERSECT** | **EXCEPT**] Y

(2) N [**INTERSECT** | **EXCEPT**] Y

(3) X [**INTERSECT** | **EXCEPT**] N

(4) N1 [**INTERSECT** | **EXCEPT**] N2

Bei den Fällen (1) und (4) sind keine weiteren Anpassungen nötig, da die direkte Anwendung der Operatoren zur gewünschten Semantik führt. Bei den Fällen (2) und (3) jedoch muß für die Menge, in der die *OrgPositionIDs* Null-Werte aufweisen können (N), noch ein Join mit der Tabelle *OrgPositionOccupation* vollzogen werden, um die Null-Werte durch die konkreten IDs der Stellen zu ersetzen, die dieser Agent besetzt.

7.7.5 Kompensation der Abweichungen vom SQL-Standard

Wie schon erwähnt unterstützen die verfügbaren RDBMS den jeweils aktuellen SQL-Standard nur teilweise. Das trifft auf die Verwaltung von Metadaten zu und ebenso auf die eben angesprochene rekursive Definition temporärer Tabellen. Von den verfügbaren Datenbanken unterstützen DB2, Oracle und SQL Server 2005[44] diese Möglichkeit, jedoch alle mit unterschiedlicher Syntax. DB2 hält sich dabei noch am ehesten an den Standard.

Um allerdings die geforderte Transparenz gegenüber der zugrundeliegenden Datenbank gewährleisten zu können, müssen zur Übersetzung der transitiven Funktionen andere Konzepte erdacht werden. Nach Möglichkeit sollte auch nicht schon in der Klasse *DatabaseBackend* (siehe Architekturzeichnung in Abbildung 7-8 auf Seite 123) zwischen den verschiedenen Fähigkeiten der Datenbanken unterschieden werden, da sonst für jede Datenbank eine eigene Übersetzung nach SQL stattfinden müßte und die Unterschiede der Datenbanken sich über die Schnittstelle *DBAccess* hinaus auswirken würden.

Aus diesem Grund muß der „kleinste gemeinsame Nenner“ der verwendeten Datenbanken gefunden und die Übersetzung darauf aufgebaut werden. Eine Möglichkeit, die alle Datenbanken bieten, ist das Neuanlegen von Tabellen mittels „`CREATE TABLE`“ und das Einfügen von Werten in diese Tabellen mittels „`INSERT INTO`“. Manche Datenbanken unterstützen auch das Anlegen von Tabellen, die in Wirklichkeit nicht real existieren. Diese nicht materialisierten Tabellen werden Sichten (*Views)* genannt und mit dem Kommando „`CREATE VIEW`“ erzeugt. Gemeinsam ist diesen Tabellen und Sichten, daß sie wie alle anderen Tabellen in der Datenbank referenziert werden können.

Genau hier ist der Punkt, an dem die Unterschiede der Datenbanken ausgeglichen werden können. Mit dem Aufruf einer Methode in der Schnittstelle *DBAccess* kann eine Datenbank eine temporäre Tabelle zur Verfügung stellen, in der die transitiven Beziehungen manifestiert sind. Somit wird die Schnittstelle *DBAccess* um folgende Methoden erweitert:

```
enum TransitiveEntity = {Role, OrgUnit, ProjectGroup}
enum Direction = {Up, Down}

void createTemporaryTable (TransitiveEntity entity,
                           List<int> IDs, Direction dir,
                           String resultTableName)
                           throws Exception
```

Generiert für die angegebene Entität eine temporäre Tabelle, die ab den Einträgen mit den geforderten ID die transitiven Beziehungen beinhaltet. Die Richtung, entlang der die Beziehung zu verfolgen ist, muß durch den Parameter *dir* angegeben sein.
Der Parameter *resultTableName* ist der Name der zu erzeugenden temporären Tabelle.

Wirft eine Exception, falls ein Datensatz mit der angegeben ID nicht existiert.

```
void dropTemporaryTables (String tablenames[])
```

löscht die temporären Tabellen der Datenbank.

Bei näherer Betrachtung lassen sich aber mit der oben beschriebenen Methode `createTemporaryTable()` die geforderten temporären Tabellen nicht erzeugen. Das liegt daran, daß zum Zeitpunkt, zu dem die Methode aufgerufen wird – der Zeitpunkt der

44 Microsoft SQL-Server 2005 ist zum Zeitpunkt, da dieser Text verfaßt wird, noch nicht erhältlich.

Übersetzung der Formel in SQL – der Eingabeparameter *IDs* noch nicht verfügbar ist. Dieser ist erst nach der Ausführung des SQL-Statements verfügbar. Das ist scheinbar ein Paradoxon, da die Eingabeparameter der Methode erst nach ihrer Ausführung zur Verfügung stehen.

Allerdings werden als Eingabeparameter nur Werte benötigt, die während der Ausführung der Methode anfallen, nicht aber das komplette Endergebnis. Das Problem läßt sich also lösen, indem von den IDs als konkreten Werten abstrahiert wird und an deren Stelle nur eine Berechnungsvorschrift für diese übergeben wird. Diese Berechnungsvorschrift läßt sich durch ein SQL-Statement angeben, das eine einspaltige Ergebnisrelation mit den IDs für den Rekursionsanfang beinhaltet.

```
void createTemporaryTable (TransEntity entity,
                           SQLStatement StartIDs,
                           Direction dir,
                           String resultTableName)
                           throws Exception
```

Zur Verdeutlichung wird noch einmal das Beispiel von Seite 143 herangezogen.

Die Übersetzung der Bearbeiterformel verläuft wie gewohnt bis zu dem Punkt, an dem eine transitive Funktion aufgerufen wird.

```
SELECT T2.ID
FROM ProjectGroup AS T0
     JOIN OrgPosition AS T1 ON T0.ManagerID = T1.ID
     JOIN OrgUnit AS T2 ON T1.OrgUnitID = T2.ID
WHERE T0.Budget > 1000000
                          → SQL-Statement startIDs
```

Das bisherige SQL-Statement liefert genau die StartIDs für den Aufbau der temporären Tabelle mit den transitiven Beziehungen. Es muß also ein Aufruf der Methode `createTemporaryTable()` erfolgen. Dieser hat für das Beispiel folgende Form:

```
createTemporaryTable(OrgUnit, startIDs, Up, T3);
```

Danach wird die Übersetzung wie gewohnt weiter fortgesetzt. Enthält eine Bearbeiterformel mehrere transitive Funktionen innerhalb eines Pfades, muß dieses Vorgehen gegebenenfalls wiederholt werden. Die Berechnungsvorschrift für die benötigten Rekursionsanfänge ist immer das bis dahin gebildete SQL-Statement. Dieses Statement bezieht sich dabei dann schon auf zuvor temporär angelegte Tabellen.

Deshalb ist es wichtig, diese Tabellen in der richtigen Reihenfolge zu erzeugen, da zur Erzeugung der temporären Tabelle das SQL-Statement `startIDs` jeweils ausgewertet werden muß, um definierte Rekursionsanfänge zu erhalten.

Das komplett übersetzte SQL-Statement lautet:

```
SELECT T5.OrgPositionID, T6.ID
FROM ProjectGroup AS T0
     JOIN OrgPosition AS T1 ON T0.ManagerID = T1.ID
     JOIN OrgUnit AS T2 ON T1.OrgUnitID = T2.ID
     JOIN T3 AS T3 ON T2.ID = T3.ID
     JOIN OrgPosition AS T4 ON T3.ManagerID = T4.ID
     JOIN OrgPositionOccupation AS T5 ON T4.ID =
                                         T5.OrgPositionID
     JOIN Agent AS T6 ON T5.AgentID = T6.ID
WHERE T0.Budget > 1000000
```

,wobei T3 eine temporär erzeugte Tabelle ist

Mit Hilfe der beiden Methoden

- `createTemporaryTable()`
- `dropTemporaryTables()`

ist es nun möglich, transitive Funktionen auch ohne direkte Unterstützung von rekursiven Sichten zu implementieren. Auf welche Art und Weise eine zugrundeliegende Datenbank diese Tabellen zur Verfügung stellt ist unerheblich. Im Falle von DB2 kann das durch Erzeugen einer Sicht mit rekursiver Definition geschehen. Die Syntax ähnelt dabei der in Abschnitt 7.7.3 beschriebenen.
Unterstützt eine Datenbank diese Möglichkeit aber nicht, so muß die für diese Datenbank implementierende Klasse eine Tabelle oder Sicht erzeugen und deren Inhalt iterativ bestimmen.

Nach dem Ausführen der Anfrage sollten die verwendeten, zusätzlich angelegten Tabellen und Sichten mit der Funktion `dropTemporaryTables()` gelöscht werden.

7.8 Zusammenfassung

In diesem Kapitel wurden Entscheidungen für eine konkrete Implementierung diskutiert. Ausgangspunkt für diesen Entwurf sind die konzeptionellen Betrachtungen der vorangegangenen Kapitel. Zunächst wurde die Frage nach einem geeigneten Backend für die Speicherung des Organisationsmodells beantwortet. Mehrere Möglichkeiten wurden dazu in Betracht gezogen, unter anderem auch der Einsatz eines Verzeichnisdienstes. Um diese Entscheidung fällen zu können, wurden die Eigenschaften eines Verzeichnisdienstes ausführlich besprochen. Ein Verzeichnisdienst wie LDAP wird zwar nicht kategorisch für eine Implementierung ausgeschlossen, aber wegen der besseren Eignung werden relationale Datenbanken vorgezogen.

Da Schnittstellen (oder: *Interfaces*) den Schlüssel zu einem sauberen Entwurf darstellen, wurden diese ausführlich vorgestellt – insbesondere die Schnittstellen *PolicyResolution*, *ChangeOperation*, *Authentication* und *Discovery* , die die Funktionalität der Komponente nach außen definieren.

Um auch Modellierern, die die Methodik des Programmierens nicht beherrschen, den Entwurf von Bearbeiterformeln zu ermöglichen, wurden Überlegungen angestellt, diese Aufgabe komplett durch die Interaktion mit einer graphischen Benutzeroberfläche (GUI) zu erledigen.

Resultat dieser Überlegungen ist eine Benutzerschnittstelle, mit der schon allein durch Aktionen mit einem Zeigegerät Bearbeiterformeln modelliert werden können.

Die gewonnenen Erkenntnisse wurden in einem Entwurf vereint, wobei auch die Frage beantwortet wurde, wie die Funktionalität bezüglich der Organisationsverwaltung im Gesamtsystem zu verteilen ist. Ein Teilaspekt ist dabei auch die Frage nach dem Ort, an dem die Bearbeiterformeln verwaltet werden sollen. Dies wurde ausführlich diskutiert. Eine Frage des Entwurfs ist auch die Unabhängigkeit einzelner Softwareschichten voneinander. Unter diesem Aspekt wurde auch die Schnittstelle *DBAccess* besprochen, die die Abweichungen vom SQL-Standard kompensiert, was durch die Verwendung verschiedener Datenbanksysteme unausweichlich ist.

Im Hinblick auf die Wahl eines relationalen Datenbanksystems für die Verwaltung des Organisationsmodells wurde das Datenmodell in Form eines Relationenschemas beschrieben. Ebenso wurde gezeigt, wie der modulare Aufbau des Metamodells durch eine relationale Datenbank realisiert werden kann. Schließlich war auch noch die Übersetzung der Sprache für die Definition von Bearbeiterformeln in SQL ein zentrales Thema dieses Kapitels.

Ziel dieses Entwurfes ist eine Implementierung des Org.Modell-Servers zur Integration der Komponente in ADEPT2, dem zweiten Prototypen des Workflow-Management-Systems der Universität Ulm. Die Implementierung selbst ist auf Grund des anzunehmenden Aufwandes nicht Teil der Aufgabenstellung dieser Arbeit. Zur Zeit, da dieser Text verfaßt wird, befaßt sich ein Praktikum in der Abteilung DBIS (Datenbanken und Informationssysteme) mit der Umsetzung dieses Entwurfes. Die Implementierung des OMM-Servers ist zum aktuellen Zeitpunkt noch nicht abgeschlossen. Die meisten bisher aufgetauchten Probleme beziehen sich auf die Unterschiede der geforderten Datenbanksysteme.

8 Weitergehende Überlegungen

An dieser Stelle sollen Aspekte erwähnt werden, die nicht durch diese Arbeit abgedeckt sind. Unter ihnen finden sich sowohl Ideen, mit denen der erarbeitete Entwurf in seiner Effizienz verbessert werden kann als auch neue Anforderungen, die erst durch die nähere Beschäftigung mit dem Thema aufgekommen sind und die Entwicklung von Komponenten zur Organisationsmodellverwaltung zukünftig beeinflussen könnten.

8.1 Caches

Was im Rahmen dieser Arbeit nicht betrachtet wurde, sind Konzepte, wie die eingesetzten Algorithmen effizient unterstützt werden können. Als Beispiel hierzu wäre ein Cachingmechanismus im OMM-Server denkbar. Wird eine Bearbeiterformel an den OMM-Server übergeben, kann dieser aus der Formel mittels eines Hashverfahrens (beispielsweise MD5[45]) einen eindeutigen Wert für diese Formel ermitteln, die ihre Identität widerspiegelt. Dieser eindeutige Hashwert wird benötigt, um die Einträge im Cache den Formeln zuordnen zu können.
Ein Cache kann die Auflösung von Bearbeiterformeln nun in zwei Ebenen unterstützen:

- Die Einträge im Cache sind die Übersetzungen von Bearbeiterformeln. Wird eine Bearbeiterauflösung von der Org.Modell-Komponente gefordert, so kann der Cache nach Feststellung der Identität der Bearbeiterformel unter Umständen schon eine Übersetzung in die Anfragesprache des zugrundeliegenden Datenhaltungssystems liefern (im Entwurf für die Erstimplementierung ist das ein SQL-Statement). In diesem Fall entfällt das aufwendige Parsen und Übersetzen der Bearbeiterformel.

- Die Einträge im Cache sind Referenzen auf Mengen von potentiellen Bearbeitern. Wird eine Bearbeiterformel also in Aufgabenträger aufgelöst, wird die Ergebnismenge nicht nur als Rückgabewert an den Client übermittelt, sondern findet auch Eingang in den Cache des OMM-Servers. Wird nun dieselbe Bearbeiterformel ein weiteres Mal zur Auflösung an den OMM-Server übergeben, kann nach Feststellung der Identität der Bearbeiterformel ohne weitere Überprüfung die zuvor schon ermittelte Menge von potentiellen Bearbeitern zurückgeliefert werden.

Effizienter ist der Einsatz der zweiten Variante, da hier bereits fertige Ergebnismengen vorgehalten werden. Allerdings spricht auch nichts gegen den parallelen Einsatz beider Varianten. Eine allgemeine Problematik bei der Verwendung von Caches ist die Frage nach der Gültigkeit der Einträge. Werden nur wie in der ersten Variante die Übersetzungen der Bearbeiterformel abgelegt, sind die Einträge ohne Einschränkung zu jedem Zeitpunkt gültig. Das liegt daran, daß die Übersetzung eine Funktion der Form

$$SQLStatement = f(Bearbeiterformel)$$

darstellt. Das Ergebnis ist alleine von der Bearbeiterformel abhängig. Ändert sich die Bearbeiterformel, besitzt sie auch nicht mehr dieselbe Identität und kann auch nicht mehr mit dem vorhandenen Eintrag in Verbindung gebracht werden. Im Gegensatz dazu ist die Menge der potentiellen Bearbeiter eine Funktion der Form:

$$Bearbeitermenge = h(Bearbeiterformel) = g(f(Bearbeiterformel), OrgModell)$$

45 Siehe [Riv92]

Offensichtlich ist das Ergebnis der Funktion noch von dem zugrundeliegenden Org.Modell als zusätzlichem Parameter abhängig. Die Funktion kann also nach Änderungen am Org.Modell unterschiedliche Ergebnismengen liefern. Aus diesem Grund ist bei diesem Verfahren der Cache mit berechneten Ergebnismengen nach Änderungen am Org.Modell zu invalidieren.

8.2 Stored Procedures

Die in Abschnitt 7.5.2 getroffene Entscheidung, Bearbeiterformeln nicht serverseitig zu speichern, ist wie erwähnt lediglich ein Kompromiß. Dabei hätte die serverseitige Speicherung zumindest den Vorteil des effizienten Zugriffs auf die Bearbeiterformeln, falls nach Änderungen am Organisationsmodell Anpassungen an diesen notwendig sind.

Dieses Konzept stellte eine Analogie zum Einsatz von „*Stored Procedures*“ im Bereich der Datenbanken dar[46]. Ein konkretes Beispiel dafür sind die „*Prepared Statements*“ in JDBC[47]. Mit einem Befehl wie

```
int storeOrgPolicy (String orgPolicy)
```

würde ein Client den Server dazu veranlassen, eine angegebene Bearbeiterformel im Server zu speichern. Der Rückgabewert entspräche dabei der Referenz auf die gespeicherte Bearbeiterformel. Um den Server zur Auflösung einer gespeicherten Bearbeiterformel zu veranlassen, müßte dann im Interface *PolicyResolution* parallel zur Methode

```
List<AgentResult> resolvePolicy (String orgPolicy)
```

noch eine Methode

```
List<AgentResult> resolvePolicy (int storedOrgPolicy)
```

existieren, die keine Bearbeiterformel als Parameter erwartet, sondern nur die Referenz auf eine im Server gespeicherte Bearbeiterformel. Bei den Aktivitäten müßten dann nicht mehr die kompletten Bearbeiterformeln hinterlegt werden, sondern nur noch die Referenzen auf die im Server verwalteten Formeln.
Möglich wäre auch die serverseitige Speicherung von parametrisierten Prozeduren für abhängige Bearbeiterzuordnungen.
Dazu müßte die Auflösung der abhängigen Ausdrücke zwar ebenfalls schon vor der Speicherung aufgelöst werden. Allerdings würden die Positionen, an denen die konkreten Werte einzusetzen sind, und die erst zur Laufzeit der Prozeßinstanz verfügbar sind, in den Bearbeiterformeln mit einem Marker versehen werden. Bei einem Aufruf einer solchen parametrisierten benutzerdefinierten Funktion müßten diese ermittelten Werte dann mit übergeben werden. Eine Methode für einen solchen Aufruf hätte folgende Signatur:

```
List<AgentResult> resolvePolicy (int storedOrgPolicy,
                                 List<Parameter>)
```

Der Client muß aber in einem bestimmten Maß dabei trotz allem die Semantik der referenzierten Bearbeiterformel kennen, um die Werte zu ermitteln. Dieser Umstand wird nicht zu vermeiden sein.

46 Siehe [HäRa01]
47 Siehe [PrepStmt]

8.3 Deontische Modi

Bei allen bisherigen Überlegungen wurde ohne weiteres Hinterfragen angenommen, daß sobald ein Benutzer einen Eintrag in seiner Arbeitsliste vorfindet, er die zugehörige Aufgabe auch zu bearbeiten hat. Wenn nun aber eine Aufgabe in den Arbeitslisten verschiedener Benutzer auftaucht, muß jeder von ihnen annehmen, daß es seine Pflicht ist, diese Aufgabe zu erledigen. Werden jedem Benutzer alle Workitems präsentiert, welche er in der Lage ist abzuarbeiten, entsteht für ihn der Eindruck, daß die ganze Arbeit nicht zu bewältigen ist.

Eine Möglichkeit, diese Problematik näher zu beleuchten, bietet die deontische Logik. Es handelt sich dabei um eine Erweiterung der modalen Logik und beantwortet Fragen nach der Art der Handlungsaufforderung. Modalitäten dieser Logik sind Ausdrücke wie *geboten*, *freigestellt*, *erlaubt* und *verboten.*

[Buss98] enthält dazu einen kurzen Abriss, wie mit dieser Logik umzugehen ist und inwieweit sie im Zusammenhang mit WfMS zum Tragen kommt. Mit Hilfe dieser Logik wäre es möglich, für jedes Workitem, das einem Benutzer zugewiesen wird, dessen deontischen Modus auszumachen (geboten, freigestellt, erlaubt, verboten).

Der deontische Modus einer zu bearbeitenden Aktivität, die für einen Benutzer gilt, kann dabei entweder bei der Aktivität oder auch dem Benutzer hinterlegt sein. An dieser Stelle ist es auch lohnenswert, sich über verschiedene Zuteilungsverfahren Gedanken zu machen, mit denen der deontische Modus für eine Aktivität bestimmt werden kann. Möglich wäre hier ein Round-Robin-Verfahren über alle Benutzer, die sich als potentielle Bearbeiter der Aktivität qualifizieren. Möglicherweise ist die Zuteilung einer Aufgabe auch gerechter, wenn sie dem Bearbeiter zugewiesen wird, der die wenigsten Einträge in seiner Arbeitsliste hat. Alle andern möglichen Bearbeiter sehen diese Aktivität dann mit dem Vermerk „freigestellt“.

8.4 Versionierung

Für verschiedene Zwecke kann es nützlich sein, zu einem späteren Zeitpunkt als dem Moment der Ausführung die potentiellen Bearbeiter einer Aktivität ausfindig zu machen. Eine mögliche Fragestellungen wäre:

„*Wer war außer dem tatsächlichen Bearbeiter der Aktivität noch für deren Ausführung vorgesehen?*“

Um eine solche Frage zu beantworten, genügt es nicht, die Bearbeiterformel der Aktivität zu einem späteren Zeitpunkt noch einmal aufzulösen. Da das Org.Modell sich ständig durch Anwendung von Änderungsoperationen ändern kann, ist das Ergebnis einer Bearbeiterformel auch vom Zeitpunkt abhängig, zu dem sie aufgelöst wird. Um zu einem späteren Zeitpunkt das Ergebnis einer Bearbeiterformel noch einmal wiederherstellen zu können, muß die genaue Ausprägung des Organisationsmodells bekannt sein. Deshalb bietet es sich an, über Änderungsoperationen ein Protokoll zu führen. Dieses Protokoll kann auch in Form einer Versionsverwaltung konzipiert sein.

Auch dieser Aspekt ist durch die vorliegende Arbeit noch nicht behandelt worden, dürfte aber für zukünftige Überlegungen im Bereich der Organisationsverwaltung noch von großem Interesse sein.

8.5 Zugriffsrechte

In der Org.Modell-Komponente lassen sich alle Daten einer Organisation verwalten, die sich auf die Entitäten des Organisationsmetamodells beziehen. Somit bietet es sich an, die Personalstammdatenverwaltung über die Org.Modell-Komponente zu realisieren. Darin werden dann auch personenbezogene Daten wie das Gehalt oder Beurteilungen von Mitarbeitern abgelegt. Diese Daten sind offensichtlich besonders schützenswert und so ergibt sich die Forderung nach Konzepten zur Festlegung von Berechtigungen auf den Daten des Organisationsmodells. Die Regelung von Zugriffsberechtigungen auf bestimmte Attribute allein genügt dabei nicht. Ebenso muß geklärt werden, wer die Werte bestimmter Attribute setzen oder verändern darf und wer welche Änderungsoperationen auf welchen Datensätzen ausführen darf.

Da eine Bearbeiterformel jede Entität und jedes Attribut referenzieren kann, muß zur Auflösung von Bearbeiterformeln lesender Zugriff auf dem gesamten Org.Modell erlaubt sein. Deshalb sind auch Überlegungen anzustellen, wie sichergestellt werden kann, daß ein Benutzer durch die Kenntnis einer Bearbeiterformel und die Menge der resultierenden potentiellen Bearbeiter keine Rückschlüsse auf Daten ziehen kann, für die er keine Berechtigung besitzt.

8.6 Schlußfolgerung

Die im Laufe der Zeit aufgetretenen Anforderungen an eine Organisationsmodellverwaltung ähneln offenbar denen, die auch bei der Entwicklung von Datenbankmanagementsystemen während ihrer Entwicklung aufgetreten sind. Auch bei den DBMS kamen über die Zeit z.B. Forderungen nach systemseitiger Integritäts- und Konsistenzwahrung auf. Genau wie viele andere Aspekte wurden diese Konzepte in die Datenbanksysteme mit aufgenommen. Beispiele für Anforderungen, die sowohl bei Datenbanken als auch bei der Organisationsmodell-verwaltung eine Rolle spielen, sind:

- Systemseitige Integritäts- und Konsistenzwahrung der verwalteten Daten
- Änderungen am Datenbestand durch Operationen mit präzise definierter Semantik
  DBMS: (*SQL) UPDATE, ALTER TABLE*
  Org.Modell: *Änderungsoperationen*
- Transaktionslogik
- Maximal ausdrucksmächtige Anfragesprache
- Zugriffsberechtigungen
- Caches
- Systemseitig gespeicherte, benutzerdefinierte Funktionen (Stored Procedures)
- Versionierung

Offenbar durchlaufen Workflow-Management-Systeme hinsichtlich der geforderten Funktionalität eine ähnliche Entwicklung wie die Datenbanksysteme. Das ist nicht verwunderlich, da ein WFMS unter gewissen Aspekten für dieselben Aufgaben wie eine

Datenbank konstruiert ist – nämlich als Informationssystem.
Der Unterschied zwischen DBMS und WfMS ist, daß ein WfMS die Informationen auf semantisch höherer Ebene präsentiert. Zu diesem Zweck bietet sich der Aufbau eines WfMS auf Basis eines DBMS an. So ist es nicht verwunderlich, daß die Konzepte von Datenbanksystemen, deren Entwicklung den WfMS chronologisch vorangeht, mit der Zeit auch auf semantisch höheren Schichten gefordert werden.

So liegt die Vermutung nahe, daß zukünftige Anforderungen für Workflow-Management-Systeme nicht aus dem Nichts auftauchen, sondern ihr Ursprung mit hoher Wahrscheinlichkeit im Bereich der Datenbanksysteme liegt. Ein Blick auf die dort umgesetzten Konzepte lohnt sich somit immer und die Chance ist groß, daß sich mit dem Herstellen von Analogien, die bei Datenbanken verwirklichten Lösungen auch im Bereich der Workflow-Management-Systeme umsetzen lassen.

9 Zusammenfassung

Aufgabe der Arbeit war die Konzeption und der Entwurf einer Softwarekomponente für Organisationsmodelle. Anlaß für die Aufgabenstellung ist der Bedarf einer solchen Komponente im Kontext eines Workflow-Management-Systems (WfMS). Die gestellten Anforderungen lassen sich dabei eingrenzen mit der Modellierung und Verwaltung von Organisationsstrukturdaten und der Beantwortung von Fragestellungen, die sich auf diese Daten beziehen.

Dazu wurde ein ausdrucksmächtiges Organisationsmetamodell vorgestellt, welches die Möglichkeit bietet, die allermeisten Organisationsformen in ein konkretes Modell abzubilden. Auf den Einsatz eines Metametamodells, wie in [Buss98] beschrieben, wurde dabei bewußt verzichtet. Der Grund dafür ist der geringe Vorteil in der Gestaltung eines eigenen Metamodells, den der Benutzer aber mit dem Preis einer hohen Komplexität erkaufen muß. Der Anforderung nach Flexibilität des Metamodells wird dadurch Rechnung getragen, daß abgesehen von den Basisentitäten *OrgUnit*, *OrgPosition* und *Agent* die restlichen Entitäten des Metamodells je nach den gestellten Anforderungen entweder genutzt werden können oder nicht.

Da es sich bei einer Organisationsstruktur im Regelfall nicht um ein statisches Gebilde handelt, sondern um eine dynamische Struktur, wurden Operationen entwickelt, die Manipulationen auf dem Organisationsmodell ermöglichen, ohne dessen Integritätsbedingungen zu verletzen. Im Umfang dieser Arbeit findet sich dazu ein minimaler, vollständiger Satz an einfachen Änderungsoperationen, die je nach Belieben zu komplexen Änderungsoperationen aggregiert werden können.

Neben der Unterstützung der Modellierung einer Organisation ist die Beantwortung von Fragestellungen bezüglich des Organisationsmodells die zweite wichtige Forderung, die an die Org.Modell-Komponente gestellt wird. Konkret sind im Zusammenhang mit einem WfMS dabei Fragen nach potentiellen Bearbeitern einer Aktivität zu beantworten. Da sich ein potentieller Bearbeiter einer Aktivität aufgrund jeder erdenklichen Begebenheit, die im Organisationsmodell berücksichtigt ist, qualifizieren können soll, war die Bereitstellung einer geeigneten Anfragesprache gefordert. Diese Anfragesprache sollte so weit als möglich der Ausdrucksmächtigkeit des Organisationsmetamodells gerecht werden. Untersucht wurden dazu drei verschiedene Konzepte, wobei die Wahl auf das Konzept des navigierenden Zugriffs mittels Funktionen fiel. Insbesondere wurde dabei auch großer Wert auf die Möglichkeit der Formulierung abhängiger Bearbeiterzuordnungen gelegt. Die Abhängigkeit bedeutet dabei einen Bezug auf vergangene Ereignisse im Verlauf der Ausführung einer Prozeßinstanz, die z.B. Rückschlüsse darauf erlauben, wer einen bestimmten Prozeßschritt bearbeitet hat.

Da sich die Bearbeiterformeln auf die dynamische Datenstruktur eines Organisationsmodells beziehen, ist es möglich, daß nach Ausführung von bestimmten Änderungsoperationen Daten im Organisationsmodell nicht mehr existieren, die aber von Bearbeiterformeln noch direkt referenziert werden. Aus diesem Grund müssen nach bestimmten Änderungsoperationen die Bearbeiterformeln geprüft und eventuell angepaßt werden, um solche leeren Referenzen zu vermeiden.

Nach der Konzeption der Org.Modell-Komponente war die zweite wichtige Aufgabe der Arbeit, die Erkenntnisse der Konzeption im Rahmen eines konkreten Entwurfs in eine Lösung

umzusetzen. Eine wichtige Entscheidung in diesem Zusammenhang ist die Realisierung der Org.Modell-Komponente als Server (OMM-Server), der sowohl als als Server über eine Netzwerk-Verbindung, als auch als lokale Komponente eingebunden werden kann.

Nach einer näheren Betrachtung von LDAP (bzw. von Verzeichnisdiensten im Allgemeinen) wurde die Entscheidung gefällt, daß eine relationale Datenbank für die Zwecke der Org.Modell-Komponente besser geeignet ist als ein Verzeichnisdienst. Gründe dafür sind, daß die Abbildung von (n:m)-Beziehungen auf die Struktur eines Verzeichnisses mit den Mitteln eines Verzeichnisses nicht möglich ist, da ein Verzeichnis eine Baumstruktur beschreibt. Bedient man sich der Möglichkeit, die Position eines Objektes im Baum und deren Beziehungen zueinander mit Hilfe von Attributen zu beschreiben, läßt sich diese Beschränkung umgehen. Die entspricht dann aber nicht dem ursprünglichen Einsatzzweck eines Verzeichnisses. Problematisch ist auch die Mitbenutzung eines vorhandenen Verzeichnisses durch die Org.Modell-Komponente, da für deren korrekte Funktion nicht nur die Integritätsbedingungen der verwalteten Daten beachtet werden müssen, sondern nach Änderungen der Daten auch bestimmte Aktionen ausgelöst werden müssen (Anpassungen der Bearbeiterformeln nach Änderungen am Organisationsmodell).
Da also ein Verzeichnisdienst für die geforderten Zwecke nur ineffizient genutzt werden könnte und der exklusive Zugriff auf die zugrundeliegenden Daten von Vorteil ist, wurde für den Entwurf des OMM-Servers eine relationale Datenbank angenommen. Dennoch ist der Entwurf derart gestaltet, daß auch ein Verzeichnisdienst zur Datenhaltung herangezogen werden kann, wenn eine adaptierende Schicht dazu entwickelt wird.

Um auch Benutzern ohne die exakte Kenntnis der Syntax von Bearbeiterformeln die Modellierung derselben zu ermöglichen, wurde eine graphische Benutzerschnittstelle (GUI) entworfen. Mit Hilfe eines Clientprogrammes, welches diese Schnittstelle umsetzt, können durch bloße Interaktion mit der Maus syntaktisch korrekte Bearbeiterformeln modelliert werden.

Die externen und internen Schnittstellen des OMM-Servers als essentielles Ausdrucksmittel der Softwaretechnik wurden ausführlich besprochen. Von ebenso großem Interesse ist auch die Frage nach der Verteilung von Verantwortlichkeiten und Aufgaben im Gesamtsystem, in dessen Kontext die Org.Modell-Komponente nur einen Teil darstellt. In diesem Zusammenhang wurden auch die Vor- und Nachteile im Detail erörtert, die sich ergeben, wenn man die Bearbeiterformeln zum einen außerhalb und zum anderen innerhalb der Org.Modell-Komponente verwaltet. Die Entscheidung der Verwaltung von Bearbeiterformeln auf Clientseite stellt nur einen Kompromiß dar, auf den im Rahmen von weitergehenden Überlegungen im vorherigen Kapitel noch einmal eingegangen wurde.

Weitere Punkte des Entwurfes, wie den einer grundlegenden Architektur wurden ebenso betrachtet wie ein konkretes Datenmodell in Form eines Relationenschemas. Von großem Interesse ist dabei auch die Vorgehensweise, wie die entworfene Sprache zur Bestimmung von Aufgabenträgern auf die Anfragesprache des zugrundeliegenden Datenhaltungssystems abgebildet werden kann. Im Fall einer relationalen Datenbank handelt es sich dabei um eine Übersetzung nach SQL. Dieses Thema wurde ausführlich behandelt. Allerdings ist die tatsächliche Umsetzung in Code weit weniger elegant als es der Entwurf vermuten läßt. Denn obwohl die Übersetzung standardkonformes SQL darstellt, wird es in dieser Form von keiner der geforderten Datenbanken akzeptiert werden: entweder, weil deren Syntax vom Standard abweicht, oder weil bestimmte Eigenschaften des Standards – wie Rekursion – nicht implementiert sind.

Literaturverzeichnis

Bei Quellenverweisen im Internet ist zusätzlich das Datum angegeben, an dem der Link zuletzt geprüft wurde.

[ADEPT] ADEPT – Next Generation Workflow Technology
http://www.informatik.uni-ulm.de/dbis/01/staff/reichert/
projects/project_adept.htm (Stand: 06.06.2005)

[Alv97] Inoffizielle OID Registry von Harald Tveit Alvestrand
http://www.alvestrand.no/objectid/ (Stand 26.01.2004)

[Berr00] Berroth, M.: Logische und physikalische Struktur des Second Extended Filesystems für Linux, Seminararbeit Proseminar Linux, Universität Ulm, Abteilung Verteilte Systeme, Juli 2000

[Buss96] Bussler Chr.: Analysis of the Organisation Modeling Capability of Workflow-Management-Systems. In: Proceedings of the PRIISM '96 Conference, Maui, Hawaii, USA, Januar 1996

[Buss98] Bussler Chr.: Organisationsverwaltung in Workflow-Management-Systemen, Dissertation an der Universität Erlangen-Nürnberg, Institut für Informatik, 1998.

[CloMe03] Clocksin W., Mellish C.: Programming in Prolog, Springer Verlag Berlin, 2003

[COAH94] COSA Administrator Handbuch. Version 1.4, Software-Ley GmbH, Mai 1994

[Cook80] Cook, C.: Streamlining office procedures – An analysis using the information control net model. In: AFIPS Conference Proceedings 1980, National Computer Conference, Anaheim, California, Mai 1980

[COPH94] COSA Programmierer Handbuch. Version 1.4, Software-Ley GmbH, Mai 1994

[CORH94] COSA Referenz Handbuch. Version 1.4, Software-Ley GmbH, Mai 1994

[DaDa96] Date C. J., Darwen H.: A Guide to SQL Standard (4. Auflage), Addison-Wesley Professional 1996

[DB2] IBM - DB2 Universal Database V8.2
http://www-306.ibm.com/software/data/db2/ (Stand 25.06.2005)

[Derby] The Apache Derby Project
http://incubator.apache.org/derby/ (Stand 25.06.2005)

[Dud93] Duden Informatik, Dudenverlag Mannheim, Leipzig, Wien, Zürich, 1993

[ECAG92] E.C.H.O. 2.1. Administrator`s Guide. Digital Equipment, November 1992

[ElBe82] Ellis, C.; Bernal, M: OfficeTalk-D: An experimental Office Information System. In: Proceedings First SIGOA Conference on Office Information Systems, Juni 1982

[Elli83] Ellis, C.: Formal and Informal Models of Office Activity. In: R.E.A. Mason (Hrsg.): Information Processing. Elsevier Science Publishers B.V. (North-Holland), 1983

[ElNu80] Ellis, C,; Nutt, G.: Office Information Systems and Computer Science. In: Computing Surveys, Vol. 12, No. 1, März 1980

[ElWa93] Ellis, C.; Wainer, J.: Goal Based Models of Groupware. Technischer Report, University of Colrado at Boulder, Department of Computer Science, CU-CS-668-93, August 1993

[GrKe98] Grover, V.; Kettinger, W.: Business Process Change: Reengineering Concepts, Methods and Technologies, Idea Group Publishing, 1998

[GWB+05] Göser, K.; Weitmann, T.; Berroth, M.; Tscheho, J.; Stieger, A.: Dokumentation - Implementierung eines Prozeßeditors für ADEPT 2, Universität Ulm, Abteilung DBIS, 2005

[HaCh96] Hammer, M.; Champy, J.:Business Reengineering. Die Radikalkur für das Unternehmen, Campus Fachbuch, 1996

[HäRa01] Härder, T.; Rahm E.: Datenbanksysteme, Springerverlag Berlin, 2001

[IBPG95] IBM FlowMark , Programming Guide. Version 2.1. International Buisness Machines, 1995

[IBMW95] IBM FlowMark. Modelling Workflow. Version 1.1 International Buisness Machines, 1995

[ITU01_1] ITU - International Telecommunication Union: Information technology - Open Systems Interconnection - The Directory: Overview of concepts, models and services
http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-X.500 (Stand: 22.03.2005)

[ITU01_2] ITU - International Telecommunication Union: Information technology – Open Systems Interconnection – The Directory: Selected object classes
http://www.itu.int/rec/recommendation.asp?type=folders&lang=e&parent=T-REC-X.521 (Stand: 22.03.2005)

[Jab95] Jablonski, S. et al.: Workflow-Management-Systeme, Modellierung und Architektur, Thomson Publishing Bonn, 1995.

[Java5] Java 5 - Core Java J2SE 5.0
http://java.sun.com/j2se/1.5.0/index.jsp (Stand 24.06.2005)

[KHK+91] Kreifelts, T.; Hinrichs, E.; Klein, K.-H.; Seuffert, P.;Woetzel G.: Experiences with the DOMINO Office Procedure System. In: Proceedings of the Second European Conference on Computer-Supported Cooperative Work (WCSCW '91), Amsterdam, The Netherlands, September 1991

[KlüLa03] Klünter D., Laser J.: LDAP verstehen, OpenLDAP einsetzen, dpunkt.verlag Heidelberg 2003.

[Kony96] Konyen, I.: Organisations- und Ablaufstrukturen im Krankenhaus – Anforderungen, Werkzeuge und deren Anwendung, Diplomarbeit an der Universität Ulm, Fakultät für Informatik, 1996

[Krei91] Kreifelts, T.: Coordination of Distributed Work: From Office Procedures to Customizable Activities. In: In: W. Brauer, D. Hernandez (Hrsg.) Verteilte Künstliche Intelligenz und kooperatives Arbeiten. 4. Internatialer GI-Kongreß wissenbasierte Systeme, München, Oktober 1991 (Informatik Fachbericht 291, Springer-Verlag)

[KrHW93) Kreifelts, T.;Hinrichs,E.;Woetzel, G.:Sharing To-Do Lists with Distributed Task Manager. In: Proceedings Third European Conference on Computer-Supported Cooperative Work (ECSCW`93), Milano, Italy, September 1993

[KrSW87] Kreifelts, T.;Seuffert, P.; Woetzel, G.: Ausnahmebehandlung, Verteilung und Adressierung im Vorgangssystem DOMINO. In: M. Paul (Hrsg.) GI – 17. Jahrestagung Computerintegrierter Arbeisplatz im Büro. Proceedings, Springer-Verlag, Oktober 1987

[Kubi98] Kubicek, M.: Organisatorische Aspekte in flexiblen Workflow-Management-Systemen, Diplomarbeit an der Universität Ulm, Fakultät für Informatik, 1998.

[LeAl94] Leymann, F.; Altenhuber, W.: Managing buisness processes as an information resource. In. IBM Systems Journal, Vol. 33, No. 2, 1994

[MeSi01] Melton, J.; Simon, R.: SQL: 1999 - Understanding Relational Language Components (1.Auflage), Morgan Kaufmann 2001

[MLinx] www.mitlinx.de - LDAP verstehen
http://www.mitlinx.de/ldap/index.html?http://www.mitlinx.de/ldap/main.htm
(Stand: 10.06.2005) oder
http://gewilab.uni-graz.at/doc/allgemeines/LDAP/mitlinx_LDAP_GUIDE.pdf
(Stand: 10.06.2005)

[MQWF04] IBM WebSphere MQ Workflow - Concepts and Architecture Version 3.5 Seventh Edition, Januar 2004 (Stand 07.06.2005)
http://www-306.ibm.com/software/integration/wmqwf/
library/manuals/wmqwf34.html

[MS03] Microsoft® Active Directory® LDAP Compliance, Microsoft ® Windows Server ™ 2003 White Paper. Quelle (Stand 21.12.2004): *http://www.microsoft.com/windowsserver2003/techinfo/overview/ldapcomp.mspx*

[MySQL] MySQL Datenbank Dokumentation, Abschnitt: Tabellentyp '*Memory*' *http://dev.mysql.com/doc/mysql/en/memory-storage-engine.html* (Stand: 09.06.2005)

[Oracle] Oracle 10g *http://www.oracle.com/database/index.html* (Stand 25.06.2005)

[ORGR94] ORM Organisations- und Ressourcenmanagement, Grundlagenhandbuch. Version 1.2, Siemens-Nixdorf-Informationssysteme, August 1994

[ORRE94] ORM Organisations- und Ressourcenmanagement, API-Referenzbuch. Version 1.2, Siemens-Nixdorf-Informationssysteme, August 1994

[OrWG96] Oracle Workflow Guide. Release 10SC, Beta Documentation, Oracle Corporation, April 1996

[PrepStmt] The Java Tutorial - JDBC Basics - Using Prepared Statements *http://java.sun.com/docs/books/tutorial/jdbc/basics/prepared.html* (Stand: 12.06.2005)

[Reich04] Reichert M.; Catrinescu-Wiedemuth, U.; Rinderle S.: Evolution von Zugriffsregelungen in Informationssystemen, Proc. Conf. Elektronische Geschäftsprozesse (EGP'04), September, Klagenfurt (2004)

[Riv92] Rivest, R. et.al.: RFC 1321 - The MD5 Message-Digest Algorithm; MIT Laboratory for Computer Science and RSA Data Security, Inc., April 1992 *http://www.faqs.org/rfcs/rfc1321.html* (Stand: 12.06.2005)

[Rupi92] Rupietta, W.: Organisationsmoedllierung zur Unterstützung kooperativer Vorgangsbearbeitung. In: Wirtschaftsinformatik, 34. Jahrgang, Heft 1, Februar 1992

[Rupi94] Rupietta, W.: OriganizationModels for Cooperative Office Applications. In:D. Karagiannis (Hrsg.) Database and Expert Systems Applications. Proseedings 5th International Conference, DEXA `94, Athens, Greece, September 1994 (Springer-Verlag)

[Schö00] Schöning, U.: Boolesche Funktionen, Logik, Grammatiken und Automaten, Vorlesungsskript 2000, Universität Ulm

[SIM+94a] Swenson, K.; Irwin, K.; Matsumoto, T.; Maxwell, R.; Saghari, B.: Collaborative Planing: Empowering the User in a Process Support Environment. In: Proceedings of the Computer Supported Cooperative Work Conference (CSCW), 1994

[SIM+94a] Swenson, K.; Irwin, K.; Matsumoto, T.; Maxwell, R.; Saghari, B.: Collaborative Planing: A Process to Develop Software Processes. Submission on the 9th International Software Process Workshop, Arlie, Virginia, USA, Oktober 1994

[SMM+94] Swenson, K.; Maxwell, R.; Matsumoto, T.; Saghari, B.; Irwin, K.: A Business Process Environment Supporting Collaborative Planing. In: The Journal of Collaborative Computing, Vol. 1, No. 1, 1994

[Swen93a] Swenson, K.: A Visual Language to Describe Collaborative Work. In: Proceedings of the International Workshop for Visual Languages, Bergen, Norwegen, 1993

[Swen93b] Swenson, K.: Visual Support for Reengineering Work Processes. Proceedings of the Conference on Organisational Computing Systems, November 1993

[Swen93c] Swenson, K.: Regatta Project. A Tool for Business Process Reengineering. Submitted to Applica '93 Conference.

[Spar01] Sparr, S.:Ausführungs-, Zugriffs- und Anderungsrechte in adaptiven Prozess-Management-Systemen, Diplomarbeit an der Universität Ulm, Fakultät für Informatik, 2001

[Staff99] Staffware for X.500 Administrators's Supplement (for Windows NT), Systemdokumentation Staffware 2000, März 1999

[SUSE] SUSE LINUX – Administrationshandbuch - Kapitel 14. Grundlagen der Vernetzung / 14.7. LDAP – Ein Verzeichnisdienst *http://sman.informatik.htw-dresden.de/doc/manual/suselinux-adminguide_de/html/ch14s07.html* (Stand: 11.06.2005)

[VanH93] Van Hattem, M.: Case Type Definition Language Facility. Digital Equipment, Oktober 1993

[Vuur92] Vuurboom, R.: Functional Requirement – Overview. Digital Equipment, September 1992

[Wang00] Wang X., Schulzrinne H., Kandlur D., Verma D.: Measurement and Analysis of LDAP Performance, In International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'2000), Santa Clara, CA, pp. 156--165, Juni 2000. *http://www.cse.buffalo.edu/~xwang8/public/paper/ldap_sigmetrics.pdf* (Stand: 07.06.2005)

[Wied02] Wiedemuth-Catrinescu U.: Evolution von Organisationsmodellen in Workflow-Management-Systemen, Diplomarbeit an der Universität Ulm, Fakultät für Informatik, 2002.

[WoKr93] Woetzel, G,; Kreifelts, T.: The Use of Petri Nets for Modeling Workflow in the DOMINO System. In: Workshop on CSCW, Petri Nets and Related Formalismus, International Conference on Application and Theory of Petri Nets, Chicago, Il, Juni 1993

[WPE00] Wielsch, M.; Prahm, J.; Eßer, H.: Linux intern, Data Becker, August 2000

[Zern95] Zerndt, A.: Untersuchung ausgewählter Vorgangssteuerungssysteme mit dem Ziel der Erstellung eines umfassenden Datenbankschemas. Diplomarbeit, Technische Universität Dresden, Institut für Betriebssysteme, Datenbanken und Rechnernetze, 1995

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang

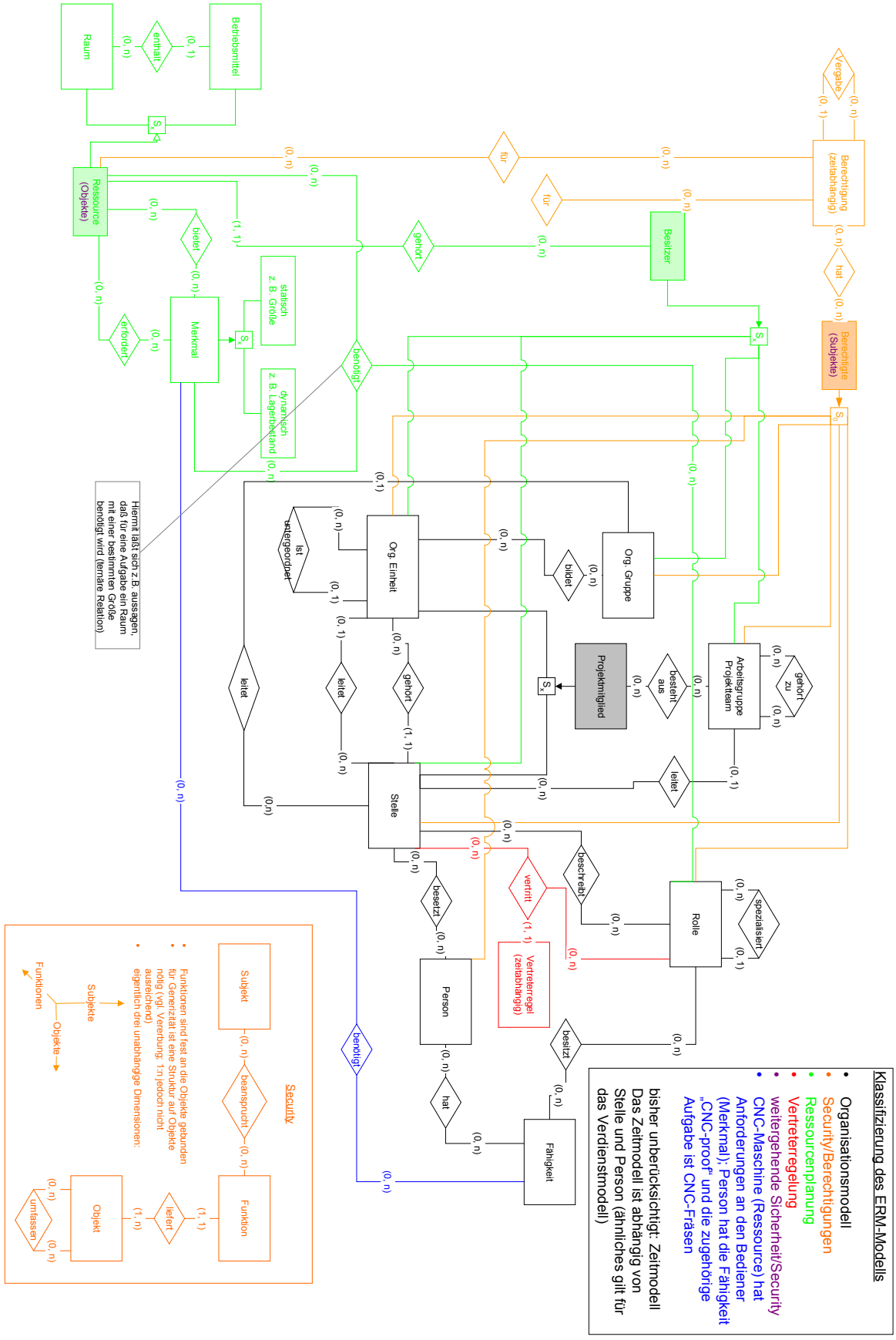
A Glossar

ANSI	American National Standards Institute (*www.ansi.org*)
API	Application Programming Interface – Schnittstelle für Anwendungsprogramme
DB	Datenbank – Database
DIT	Directory Information Tree – Verzeichnisbaum in LDAP
DHS	Datenhaltungssystem – Bspw. Dateisystem, Datenbank, Verzeichnisdienst
DN	Distinguished Name – Qualifizierender Name für Objekte in einem Verzeichnis
DNS	Domain Name System – Hierarchisches Namenssystm im Internet
DSA	Directory Service Agent – LDA-Server
DUA	Directory User Agent – LDAP-Client
EER-Diagramm	Extended Entity-Relationship Diagramm – Formale graphische Beschreibungsmöglichkeit von Informationssystemen (Datenbanken)
FDL	FlowMark Definition Language – Datenaustauschformat von IBM Websphere Workflow (Buildtime ↔ Runtime)
GUI	Graphical User Interface (Graphische Benutzerschnittstelle)
ITU	Internation Telecommunication Union (*www.itu.int*)
JDBC	Java Database Connectivity – Java-Bibliothek zur Kommunikation mit relationalen Datenbanken
LDAP	Lightweight Directory Access Protocol – Kommunkationsprotokoll für Verzeichnisdienste
OID	ObjectID – Eindeutige Identifikation für X.500-Objektklassen
OMM	Organisationsmodell Manager (Org.Model Manager) – Thema dieser Arbeit
OO	Objektorientierung
PMS	Prozeß Management System – neuere Bezeichnung für WfMS

RDBMS	relationales Datenbank Management System (relationale Datenbank)
RFC	Request for Comment – zur Standardisierung vorgeschlagene Verfahrensweisen, die auch nach der Standardisierung diese Bezeichnung weiterführen (auch: Quasi-Standard, „Best Practices“)
SDK	Software Development Kit – vergleichbar mit API
SQL	Structured Query Language – Anfragesprache für relationale Datenbanksysteme (RDBMS)
WfMS	Workflow-Management-System – Prozessunterstützendes System

B Ursprüngliches Organisationsmetamodell

mit Eingliederung der noch nicht existierenden Komponenten Ressourcenmanagement und Securitymanagement



C Schemadefinitionen für X.500-Objektklassen

Für die Objektklassen *Person*, *residentialPerson*, *organisationalPerson* und *InetOrgPerson*:

objectclass (*2.5.6.6*
NAME 'person'
SUP top
STRUCTURAL
MUST (sn $ cn)
MAY (userPassword $ telephoneNumber $ seeAlso $ description)
)

objectclass (*2.5.6.10*
NAME 'residentialPerson'
SUP person
STRUCTURAL
MUST (l)
MAY (businessCategory $ x121Address $ registeredAddress $ destinationIndicator $ preferredDeliveryMethod $ telexNumber $ teletexTerminalIdentifier $ telephoneNumber $ internationaliSDNNumber $ facsimileTelephoneNumber $ street $ postOfficeBox $ postalCode $ postalAddress $ physicalDeliveryOfficeName $ st $ l)
)

objectclass (*2.5.6.7*
NAME 'organizationalPerson'
SUP person
STRUCTURAL
MAY (title $ x121Address $ registeredAddress $ destinationIndicator $ preferredDeliveryMethod $ telexNumber $ teletexTerminalIdentifier $ telephoneNumber $ internationaliSDNNumber $ facsimileTelephoneNumber $ street $ postOfficeBox $ postalCode $ postalAddress $ physicalDeliveryOfficeName $ ou $ st $ l)
)

objectclass (*2.16.840.1.113730.3.2.2*
NAME 'inetOrgPerson'
DESC 'RFC2798: Internet Organizational Person'
SUP organizationalPerson
STRUCTURAL
MAY (audio $ businessCategory $ carLicense $ departmentNumber $ displayName $ employeeNumber $ employeeType $ givenName $ homePhone $ homePostalAddress $ initials $ jpegPhoto $ labeledURI $ mail $ manager $ mobile $ o $ pager $ photo $ roomNumber $ secretary $ uid $ userCertificate $ x500uniqueIdentifier $ preferredLanguage $ userSMIMECertificate $ userPKCS12)
)

D Schemadefinitionen für X.500-Attributtypen

Für *telephoneNumber* und *mail*:

attributetype (*2.5.4.20*
NAME 'telephoneNumber'
EQUALITY telephoneNumberMatch
SUBSTR telephoneNumberSubstringsMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.50{32}
)

attributetype (*0.9.2342.19200300.100.1.3*
NAME ('mail' 'rfc822Mailbox')
DESC 'RFC1274: RFC822 Mailbox'
EQUALITY caseIgnoreIA5Match
SUBSTR caseIgnoreIA5SubstringsMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.26{256}
)

E LDAP-SDKs und -APIs

Diese Liste erhebt keinen Anspruch auf Vollständigkeit. Sie soll nur einen Überblick über die vielfältig verfügbaren APIs für Zugriffe auf Verzeichnisdienste aufzeigen.

Die Informationen fanden sich durch eine Suche in der FAQ von http://www.openldap.org/faq/
.
Für die Erreichbarkeit der angegebenen Quelle kann keine Garantie gegeben werden. Zum Stand vom 26.01.2005 waren alle angegebenen Quellen verfügbar.

- OpenLDAP SDK für C (zusammen mit OpenLDAP)
- OpenLDAP Interface für C++ (zusammen mit OpenLDAP)
- NeoSoft Tcl API (zusammen mit OpenLDAP)
- Mozilla Directory SDKs: http://www.mozilla.org/directory/
- Net::LDAP Perl Modul
- PHP: LDAP-Unterstützung durch Angabe des entsprechenden Parameters bei der Kompilation von PHP (siehe http://www.php.net/manual/en/ref.ldap.php)
- Python: siehe http://python-ldap.sourceforge.net
- Java Naming & Directory Interface (JNDI):
  (siehe: http://java.sun.com/products/jndi)
- Microsoft ADSI SDK für Microsoft Windows (Active Directory Service Interface): SDK für VisualBasic und C/C++ zum Zugriff auf das Active Directory (siehe www.microsoft.com/adsi/)

F Use Cases für Bearbeiterzuordnungen und deren Übersetzung in SQL

Unabhängige Bearbeiterzuordnungen

Use Case 1: Der Bearbeiter besetzt die Stelle 'Hausmeister1'

(OrgPosition.Name = 'Hausmeister1').getAgent()

1. Selektion: *(OrgPosition.Name = 'Hausmeister1')*
   $Count = 0

```
SELECT    T0.ID
FROM      OrgPosition AS T0
WHERE     T0.Name = 'Hausmeister1'
```

2. Funktion: *OrgPosition.getAgent()*
   $Count = $Count++ = 1

```
SELECT    T1.OrgPositionID, T2.ID
FROM      OrgPosition AS T0
          JOIN OrgPositionOccupation AS T1
          ON T0.ID = T1.OrgPositionID
          JOIN Agent AS T2 ON T1.AgentID = T2.ID
WHERE     T0.Name = 'Hausmeister1'
```

Use Case 2: Der Bearbeiter besetzt die Stelle Hausmeister1 oder ist irgendein Mitarbeiter, der eine Stelle in der Abteilung 'Instandhaltung' besetzt.

(OrgPosition.Name = ‘ Hausmeister1’).getAgent
OR (OrgUnit.Name = ‘Instandhaltung’).getOrgPosition.getAgent()

1. Selektion: *(OrgPosition.Name = ‘ Hausmeister1’)*
   $Count = 0

```
SELECT    T0.ID
FROM      OrgPosition AS T0
WHERE     T0.Name = 'Hausmeister1'
```

2. Funktion: *OrgPosition.getAgent()*
   $Count= $Count++ = 1

```
SELECT    T1.OrgPositionID, T2.ID
FROM      OrgPosition AS T0
          JOIN OrgPositionOccupation AS T1
          ON T0.ID = T1.OrgPositionID
          JOIN Agent AS T2 ON T1.AgentID = T2.ID
 WHERE     T0.Name = 'Hausmeister1'
```

   $Count = $Count++ = 3

3. Operator: *OR*

```
SELECT   T1.OrgPositionID, T2.ID
FROM     OrgPosition AS T0
         JOIN OrgPositionOccupation AS T1
         ON T0.ID = T1.OrgPositionID
         JOIN Agent AS T2 ON T1.AgentID = T2.ID
WHERE    T0.Name =  ' Hausmeister1'
UNION
```

4. Selektion: *(OrgUnit.Name = 'Instandhaltung')*
   $Count= $Count++ = 4

```
SELECT   T1.OrgPositionID, T2.ID
FROM     OrgPosition AS T0
         JOIN OrgPositionOccupation AS T1
         ON T0.ID = T1.OrgPositionID
         JOIN Agent AS T2 ON T1.AgentID = T2.ID
WHERE    T0.Name = 'Hausmeister1'
UNION
SELECT   T4.ID
FROM     OrgUnit AS T4
WHERE    T4.Name = 'Instandhaltung'
```

5. Funktion: *OrgUnit.getOrgPosition()*
   $Count = $Count++ = 5

```
SELECT   T1.OrgPositionID, T2.ID
FROM     OrgPosition AS T0
         JOIN OrgPositionOccupation AS T1
         ON T0.ID = T1.OrgPositionID
         JOIN Agent AS T2 ON T1.AgentID = T2.ID
WHERE    T0.Name = 'Hausmeister1'
UNION
SELECT   T5.ID
FROM     OrgUnit AS T4
         JOIN OrgPosition AS T5 ON T4.ID = T5.OrgUnitID
WHERE    T4.Name = 'Instandhaltung'
```

6. Funktion: *OrgPosition.getAgent()*
   $Count = $Count++ = 6

```
SELECT   T1.OrgPositionID, T2.ID
FROM     OrgPosition AS T0
         JOIN OrgPositionOccupation AS T1
         ON T0.ID = T1.OrgPositionID
         JOIN Agent AS T2 ON T1.AgentID = T2.ID
WHERE    T0.Name = 'Hausmeister1'
UNION
SELECT   T6.OrgPositionID, T7.ID
FROM     OrgUnit AS T4
         JOIN OrgPosition AS T5
         ON T4.ID = T5.OrgUnitID
         JOIN OrgPositionOccupation AS T6
         ON T5.ID = T6.OrgPositionID
         JOIN Agent AS T7 ON T6.AgentID = T7.ID
WHERE    T4.Name = 'Instandhaltung'
```

Use Case 3: Der Bearbeiter ist irgendein Mitarbeiter, der eine Stelle in der Abteilung 'Instandhaltung' besetzt, und gleichzeitig die Fähigkeit 'Rasenmähen' besitzt, oder die Person, die die Stelle 'Hausmeister1' besetzt

(OrgUnit.Name = 'Instandhaltung').getOrgPosition.getAgent()
AND (Ability.Name = 'Rasenmähen').getAgent()
OR (OrgPosition.Name = 'Hausmeister1'). GetAgent()

1. Selektion: *(OrgUnit.Name = 'Instandhaltung')*
   $Count = 0

```
SELECT    T0.ID
FROM      OrgUnit AS T0
WHERE     T0.Name = 'Instandhaltung'
```

2. Funktion: *OrgUnit.getOrgPosition()*
   $Count = $Count++ = 1

```
SELECT    T1.ID
FROM      OrgUnit AS T0
          JOIN OrgPosition AS T1 ON T0.ID = T1.OrgUnitID
WHERE     T0.Name = 'Instandhaltung'
```

3. Funktion: *OrgPosition.getAgent()*
   $Count= $Count++ = 2

```
SELECT    T2.OrgPositionID, T3.ID
FROM      OrgUnit AS T0
          JOIN OrgPosition AS T1 ON T0.ID = T1.OrgUnitID
          JOIN OrgPositionOccupation AS T2
          ON T1.ID = T2.OrgPositionID
          JOIN Agent AS T3 ON T2.AgentID = T3.ID
WHERE     T0.Name = 'Instandhaltung'
```

   $Count = $Count++ = 3

4. Operator: *AND*

```
SELECT    T2.OrgPositionID, T3.ID
FROM      OrgUnit AS T0
          JOIN OrgPosition AS T1 ON T0.ID = T1.OrgUnitID
          JOIN OrgPositionOccupation AS T2
          ON T1.ID = T2.OrgPositionID
          JOIN Agent AS T3 ON T2.AgentID = T3.ID
WHERE     T0.Name = 'Instandhaltung'
 INTERSECT
```

5. Selektion: *(Ability.Name = 'Rasenmähen')*
   $Count = $Count++ = 4

```
SELECT    T2.OrgPositionID, T3.ID
FROM      OrgUnit AS T0
          JOIN OrgPosition AS T1 ON T0.ID = T1.OrgUnitID
          JOIN OrgPositionOccupation AS T2
          ON T1.ID = T2.OrgPositionID
          JOIN Agent AS T3 ON T2.AgentID = T3.ID
WHERE     T0.Name = 'Instandhaltung'
INTERSECT
SELECT    T4.ID
FROM      Ability AS T4
WHERE     T4.Name = 'Rasenmähen'
```

6. Funktion: *Ability.getAgent()*
   $Count = $Count++= 5

```
SELECT    T2.OrgPositionID, T3.ID
FROM      OrgUnit AS T0
          JOIN OrgPosition AS T1 ON T0.ID = T1.OrgUnitID
          JOIN OrgPositionOccupation AS T2
          ON T1.ID = T2.OrgPositionID
          JOIN Agent AS T3 ON T2.AgentID = T3.ID
WHERE     T0.Name = 'Instandhaltung'
INTERSECT
SELECT    T6.ID
FROM      Ability AS T4
          JOIN AgentAbility AS T5 ON T4.ID = T5.Ability
          JOIN Agent AS T6 ON T5.AgentID = T6.ID
WHERE     T4.Name = 'Rasenmähen'
```

7. *Sonderfall mit INTERSECT (siehe Kapitel 7.7.4)*
   $Count = $Count++= 6

```
SELECT    T2.OrgPositionID, T3.ID
FROM      OrgUnit AS T0
          JOIN OrgPosition AS T1 ON T0.ID = T1.OrgUnitID
          JOIN OrgPositionOccupation AS T2
          ON T1.ID = T2.OrgPositionID
          JOIN Agent AS T3 ON T2.AgentID = T3.ID
WHERE     T0.Name = 'Instandhaltung'
INTERSECT
SELECT    T7.OrgPositionID, T6.ID
FROM      Ability AS T4
          JOIN AgentAbility AS T5 ON T4.ID = T5.Ability
          JOIN Agent AS T6 ON T5.AgentID = T6.ID
          JOIN OrgGroupOccupation AS T7
          ON T6.ID = T7.AgentID
WHERE     T4.Name = 'Rasenmähen'
```

8. Operator: *OR*

```
SELECT   T2.OrgPositionID, T3.ID
FROM     OrgUnit AS T0
         JOIN OrgPosition AS T1 ON T0.ID = T1.OrgUnitID
         JOIN OrgPositionOccupation AS T2
         ON T1.ID = T2.OrgPositionID
         JOIN Agent AS T3 ON T2.AgentID = T3.ID
WHERE    T0.Name = 'Instandhaltung'
INTERSECT
SELECT   T7.OrgPositionID, T6.ID
FROM     Ability AS T4
         JOIN AgentAbility AS T5 ON T4.ID = T5.Ability
         JOIN Agent AS T6 ON T5.AgentID = T6.ID
         JOIN OrgGroupOccupation AS T7
         ON T6.ID = T7.AgentID
WHERE    T4.Name = 'Rasenmähen'
UNION
```

9. Selektion: *(OrgPosition.Name ='Hausmeister1')*
$Count = $Count++ = 8

```
SELECT   T2.OrgPositionID, T3.ID
FROM     OrgUnit AS T0
         JOIN OrgPosition AS T1 ON T0.ID = T1.OrgUnitID
         JOIN OrgPositionOccupation AS T2
         ON T1.ID = T2.OrgPositionID
         JOIN Agent AS T3 ON T2.AgentID = T3.ID
WHERE    T0.Name = 'Instandhaltung'
INTERSECT
SELECT   T7.OrgPositionID, T6.ID
FROM     Ability AS T4
         JOIN AgentAbility AS T5 ON T4.ID = T5.Ability
         JOIN Agent AS T6 ON T5.AgentID = T6.ID
         JOIN OrgGroupOccupation AS T7
         ON T6.ID = T7.AgentID
WHERE    T4.Name = 'Rasenmähen'
UNION
SELECT   T8.ID
FROM     OrgPosition AS T8
WHERE    T8.Name = 'Hausmeister1'
```

10. Funktion: *OrgPosition.getAgent()*

$Count = $Count++ = 9

```
SELECT    T2.OrgPositionID, T3.ID
FROM      OrgUnit AS T0
          JOIN OrgPosition AS T1 ON T0.ID = T1.OrgUnitID
          JOIN OrgPositionOccupation AS T2
          ON T1.ID = T2.OrgPositionID
          JOIN Agent AS T3 ON T2.AgentID = T3.ID
WHERE     T0.Name = 'Instandhaltung'
INTERSECT
SELECT    T7.OrgPositionID, T6.ID
FROM      Ability AS T4
          JOIN AgentAbility AS T5 ON T4.ID = T5.Ability
          JOIN Agent AS T6 ON T5.AgentID = T6.ID
          JOIN OrgGroupOccupation AS T7
          ON T6.ID = T7.AgentID
WHERE     T4.Name = 'Rasenmähen'

UNION
SELECT    T9.OrgPositionID, T10.ID
FROM      OrgPosition AS T8
          JOIN OrgPositionOccupation AS T9
          ON T8.ID = T9.OrgPositionID
          JOIN Agent AS T10 ON T9.AgentID = T10.ID
WHERE     T8.Name = 'Hausmeister1'
```

Use Case 4: Die Mitglieder der Abteilungen mit Namen 'Entwicklung', die an Projekten in Europa beteiligt sind

(OrgUnit.Name = 'Entwicklung').getProjectGroup(ProjectGroup.Ort = 'Europa').getOrgUnit().getOrgPosition().getAgent()

1. Selektion: *(OrgUnit.Name = 'Entwicklung')*

$Count = 0

```
SELECT    T0.ID
FROM      OrgUnit T0
WHERE     T0.Name = 'Entwicklung'
```

2. Funktion: *OrgUnit.getProjectGroup()*

$Count = $Count++ = 1

```
SELECT    T2.ID
FROM      OrgUnit AS T0
          JOIN ProjectInvolvement AS T1
          ON T0.ID= T1.InvolvedID
          JOIN ProjectGroup AS T2 ON T1.ProjectID = T2.ID
WHERE     T0.Name = 'Entwicklung'
          AND T1.Discrimination = ORGUNIT
```

$Count = $Count++ = 2

3. Selektion: *(ProjectGroup.Ort = 'Europa')*

```
SELECT    T2.ID
FROM      OrgUnit AS T0
          JOIN ProjectInvolvement AS T1
          ON T0.ID= T1.InvolvedID
          JOIN ProjectGroup AS T2 ON T1.ProjectID = T2.ID
WHERE     T0.Name = 'Entwicklung'
          AND T1.Discrimination = ORGUNIT
          AND T2.Ort = 'Europa'
```

4. Funktion: *ProjectGroup.getOrgUnit()*

$Count = $Count++ = 3

```
SELECT    T4.ID
FROM      OrgUnit AS T0
          JOIN ProjectInvolvement AS T1
          ON T0.ID= T1.InvolvedID
          JOIN ProjectGroup AS T2 ON T1.ProjectID = T2.ID
          JOIN ProjectInvolvement AS T3
          ON T2.ID = T3.ProjectID
          JOIN OrgUnit AS T4 ON T3.InvolvedID = T4.ID
WHERE     T0.Name = 'Entwicklung'
          AND T1.Discrimination = ORGUNIT
          AND T2.Ort = 'Europa'
          AND T3.Discrimination = ORGUNIT
```

$Count = $Count++ = 4

5. Funktion: *OrgUnit.getOrgPosition()*

$Count = $Count++ = 5

```
SELECT    T5.ID
FROM      OrgUnit AS T0
          JOIN ProjectInvolvement AS T1
          ON T0.ID= T1.InvolvedID
          JOIN ProjectGroup AS T2 ON T1.ProjectID = T2.ID
          JOIN ProjectInvolvement AS T3
          ON T2.ID = T3.ProjectID
          JOIN OrgUnit AS T4 ON T3.InvolvedID = T4.ID
          JOIN OrgPosition AS T5 ON T4.ID = T5.OrgUnitID
WHERE     T0.Name = 'Entwicklung'
          AND T1.Discrimination = ORGUNIT
          AND T2.Ort = 'Europa'
          AND T3.Discrimination = ORGUNIT
```

6. Funktion: OrgPosition.getAgent()
   $Count = $Count++ = 6

```
SELECT    T6.OrgPositionID, T7.ID
FROM      OrgUnit AS T0
          JOIN ProjectInvolvement AS T1
          ON T0.ID= T1.InvolvedID
          JOIN ProjectGroup AS T2
          ON T1.ProjectID = T2.ID
          JOIN ProjectInvolvement AS T3
          ON T2.ID = T3.ProjectID
          JOIN OrgUnit AS T4 ON T3.InvolvedID = T4.ID
          JOIN OrgPosition AS T5 ON T4.ID = T5.OrgUnitID
          JOIN OrgPositionOccupation AS T6
          ON T5.ID = T6.OrgPositionID
          JOIN Agent AS T7 ON T6.AgentID = T7.ID
WHERE     T0.Name = 'Entwicklung'
          AND T1.Discrimination = ORGUNIT
          AND T2.Ort = 'Europa'
          AND T3.Discrimination = ORGUNIT
```

   $Count = $Count++ = 7

Use Case 5: Die Leiter der Unterabteilungen in Ulm, die der Abteilungen 'Entwicklug' untergeordnet sind

(OrgUnit.Name = 'Entwicklung').getSubOrgUnit(OrgUnit.Standort = 'Ulm'). getManager().getAgent()

1. Selektion: *(OrgUnit.Name = 'Entwicklung')*
   $Count = 0

```
SELECT    T0.ID
FROM      OrgUnit AS T0
WHERE     T0.Name = 'Entwicklung'
```

2. Funktion: *OrgUnit.getSubOrgUnit()*
   $Count = $Count++ = 1

```
SELECT    T1.ID
FROM      OrgUnit AS T0
          JOIN OrgUnit AS T1 ON T0.ID = T1.SupID
WHERE     T0.Name = 'Entwicklung'
```

3. Selektion: *(OrgUnit.Standort = 'Ulm')*

```
SELECT    T1.ID
FROM      OrgUnit AS T0
          JOIN OrgUnit AS T1 ON T0.ID = T1.SupID
WHERE     T0.Name = 'Entwicklung' AND T1.Standort = 'Ulm'
```

4. Funktion: *OrgUnit.getManager()*
   $Count = $Count++ = 2
```
SELECT    T2.ID
FROM      OrgUnit AS T0
          JOIN OrgUnit AS T1 ON T0.ID = T1.SupID
          JOIN OrgPosition AS T2 ON T1.ManagerID = T2.ID
WHERE     T0.Name = 'Entwicklung' AND T1.Standort = 'Ulm'
```

5. Funktion: *OrgPosition.getAgent()*
   $Count = $Count++ = 3
```
SELECT    T3.OrgPositionID, T4.ID
FROM      OrgUnit AS T0
          JOIN OrgUnit AS T1 ON T0.ID = T1.SupID
          JOIN OrgPosition AS T2 ON T1.ManagerID = T2.ID
          JOIN OrgPositionOccupation AS T3
          ON T2.ID = T3.OrgPositionID
          JOIN Agent AS T4 ON T3.AgentID = T4.ID
WHERE     T0.Name = 'Entwicklung' AND T1.Standort = 'Ulm'
```
   $Count = $Count++ = 4

Use Case 6: Die Mitarbeiter, die eine Fähigkeit besitzen, die auch die Rolle Hausmeister beschreibt

(Role.Name = 'Hausmeister').getAbility.getAgent()

1. Selektion: *(Role.Name = 'Hausmeister')*
   $Count = 0
```
SELECT    T0
FROM      Role AS T0
WHERE     T0.Name = 'Hausmeister'
```

2. Funktion: Role.getAbility()
   $Count = $Count++ =1
```
SELECT    T2.ID
FROM      Role AS T0
          JOIN RoleDescription AS T1 ON T0 = T1.RoleID
            JOIN Ability AS T2 ON T1.AbilityID = T2.ID
WHERE     T0.Name = 'Hausmeister'
```
   $Count = $Count++ =2

3. Funktion: Ability.getAgent()
   $Count = $Count++ = 3
```
SELECT    NULL, T4.ID
FORM      Role AS T0
          JOIN RoleDescription AS T1 ON T0 = T1.RoleID
          JOIN Ability AS T2 ON T1.AbilityID = T2.ID
          JOIN AgentAbility AS T3 ON T2.ID = T3.AbilityID
          JOIN Agent AS T4 ON T3.AbilityID = T4.ID
WHERE     T0.Name = 'Hausmeister'
```

Use Case 7: Die Stellen, die die Stelle 'Hausmeister1' in der Rolle 'Rasenpfleger' vertreten

(OrgPosition.Name = 'Hausmeister1').
getSubstitutionRule (Role.Name = 'Rasenpfleger').getSubstitute().getAgent()

1. Selektion: *(OrgPosition.Name = 'Hausmeister1')*
   $Count = 0

```
SELECT    T0.ID
FROM      OrgPosition AS T0
WHERE     T0.Name = 'Hausmeister1'
```

2. Funktion: *OrgPosition.getSubstitutionRule()*
   $Count = $Count++ = 1

```
SELECT    T1.ID
FROM      OrgPosition AS T0
          JOIN SubstitutionRule AS T1
          ON T0.ID = T1.OrgPositionID
WHERE     T0.Name = 'Hausmeister1'
```

3. Selektion : *(Role.Name = 'Rasenpfleger')*
   $Count = $Count++ = 2

```
SELECT    T1.ID
FROM      OrgPosition AS T0
          JOIN SubstitutionRule AS T1
          ON T0.ID = T1.OrgPositionID
          JOIN Role AS T2 ON T1.RoleID = T2.ID
WHERE     T0.Name = 'Hausmeister1'
          AND T2.Name = 'Rasenpfleger'
```

4. Funktion : *SubstitutionRule.getSubstitute()*
   $Count = $Count++ = 3

```
SELECT    T3.ID
FROM      OrgPosition AS T0
          JOIN SubstitutionRule AS T1
          ON T0.ID = T1.OrgPositionID
          JOIN Role AS T2 ON T1.RoleID = T2.ID
          JOIN OrgPosition AS T3 ON T1.SubstituteID = T3.ID
WHERE     T0.Name = 'Hausmeister1'
          AND T2.Name = 'Rasenpfleger'
```

5. Funktion : *OrgPosition.getAgent()*
   $Count = $Count++ = 4

```
SELECT    T4.OrgPositionID, T5.ID
FROM      OrgPosition AS T0
          JOIN SubstitutionRule AS T1
          ON T0.ID = T1.OrgPositionID
          JOIN Role AS T2 ON T1.RoleID = T2.ID
          JOIN OrgPosition AS T3 ON T1.SubstituteID = T3.ID
          JOIN OrgPositionOccupation AS T4
          ON T3.ID = T4.OrgPositionID
          JOIN Agent AS T5 ON T4.AgentID = T5.ID
WHERE     T0.Name = 'Hausmeister1'
          AND T2.Name = 'Rasenpfleger'
```

Use Case 8: Beschreibt alle Stellen, die eine Rolle 'Krankenschwester' vertreten.

(Role.Name = 'Krankenschwester').getSubstitutionRule().getSubstitute().getAgent()

1. Selektion: *(Role.Name = 'Krankenschwester')*
   Count$ = 0

```
SELECT    T0.ID
FROM      Role AS T0
WHERE     T0.Name = 'Krankenschwester'
```

2. Funktion: *Role.getSubstitutionRule()*
   $Count = $Count++ = 1

```
SELECT    T1.ID
FROM      Role AS T0
          JOIN SubstitutionRule AS T1 ON T0.ID = T1.RoleID
WHERE     T0.Name = 'Krankenschwester'
```

3. Funktion: *SubstitutionRule.getSubstitute()*
   $Count = $Count++ = 2

```
SELECT    T2.ID
FROM      Role AS T0
          JOIN SubstitutionRule AS T1
          ON T0.ID = T1.RoleID
          JOIN OrgPosition AS T2 ON T1.SubstituteID = T2.ID
WHERE     T0.Name = 'Krankenschwester'
```

4. Funktion: *OrgPosition.getAgent()*
   $Count = $Count++ = 3

```
SELECT    T3.OrgPositionID, T4.ID
FROM      Role AS T0
          JOIN SubstitutionRule AS T1
          ON T0.ID = T1.RoleID
          JOIN OrgPosition AS T2 ON T1.SubstituteID = T2.ID
          JOIN OrgPositionOccupation AS T3
          ON T2.ID = T3.OrgPositionID
          JOIN Agent AS T4 ON T3.AgentID = T4.ID
WHERE     T0.Name = 'Krankenschwester'
```

   $Count = $Count++ = 4

Use Case 9: beschreibt alle Stellen, die in der Rolle Krankenschwester vertreten werden.

(Role.Name = 'Krankenschwester').getSubstitutionRule().getSubstituted().getAgent()

1. Selektion: *(Role.Name = 'Krankenschwester')*
   $Count = 0
```
SELECT    T0.ID
FROM      Role AS T0
WHERE     T0.Name = 'Krankenschwester'
```

2. Funktion: *Role.getSubstitutionRule()*
   $Count = $Count++ = 1
```
SELECT   T1.ID
FROM     Role AS T0
         JOIN SubstitutionRule AS T1 ON T0.ID = T1.RoleID
WHERE    T0.Name = 'Krankenschwester'
```

3. Funktion: *SubstitutionRule.getSubstituted()*
   $Count = $Count++ = 2
```
SELECT    T2.ID
FROM      Role AS T0
          JOIN SubstitutionRule AS T1 ON T0.ID = T1.RoleID
          JOIN OrgPosition AS T2 ON T1.OrgPositionID = T2.ID
WHERE     T0.Name = 'Krankenschwester'
```

4. Funktion: *OrgPosition.getAgent()*
   $Count = $Count++ = 3
```
SELECT    T3.OrgPositionID, T4.ID
FROM      Role AS T0
          JOIN SubstitutionRule AS T1 ON T0.ID = T1.RoleID
          JOIN OrgPosition AS T2 ON T1.OrgPositionID = T2.ID
          JOIN OrgPositionOccupation AS T3
          ON T2.ID = T3.OrgPositionID
          JOIN Agent AS T4 ON T3.AgentID = T4.ID
WHERE     T0.Name = 'Krankenschwester'
```
   $Count = $Count++ = 4

Use Case 10: Alle Stellen, die die Stelle 'Generaldirektor' im Jahr 2005 vertreten (egal in welcher Rolle)

(OrgPosition.Name='Generaldirektor').
getSubstitutionRule(SubstitutionRule.ZeitRaum='2005').getSubstitute().getAgent()

1. Selektion: *(OrgPosition.Name = 'Generaldirektor')*
   $Count = 0
```
SELECT    T0.ID
FROM      OrgPosition AS T0
WHERE     T0.Name = 'Generaldirektor'
```

2. Funktion: *OrgPosition.getSubstitutionRule()*
   $Count = $Count++ = 1

```
SELECT    T1.ID
FROM      OrgPosition AS T0
          JOIN SubstitutionRule AS T1
          ON T0.ID = T1.OrgPositionID
WHERE     T0.Name = 'Generaldirektor'
```

3. Selektion: *(SubstitutionRule.ZeitRaum='2005')*

```
SELECT    T1.ID
FROM      OrgPosition AS T0
          JOIN SubstitutionRule AS T1
          ON T0.ID = T1.OrgPositionID
WHERE     T0.Name = 'Generaldirektor'
          AND T1.Zeitraum = '2005'
```

4. Funktion: *SubstitutionRule.getSubstitute()*
   $ = $++ = 2

```
SELECT    T2.ID
FROM      OrgPosition AS T0
          JOIN SubstitutionRule AS T1
          ON T0.ID = T1.OrgPositionID
          JOIN OrgPosition AS T2 ON T1.SubstituteID = T2.ID
WHERE     T0.Name = 'Generaldirektor'
          AND T1.Zeitraum = '2005'
```

5. Funktion: *OrgPosition.getAgent()*
   $Count = $Count++ = 3

```
SELECT    T3.OrgPositionID, T4.ID
FROM      OrgPosition AS T0
          JOIN SubstitutionRule AS T1
          ON T0.ID = T1.OrgPositionID
          JOIN OrgPosition AS T2 ON T1.SubstituteID = T2.ID
          JOIN OrgPositionOccupation AS T3
          ON T2.ID = T3.OrgPositionID
          JOIN Agent AS T4 ON T3.AgentID = T4.ID
WHERE     T0.Name = 'Generaldirektor' AND
          T1.Zeitraum = '2005'
```

   $Count = $Count++ = 4

Abhängige Bearbeiterzuordnungen:

Use Case 1: Derselbe Bearbeiter wie der von Aktivität 0815

(Activity.ID='0815').getAgent()

1. Ersetzen der abhängigen Teile laut Tabelle 5-8:
   (Agent.ID='007')

2. Erste Selektion:
   $Count = 0
```
SELECT    NULL, T0.ID
FROM      Person AS T0
WHERE     T0.ID = '007'
```

Use Case 2: Der Vorgesetzte des Bearbeiters von Aktivität 0815

(Activity.ID='0815').getOrgPosition().getOrgUnit().getManager().getAgent()

1. Ersetzen der abhängigen Teile laut Tabelle:
   (OrgPosition.ID='4711') .getOrgUnit().getManager().getAgent()

2. Erste Selektion: *(OrgPosition.ID='4711')*
   $Count = 0
```
SELECT    T0.ID
FROM       OrgPosition AS T0
WHERE     T0.ID = '4711'
```

3. Funktion: *OrgPosition.getOrgUnit()*
   $Count = $Count++ = 1
```
SELECT    T1.ID
FROM      OrgPosition AS T0
          JOIN OrgUnit AS T1 ON T0.OrgUnitID = T1.ID
WHERE     T0.ID = '4711'
```

4. Funktion: *OrgUnit.getManager()*
   $Count = $Count++ = 2
```
SELECT    T2.ID
FROM      OrgPosition AS T0
          JOIN OrgUnit AS T1 ON T0.OrgUnitID = T1.ID
          JOIN OrgPosition AS T2 ON T1.ManagerID = T2.ID
WHERE     T0.ID = '4711'
```

5. Funktion: *OrgPosition.getAgent()*
   $Count = $Count++ = 3
```
SELECT    T3.OrgPositionID, T4.ID
FROM      OrgPosition AS T0
          JOIN OrgUnit AS T1 ON T0.OrgUnitID = T1.ID
          JOIN OrgPosition AS T2 ON T1.ManagerID = T2.ID
          JOIN OrgPositionOccupation AS T3
          ON T2.ID = T3.OrgPositionID
          JOIN Agent AS T4 ON T3.AgentID = T4.ID
WHERE     T0.ID = '4711'
```
   $Count = $Count++ = 4

Use Case 3: Bearbeiter ist aus derselben Org.Einheit wie der Prozeßverantwortliche

Instance.getSupervisor().getOrgUnit().getOrgPosition().getAgent()

1. Ersetzen der abhängigen Teile laut Tabelle 5-8:
   (OrgPosition.ID='4711').*getOrgUnit().getOrgPosition().getAgent()*

2. Erste Selektion: *(OrgPosition.ID='4711')*
   $Count = 0
```
SELECT    T0.ID
FROM      OrgPosition AS T0
WHERE     T0.ID = '4711'
```

3. Funktion: *OrgPosition.getOrgUnit()*
   $Count = $Count++ = 1
```
SELECT    T1.ID
FROM      OrgPosition AS T0
          JOIN OrgUnit AS T1 ON T0.OrgUnitID = T1.ID
WHERE     T0.ID = '4711'
```

4. Funktion: *OrgUnit.getOrgPosition()*
   $Count = $Count++ = 2
```
SELECT    T2.ID
FROM      OrgPosition AS T0
          JOIN OrgUnit AS T1 ON T0.OrgUnitID = T1.ID
          JOIN OrgPosition AS T2 ON T1.ID = T2.OrgUnitID
WHERE     T0.ID = '4711'
```

5. Funktion: *OrgPosition.getAgent()*
   $Count = $Count++ = 3
```
SELECT    T3.OrgPositionID, T4.ID
FROM      OrgPosition AS T0
          JOIN OrgUnit AS T1 ON T0.OrgUnitID = T1.ID
          JOIN OrgPosition AS T2 ON T1.ID = T2.OrgUnitID
          JOIN OrgPositionOccupation AS T3
          ON T2.ID = T3.OrgPositionID
          JOIN Agent AS T4 ON T3.AgentID = T4.ID
WHERE     T0.ID = '4711'
```
   $Count = $Count++ = 4

Use Case 4: Bearbeiter ist aus derselben Org.Einheit wie der Bearbeiter von Aktivität 0815, ist aber nicht dieser Arbeiter

(Activity.ID = '0815').getOrgPosition().getOrgUnit().getOrgPosition().getAgent() AND
(Activity.ID != '0815').getOrgPosition_I().getAgent()

1. Ersetzen der abhängigen Teile laut Tabelle 5-8:
   (OrgPosition.ID = '4711').getOrgUnit().getOrgPosition().getAgent() AND (OrgPosition.ID != '4711').getAgent()

2. Selektion: *(OrgPosition.ID = '4711')*
   $Count = 0
```
SELECT    T0.ID
FROM      OrgPosition AS T0
WHERE     T0.ID = '4711'
```

3. Funktion: *OrgPosition.getOrgUnit()*
   $Count = $Count++ = 1
```
SELECT    T1.ID
FROM      OrgPosition AS T0
          JOIN OrgUnit AS T1 ON T0.OrgUnitID = T1.ID
WHERE     T0.ID = '4711'
```

4. Funktion: *OrgUnit.getOrgPosition()*
   $Count = $Count++ = 2
```
SELECT    T2.ID
FROM      OrgPosition AS T0
          JOIN OrgUnit AS T1 ON T0.OrgUnitID = T1.ID
          JOIN OrgPosition AS T2 ON T1.ID = T2.OrgUnitID
WHERE     T0.ID = '4711'
```

5. Funktion: *OrgPosition.getAgent()*
   $Count = $Count++ = 3
```
SELECT    T3.OrgPositionID, T4.ID
FROM      OrgPosition AS T0
          JOIN OrgUnit AS T1 ON T0.OrgUnitID = T1.ID
          JOIN OrgPosition AS T2 ON T1.ID = T2.OrgUnitID
          JOIN OrgPositionOccupation AS T3
          ON T2.ID = T3.OrgPositionID
          JOIN Agent AS T4 ON T3.AgentID = T4.ID
WHERE     T0.ID  = '4711'
```
   $Count = $Count++ = 4

6. Operator: *AND*
```
SELECT    T3.OrgPositionID, T4.ID
FROM      OrgPosition AS T0
          JOIN OrgUnit AS T1 ON T0.OrgUnitID = T1.ID
          JOIN OrgPosition AS T2 ON T1.ID = T2.OrgUnitID
          JOIN OrgPositionOccupation AS T3
          ON T2.ID = T3.OrgPositionID
          JOIN Agent AS T4 ON T3.AgentID = T4.ID
WHERE     T0.ID = '4711'
INTERSECT
```

7. Selektion: *(OrgPosition.ID != '4711')*
   $Count = $Count++ = 5

```
SELECT   T3.OrgPositionID, T4.ID
FROM     OrgPosition AS T0
         JOIN OrgUnit AS T1 ON T0.OrgUnitID = T1.ID
         JOIN OrgPosition AS T2 ON T1.ID = T2.OrgUnitID
         JOIN OrgPositionOccupation AS T3
         ON T2.ID = T3.OrgPositionID
         JOIN Agent AS T4 ON T3.AgentID = T4.ID
WHERE    T0.ID = '4711'
INTERSECT
SELECT   T5.ID
FROM     OrgPosition AS T5
WHERE    T5.ID != '4711'
```

8. Funktion: *OrgPosition.getAgent()*
   $Count = $Count++ = 6

```
SELECT   T3.OrgPositionID, T4.ID
FROM     OrgPosition AS T0
         JOIN OrgUnit AS T1 ON T0.OrgUnitID = T1.ID
         JOIN OrgPosition AS T2 ON T1.ID = T2.OrgUnitID
         JOIN OrgPositionOccupation AS T3
         ON T2.ID = T3.OrgPositionID
         JOIN Agent AS T4 ON T3.AgentID = T4.ID
WHERE    T0.ID = '4711'
INTERSECT
SELECT   T6.OrgPositionID, T7.ID
FROM     OrgPosition AS T5
         JOIN OrgPositionOccupation AS T6
         ON T5.ID = T6.OrgPositionID
         JOIN Agent AS T7 ON T6.AgentID = T7.ID
WHERE    T5.ID != '4711'
```

   $Count = $Count++ = 7

Use Case 5: Bearbeiter ist der Prozeßinitiator

Instance.getInitiatorByPerson()

1. Ersetzen der abhängigen Teile laut Tabelle 5-8:
   (Agent.ID = '007')

2. Selektion: *(Agent.ID = '007)*
   $Count = 0

```
SELECT   NULL, T0.ID
FROM     Agent AS T0
WHERE    T0.ID = '007'
```

Use Case 6: Bearbeiter ist aus derselben Org.Einheit, wie der Prozeßverantwortliche

Instance.getSupervisor().getOrgUnit().getOrgPosition().getAgent()

1. Ersetzen der abhängigen Teile laut Tabelle 5-8:
   (OrgPosition.ID = '4711').getOrgUnit().getOrgPosition().getAgent()
2. Selektion *(OrgPosition.ID = '4711')*

   $Count = 0

```
SELECT    T0.ID
FROM      Orgposition AS T0
WHERE     T0.ID = '4711'
```

3. Funktion: *Orgposition.getOrgUnit()*

   $ = $++ = 1

```
SELECT    T1.ID
FROM      OrgPosition AS T0
          JOIN OrgUnit AS T1 ON T0.OrgUnitID = T1.ID
WHERE     T0.ID  = '4711'
```

4. Funktion: *OrgUnit.getOrgPosition()*

   $Count = $Count++ = 2

```
SELECT    T2.ID
FROM      OrgPosition AS T0
          JOIN OrgUnit AS T1 ON T0.OrgUnitID = T1.ID
          JOIN OrgPosition AS T2 ON T1.ID = T2.OrgUnitID
WHERE     T0.ID = '4711'
```

5. Funktion: *OrgPosition.getAgent()*

   $Count = $Count++ = 3

```
SELECT    T3.OrgPositionID, T4.ID
FROM      OrgPosition AS T0
          JOIN OrgUnit AS T1 ON T0.OrgUnitID = T1.ID
          JOIN OrgPosition AS T2 ON T1.ID = T2.OrgUnitID
          JOIN OrgPositionOccupation AS T3
          ON T2.ID = T3.OrgPositionID
          JOIN Agent AS T4 ON T3.AgentID = T4.ID
WHERE     T0.ID = '4711'
```

   $Count = $Count++ = 4

Use Case 7: Der Vorgesetzte der Einheit, die der Einheit übergeordnet ist, in der die Aktivität ausgeführt wurde

(Activity.ID = '0815').getOrgPosition().getOrgUnit().getSupOrgUnit()
.getManager().getAgent()

1. Ersetzen der abhängigen Teile laut Tabelle 5-8:
   (OrgPosition.ID = '4711').getOrgUnit().getSupOrgUnit().getManager().getAgent()

2. Selektion: (OrgPosition.ID = '4711')
   $Count = 0
```
SELECT    T0.ID
FROM      OrgPosition AS T0
WHERE     T0.ID = '4711'
```

3. Funktion: *OrgPosition.getOrgUnit()*
   $Count = $Count++ = 1
```
SELECT    T1.ID
ROM       OrgPosition AS T0
          JOIN OrgUnit AS T1 ON T0.OrgUnitID = T1.ID
WHERE     T0.ID = '4711'
```

4. Funktion: *OrgUnit.getSupOrgUnit()*
   $Count = $Count++ = 2
```
SELECT    T2.ID
FROM      OrgPosition AS T0
          JOIN OrgUnit AS T1 ON T0.OrgUnitID = T1.ID
          JOIN OrgUnit AS T2 ON T1.SupID = T2.ID
WHERE     T0.ID = '4711'
```

5. Funktion: *OrgUnit.getManager()*
   $Count = $Count++ = 3
```
SELECT    T3.ID
FROM      OrgPosition AS T0
          JOIN OrgUnit AS T1 ON T0.OrgUnitID = T$.ID
          JOIN OrgUnit AS T2 ON T1.SupID = T2.ID
          JOIN OrgPosition AS T3 ON T2.ManagerID = T3.ID
WHERE     T0.ID = '4711'
```

6. Funktion: *OrgPosition.getAgent()*
   $Count = $Count++ = 4
```
SELECT    T4.OrgPositionID, T5.ID
FROM      OrgPosition AS T0
          JOIN OrgUnit AS T1 ON T0.OrgUnitID = T$.ID
          JOIN OrgUnit AS T2 ON T1.SupID = T2.ID
          JOIN OrgPosition AS T3 ON T2.ManagerID = T3.ID
          JOIN OrgPositionOccupation AS T4
          ON T3.ID = T4.OrgPositionID
          JOIN Agent AS T5 ON T4.AgentID = T5.ID
WHERE     T0.ID = '4711'
```
   $Count = $Count++ = 5

Name: Marco Berroth Matrikelnummer: 404244

Ich erkläre, daß ich die Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den ....
(Unterschrift)

Danksagung:

An dieser Stelle möchte ich all denjenigen danken, die zum Gelingen dieser Arbeit beigetragen haben.

Ein besonderer Dank gilt hierbei Herrn Prof. Dr. Peter Dadam von der Abteilung DBIS der Universität Ulm für die ausführlichen Diskussionen zum Thema der Arbeit.

Ebenso danken möchte ich Herrn Dr. Manfred Reichert von der Universität Twente für die konstruktive Kritik am Inhalt dieser Arbeit.

Bedanken möchte ich mich auch bei meinem direkten Ansprechpartner Dipl.-Inf. Ulrich Kreher für die gute Betreuung und das starke Interesse am Fortschritt der Diplomarbeit und die ständige Diskussionsbereitschaft.

Im gleichen Atemzug sind dabei Dipl.-Inf. Hilmar Acker und Dipl.-Inf. Markus Lauer zu nennen.

Lin Thao Ly danke ich für die Diskussion zum Thema Organisationsstrukturmetamodell, da sie die einzelnen Elemente des Modells kritisch hinterfragte.

In zeitlicher Nähe zur Durchführung dieser Arbeit wurde auch ein Praktikum durchführt, um die gewonnenen Erkenntnisse und den Entwurf in Software zu implementieren. Auch den Praktikumsteilnehmern gebührt Dank für die Hinweise auf inhaltliche und logische Fehler. Außerdem haben sie die Eigenheiten der eingesetzten Datenbanken erarbeitet. Es sind:
Xin Hong, Stefan Michaelsen, Bertrand Tchoumkeu, Liu Yaning, Zhongda Zhao und Jun Zhu

Durch ihre Korrektur haben mich meine Freunde Christian Hahn, Michael Schellhammer und Kai Vanhöfen vor dem orthographischen Fehlerteufel bewahrt.

Mein ganz herzlicher Dank gilt nicht zuletzt meinen Eltern, die mich stets unterstützt und an mich geglaubt haben und denen ich viel zu selten dafür gedankt habe.