Studiengang:	Informatik
Prüfer:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Betreuer:	Dipl.-Inform. Jürgen Klarmann
begonnen am:	1. Februar 2001
beendet am:	31. Juli 2001
CR-Klassifikation:	F.2.2, G.2.2, H.4.1, I.2.1, I.2.3, I.2.4, I.2.8, J.1

Diplomarbeit Nr. 1913

Entwicklung einer wissensbasierten Zuweisung von Aufgabenträgern in einem Workflow-Management-System

Frank Dürr

Institut für Parallele und
Verteilte Höchstleistungsrechner
Universität Stuttgart
Breitwiesenstraße 20-22
D-70565 Stuttgart

Zusammenfassung

Das Umfeld eines heutigen Unternehmens ist von ständigem und beschleunigtem Wandel geprägt. Verschiedene Faktoren, wie z.B. kurze Produktlebenszyklen, wachsender Konkurrenzdruck, anspruchsvollere Kunden, sich ändernde Rechtsvorschriften und kurze Entwicklungszeiten für neue Produkte, zwingen Unternehmen, einen hohen Grad an Flexibilität und Wandlungsfähigkeit zu entwickeln, um in diesem turbulenten Umfeld erfolgreich bestehen zu können.

Die Forderung nach Flexibilität und Wandlungsfähigkeit läßt sich direkt auf Workflow-Management-Systeme übertragen, denn sie sind für die Steuerung und Kontrolle von Arbeitsabläufen innerhalb eines Unternehmens verantwortlich. Diese Systeme müssen daher in der Lage sein, Änderung an Abläufen und an der Organisationsstruktur mitzuvollziehen. Bisherige Arbeiten konzentrierten sich dabei hauptsächlich auf die flexible Ablauforganisation. Doch gerade im Bereich der Aufbauorganisation und bei der Kopplung von Ablauf- und Aufbauorganisation weisen heutige Workflow-Management-Systeme Defizite auf. Vor allem die Kopplung ist bisher sehr starr und unflexibel, wodurch sich Änderungen an der Aufbauorganisation direkt auf die Ablauforganisation auswirken und umgekehrt. Dieses Manko ist hauptsächlich auf die Metamodelle aktueller Systeme zurückzuführen, die es nicht ermöglichen, im Modell der Organisationsstruktur genügend und vor allem geeignete Informationen unterzubringen, um damit eine flexible Zuweisung von Aktivitäten zu Aufgabenträgern zu ermöglichen.

Diese Arbeit zeigt, wie durch eine wissensbasierte Modellierung der Aufbauorganisation eine wesentlich höhere Entkopplung von Ablauf- und Aufbauorganisation erreicht werden kann, als das zur Zeit der Fall ist. Sowohl zur Modellierung der Aufbauorganisation als auch zur Selektion geeigneter Aufgabenträger werden hierbei Konzeptgraphen eingesetzt, die ein sehr anschauliches Mittel zur Wissensrepräsentation darstellen, aber gleichzeitig auch durch ihre Abbildung auf Formeln der Prädikatenlogik 1. Stufe eine solide theoretische Basis besitzen. Es wird ein auf Konzeptgraphen basierendes Metamodell vorgestellt, das allgemein genug ist, verschiedene Organisationsstrukturen zu modellieren und außerdem die Möglichkeit zur individuellen Anpassung bietet. Außerdem werden verschiedene Algorithmen zur Selektion von Aufgabenträgern vorgestellt und bezüglich ihres Antwortzeitverhaltens miteinander verglichen. Es erfolgt eine prototypische Realisierung des vorgestellten Ansatzes und eine Integration in das Workflow-Management-System Changengine.

Vorwort

Diese Arbeit entstand im Rahmen des Teilprojekts C4 des Sonderforschungsbereiches 467 (SFB 467) *Wandlungsfähige Unternehmensstrukturen für die variantenreiche Serienproduktion*. Dieser SFB, der im Januar 1997 gegründet wurde, hat als zentrale Zielsetzung die Erarbeitung eines theoretisch fundierten, fachübergreifend überprüften und umsetzbaren Gestaltungskonzepts für wandlungsfähige Unternehmen. Das Teilprojekt C4 beschäftigt sich dabei mit der informationstechnischen Kommunikation und Kooperation. Innerhalb des Teilprojekts C4 ist die Abteilungen Bildverstehen (BV) des Instituts für parallele und verteilte Höchstleistungsrechner (IPVR) der Universität Stuttgart für die Entwicklung geeigneter Modelle für die agentenbasierte Kooperation und Kommunikation zuständig, während sich die Abteilung Verteilte Systeme (VS) des IPVR mit Modellen für die menschliche Kooperation beschäftigt.

Mein Dank gilt vor allem Herrn Dipl.-Inform. Jürgen Klarmann, der als Betreuer dieser Arbeit immer ein offenes Ohr für Fragen hatte und jederzeit für Diskussionen zur Verfügung stand.

Inhaltsverzeichnis

1 Einleitung und Motivation der Arbeit **1**

1.1 Ursprünge heutiger Unternehmensstrukturen 1

1.2 Wandel des Unternehmensumfeldes 2

1.2.1 Treibende Kräfte 2

1.2.2 Unternehmen im turbulenten Umfeld 3

1.3 Business Reengineering 3

1.3.1 Grundlagen des Business Reengineerings 4

1.3.2 Fallbeispiel 4

1.3.3 Auswirkungen auf das Unternehmen 6

1.4 Konsequenzen des Wandels für Workflow-Management-Systeme 7

1.4.1 Formen des Wandels 8

1.4.2 Bestehende Lösungsansätze 8

1.5 Aufgabenstellung und Zielsetzung dieser Arbeit 10

2 Grundlagen **13**

2.1 Betriebswirtschaftliche Grundlagen 13

2.1.1 Ablauforganisation 13

2.1.2 Aufbauorganisation 14

2.1.2.1 Organisationseinheiten 14

2.1.2.2 Grundmodelle der Aufbauorganisation 16

2.2 Workflow-Management-Systeme 19

2.2.1 Definitionen 20

2.2.2 Aspekte eines Workflow-Management-Systems 21

2.2.3 Komponenten eines Workflow-Management-Systems 25

3 Analyse bestehender Ansätze **27**

3.1 Ansätze zur Modellierung der Organisationsstruktur 27

3.1.1 WorkParty 28

3.1.2 FlowMark 29

3.1.3 Leu . . . 30
3.1.4 Diplomarbeit Nr. 1763 an der Universität Stuttgart . . . 31
3.1.5 Workflow Management Coalition . . . 32
3.1.6 MOBILE (Forschung Universität Erlangen) . . . 32
3.2 Diskussion der Modellierungsansätze . . . 33
3.3 Ansätze zur Zuweisung von Aufgabenträgern . . . 35
3.3.1 FlowMark . . . 36
3.3.2 WorkParty . . . 37
3.3.3 Leu . . . 37
3.3.4 Workflow Management Coalition . . . 37
3.3.5 Diplomarbeit Nr. 1763 an der Universität Stuttgart . . . 37
3.3.6 MOBILE (Forschung Universität Erlangen) . . . 38
3.4 Diskussion der Ansätze zur Zuweisung von Aufgabenträgern . . . 40
3.4.1 Vergleich . . . 40
3.4.2 Bewertung . . . 42
3.5 Zusammenfassung . . . 42

4 Konzeptgraphen 45
4.1 Einführung und Notationen . . . 45
4.2 Konzepte und Relationen . . . 46
4.2.1 Kanonische Konzeptgraphen . . . 47
4.2.2 Konzeptarten . . . 48
4.2.2.1 Generische Konzepte . . . 48
4.2.2.2 Individuelle Konzepte . . . 48
4.2.3 Konzepttypen und Konzepttyphierarchien . . . 50
4.2.4 Relationstypen und Relationstyphierarchien . . . 53
4.3 Formale Definition . . . 53
4.4 Logische Grundlagen . . . 54
4.5 Operationen auf Konzeptgraphen . . . 56
4.5.1 Kanonische Formationsregeln . . . 56
4.5.2 Generalisierung und Spezialisierung . . . 57
4.6 Erweiterte Konzeptgraphen . . . 58
4.7 Ontologien . . . 59

5 Entwurf der Organisationsverwaltung 61
5.1 Anforderungsanalyse . . . 61
5.2 Ablauf der Aufgabenträgerzuweisung . . . 62
5.3 Entwurf der Organisationsstrukturverwaltung . . . 64
5.3.1 Basis-Metamodell der Organisationsstrukturverwaltung . . . 65
5.3.2 Entwurf der Wissensbasis . . . 66
5.3.2.1 Fakten . . . 67
5.3.2.2 Regeln . . . 67
5.3.2.3 Formale Definition der Wissensbasis . . . 70
5.3.3 Selektion von Aufgabenträgern . . . 70
5.3.3.1 Anfragegraphen . . . 71
5.3.3.2 Ergebnis einer Anfrage . . . 71
5.3.4 Selektion durch Graph-Matching . . . 72
5.3.4.1 Korrektheit . . . 73
5.3.4.2 Vollständigkeit . . . 73
5.3.4.3 Komplexität der Matching-Operation . . . 77
5.3.4.4 Ein P-Algorithmus für die Matching-Operation . . . 78
5.3.4.5 Ein NP-Algorithmus für die Matching-Operation . . . 83
5.3.4.6 Graph-Matching und Regeln . . . 88
5.3.4.7 Problem des Graph-Matching-Ansatzes . . . 91
5.3.5 Selektion durch logische Programmierung . . . 93
5.3.5.1 Transformation der Fakten . . . 94
5.3.5.2 Transformation der Typhierarchien . . . 95
5.3.5.3 Transformation der Regeln . . . 96
5.3.5.4 Anfragen . . . 96
5.3.6 Negierte Selektionen . . . 99
5.3.7 Partielle Projektionen . . . 100
5.3.7.1 P-Algorithmus . . . 101
5.3.7.2 NP-Algorithmus . . . 102
5.3.7.3 Bewertung von partiellen Projektionen . . . 109
5.3.8 Zuweisungsregelverwaltung . . . 114
5.3.9 Klassenstruktur . . . 115
5.3.9.1 Basisklassen . . . 115
5.3.9.2 Klassen der Wissensbasis . . . 115
5.3.9.3 Klassen der Anfragegraphen-Verwaltung . . . 116
5.3.9.4 Klassen der Taskholder-Finder-Komponente . . . 118
5.4 Integration in Changengine . . . 118

5.4.1 Aufgabenträgerzuweisung in Changengine . . . 118
5.4.2 Einbindung der erstellten Organisationsverwaltung . . . 120
5.5 Integration der Benutzungsoberfläche . . . 121
5.6 Entwurf einer Unternehmens-Ontologie . . . 122
5.6.1 Organisationsstruktur . . . 123
5.6.2 Fähigkeiten . . . 125
5.6.3 Rechte . . . 127
5.6.4 Rollen . . . 127
5.6.5 Metamodell . . . 128
5.6.6 Konsistenzbedingungen . . . 129

6 Implementierung und Evaluation 131
6.1 Implementierung . . . 131
6.1.1 Basisklassen . . . 131
6.1.2 Wissensbasis . . . 131
6.1.3 Anfragegraphen-Verwaltung . . . 133
6.1.4 Taskholder-Finder . . . 133
6.1.5 Taskholder-Finder-Client . . . 136
6.1.6 Testumgebung . . . 139
6.2 Evaluation . . . 139
6.2.1 Bewertung der Matching-Algorithmen . . . 139
6.2.1.1 Testfälle . . . 140
6.2.1.2 Messungen 1 . . . 142
6.2.1.3 Messungen 2 . . . 142
6.2.1.4 Fazit der Messungen . . . 144
6.2.2 Bewertung der erreichten Entkopplung . . . 146
6.2.2.1 Auswirkungen von Änderungen . . . 146
6.2.2.2 Fazit . . . 151
6.2.3 Abschließende Bewertung . . . 151

7 Zusammenfassung und Ausblick 153

Abbildungsverzeichnis 157

A Enterprise-Ontologie 159
A.1 Begriffe der Enterprise-Ontologie . . . 159
A.2 Meta-Ontologie . . . 160

B Begriffe der entworfenen Unternehmensontologie **161**

B.1 Konzepte 161

B.2 Relationen 162

C CORBA-Schnittstellen der Organisationsverwaltung **165**

C.1 Wissensbasis 165

C.2 Querybase 171

C.3 Taskholder-Finder 174

D Messergebnisse der Laufzeittests **177**

Kapitel 1

Einleitung und Motivation der Arbeit

Dieses einleitende Kapitel liefert die grundlegende Motivation für die weitere Arbeit. Es wird aufgezeigt, wie und warum sich Unternehmensprozesse und Unternehmensstrukturen im Laufe der Zeit gewandelt haben, warum es heute mehr denn je notwendig ist, eine hohe Flexibilität in diesen Punkten zu erreichen, und warum die heute von Unternehmen geforderte Flexibilität ebenso für Workflow-Management-Systeme gilt.

Dieses Kapitel stützt sich im Wesentlichen auf die Erkenntnisse von Hammer und Champy, wie sie in [**?**] beschrieben werden.

1.1 Ursprünge heutiger Unternehmensstrukturen

Der Großteil der heute vorzufindenden Organisationsstrukturen hat seine Wurzeln in Prinzipien, die bereits 1776 in dem Buch *Der Wohlstand der Nationen* von Adam Smith beschrieben wurden. Im Gegensatz zur bis dahin üblichen Vorgehensweise, ein ganzes Produkt oder zumindest große Teile davon von einem Handwerker (einem Generalisten) fertigen zu lassen, schlägt Smith vor, spezialisierten Arbeitskräften die Durchführung kleiner Teilarbeitsschritte zu überlassen. Jeder dieser Spezialisten ist in der Lage, seinen Arbeitsschritt wesentlich schneller durchzuführen, als es der Generalist je könnte, der ständig zwischen den Arbeitsschritten wechseln muß und der auch nicht die Routine in den jeweiligen Arbeitsschritten erlangt. Durch dieses Prinzip der *Arbeitsteilung* ist eine wesentliche Produktivitätssteigerung gegenüber der traditionellen handwerklichen Fertigung zu erreichen. Die Einführung neuer Maschinen und des Fließbandes führte zu einer Perfektionierung dieses Prinzips.

Ständig expandierende Unternehmen erforderten in der Folge einen immer größeren organisatorischen Aufwand, um das Unternehmen zu leiten. Alfred Sloan begegnete diesem Aufwand bei General Motors durch Gliederung des Unternehmens in kleine, dezentral Bereiche (Divisionen; siehe auch Abs. 7). Diese Bereiche wurden jeweils von einem Management von der Konzernzentrale aus überwacht und geleitet. Spezialisten mit entsprechendem Fachwissen übernahmen die Überwachung einzelner Fachbereiche. Das bereits auf Fertigungsebene eingeführte Prinzip der Arbeitsteilung wurde dadurch quasi auf der Führungsebene fortgesetzt.

Die vor allem in der Nachkriegszeit stetig wachsende Nachfrage nach Konsumgütern führte zu einer Expansion vieler Unternehmen, die als Hauptziel die Kapazitätssteigerung hatten, um diese Nachfrage befriedigen zu können. Gerade die oben beschriebene Gliederung der Unternehmen

machte eine solche Expansion relativ einfach, da einfach entsprechende Abteilungen in der Organisation hinzugefügt werden konnten. Es bildeten sich die auch noch heute oft anzutreffenden tiefen und hierarchischen Organisationsstrukturen mit ausgeklügelten Befugnissen, Kompetenzen, Aufgaben und Richtlinien, die für eine reibungslose Zusammenarbeit der einzelnen Abteilungen, Divisionen, usw. sorgen sollten. Kurzgefaßt lassen sich die heutigen Unternehmensstrukturen durch folgendes Zitat aus [**?**] beschreiben:

> Unternehmen zerlegen heute Arbeiten in bedeutungslose Aufgaben, weil sie auf diese Weise einst ihre Effizienz sicherstellten. Sie verteilen Macht und Verantwortung in riesigen Bürokratien, weil sie auf diese Weise lernten, ihre weitverzweigten Unternehmungen zu steuern.[**?**]

1.2 Wandel des Unternehmensumfeldes

Unbestritten waren die oben genannten hierarchischen Unternehmensstrukturen, basierend auf dem Prinzip der Arbeitsteilung, bisher sehr erfolgreich. Man kann sich also zu Recht fragen, warum sich das nun mehr oder wenig plötzlich ändern sollte, und warum es nun notwendig sein soll, die alten Strukturen aufzugeben, und neue aufzubauen, was sicherlich mit einem erheblichen Aufwand und auch Risiko verbunden ist. Auch hierauf geben Hammer und Champy in [**?**] eine Antwort und nennen drei Kräfte, die diese Änderung vorantreiben:

- Kunde
- Wettbewerb
- Wandel

1.2.1 Treibende Kräfte

Kunde und Wettbewerb Der Kunde ist heute in der Lage, selbst zu bestimmen, was er will, wann er es will, und wieviel er zu zahlen bereit ist. Das bedeutet einen grundlegenden Unterschied zur früheren Marksituation, bei der ein Standard-Produkt genügte, da der Kunde keine großen Möglichkeiten hatte, unter Produkten verschiedener Anbieter zu wählen. Heutige Kunden erwarten Produkte, die auf ihre persönlichen Bedürfnisse zugeschnitten sind. Das gilt für den privaten Kunden ebenso, wie für den Geschäftskunden. Und für den heutigen Kunden ist es zudem sehr einfach, aufgrund des durch Zeitung, Fernsehen, Radio, Internet und weiteren Medien zugänglichen Informationsangebotes, den Anbieter zu finden, der seinen persönlichen Ansprüchen genügt.

Ein gutes Beispiel sowohl für den privaten als auch den Geschäftskunden ist die Automobilindustrie. Kaum ein Anbieter kann es sich heute noch leisten, nur ein Standard-Modell für die entsprechenden Klassen anzubieten. Der Kunde erwartet verschiedenste Extras und Sonderausstattungen. Das fängt an bei der Wahl zwischen zig Lackierungen und endet sicher nicht bei der Wahl des Autoradios. Selbst die ausgefallensten Wünsche werden von kleinen Drittanbietern erfüllt, die noch Nischen ausfüllen können, von denen die großen Hersteller alleine aufgrund des kleinen Kundenkreises und der geringen Umsätze absehen müssen. Im geschäftlichen Bereich ist das nicht anders. Ein Hersteller hat heute die Wahl zwischen verschiedensten Zulieferern und kann die Preise und Leistungen entsprechend vorgeben, so lange sich nur einer findet, der bereit ist, zu diesen Konditionen zu liefern. Dadurch entsteht ein sehr viel stärkerer Wettbewerb, als das früher der Fall war.

Wandel Die dritte Kraft, die als Ursache für eine Änderung der Organisationstruktur eines Unternehmens verantwortlich ist, ist der stetige und beschleunigte Wandel. Die Produktlebenszyklen sind heute wesentlich kürzer als früher. Ein extremes Beispiel ist hier sicherlich die Computerindustrie. Was hier heute aktuell ist, kann in kürzester Zeit ein Ladenhüter sein.

Aber nicht nur die Produktlebenszyklen haben sich drastisch verkürzt. Auch die Zeit, die für die Entwicklung neuer Produkte zur Verfügung steht, wird immer geringer. Es gilt, die aktuellen Trends rasch zu erkennen, und dann möglichst mit einer Innovation vor der Konkurrenz auf dem Markt zu sein, denn so lange diese nicht mit entsprechenden Produkten aufwarten kann, hat man die besten Chancen, die eigenen an den Mann zu bringen. Neudeutsch spricht man davon, eine möglichst kurze „Time-to-Market“ zu erreichen. Das ist natürlich nur möglich, wenn das Unternehmen flexibel und reaktionsfähig genug ist, um erstens die Trends rechtzeitig zu erkennen, und zweitens die entsprechenden Produkte möglichst rasch zu entwickeln. Eine schwerfällige Unternehmensstruktur ist hier das größte Hindernis.

Außerdem sind Unternehmen sich ständig wandelnden Rechtsvorschriften unterworfen. So werden z.B. im Zuge eines vereinigten Europas nationale Gesetze angepaßt, um den Unternehmen innerhalb der EU eine gewisse Chancengleichheit einzuräumen. Grenzen werden abgebaut, und es entsteht dadurch ein größerer Markt aber auch mehr Konkurrenz. Andere Gesetzte, z.B. im Bereich des Umwelt- und Datenschutzes, können Unternehmen ebenfalls dazu zwingen, ihre Abläufe anzupassen, um den gängigen Rechtsvorschriften zu genügen.

Der hier beschriebene Wandel wird außerdem keine vorübergehende Erscheinung, sondern die Regel im heutigen Wettbewerb sein. Etwas paradox ausgedrückt kann man sagen, der Wandel ist heute die einzige Konstante, oder wie Hammer und Champy sagen, „der Wandel ist heute die Norm“.

1.2.2 Unternehmen im turbulenten Umfeld

Wachsender Konkurrenzdruck, anspruchsvollere Kunden, breitere Produktspektren, kurze Produktlebenszyklen, kurze Entwicklungszeiten und neue Gesetzte bilden heute das turbulente Umfeld eines Unternehmens. Um konkurrenzfähig zu bleiben, muß sich ein Unternehmen heute ständig an dieses sich rasch ändernde Umfeld anpassen. *Flexibilität* ist daher heute die wesentliche Anforderung an ein Unternehmen. Ein nicht flexibles Unternehmen wird einem flexiblen immer einen Schritt hinterher sein, was sich gerade in einem sich rasch ändernden Umfeld viel stärker auswirkt, als unter stabilen Bedingungen.

Flexibilität bedeutet vor allem, die Abläufe innerhalb eines Unternehmens so zu organisieren, daß eine möglichst effiziente Aufgabenerfüllung möglich ist. Dazu muß auch die Organisationsstruktur des Unternehmens ständig angepaßt werden, denn nur eine entsprechende Organisationsstruktur kann die Abläufe ideal unterstützen und macht somit erst die effiziente Durchführung möglich.

1.3 Business Reengineering

In diesem Abschnitt soll anhand eines konkreten Beispieles gezeigt werden, was es für ein Unternehmen bedeutet, die Arbeitsorganisation effizienter zu gestalten. Dieses Beispiel kommt aus dem Bereich des sogenannten Business Reengineerings, dessen Grundlagen zunächst erläutert werden, ehe ein konkretes Beispiel aus der Praxis vorgestellt und hinsichtlich seiner Auswirkungen auf das Unternehmen analysiert wird.

1.3.1 Grundlagen des Business Reengineerings

Wie bereits angesprochen, ist eine wesentlich Eigenschaft für eine Unternehmen Flexibilität, und die ist mit der traditionellen, tiefen Hierarchie und dem Prinzip der Zerlegung der Arbeit in kleinste Aufgaben und der Orientierung an Funktionen, wenn überhaupt, so doch nur sehr schwer zu erreichen. Hammer und Champy schreiben:

> Das Problem vieler Unternehmen besteht darin, daß sie an der Schwelle zum einundzwanzigsten Jahrhundert die Bürde eines organisatorischen Aufbaus zu tragen haben, der im neunzehnten Jahrhundert entstand ist und im zwanzigsten gute Dienste geleistet hat. Wir brauchen etwas Grundverschiedenes.[**?**]

Hammer und Champy fordern daher eine konsequente Ausrichtung der Arbeit an Geschäftsprozessen. Sie schlagen einen sehr umfassenden wie radikalen Ansatz vor, der das Unternehmen unter Umständen vollständig verändert. Dieser Ansatz mit dem Namen *Business Reengineering*[1] wird wie folgt definiert:

> Business Reengineering ist fundamentales Überdenken und radikales Redesign von Unternehmen oder wesentlichen Geschäftsprozessen. Das Resultat sind Verbesserungen um Größenordnungen in entscheidenden, heute wichtigen und meßbaren Leistungsgrößen in den Bereichen Kosten, Qualität, Service und Zeit. [**?**]

Es geht also um grundlegende und radikale Änderungen, d.h. keine Änderungen von kleinen Teilprozessen, sondern des gesamten Prozesses. Ein wesentlicher Punkt ist dabei die Ausrichtung am *Geschäftsprozeß*, anstatt einer funktionsorientierten Sicht. Es ist daher angebracht, zunächst eine Definition dieses zentralen Begriffes vorzunehmen. Die Literatur macht hierzu mehrere Vorschläge:

- Ein Bündel von Aktivitäten, für das ein oder mehrere Inputs benötigt werden und das für den Kunden ein Ergebnis von Wert erzeugt. [**?**]
- A set of one or more linked procedures or activities which collectively realise a business objective or policy goal, normally within the context of an organisational structure defining functional roles and relationships. [**?**]

Beide Definitionen sprechen von mehreren Aktivitäten und der Ausrichtung auf ein bestimmtes Ziel, wobei die erste Definition eher den Wert für den Kunden[2] hervorhebt.

1.3.2 Fallbeispiel

Die Auswirkungen des Business Reengineerings auf ein Unternehmen lassen sich am besten anhand eines Fallbeispieles erklären und verdeutlichen. Dieses Fallbeispiel ist [**?**] entnommen und stammt ursprünglich aus der Praxis, wird aber zur Veranschaulichung teilweise modifiziert. Die IBM Credit, ein Tochterunternehmen der Firma IBM, finanziert den Kauf von Produkten aus dem Hause IBM. Bis zur Genehmigung eines Finanzierungsantrags sind folgende Punkte zu durchlaufen:

1. Ein Außendienstmitarbeiter nimmt die Anfrage des Kunden entgegen.

[1] Teilweise auch als *Business Process Reengineering* bezeichnet.
[2] Der Kunde kann auch durchaus im eigenen Unternehmen zu finden oder selbst wieder ein Unternehmen sein.

2. Der Außendienstmitarbeiter übermittelt die Daten telefonisch an einen Mitarbeiter der IBM Credit. Dieser notiert den Auftrag auf Papier.

3. Der Auftrag wird an einen Mitarbeiter der Kreditabteilung übergeben. Dieser Spezialist überprüft mit Hilfe eines Computersystems die Kreditwürdigkeit des Antragstellers und notiert sein Ergebnis auf Papier.

4. Der Auftrag wird an die Vertragsabteilung übergeben, die ihn an die Anforderungen des Kunden anpaßt. Die Ergebnisse dieses Arbeitsschritts werden wiederum auf Papier notiert und dem Antrag beigefügt.

5. Ein weiterer Sachbearbeiter berechnet mit Hilfe einer Tabellenkalkulation den zugehörigen Zinssatz, notiert diesen auf dem Antrag und übergibt ihn einem Verwaltungsmitarbeiter.

6. Der Verwaltungsmitarbeiter erstellt ein Angebotsschreiben und übermittelt dieses postalisch an den Außendienstmitarbeiter.

Ein erster Verbesserungsversuch: Automatisierung Aufgrund der langen Durchlaufzeit – die Bearbeitung eines Antrages dauerte durchschnittlich sechs Tage, teilweise auch bis zu zwei Wochen – soll der Prozeß nun überarbeitet werden. Folgendes, in der Praxis nicht untypisches, Vorgehen ist denkbar.[3]

Um die fünf Auftragsübergaben zu automatisieren, wird ein Workflow-Management-System eingeführt. Ohne hier bereits auf die Details eines solchen Systems einzugehen[4], kann man sich vorstellen, daß dieses System die Reihenfolge der auszuführenden Schritte überwacht und nach Beendigung eines Arbeitsschrittes automatisch den nächsten einleitet. Anstatt die Ergebnisse eines Arbeitsschrittes auf Papier zu notieren, werden die Daten nun in eine Maske eingegeben und auch automatisch dem nächsten Bearbeiter zur Verfügung gestellt. Außerdem werden die Aufträge nun elektronisch archiviert, so daß deren Daten jederzeit abrufbar sind. Nun liegt ein Ablauf vor, wie er heute in vielen Unternehmen anzutreffen ist (man denke nur z.B. an die computerunterstützte Bearbeitung von Geschäftsprozessen in Banken und Versicherungen). Unternehmensprozesse werden mit Hilfe eines Workflow-Management-Systems bearbeitet, trotzdem bleibt oft die alte Einteilung in Abteilungen und Ressorts erhalten.

Leider läßt sich durch diese rein technische Lösung keine wesentlich größere Effizienz des Gesamtprozesses erreichen, denn der Prozeß hat weiterhin dramatische Nachteile:

- Viele Mitarbeiter sind an einem Antrag beteiligt. Daher steigt die Wahrscheinlichkeit für Fehler.

- Kein Mitarbeiter trägt alleine die Verantwortung für den Auftrag. Im Fehlerfall wird daher auch keiner die Verantwortung übernehmen.

- Es existieren immer noch viele Übergabestellen zwischen den Abteilungen, die den Gesamtprozeß trotz Automatisierung wesentlich verlangsamen.

- Ressortdenken: Jeder hat nur seinen kleinen Teil des Gesamtprozesses im Auge. Das wesentliche Ziel, nämlich die rasche Bearbeitung und Zufriedenheit des Kunden, wird aus dem Auge verloren.

[3] Dieser Schritte wurden in diesem Beispiel in Wirklichkeit nicht durchgeführt.
[4] Eine nähere Betrachtung von Workflow-Management-Systemen erfolgt in Kapitel 2.2.

Es ließen sich sicher noch weitere Nachteile aufzählen, aber dieses Beispiel soll nur aufzeigen, daß die Automatisierung *alleine* kein Allheilmittel darstellt. Oft gibt es auch nach der Einführung unterstützender Computersysteme noch genügend Potential für Verbesserungen.

Zweiter Verbesserungsversuch: Business Reengineering Die Änderungen, die Hammer und Champy im Zuge des Business Reengineerings vorschlagen, sind wesentlich radikaler, als die des ersten Verbesserungsversuches. Sie schlagen vor, einen Kreditantrag vollständig von *einem* Mitarbeiter bearbeiten zu lassen. Dieser eine Mitarbeiter ersetzt als Generalist nun mehrere Spezialisten. Ihm stehen allerdings dieselben Mittel (z.B. Hilfsprogramme) zur Verfügung, wie den Experten. Diese Verbesserung beruht auf der Erkenntnis, daß der Großteil der Anträge sich nicht wesentlich voneinander unterscheidet, und daher oft kein tiefes Expertenwissen notwendig ist. Sollte das einmal der Fall sein, so stehen dem Mitarbeiter ein Stab von Experten zur Verfügung. Mit Hilfe dieser Verbesserung konnten Durchlaufzeiten pro Antrag von ca. vier Stunden erreicht werden.

Der wesentliche Unterschied zum ersten Verbesserungsversuch besteht darin, nicht zu versuchen, die Defizite alter Abläufe durch Detailverbesserungen zu beheben, sondern den Gesamtprozeß im Auge zu behalten und sich zu fragen, wie dieser verbessert werden kann. Es geht also nicht darum, einzelne Teilaufgaben möglichst effizient zu erfüllen, sondern die effektivste Abwicklung eines Prozesses zu ermöglichen. Dabei wird keine Rücksicht auf bestehende Strukturen genommen (das Beispiel zeigt deutlich die Abkehr von der funktions- und abteilungsorientierten Sichtweise).

Meist wird argumentiert, daß die Einführung von Workflow-Management-Systemen zu einer solchen radikalen Verbesserung beitragen, da Geschäftsprozesse vor der Einführung neu überdacht werden. So schreibt z.B. Jablonski in [**?**]:

> Workflow-Management-Systeme erfordern die Überarbeitung (Reengineering) der für ein Unternehmen wesentlichen Prozesse. [**?**]

Oft werden allerdings einfach alte Strukturen und Abläufe übernommen und der Ablauf nur automatisiert (siehe Verbesserungsversuch 1). Es erfordert erheblichen Aufwand und Überwindung, sich von alten Strukturen zu lösen, gerade wenn diese in der Vergangenheit so erfolgreich waren, und dieser Aufwand wird teilweise gescheut. In solchen Fällen sind spätere Änderungen am Ablauf und der Organisationsstruktur sinnvoll. Auch Abläufe, die heute optimal gelöst scheinen, sind aufgrund des bereits angesprochenen schnellen Wandels in der Zukunft potentielle Kandidaten für Veränderungen. Nur dadurch, *zusammen* mit dem kreativen Einsatz der Informationstechnologie, läßt sich die Effizienz und Effektivität des Gesamtprozesses gewährleisten.

Man kann sich natürlich darüber streiten, ob wirklich die radikalen Änderungen notwendig sind, die Hammer und Champy fordern, oder ob sich Änderungen eher stetig und evolutionär vollziehen werden. Sicher ist aber, daß sich Änderungen einstellen werden.

1.3.3 Auswirkungen auf das Unternehmen

Welche Auswirkungen haben nun solche Änderungen, wie sie beim Business Reengineering auftreten? Zunächst muß festgehalten werden, daß sich sowohl die Organisation der Abläufe, als auch die des Unternehmens selbst stark verändern kann. Einzelne Arbeitsschritte können modifiziert werden oder auch ganz wegfallen, andere Arbeitsschritte kommen hinzu.

Außerdem verschieben sich Kompetenzen, Pflichten und Aufgaben innerhalb der Unternehmens. Im obigen Beispiel haben sich sogar die Fähigkeiten zumindest scheinbar verschoben, denn man

gesteht nun auch einem Nicht-Spezialisten die Fähigkeit zu, zumindest einfache Anträge selbständig zu bearbeiten. Änderungen der Organisationsstruktur ergeben sich auch in anderen Ansätzen als dem Business Reengineering. Zum Beispiel zielt das sogenannte *Lean Management* auf den Abbau von Hierarchie-Stufen innerhalb der Organisation ab, um diese zu „verschlanken", und somit kürzere Entscheidungswege zu erreichen. Hammer und Champy weisen zwar darauf hin, daß Business Reengineering nicht primär die Ausdünnung und Abflachung der Hierarchie zum Ziel hat, denn *alleine* eine neue Organisationsstruktur zusammen mit alten Abläufen könne keine wesentlichen Verbesserungen bringen, trotzdem sind schlankere Organisationen oft auch das Resultat einer an Geschäftsprozessen orientierten Ablauf- und Aufbauorganisation.

Im obigen Beispiel werden die Veränderungen an der Aufbauorganisation sehr deutlich. Mehrere Spezialisten wurden praktisch durch eine Person ersetzt, die nun die Verantwortung und Befugnisse dieser Mitarbeiter übernimmt. Einzelne Abteilungen werden umstrukturiert oder fallen ganz weg, dafür können andere Organisationsformen entstehen. Zum Beispiel sind Teams dann sinnvoll, wenn eine Aufgabe für eine Person zu schwer oder zu umfangreich wird (vgl. auch [**?**]: „Caseteam").

Interessant ist auch die Verschiebung der Sichtweise beim prozeßorientierten Vorgehen. Dominierte bisher die Aufbauorganisation, innerhalb derer sich die Abläufe abspielten, steht nun der Prozeß im Vordergrund. Das heißt, es passen sich nicht mehr die Abläufe an die Unternehmensstruktur an, sondern umgekehrt (Stichwort „structure follows process", vgl. auch [**?**]). Eine Änderung der Aufbauorganisation ist dadurch sehr viel wahrscheinlicher als bisher.

Änderungen bezüglich der Aufbau- und Ablauforganisation, die im Zuge des Business Reengineering oft anzutreffen sind, lassen sich wie folgt zusammenfassen:

- Zusammenfassung von Positionen; Bildung von Teams zur Bearbeitung von Aufgaben.
- Verschiebung von Kompetenzen und Pflichten; Stärkung der Kompetenzen und Verantwortung von vormals unterstellten Mitarbeitern.
- Überdenken von Fähigkeiten.
- Entstehung flacher Organisationsstrukturen.
- Umstrukturierung von Abläufen; dabei Ausrichtung an der „natürlichen" Reihenfolge der einzelnen Schritte eines Prozesses (Prozeßorientierung)
- Evtl. Entstehung mehrerer Prozeßvarianten („Ende der Standardisierung" [**?**])

1.4 Konsequenzen des Wandels für Workflow-Management-Systeme

Der vorige Abschnitt hat an einem sehr anschaulichen Beispiel gezeigt, welche Änderungen innerhalb eines Unternehmens sowohl an der Aufbau- als auch an der Ablauforganisation auftreten können. Die Aspekte der Ablauf- und Aufbauorganisation haben direkten Einfluß auf Workflow-Management-Systeme, also Computersysteme, die die Prozeßdurchführung koordinieren und überwachen.[5] Zum einen muß festgelegt werden, welche einzelnen Aktivitäten eines Prozesses in welcher Reihenfolge unter welchen Bedingungen durchzuführen sind, zum anderen

[5] Eine genaue Definition und nähere Betrachtung der Details von Workflow-Management-Systemen erfolgt in Abs. 2.2; an dieser Stelle sollte diese knappe Beschreibung für das Verständnis genügen.

muß bestimmt werden, wer diese Aktivitäten durchzuführen hat. Da sich beides ändern kann, müssen diese Systeme die entsprechenden Änderungen mitvollziehen können. Dabei müssen die Systeme dieselbe Flexibilität aufweisen, die heute auch von den Unternehmen gefordert wird, denn sie spiegeln deren Organisationsstrukturen und Abläufe wider. Diese Forderung nach Flexibilität wird durch den bereits beschriebenen raschen Wandel im heutigen Unternehmensumfeld und der Prozeßorientierung verstärkt. Ein System, das Änderungen nicht mitvollziehen kann, wird zum Hemmschuh und behindert somit die effiziente Arbeitsorganisation.

1.4.1 Formen des Wandels

Wenn Flexibilität von Workflow-Management-Systemen gefordert wird, um dem Wandel erfolgreich begegnen zu können, so müssen zunächst die verschiedenen Formen des innerbetrieblichen Wandels genannt werden, denn verschiedene Formen des Wandels bedürfen unterschiedlicher Unterstützung seitens des Workflow-Management-Systems.

- *Änderungen von Prozessen*: Prozesse ändern sich heute aus den in Abs. 1.2 genannten Gründen wesentlich rascher und auch unvorhersehbarer als früher. Ein Extrem bilden hierbei Prozesse mit einer nur schwach ausgeprägten Struktur und vielen möglichen Ausnahmesituationen, die sich im voraus oft nicht oder nur unvollständig spezifizieren lassen. Diese Prozesse sind so speziell, daß sich oft erst während ihrer Ausführung entscheidet, wie weiter zu verfahren ist. Aber auch bei ansich stark strukturierten Prozessen sind Änderungen im Zuge der Prozeßoptimierung häufiger als früher anzutreffen, und wie das Beispiel aus Abs. 4 zeigt, sind die Änderungen hierbei oft radikal.
- *Änderung der Organisationsstruktur*: Änderungen an Prozesse ziehen oft Änderungen in der Organisationsstruktur nach sich, denn nur eine „passende“ Organisationstruktur kann als ideale Ausführungsplattform für die effiziente Durchführung von Prozessen dienen. Allgemein sind die möglichen Änderungen an der Organisationsstruktur sehr vielschichtig, wie die Auflistung aus [**?**] zeigt:

  1. Änderung der Position eines Mitarbeiters innerhalb des Unternehmens (z.B. Einstellung, Entlassung, Beförderung, Änderung des Aufgabenbereiches eines Mitarbeiters, usw.) unter Beibehaltung der Organisationsform.
  2. Zusammenlegung/Aufteilung von Elementen der Organisationsstruktur. Hierunter fällt z.B. die Zusammenfassung von Stellen (wie im obigen Fallbeispiel des Business Reengineerings) oder das Aufteilen von Verantwortungen/Kompetenzen von vormals einer auf nun mehrere neue Stellen.
  3. Neuaufteilung von organisatorischen Elementen, z.B. die Neuverteilung von Verantwortungen/Kompetenzen auf verschiedene existierende Stellen.
  4. Reines Entfernen/Hinzufügen von Elementen, z.B. das Hinzufügen einer Stelle, die keinerlei Aufgaben oder Kompetenzen einer bereits existierenden Stelle übernimmt oder das ersatzlose Streichen einer Stelle.
  5. Austausch von Organisationseinheiten, d.h. die Durchführung eines Prozesses innerhalb einer anderen Organisation.

1.4.2 Bestehende Lösungsansätze

Lösungsansätze werden hier vor allem im Hinblick auf die Kopplung von Aufbau- und Ablauforganisation betrachtet, da diese der zentrale Punkt dieser Arbeit ist. Traditionell werden

Prozesse und Organisationsstruktur innerhalb eines Workflow-Management-Systems in verschiedenen Modellen definiert, da sie unterschiedliche Aspekte eines Unternehmens darstellen (siehe Abs. 2.2.2). Zwangsläufig muß ein Bindeglied zwischen diesen beiden Beschreibungen existieren, denn Aufgaben müssen schließlich von Mitarbeitern der Organisation durchgeführt werden. Dieses Bindeglied stellt die Zuweisung von Aufgaben zu Aufgabenträgern (Akteuren) dar, bzw. die jeweilige Zuweisungsregel, die diese Zuordnung beschreibt. Da Änderungen sowohl an der Ablauf- als auch an der Aufbauorganisation nicht vermieden werden können, sondern wie bereits mehrfach angedeutet immer häufiger werden, sollte eine möglichst starke Entkopplung dieser beiden Beschreibungen angestrebt werden. Durch die Entkopplung sollten sich Änderungen innerhalb der Organisationsstruktur nach Möglichkeit nicht auf die Prozeßbeschreibungen auswirken und umgekehrt. Wird diese Entkopplung nicht erreicht, so können bereits kleine Änderungen an der Organisationsstruktur einen großen Aufwand für die Anpassung der Prozeßbeschreibungen nachsichziehen, die die veränderten Elemente zunächst weiterhin referenzieren und somit nach der Veränderung nicht mehr konsistent mit der unterlagerten Organisationsstruktur sind. Man muß sich hier nur vor Augen halten, was passiert, wenn eine Umstrukturierung stattfindet, um z.B. einen bestimmten Prozeß durch eine geeignete Organisationsstruktur besser unterstützen und somit effizienter ausführen zu können. Von dieser Änderung sind i.a. sehr viele bestehende Workflows betroffen, die eigentlich wie bisher weiterlaufen sollten, aber nun evtl. keine geeigneten Akteure für die Ausführung einzelner Aktivitäten mehr finden, da z.B. referenzierte Rollen durch die Umstrukturierung weggefallen sind. Oft bleibt hier bisher nur die manuelle Überprüfung und ggf. Nachführung der Zuweisungen, was natürlich nicht als ideal bezeichnet werden kann.

Lösungsansätze für schwach strukturierte Prozesse Bisherige Systeme konzentrierten sich auf hochgradig strukturierte Prozesse. Durch den Einsatz von Systemen, die sogenannte Ad-hoc-Workflows, also Workflows für schwach strukturierte Prozesse mit starker Änderungswahrscheinlichkeit und/oder häufigen Ausnahmesituationen, unterstützen, wird versucht, auch bei solchen Prozessen eine Effizienzsteigerung zu erreichen. Die Unterstützung betrifft dabei hauptsächlich die Prozeßdefinition, die bei Ad-hoc-Workflows aufgrund deren Natur erst kurz vor deren Ausführung erfolgen kann. Möglichkeiten zur Umplanung in Ausnahmesituationen werden ebenfalls im Bereich der Ad-hoc-Workflows betrachtet. Der Zuweisung von Aufgaben zu Akteuren schenken diese Ansätze kaum Beachtung, d.h. die Zuweisung erfolgt meist manuell.

Ansätze zur Begegnung des Wandels innerhalb der Organisationsstruktur Interessant für die Kopplung von Aufbau- und Ablauforganisation unter dem Aspekt des Wandels der Organisationsstruktur sind vor allem die oben genannten Punkte 1,2,3 und 5. Punkt 4, also das reine Hinzufügen und Entfernen von Elementen der Organisationsstruktur, ist wie in [**?**] beschrieben wird relativ uninteressant. Bei neuen Elementen bestehen keine Referenzen aus Prozeßdefinitionen bzw. Zuweisungsregeln auf diese neuen Elemente, so daß auch keine Anpassung dieser Definitionen erforderlich werden könnte (dieser Fall ist nicht mit dem Aufteilen aus Punkt 2 zu verwechseln, da hier ein bereits existierendes Element auf mehrere neue verteilt wird). Beim reinen Entfernen eines Elements kann davon ausgegangen werden, daß dieses Element nicht mehr benötigt und somit auch nicht mehr aus Prozeßdefinitionen heraus referenziert wird (ansonsten handelt es sich nicht um ein reines Entfernen, sondern um eine Neuaufteilung nach Punkt 3).

Eine ausführliche Diskussion der bestehenden Ansätze würde den Rahmen dieses einführenden Kapitels sprengen und ist auch erst sinnvoll, wenn die dafür notwendigen Grundlagen erläutert

wurden. Daher erfolgt an dieser Stelle nur eine knappe, allgemeingehaltene Erläuterung der bestehenden Ansätze. Eine ausführliche Diskussion ist in Abs. 3.4 zu finden.

Der Änderung der Position eines Mitarbeiters innerhalb eines Unternehmens wurde in Workflow-Management-Systemen relativ früh begegnet, wohl auch weil es sich hierbei um eine sehr häufige Änderung handelt. Ein Workflow-Management-System, das Änderungen innerhalb dieses Punktes nicht unterstützt, ist in der Praxis kaum sinnvoll einzusetzen. Die Lösung für dieses Problem ist relativ einfach. Man abstrahiert vom konkreten Mitarbeiter und verwendet für die Zuweisung Rollen, die wiederum von konkreten Mitarbeitern eingenommen werden können. Ändert sich die Position des Mitarbeiters, so ändert sich die Zuordnung zu einer Rolle. Da sich die Rolle selbst nicht ändert und nur diese von der Prozeßdefinition referenziert wird, sind keine Änderungen an der Prozeßdefinition bzw. Zuweisungsregel notwendig. Somit kann diese Art der Änderung der Organisationsstruktur heute als gelöst betrachtet werden.

Anders sieht es dagegen bei den obigen Punkten 2,3 und 5 aus. Die bisherigen Ansätze bieten hier keine oder nur sehr unzureichende Unterstützung (siehe Abs. 3.4), d.h. Änderungen dieser Art führen meist unweigerlich zu Änderungen im Prozeßmodell , da referenzierte Elemente nicht mehr oder nur noch in veränderter Form im Modell der Organisationsstruktur vorhanden sind. Wenn überhaupt eine Unterstützung seitens des Workflow-Management-Systems für diese Fälle angeboten wird, so beschränkt sich diese auf die Möglichkeit, Änderungen durch Versionierung von Elementen der Organisationsstruktur und Versionierung der Zuweisungsregeln nachvollziehbar zu machen und den Workflows somit ein konsistentes Bild der Organisationsstruktur zu bieten, so daß weiterhin eine Aufgabenträgerzuweisung erfolgen kann. Dieser Ansatz ist aber, wie Abs. 3.4 noch zeigen wird, nicht für alle relevanten Änderungen am Modell der Organisation ohne größeren Aufwand anwendbar und bietet somit nur unzureichende Unterstützung.

Ein wesentlicher Punkt, warum die Punkten 2,3 und 5 zur Zeit nur unzureichend unterstützt werden können, sind die zu eingeschränkten Metamodelle, die die Workflow-Management-Systeme zur Modellierung der Aufbauorganisation bereitstellen (siehe Abs. 3.1). Meist sind diese Metamodelle nicht in der Lage, die verschiedenen Organisationsformen zu beschreiben, so daß die Flexibilität bereits durch diesen Punkt wesentlich eingeschränkt wird.

1.5 Aufgabenstellung und Zielsetzung dieser Arbeit

Wie dieses einleitende Kapitel gezeigt hat, ist eine wesentliche Anforderung an heutige Unternehmen die Wandlungsfähigkeit. Die Wandlungsfähigkeit steht auch im Mittelpunkt des Sonderforschungsbereichs 467, der die Zielsetzung hat, ein Gestaltungskonzept für wandlungsfähige Unternehmensstrukturen für die variantenreiche Serienproduktion zu erstellen. Das Teilprojekt C4 des SFB 467 beschäftigt sich hierbei mit der informationstechnischen Kommunikation und Kooperation und bildet gleichzeitig den Rahmen dieser Arbeit.

Kernpunkt dieser Arbeit ist die flexible Kopplung der Aufbau- und Ablauforganisation unter Verwendung eines wissensbasierten Ansatzes zur Modellierung der Aufbauorganisation. Die Aufbau- und Ablauforganisation dürfen nicht isoliert voneinander betrachtet werden. Eine Änderung in der Ablauforganisation hat oft auch gleichzeitig eine Änderung der Aufbauorganisation zur Folge. Trotzdem sind Ablauf- und Aufbauorganisation zwei unterschiedliche Aspekte eines Unternehmens, die innerhalb von Workflow-Management-Systemen in verschiedenen Modellen beschrieben werden. Es ist daher sinnvoll, eine Modellierung dieser beiden Aspekte anzustreben, die Änderungen an einem Aspekt zuläßt, ohne den anderen Aspekt direkt zu beeinflussen. Eine Entkopplung kann allerdings nur erreicht werden, wenn die Zuweisung von Aufgabenträgern

zu Aktivitäten entsprechend gestaltet wird, denn sonst ziehen Veränderungen an der Organisationsstruktur zwangsläufig Änderungen an Prozessen nach sich, die u.U. nicht mehr aktuelle Elemente der Organisationsstruktur referenzieren, und umgekehrt. Hierzu ist ein mächtigeres Organisationsmodell notwendig, als es bisher bei den aktuellen Workflow-Management-Systemen anzutreffen ist. Dieses Organisationsmodell muß genügend Informationen über die Organisation bereitstellen, um auf Änderungen entsprechend reagieren zu können.

Ziel dieser Arbeit ist es daher zu zeigen, in wie weit es möglich ist, mit einem wissensbasierten Ansatz eine Verbesserung bei der Entkopplung von Ablauf- und Aufbauorganisation in Workflow-Management-Systemen zu erreichen. Es werden sowohl die diesbezüglichen Defizite bisheriger Systeme aufgezeigt, als auch die Vorteile, die sich durch die wissensbasierte Modellierung der Aufbauorganisation ergeben. Es werden außerdem entsprechende Möglichkeiten der Zuweisung von Aufgaben zu Aufgabenträgern erarbeitet. Als Mittel zur Wissensrepräsentation dienen hierbei Konzeptgraphen, die eine grafische und sehr anschauliche Form der Wissensrepräsentation darstellen.

Die weitere Arbeit gliedert sich wie folgt: Zunächst werden im 2. Kapitel die Grundlagen für die weitere Arbeit beschrieben. Es erfolgt eine Einführung in die Grundlagen der Betriebsorganisation, sowie eine Beschreibung der Grundlagen der Workflow-Management-Systeme.

Nachdem die entsprechenden Grundlagen vermittelt wurden, erfolgt im 3. Kapitel eine Analyse bestehender Workflow-Management-System bezüglich ihrer Möglichkeiten zur Modellierung der Aufbauorganisation und der Kopplung von Ablauf- und Aufbauorganisation, d.h. der Zuweisung von Aufgaben zu Aufgabenträgern. Dabei werden nicht nur kommerzielle Systeme betrachtet, sondern auch diverse Forschungsansätze diskutiert. Ziel dieses Abschnitts ist es zum einen einen Überblick über den aktuellen Stand der Technik zu erhalten, und zum anderen die Defizite aktueller Ansätze herauszuarbeiten.

Bevor in Kapitel 5 der Entwurf einer Organisationsverwaltung mit Hilfe von Konzeptgraphen erfolgt, werden in Kapitel 4 zunächst die Grundlagen der Konzeptgraphen erläutert.

Das Kapitel 6 beschreibt die Implementierung des Enwurfes und beinhaltet eine Evaluation des vorgestellten Entwurfes.

Kapitel 2

Grundlagen

In diesem Kapitel werden Begriffe aus der Betriebswirtschaftslehre und aus dem Bereich der Workflow-Management-Systeme definiert und erläutert, die für die weitere Arbeit von Bedeutung sind. Damit bildet dieses Kapitel die Grundlage der weiteren Arbeit.

2.1 Betriebswirtschaftliche Grundlagen

Eine effektive und effiziente Aufgabenerfüllung ist nur dann möglich, „wenn Regeln existieren, die ein arbeitsteiliges Vorgehen und Zusammenwirken der Personen möglichst zielwirksam gestalten ... Die Objekte der organisatorischen Regelung sind Aufgaben, Menschen, Informationen und Sachmittel.“ [**?**].

Zwei grundlegende Begriffe in der Organisationslehre sind dabei die Aufbau- und Ablauforganisation, die in den folgenden Abschnitten näher erläutert werden, wobei sich diese Ausführungen an [**?**][**?**][**?**] orientieren.

2.1.1 Ablauforganisation

> Die Ablauforganisation betrifft die Regelung der Aufgabenerfüllungsprozesse eines Unternehmens in zeitlicher, räumliche, mengenmäßiger und sachlogischer Hinsicht. [**?**]

Typische Fragen der Ablauforganisation sind also:

- *Was* ist zu erledigen? Das heißt, was sind die zu erledigenden Aufgaben?
- *Wann* sind die Aufgaben zu erledigen? Welche Bedingungen müssen erfüllt sein, daß eine Aufgabe ausgeführt werden kann? In welcher Reihenfolge sind die Aufgaben zu erledigen?
- *Wo* sind die Aufgaben zu erledigen (räumliche Zuordnung)?
- *Womit* sind die Aufgaben zu erledigen? Welche Hilfsmittel (z.B. Maschinen) sind notwendig, um die Aufgabe zu erfüllen?
- Welche *Informationen* werden zur Erfüllung der Aufgabe benötigt? Entscheidende Punkte sind hier Informations- und Kommunikationssysteme.

Für diese Arbeit ist vor allem die Kopplung von Ablauf- und Aufbauorganisation entscheidend, d.h. die Zuweisung von Aufgabenträgern zu den jeweiligen Aufgaben (Aktivitäten). Für weiterführende Informationen zur Ablauforganisation wird daher auf die entsprechende Literatur (z.B.[**?**][**?**][**?**]) verwiesen.

2.1.2 Aufbauorganisation

Die Fragestellung für die Aufbauorganisation lautet: *Wer* hat bestimmte Aufgaben auszuführen?

> Die Aufbauorganisation befaßt sich mit der Gliederung von Unternehmungen in aufgabenteilige, funktionsfähige Teileinheiten sowie mit der Regelung von Kompetenz- und Unterstellungsverhältnissen. [**?**]

Entscheidend für eine effiziente Aufgabenerfüllung ist dabei die sinnvolle Gliederung der Unternehmens, d.h. die Bildung von Subsystemen (Organisationseinheiten) und die Koordination und Integration der entstandenen Einheiten. Wie bereits im einführenden Kapitel beschrieben wurde, erkannte bereits Adam Smith im 18-ten Jahrhundert die Vorteile der Arbeitsteilung. Seine artenmäßige Arbeitsteilung (Funktionsspezialisierung) bewirkte bereits eine wesentliche Effizienzsteigerung bei der Produktion (siehe auch Abs. 1.1). Bei der Überschreitung von Kapazitätsgrenzen ist außerdem eine mengenmäßige Arbeitsteilung sinnvoll. Diese Art der Trennung nach auszuführenden Tätigkeiten nennt man *horizontale Arbeitsteilung*. Neben der horizontalen Arbeitsteilung kann ebenfalls eine Unterscheidung von planenden und ausführenden Tätigkeiten erfolgen, was als *vertikale Arbeitsteilung* bezeichnet wird. F.W. Taylor beschrieb dieses Prinzip bereits im Jahre 1911 in *Principles of Scientific Management*.

2.1.2.1 Organisationseinheiten

Im folgenden werden verschiedenen Organisationseinheiten eingeführt (vgl. auch [**?**]), aus denen sich die Aufbauorganisation eines Unternehmens zusammensetzen kann. Bevor dies geschehen kann, ist es sinnvoll, eine Unterteilung der Organisationsstruktur in Primär- und Sekundärorganisation vorzunehmen, wie sie in [**?**] vorgeschlagen wird. Die Primärorganisation umfaßt alle Organisationseinheiten, die sich mit der Erfüllung von Dauer- und Routineaufgaben beschäftigen. Die Sekundärorganisation dagegen befaßt sich mit der Bearbeitung von komplexen, einmaligen Spezialaufgaben oder zumindest von Aufgaben, deren Wiederholungsgrad gering ist. Sie setzt sich daher zumeist aus zeitlich befristeten Teams zusammen (siehe unten).

In [**?**] wird die Bedeutung der Sekundärorganisation in einem Umfeld ständigen Wandels unterstrichen:

> Die Geltungsdauer organisatorischer Lösungen hat sich stark verkürzt, wodurch das Merkmal Dauerhaftigkeit zunehmend an Bedeutung verliert. Es geht demgemäß verstärkt darum, Wandel zu organisieren, d.h. die Organisation flexibel zu halten. Dauerhafte Strukturen werden in immer kürzeren Intervallen geändert und die Organisation auf Zeit erhält eine immer größere Bedeutung ... Für die Leiter und Mitglieder wechselnder Projekte wird die Veränderung des Bestehenden von einer befristeten Spezialaufgabe zur Daueraufgabe. [**?**]

Auch hier wird nochmals die Forderung der Flexibilität in der Aufbauorganisation hervorgehoben, die bereits in Kapitel 1 angeklungen ist.

Stellen Die Zusammenfassung einer Menge von Teilaufgaben führt zur Definition der elementaren Organisationeinheit, der Stelle. Die Zusammenfassung von Aufgaben orientiert sich dabei an der „qualitativen und quantitativen Kapazität einer Person“ und weist dadurch einen Personenbezug auf [**?**]. Meist wird hierbei von der konkreten Person abstrahiert, so daß eine Unabhängigkeit der Stellenbeschreibung von den konkreten Aufgabenträgern erreicht wird.[1]

Es können verschiedene Arten von Stellen unterschieden werden:

- *Instanzen* (*Leitungsstellen*): Stellen mit Führungsaufgaben werden als Instanz oder Leitungsstelle bezeichnet. Eine Instanz hat das Recht, den nachgelagerten Stellen Anweisungen zu erteilen (Weisungsbefugnis), und die Ausführung dieser Anweisungen zu kontrollieren, d.h. die nachgelagerten Stellen haben gegenüber der Instanz eine Rechenschaftspflicht.
- *Ausführungsstelle*: Im Gegensatz zur Instanz hat die Ausführungsstelle keine Leitungsaufgaben und ist nur mit den Kompetenzen ausgestattet, die zur Erfüllung der angewiesenen Aufgabe notwendig sind.
- *Stabsstelle*: Stabsstellen haben ebenfalls keine Weisungsbefugnis. Sie dienen der Entlastung der Leitungsstellen, indem sie diese bei deren Entscheidungs- und Kontrollaufgaben unterstützen.

Um dem Stelleninhaber die Erfüllung seiner Aufgaben zu ermöglichen, ist dieser mit den entsprechenden Befugnissen auszustatten. Das heißt, eine Stellenbeschreibung beinhaltet neben einer Menge von Aufgaben auch eine Menge von Kompetenzen und aufgrund der anerkannten Regel, Kompetenz und Verantwortung nicht voneinander zu trennen, ebenfalls die zugehörigen Rechenschaftspflichten. Man bezeichnet dies als das Kongruenzprinzip, d.h. Aufgabe, Kompetenz und Verantwortung sind deckungsgleich an die Stellen zu vergeben (keine Trennung von Aufgabe, Kompetenz und Verantwortung).

Abteilungen Die Zusammenfassung einer Leitungsinstanz und den geleiteten Ausführungsstellen führt zum Begriff der Abteilung. Eine Abteilung kann dabei einerseits wieder aus mehreren Unterabteilungen bestehen, andererseits können auch mehrere Abteilungen zu einer übergeordneten Abteilung zusammengefaßt werden.

Arbeitsgruppen Nach [**?**] ist eine Arbeitsgruppe eine Organisationseinheit, die zusammenhängende Aufgaben in Eigenverantwortung erfüllt und dazu über die notwendigen Entscheidungs- und Kontrollkompetenzen verfügt, die bisher auf übergeordneten Hierarchiestufen zu finden waren. Damit erhält eine Arbeitsgruppe einen gewissen Grad an (Teil-)Autonomie. Die Organisation innerhalb der Gruppe wird von dieser selbst bestimmt. Es erfolgt also eine Abkehr von der klassischen, strikt arbeitsteiligen Betrachtungsweise, hin zu einem höheren Maß an Selbstbestimmung.

(Projekt-)Teams Die Unterscheidung zwischen Teams und Arbeitsgruppen beruht auf der in [**?**] vorgeschlagenen Unterscheidung zwischen Primär- und Sekundärorganisation (siehe oben). Die Arbeitsgruppe befaßt sich mit Dauer- und Routine-Aufgaben, während sich das zeitlich befristete (Projekt-)Team den innovativen Spezialaufgaben widmet. Aufgrund der Komplexität

[1] In [**?**] werden auch die Möglichkeiten genannt, bei der Stellenbeschreibung auf die konkreten Fähigkeiten einer Person einzugehen (Stellenbildung ad personam) oder die zur Verfügung stehenden Sachmittel (z.B. PCs, Maschinen, usw.) zur Stellebeschreibung heranzuziehen (Stellenbeschreibung ad instrumentum).

der Spezialaufgaben konstituiert sich ein Team dabei meist aus einer Menge von Spezialisten (vgl. auch [**?**]: „Caseteam“), die eng miteinander zusammenarbeiten. Typischerweise besteht ein Team nur aus wenigen Mitgliedern, um den Abstimmungsaufwand gering zu halten. Ein Mitarbeiter kann dabei durchaus mehreren Teams angehören.

2.1.2.2 Grundmodelle der Aufbauorganisation

Das Leitungssystem einer Organisation definiert sich durch die Weisungsbefugnisse zwischen den einzelnen Stellen und dient der Koordination und Integration der einzelnen Organisationseinheiten. Dabei können verschiedene Grundformen unterschieden werden[2].

Einliniensystem Bei der traditionellen Organisationsform des Einliniensystems nimmt eine Stelle nur Weisungen von der direkt übergeordneten Instanz entgegen. Es erfolgt also keine Teilung der Weisungsbefugnis. Man nennt dies auch das Prinzip der Einheit der Auftragserteilung. Der dadurch entstehende Dienstweg ist in jedem Fall einzuhalten. Es bildet sich die klassische, pyramidenförmige Organisationsstruktur, wie sie in Abb. 2.1 zu sehen ist.

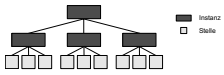

Abbildung 2.1: Einliniensystem

Mehrliniensystem Im Gegensatz zum Einliniensystem erfolgt beim Mehrliniensystem (Abb. 2.2) eine Teilung der Weisungsbefugnis, d.h. eine Stelle hat Weisungen von mehreren Instanzen entgegenzunehmen (das Prinzip der Einheit der Auftragserteilung wird hierdurch verletzt). Dieses System geht auf die sogenannten Funktionsmeister zurück, die Taylor anstatt eines einzelnen Meisters in der Fertigung einsetzte. Jeder dieser Funktionsmeister hatte aufgrund seines Fachwissens einen eigenen Zuständigkeitsbereich, wobei ein Arbeiter je nach durchzuführender Aufgaben einen anderen Vorgesetzten mit fachlicher Weisungsbefugnis hatte.

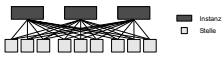

Abbildung 2.2: Mehrliniensystem

Funktionale Organisation Ein typisches Beispiel für eine Einlinienorganisation ist die funktionale Organisation, bei der die Gliederung des Unternehmens nach verrichtungsorientierten Gesichtspunkten erfolgt. Abb. 2.3 zeigt ein Beispiel für eine funktionale Organisationsstruktur.

Divisionale Organisation Bei der divisionalen Organisation handelt es sich wie bei der funktionalen Organisation um eine Einlinienorganisation. Hier erfolgt die Strukturierung allerdings nicht nach funktionalen Kriterien, sondern nach Objekten. Beispielsweise kann ein Unternehmen nach den hergestellten Produkten oder Produktgruppen organisiert werden, es ist aber ebenfalls

[2]Vor- und Nachteile der einzelnen Organisationsformen sind für diese Arbeit nicht relevant. Es wird daher auf die entsprechende Literatur (z.B. [**?**][**?**][**?**]) verwiesen.

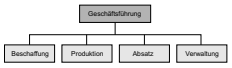

Abbildung 2.3: Funktionale Organisation

eine Organisation nach Regionen (z.B. Länder mit Niederlassungen) oder Kundengruppen (z.B. Privat-/Geschäftskunde) denkbar. Es bilden sich die vor allem für größere Unternehmen typischen Sparten, Divisionen oder Geschäftsbereiche. Abb. 2.4 zeigt die divisionale Organisation am Beispiel der Geschäftsbereiche der Bayer AG im Jahre 1994.

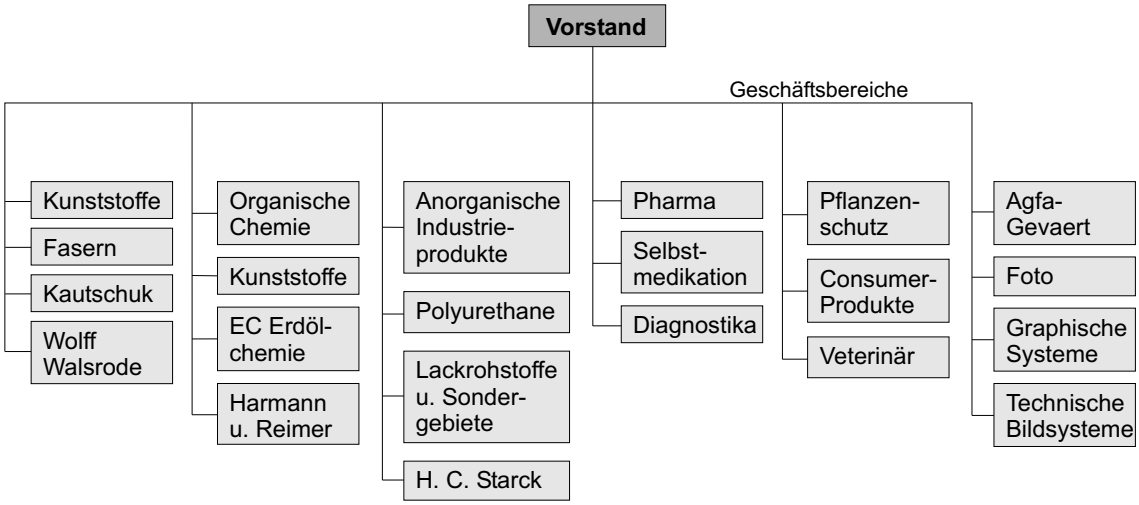

Abbildung 2.4: Divisionale Organisation nach [**?**]

Neben den einzelnen Sparten besitzen viele divisional organisierte Unternehmen noch sogenannte Zentralbereiche, die die Unternehmensleitung bei der Steuerung und die einzelnen Sparten bei ihren operativen Tätigkeiten unterstützen. Typische Zentralbereiche sind z.B. EDV, Marketing, Finanz- und Rechnungswesen, Recht, usw.

Stabliniensystem Beim Stabliniensystem wird versucht, einerseits die Einheit der Auftragserteilung beizubehalten, andererseits aber die Überlastung der Instanzen zu vermeiden. Dazu werden Stabsstellen gebildet, die die einzelnen Instanzen beraten und unterstützen und dadurch entlasten. Wie oben bereits erwähnt, haben die Stabsstellen selbst keine Weisungsbefugnis. Werden Stabsstellen mit Spezialisten besetzt, so können diese die Linieninstanz fachlich beraten. Ein Stabsgeneralist kann dagegen die Instanz in quantitativer Hinsicht unterstützen. Im Falle eines sehr hohen Unterstützungsbedarfes können auch ganze Stabsabteilungen eingerichtet werden, die ihrerseits wieder von einer Instanz geleitet werden (die Weisungsbefugnis des Stabsabteilungsleiters erstreckt sich allerdings nicht über die Stabsabteilung hinaus). Abb. 2.5 zeigt ein Stabliniensystem mit Stabsstellen und einer Stabsabteilung.

Liniensystem mit Querfunktionen Denkbar ist auch eine Organisationsform, bei der grundsätzlich das Liniensystem beibehalten wird, aber einzelne Funktionen in eigene Einheiten ausgelagert werden. Im Gegensatz zum Stabliniensystem erhalten diese Einheiten allerdings Weisungsbefugnis. So kann z.B. die Personalabteilung zusammen mit dem Linienvorgesetzten

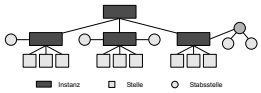

Abbildung 2.5: Stabliniensystem

einer Abteilung über die Einstellung eines neuen Mitarbeiters für die entsprechende Abteilung entscheiden.

Matrix- und Tensor-Organisation Überlagert man z.B. eine nach Verrichtungen organisierte Hierarchie mit einer objektorientierten Hierarchie (siehe Abb. 2.6), so erhält man eine Matrixorganisation, bei der sich die Weisungsbefugnisse in den Schnittpunkten der Matrix überlagern. Die Situation in den Schnittpunkten ist in Abb. 2.7 dargestellt („Matrixdiamant“ nach [**?**]). Zusätzlich zu den Instanzen der überlagerten Hierarchien wird eine übergeordnete Instanz („Matrixmanager“ nach [**?**]) eingesetzt, die ihrerseits die Instanzen der überlagerten Hierarchien leitet und somit die korrekte Funktion der Matrix garantiert. Aufgaben dieser übergeordneten Instanz sind z.B. die Durchsetzung von Unternehmenszielen in Konfliktsituationen, Versorgung mit Informationen und Ressourcen, usw. Die ausführende Stelle hat Weisungen von zwei Instanzen entgegenzunehmen, die Matrix-Organisation ist also ein Mehrliniensystem.

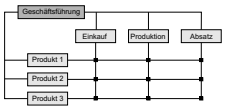

Abbildung 2.6: Matrixorganisation

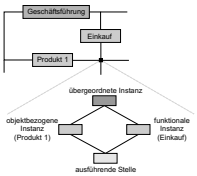

Abbildung 2.7: Matrix-Diamant nach [**?**]

Führt man die Überlagerung in drei oder mehr Dimensionen durch, so erhält man die Tensor-Organisation. So wäre es beispielsweise denkbar, neben der verrichtungs- und objektorientierten Strukturierung zusätzlich eine regionale Strukturierung vorzunehmen.

Projektorganisation Ein Projekt stellt eine zeitlich befristete Organisationsform dar und ist daher der Sekundärorganisation zuzurechnen. Allerdings existiert ein Projekt nicht im „freien Raum“, sondern muß muß mit der Primärorganisation in Verbindung gebracht werden. Aufgrund der großen Bedeutung, die Projekte in heutigen Unternehmen aus den bereits genannten Gründen haben, werden hier nun verschiedene Möglichkeiten dieser Kopplung dargestellt (vgl. auch [**?**]).

Reine Projektorganisation (Task Force) Bei der reinen Projektorganisation werden die Mitarbeiter für die Dauer des Projekts aus Stammorganisation herausgelöst und unterstehen nur dem Projektleiter. Der Projektleiter selbst ist wiederum einer Instanz der Stammorganisation unterstellt und verfügt über die Kompetenzen für alle Projektmitarbeiter und Ressourcen des Projekts. Durch die komplette Herauslösung aus der Stammorganisation entsteht eine fast autonome Parallelhierarchie (siehe Abb. 2.8).

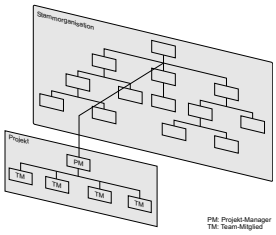

Abbildung 2.8: Reine Projektorganisation (Task Force)

Einfluß-Projektorganisation Bei der Einfluß-Projektorganisation (Abb. 2.9) hat der Projektleiter wesentlich geringere Kompetenzen, als bei der reinen Projektorganisation. Im Prinzip begleitet er eine Stabsstelle, die die Planungsaufgaben des Projekts wahrnimmt und die den Leitungsstellen der Projektmitarbeiter beratend zur Seite steht. Die Projektmitarbeiter werden nicht aus der Stammorganisation herausgelöst und bleiben dem Linienvorgesetzten unterstellt.

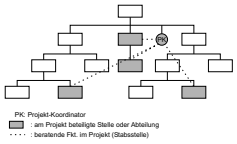

Abbildung 2.9: Einfluß-Projektorganisation

Matrix-Projektorganisation Hier gelten zunächst die Ausführungen, die bereits oben bei der Beschreibung der Matrix-Organisation gemacht wurden. Die zwei Hierarchien, die sich bei der Matrix-Projektorganisation wiederum in den Kreuzungspunkten der Matrix überlagern, sind zum einen die Hierarchie der Stammorganisation (gebildet z.B. nach funktionalen oder objektbezogenen Gesichtspunkten) und die temporäre Projektorganisation. Das heißt, ein Mitarbeiter untersteht zum einen seinem Linienvorgesetzten, zum anderen dem Projektleiter, wobei keiner die alleinige Weisungsbefugnis hat.

2.2 Workflow-Management-Systeme

Bereits im einführenden Kapitel ist die Bedeutung der Informationstechnologie für die Abwicklung von Geschäftsprozessen angeklungen. Der richtige Einsatz von Computersystemen ermöglicht sogar oft erst die Umstrukturierung dieser Prozesse im Zuge des Business-Reengineerings und ermöglicht so Effizienzsteigerungen um Größenordnungen. Workflow-Management-Systeme spielen dabei eine zentrale Rolle und sind ebenso zentrales Thema dieser Arbeit. Daher erfolgt an dieser Stelle eine Einführung in die Grundlagen der Workflow-Management-Systeme.

2.2.1 Definitionen

Die Informationstechnologie kann bei der Ausführung eines Geschäftsprozesses an vielen Stellen sinnvoll eingesetzt werden. Unter dem Stichwort *Computer Supported Cooperative Work* (CSCW), dem auch die Workflow-Management-Systeme zugeordnet werden können, lassen sich die Forschungsgebiete zusammenfassen, die sich mit der computerunterstützten Zusammenarbeit mehrere Personen und den damit verbundenen Technologien befassen. Ein Computersystem, das eine Gruppe unterstützt, welche auf ein bestimmtes Ziel hinarbeitet, wird dabei im allgemeinen als *Groupware* bezeichnet.[3] Obwohl Workflow-Management-Systeme heute allgemein nicht mehr direkt dem Begriff Groupware zugeordnet, sondern vielmehr als eigenständige Gattung von Anwendungen betrachtet werden, mag die Einordnung von Workflow-Management-Systemen in den Kontext der Groupware, wie sie in Abb. 2.10 zu sehen ist, hilfreich sein.

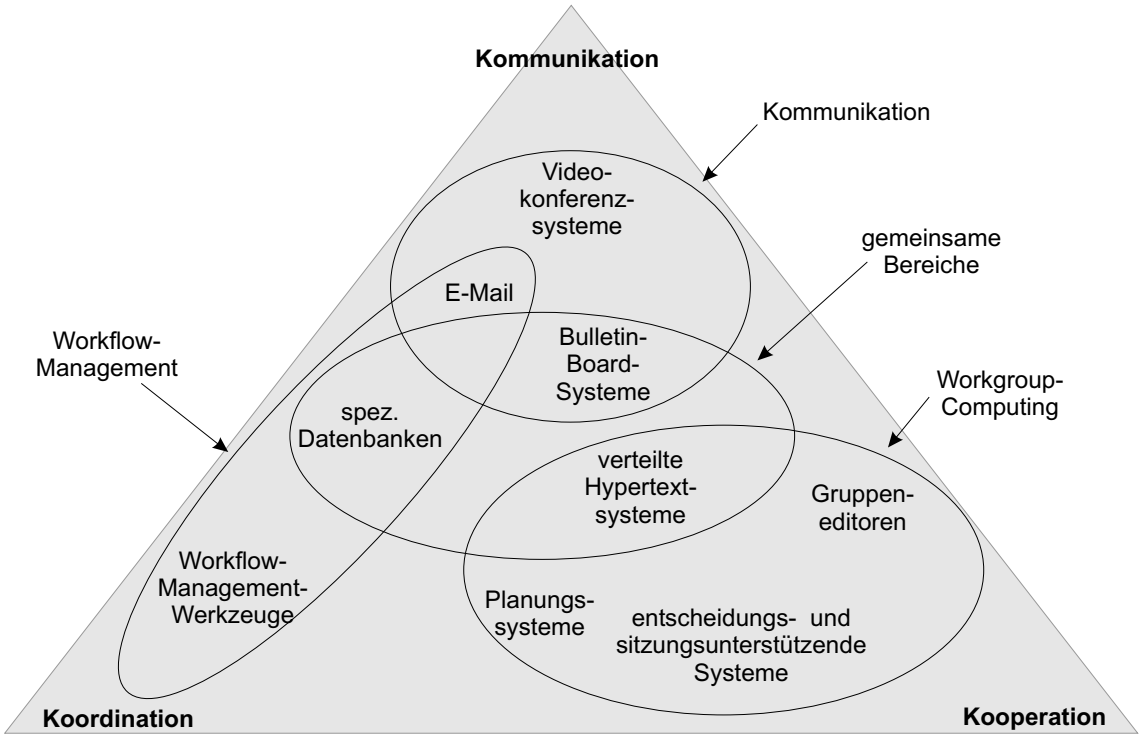

Abbildung 2.10: Klassifikation von Groupware nach [**?**]

Bevor nun definiert werden kann, was ein Workflow-Management-System tatsächlich ist, muß zunächst einmal geklärt werden, was ein Workflow ansich ist. Hier muß angemerkt werden, daß der Begriff Workflow teils sehr verschieden definiert wird. Oft wird er synonym zum Begriff Geschäftsprozeß verwendet. Andere verstehen darunter die Spezifikation eines Geschäftsprozesses, usw. (vgl. auch [**?**]). Die Workflow Management Coalition, eine internationale Organisation, bestehend aus Herstellern, Benutzern und Forschern im Bereich der Workflow-Management-Systeme, hat sich daher zum Ziel gesetzt, Standards in diesem Bereich zu etablieren, und somit eine einheitliche Terminologie zu schaffen.

Die Workflow Management Coalition definiert einen Workflow wie folgt:

[3] Der Begriff Groupware wird teilweise recht unterschiedlich definiert.

> The computerised facilitation or automation of a business process, in whole or part. [**?**]

Nach dieser Definition handelt es sich also um die vollständige oder teilweise Unterstützung oder Automatisierung eines Geschäftsprozesses durch ein Computersystem. Das hier angesprochene Computersystem führt uns direkt zur Definition eines Workflow-Management-Systems. Eine sehr ausführliche Definition liefert z.B. [**?**]:

> Workflow management software is a proactive computer system which manages the flow of work among participants, according to a defined procedure consisting of a number of tasks. It co-ordinates user and system participants, together with the appropriate data resources, which may be accessible directly by the system or off-line, to achieve defined objectives by set deadlines. The co-ordination involves passing tasks from participant to participant in correct sequence, ensuring that all fulfil their required contributions, taking default actions when necessary. [**?**]

Etwas kürzer ist die Definition des Workflow Management Coalition:

> [A Workflow management system is] a system that completely defines, manages and executes workflows through the execution of software whose order of execution is driven by a computer representation of the workflow logic. [**?**]

Sowohl die erste als auch die zweite Definition gehen auf die koordinierende und steuernde Funktion von Workflow-Management-Systemen ein. Das Workflow-Management-System überwacht und koordiniert die Ausführung der einzelnen (automatisierten) Aktivitäten eines Workflows und stellt so sicher, daß diese z.B. in der korrekten Reihenfolge ausgeführt werden und den ausführenden Stellen die richtigen Ressourcen zum richtigen Zeitpunkt zur Verfügung stehen.

Eine *Aktivität* ist dabei die Beschreibung eines logischen Schrittes innerhalb eines Prozesses[4], wobei zwischen manuellen und automatisierten Aktivitäten unterschieden wird. Im Gegensatz zu automatisierten Aktivitäten entziehen sich manuelle Aktivitäten der Kontrolle des Workflow-Management-Systems. Die Ausführung einer Aktivität führt im allgemeinen zur Zuweisung von Arbeitsaufgaben (engl. Work Items) an die meist menschlichen Bearbeiter, genannt *Akteure* (engl. Workflow Participant, Player, User, Agent), wobei einzelne Aufgaben auch komplett ohne menschliches Zutun alleine durch eine Applikation ausgeführt werden können. Die einzelnen Arbeitsaufgaben werden einem Akteur durch dessen *Arbeitsliste* (engl. Work List) zugänglich gemacht. [**?**]

2.2.2 Aspekte eines Workflow-Management-Systems

Stefan Jablonski erstellt in [**?**] ein allgemeines Workflow-Modell. Um dieses Modell in überschaubare, klar voneinander abgegrenzte Teile zu gliedern, und so eine Möglichkeit zur Modularisierung zu erhalten, führt er eine aspektorientierte Sichtweise auf Workflow-Management-Systeme ein. Dabei unterscheidet er sachliche Aspekte, die auf den Inhalt eines Workflows bezogen sind und technische Aspekte, die sich mit der softwaretechnischen Realisierung befassen.

Man beachte auch die Parallelen zu den Ausführungen in Abschnitt 2.1.1 und Abschnitt 2.1.2. Obwohl diese Abschnitte nicht speziell auf Workflow-Management-Systeme zugeschnitten sind,

[4] Die Definition eines (Geschäfts-)Prozesses erfolgte bereits auf Seite 4.

haben zumindest die sachlichen Aspekte von Jablonski Entsprechungen in den allgemeinen Ausführungen dieser Kapitel. Das ist wenig verwunderlich, denn die Fragestellungen sind beim Einsatz eines Workflow-Management-Systems zunächst dieselben (abgesehen von technischen Aspekten, die sich erst durch den Einsatz des Systems ergeben), wie bei der manuellen Bearbeitung eines Geschäftsprozesses. Trotzdem wird deutlich, wie sinnvoll eine Gliederung nach den angegebenen Aspekte ist.

Funktionaler Aspekt Der funktionale Aspekt beschreibt, *was* auszuführen ist. An dieser Stelle erfolgt die Definition der einzelnen Workflows. Workflows können dabei eine geschachtelten Struktur aufweisen, d.h. ein Workflow kann selbst wieder aus weiteren Workflows zusammengesetzt sein. Man unterscheidet daher *elementare*, d.h. nicht zusammengesetzte Workflows, die ausschließlich Applikationen referenzieren und *zusammengesetzte* Workflows. Anhand dieser Unterteilung können verschiedene Rollen von Workflows unterschieden werden. *Subworkflows* sind in *Superworkflows* enthalten; einem *Toplevel-Workflow* sind keine Superworkflows übergeordnet.

Operationaler Aspekt Der operationale Aspekt beschreibt, *wie* ein elementarer Workflow implementiert ist. Die eigentliche Implementierung erfolgt durch *Applikationen*. Das heißt, im Gegensatz zu zusammengesetzten Workflows, die der Strukturierung des Ablaufes dienen, bestimmen die Applikation die eigentliche Funktionalität.

Jablonski unterscheidet zwei Arten von Applikationen: *Programme* und *freie Applikationen*, wobei Programme nochmals in *adaptierbare Programme* und sogenannte *Legacy-Programme*[5] unterteilt werden. Adaptierbare Programme weisen konfigurierbare Schnittstellen auf, über die sie mit dem Workflow-Management-System kommunizieren können, wohingegen sich die Schnittstellen eines Legacy-Programmes nachträglich nicht verändern lassen. Bei freien Applikationen hat der Benutzer die Wahl, wie er eine Arbeit ausführt. Er kann dazu Software verwenden (z.B. ein Textverarbeitungsprogramm, eine Tabellenkalkulation, o.ä.) oder er kann die Arbeit auch vollständig manuell ausführen.

Die Implementierung der Applikationen wird heute oft durch sogenannte Middleware unterstützt.[6] Diese übernimmt folgende Aufgaben:

- Kommunikation verteilter Anwendungskomponenten, d.h. Kommunikation zwischen Workflow-Management-System und Applikation.
- Lokationstransparenz: Zugriff auf Anwendungskomponenten, ohne Wissen über deren Lokation im Netzwerk.
- Verteilte Transaktionen (siehe auch unten: Transaktionaler Aspekt)
- Sicherheit

Verhaltensbezogener Aspekt Die Fragestellung beim verhaltensbezogenen Aspekt lautet: *Wann* ist ein Workflow bzw. eine Applikation auszuführen? Um diesen Aspekt modellieren zu können, bietet ein Workflow-Management-System verschiedene Kontrollflußkonstrukte an. Mögliche Kontrollflußkonstrukte sind beispielsweise die einfach sequentielle Ausführung, bedingte

[5] deutsch Altlast, Hinterlassenschaft

[6] Strenggenommen handelt es sich bei der Middleware bereits um einen technischen Aspekt, der aber der Anschaulichkeit wegen bereits hier vorgestellt wird.

und unbedingte Verzweigungen, Schleifen, usw., wobei sich einzelne Workflow-Management-System natürlich in diesem Punkt unterscheiden können. Wichtig ist dabei nach Jablonski vor allem die übersichtliche und kompakte Beschreibung des Ablaufes.

Informationsbezogener Aspekt Dieser Aspekt befaßt sich mit dem Datenfluß zwischen den einzelnen Workflows. Zwei Arten von Daten können dabei unterschieden werden:

- *Kontrolldaten* werden durch das Workflow-Management-System verwaltet und definiert. Die Workflow Management Coalition definiert Kontrolldaten wie folgt:

  > Data that is managed by the Workflow Management System and/or a Workflow Engine. Such data is internal to the workflow management system and is not normally accessible to applications. [**?**]

  Das heißt, die Kontrolldaten stehen vollständig unter der Kontrolle des Workflow-Management-Systems und sind für Applikationen nicht zugreifbar.

- *Produktionsdaten* werden von den Applikationen verwaltet.

Allerdings sind auch Teile der Produktionsdaten für ein Workflow-Management-System von Relevanz (die sogenannten workflowrelevanten Produktionsdaten). Workflowrelevante Produktionsdaten zusammen mit den Kontrolldaten bilden daher die workflowrelevanten Daten.

Organisatorischer Aspekt Die bisherigen Aspekte betrafen vor allem die Ablauforganisation. Der organisatorische Aspekt – das zentrale Thema dieser Arbeit – dagegen betrifft die Aufbauorganisation, sowie die Zuordnung von Aufgabenträgern zu Aufgaben.

Bereits in Abs. 2.1.2.2 wurde deutliche, welche Vielfalt an Organisationsstrukturen in der Praxis existieren, wobei nur die Grundmodelle betrachtet wurden. Jablonski fordert daher zu Recht, daß sich die Modellierungsmöglichkeiten zur Definition der Aufbaustruktur eines Unternehmens, die ein Workflow-Management-System dem Benutzer anbietet, *nicht* an einem konkreten Modell orientieren dürfen, sondern eine möglichst freie, individuelle Definition des Organisationsstruktur möglich sein sollte (auch wenn die meisten verfügbaren Systeme diesen Grundsatz nicht beachten; siehe auch Abs. 3.1). Um eine möglichst freie Definition der Organisationsstruktur zu ermöglichen, schlägt Jablonski in [**?**] vor, lediglich die Modellierungskonstrukte *organisatorisches Objekt* und *organisatorische Beziehung* anzubieten, wobei sich die organisatorischen Objekte in *Agenten* und *Nicht-Agenten* unterteilen lassen. Ein Agent kann einerseits ein menschlicher Mitarbeiter oder auch ein „Stück“ Software (Server-Prozeß) sein, der bzw. das letztendlich eine Arbeitsaufgabe erfüllt, wobei die Summe aller Mitarbeiter die *Population* der Organisation darstellt. Nicht-Agenten sind dagegen z.B. Organisationseinheiten wie Stellen, Abteilungen oder auch Rollen, die ein Mitarbeiter einnehmen kann. Organisatorische Beziehungen sind Beziehungen zwischen organisatorischen Objekten. Typische Beispiel hierfür sind „(Mitarbeiter) *leitet* (Abteilung)“ oder „(Mitarbeiter) *gehört zu* (Organisationseinheit)“. Abb. 2.11 macht nochmals graphisch die Zusammenhänge zwischen den gerade eingeführten Begriffen deutlich.

Neben der Modellierung der Organisationsstruktur ist ein zweiter wichtiger Punkt des organisatorischen Aspekts die Definition von *Zuweisungsregeln* bzw. die Definition einer Zuordnungsstrategie. Das heißt, es muß festgelegt werden, welcher Aufgabenträger unter welchen Bedingungen eine anstehende Aufgabe zu erledigen hat. Werden mehrere passende Aufgabenträger gefunden, so muß ferner mit Hilfe einer sogenannten *Synchronisationsregel* bestimmt werden, welcher dieser Agenten die Aufgabe schließlich auszuführen hat, damit z.B. eine evtl. Mehrfachausführung

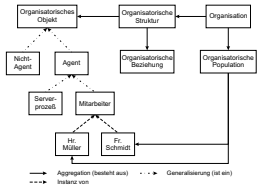

Abbildung 2.11: Beschreibungselemente der Aufbauorganisation nach [**?**]

verhindert wird, falls diese nicht erwünscht ist. Wurde eine Aufgabe einem Agenten zugewiesen, so muß dieser natürlich über die durchzuführende Aufgabe in Kenntnis gesetzt werden. Das kann über die oben beschriebenen Arbeitslisten, per E-Mail o.ä. geschehen.

Zusammenfassend lassen sich drei Hauptbestandteile der *Organisationsverwaltung* eines Workflow-Management-Systems identifizieren (vgl. auch [**?**]), die die oben beschriebenen Aufgaben übernehmen:

- *Organisationsstrukturverwaltung*
- *Zuweisungsregelverwaltung*
- *Synchronisationsregelverwaltung*

Kausaler Aspekt Der kausale Aspekt beschreibt, *warum* ein Workflow überhaupt in der definierten Art und Weise ausgeführt wird. Bestimmende Punkte sind z.B. bestehende Unternehmensvorschriften, die berücksichtigt werden müssen.

Historischer Aspekt Der historische Aspekt befaßt sich mit dem Sammeln und Mitprotokollieren von Daten, die während der Ausführung eines Workflows entstehen. Betroffen sind hiervon alle anderen bereits genannten Aspekte. So kann z.B. aufgezeichnet werden, welcher Akteur eine bestimmte Aktivität ausgeführt hat. Diese Daten können später z.B. dazu verwendet werden, um auf deren Grundlage weitere Zuweisungen durchzuführen, z.B. wenn verlangt wird, daß derselbe Mitarbeiter, der Aktivität A durchgeführt hat, auch Aktivität B durchzuführen hat.

Transaktionale Eigenschaft Die wesentlichen Eigenschaften einer Transaktion lassen sich wie folgt definieren[7]:

- *Atomizität*: Eine Transaktion wird vollständig, oder überhaupt nicht ausgeführt (Alles-oder-Nichts-Prinzip).
- *Konsistenz*: Eine Transaktion überführt das System von einem konsistenten Zustand in einen weiteren konsistenten Zustand.
- *Isolation*: Die Änderungen einer Transaktion werden für andere Transaktionen erst nach deren erfolgreichen Abschluß sichtbar.
- *Dauerhaftigkeit*: Die Änderungen einer erfolgreich beendeten Transaktion sind dauerhaft.

[7]Diese Eigenschaften werden nach den Anfangsbuchstaben der englischen Bezeichnungen **A**tomicity, **C**onsistency, **I**solation, **D**urability auch die ACID-Eigenschaften genannt.

Es liegt auf der Hand, daß diese Eigenschaften auch auf die Ausführung von Workflows anwendbar sind. Gerade die verteilte Ausführung von Workflows unter Einbeziehung verschiedener Applikationen machen den Einsatz eines Transaktionssubsystems erforderlich, falls z.B. sichergestellt werden soll, daß einzelne Operationen in konsistenter Art und Weise durchgeführt werden und deren Änderungen dauerhaft sein sollen.

2.2.3 Komponenten eines Workflow-Management-Systems

Abb. 2.12 zeigt einen groben Überblick über die Komponenten eines Workflow-Management-Systems, wie sie von der Workflow Management Coalition [**?**] vorgeschlagen werden. Es wird hierbei zwischen Komponenten der Modellierungsphase und Laufzeitkomponenten unterschieden.[8]

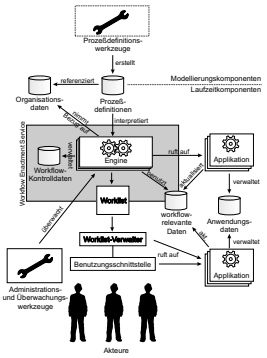

Abbildung 2.12: Komponenten eines Workflow-Management-Systems frei nach [**?**]

Komponenten der Modellierungsphase Ziel der Modellierungsphase ist es, eine formale, von der Workflow-Engine (siehe unten) interpretierbare Definition der einzelnen Prozesse zu erhalten. Hierzu müssen die Daten der einzelnen sachlichen Aspekte aus Abs. 2.2.2 definiert werden. Im Zuge des Business Reengineering geht dieser Definitions-Phase oft eine Analyse des Ist-Zustandes voraus, wobei spezielle Werkzeuge zum Einsatz kommen können, die allerdings nicht Teil des Workflow-Management-Systems sein müssen.

Bei der Definition der Prozesse bieten die meisten heute anzutreffenden Systeme umfangreiche Unterstützung durch graphische Editoren an. Mit Hilfe dieser Editoren läßt sich z.B. die Abfolge der einzelnen Aktivitäten und Strukturierung der Workflows in visueller Form darstellen, was wesentlich zur Anschaulichkeit beiträgt. Die Ausgabe der Prozeß-Definitionen ist allerdings oft in einem proprietären Format gehalten, das den Austausch zwischen unterschiedlichen Systemen verhindert. Ein Ziel der Workflow Management Coalition ist daher die Definition einer einheitlichen Schnittstelle, die die Interoperabilität in heterogenen Umgebungen sicherstellt.

Nur am Rande wird in [**?**] die Möglichkeit der Referenzierung von Elementen der Organisationsstruktur erwähnt (siehe auch Abs. 3.1.5). In Wirklichkeit ist diese Referenzierung mit einer der wichtigsten Punkte bei der Definition eines Prozesses, denn zu jeder Aktivität muß festgelegt werden, von welchem Aufgabenträger sie schließlich durchzuführen ist. Die Modellierung der Organisationsstruktur ist also wesentlicher Bestandteil eines Workflow-Management-Systems,

[8] Im Standard der Workflow Management Coalition werden diese zwei Phasen als *Build-Time* und *Runtime* bezeichnet.

und die angebotenen Modellierungskonstrukte bestimmen hierbei die Flexibilität des Systems innerhalb des organisatorischen Aspekts.

Komponenten der Laufzeit Zur Laufzeit werden aus den in der Modellierungsphase erstellten Prozeß-Definitionen Prozeß-Instanzen gebildet. Die zentrale Komponente ist hierbei die *Prozeß-Ausführungsumgebung* (engl. Workflow Enactment Service), die Prozeß-Instanzen erzeugt, verwaltet und ausführt. Die Prozeß-Ausführungsumgebung besteht aus einer oder mehren *Workflow Engines*. Die Engine ist für die eigentliche Kontrolle – erzeugen, aktivieren, unterbrechen und beenden – von Prozessen verantwortlich und verwaltet dazu eine Menge von Kontroll- und workflowrelevanten Daten, die den aktuellen Zustand einer Prozeßinstanz beschreiben.

Eine weitere Komponente von Workflow-Management-Systemen sind *Überwachungs-* und *Kontrollwerkzeuge*. Über sie bestehen Eingriffsmöglichkeiten z.B. in Ausnahmesituationen oder bei Zeitüberschreitungen. Denkbar sind auch Möglichkeiten zur manuellen Zuweisung von Aufgaben an Akteure bzw. der Änderung der Zuweisung oder auch Möglichkeiten zur Erstellung von Statistiken.

Die durchzuführenden Aktivitäten werden in der *Arbeitsliste* gespeichert, die von der Engine verwaltet wird. Die Interaktion zwischen Akteur und Arbeitsliste erfolgt über den *Arbeitslisten-Verwalter* (engl. Work List Handler), der zum einen dem Akteur die entsprechenden Aufgaben zur Verfügung stellt und diesen über anstehende Aufgaben informiert und zum anderen Rückmeldungen an die Engine erzeugt. Der Arbeitslisten-Verwalter selbst kann Teil einer *Client Application* sein, die neben den Aufgaben des Arbeitslisten-Verwalters weiteren Aufgaben wie z.B. dem An- und Abmelden von Akteuren am System dienen kann. Die *Benutzungsschnittstelle* ist dabei für die Präsentation der Aufgaben und die Entgegennahme von Benutzereingaben verantwortlich (Look&Feel).

Neben den Client Applications werden im Standard der Workflow Management Coalition die *Invoked Applications* definiert. Diese Anwendungen werden von der Workflow-Engine aufgerufen um einen Akteur bei der Erfüllung seiner Aufgabe zu unterstützen oder auch um eine Aufgabe vollständig automatisiert abzuarbeiten.

Kapitel 3

Analyse bestehender Ansätze zur Organisationsverwaltung

In diesem Kapitel werden verschiedene Ansätze zur Organisationsverwaltung analysiert, wie sie in aktuellen Workflow-Management-Systemen und verschiedenen Forschungsprototypen anzutreffen sind. Wie bereits in Abs. 2.2.2 erwähnt, lassen sich drei Hauptbestandteile der Organisationverwaltung identifizieren:

- Organisationsstrukturverwaltung
- Zuweisungsregelverwaltung
- Synchronisationsregelverwaltung

Die Analyse beschränkt sich hierbei aufgrund deren Bedeutung für die Aufgabenstellung dieser Arbeit auf die ersten beiden Punkte.

3.1 Ansätze zur Modellierung der Organisationsstruktur

Dieser Abschnitt stellt zunächst verschiedene Ansätze zur Modellierung der Organisationsstruktur innerhalb eines Workflow-Management-Systems vor. Drei kommerzielle Systeme stehen dabei stellvertretend für die große Menge kommerzieller Workflow-Management-Systeme, die schon alleine aus Platz- und Zeitgründen nicht alle vorgestellt und analysiert werden können. Die Modellierungskonstrukte der ausgewählten Kandidaten können allerdings als typisch für den aktuellen Stand der Technik gelten. Weitere Systeme werden beispielsweise in [**?**] und [**?**] vorgestellt. Unter den Forschungsansätzen kann sich vor allem der von Christoph Bußler aufgrund seiner hohen Flexibilität und Freiheit der Modellierung von den anderen Ansätzen abheben.

Viele der hier vorgestellen Systeme verwenden das Modellierungskonstrukt *Rolle*, das bisher nicht definiert wurde. Es ist daher zunächst angebracht klarzustellen, was innerhalb dieser Arbeit unter einer Rolle zu verstehen ist. Eine Auflistung verschiedener Definitionen des Rollenbegriffes bietet [**?**]). Diese Arbeit kommt zu dem Schluß, daß eine Rolle eine Menge von Fähigkeiten (Qualifikationen) und Rechten umfaßt. Zusätzlich ordnet [**?**] einer Rolle die Pflichten zu, die sich aus den Aufgaben ergeben, die ein Rolleninhaber übernommen hat. Diese Definition folgt dem in Abs. 2.1.2.1 erwähnten Kongruenzprinzip bei der Stellenbeschreibung, das fordert, Aufgabe, Kompetenz und Verantwortung nicht voneinander zu trennen. In diesem Sinne wird auch

der Rollenbegriff innerhalb dieser Arbeit als eine Menge von Fähigkeiten, Rechten und Rechenschaftspflichten definiert, die für die Durchführung einer Aufgabe notwendig sind, bzw. sich aus der Durchführung einer Aufgabe ergeben.

Der Vergleich der einzelnen Systeme erfolgt anhand des jeweiligen Metamodells der Aufbauorganisation. Diese Metamodelle werden jeweils mittels eines Entity-Relationship-Diagramms visualisiert. Leider gibt es mehrere verschiedene Varianten von Entity-Relationship-Diagrammen. Da die Darstellungen der Metamodelle in verschiedenen Notationen eher zu Verwirrung als zur Klärung beitragen und somit der Vergleich unnötig erschweren werden würde, werden die der Literatur entnommenen Diagramme wo nötig angepaßt, so daß durchgängig eine einheitliche Notation verwendet werden kann. Die verwendete Entity-Relationship-Variante weist zwei Besonderheiten auf. Eine Relation kann mit Kardinalitäten der Form „min:max“ versehen werden. Beispielsweise kann in Diagramm 3.2 ein Akteur Manager einer oder keiner Organisationseinheit sein (0:1) und eine Organisationseinheit enthält genau einen Akteur in der Funktion des Managers (1:1). Man beachte die Leserichtung! Die zweite Besonderheit ist die Möglichkeit, eine Generalisierungsbeziehung zwischen Entitäten zu kennzeichnen. Dazu wird ein Dreieck verwandt. Diagramm 3.1 zeigt hierfür ein Beispiel. Die Generalisierung der *Stammstelle* und *sonstiger Stellen* ist die allgemeine *Stelle*.

3.1.1 WorkParty

Bei WorkParty handelt es sich um ein kommerzielles Workflow-Management-System der Firma Nixdorf AG, dessen Modellierungskomponente für die Aufbauorganisation aus dem Programm ORMS (Organisations- und Ressourcen-Management-System) besteht, das näher in [**?**] beschrieben wird, und dessen Metamodell in Abb. 3.1 zu sehen ist.

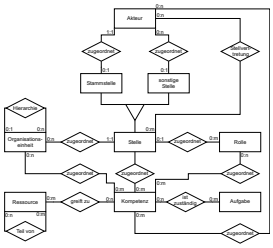

Abbildung 3.1: Organisationsstruktur-Metamodell von WorkParty nach [**?**]

WorkParty bietet zur Modellierung der Aufbauorganisation die folgenden Konstrukte an:

- *Mitarbeiter*: Ein Mitarbeiter ist gleichbedeutend mit einem Akteur. Ihm können eine oder mehrere Stellen zugeordnet werden, wobei ein Akteur genau eine Stammstelle besetzen muß. Ebenfalls vorgesehen ist eine Stellvertreterbeziehung, wobei ein Akteur Stellvertreter für mehrere Stellen sein kann.

- *Stelle*: Wie in der Definition des Stellenbegriffs aus Abs. 2.1.2.1, so ist auch der Stellenbegriff in WorkParty als eine Menge von Aufgaben zu verstehen, für die der Mitarbeiter zuständig ist, der die Stelle besetzt. Durch die Stelle wird also von der konkreten Person abstrahiert. Eine Stelle ist genau einer Organisationseinheit zugeordnet und kann einer Rolle im Sinne der Rollendefinition von WorkParty (siehe unten) zugeordnet werden.

- *Rolle*: Der Rollenbegriff, wie er von WorkParty verwendet wird, kann nach [**?**] am besten als *Stellentyp* umschrieben werden, d.h. die Rolle ist hier eine Zusammenfassung mehrerer Stellen mit ähnlichen Eigenschaften.

- *Organisationseinheit*: Eine Organisationseinheit ist eine Form der Stellenmehrheit, d.h. eine Zusammenfassung mehrerer Stellen oder weiterer, untergeordneter Organisationseinheiten. Dabei kann die Organisationseinheit sowohl dauerhaften Charakter haben, als auch in ihrer Existenz zeitlich begrenzt sein, wie das typischerweise bei einer Projektorganisation der Fall ist. Einer Organisationseinheit können Ressourcen (siehe unten) zugeordnet werden. Organisationseinheiten müssen bei WorkParty streng hierarchisch angeordnet werden, wodurch die Modellierung von Mehrliniensystemen wie z.B. der Matrixorganisation bereits unmöglich ist.

- *Kompetenz*: Eine Kompetenz beschreibt hier sowohl die Zuständigkeit für bestimmte Aufgaben, als auch die dazu notwendigen Befugnisse. Zuständigkeiten definieren sich also durch die Zuordnung von Aufgaben, Befugnisse durch den Zugriff auf Ressourcen. WorkPary definiert hierbei fünf Regeln, wie Kompetenzen „vererbt“ werden können (vgl. [**?**]):

  - Stellenregel: Akteure erben die Kompetenzen der ihnen zugeordneten Stellen.
  - Rollenregel: Stellen erben die Kompetenzen der zugeordneten Rollen (im Sinne der WorkParty-Definition).
  - Verteilerregel: Die Kompetenzen einer Organisationeinheit gehen auf deren Stellen über.
  - Leitungsregel: Die Instanz einer Organisationseinheit erbt deren Kompetenzen.
  - Hierarchieregel: Eine Organisationseinheit erbt die Kompetenzen der übergeordneten Organisationseinheit.

  Hierbei sind die ersten beiden Regeln immer gültig, die anderen optional.

- *Ressourcen*: Ressourcen sind die Sachmittel, die zur Erfüllung einer Aufgabe benötigt werden. Ressourcen haben einen eindeutigen Besitzer und können hierarchisch angeordnet werden, d.h. eine Ressource kann in einer anderen enthalten und selbst wieder einer weiteren untergeordnet sein.

3.1.2 FlowMark

FlowMark ist ein kommerzielles Workflow-Management-System der Firma IBM, dessen organisatorischer Aspekt in [**?**] beschrieben wird. Ebenfalls in dieser Quelle wird ein Metamodell dieses Aspekts erstellt, das in Abb. 3.2 zu sehen ist.

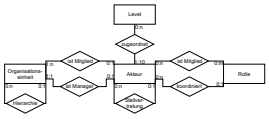

Abbildung 3.2: Organisationsstruktur-Metamodell von FlowMark nach [**?**]

Das Metamodell enthält die folgenden Modellierungskonstrukte:

- *Person*: Eine Person entspricht in FlowMark einem Akteur. Einer Person kann eine Organisationseinheit zugeordnet werden. Gehört eine Person einer Organisationeinheit an, so kann sie ein *Manager* dieser Abteilung sein und Rollen koordinieren. Eine Stellvertreter-Regelung ist möglich, wobei eine Person Stellvertreter mehrerer anderer Personen sein kann.

- *Organisationeinheit*: Einer Organisationeinheit können mehrere Akteure zugeordnet werden, von denen genau einer der Manager dieser Organisationeinheit sein muß. Organisationseinheiten können hierarchisch angeordnet werden, wobei wie bei WorkParty nur strenge Hierarchien möglich sind und somit auch hier nur Einliniensysteme modellierbar sind.

- *Rolle*: Eine Rolle ist in FlowMark gleichzusetzen mit einer Menge von Fähigkeiten, die ein Akteur hat, oder einer Funktion, die ein Akteur ausübt.

- *Ebenen*: Eine Besonderheit des FlowMark-Systems ist die Möglichkeit, Personen einer von zehn Ebene (engl. Level) zuzuordnen, wobei die Zuordnung nach einem beliebigen Kriterium erfolgen kann.

3.1.3 Leu

Ein weiteres Beispiel für ein kommerzielles Workflow-Management-System ist Leu von der Firma o.tel.o., dessen Metamodell der Aufbauorganisation in [**?**] und [**?**] beschrieben wird und in Abb. 3.3 dargestellt ist.

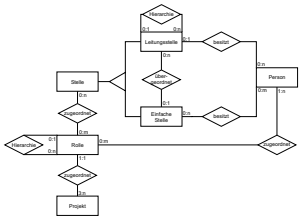

Abbildung 3.3: Organisationsstruktur-Metamodell von Leu nach [**?**]

Es folgt eine Erläuterung der einzelnen Modellierungskonstrukte:

- *Person*: Einer Person können in Leu eine oder mehrere Rollen zugeordnet werden. Sie kann außerdem mehrere Stellen besetzen, wobei keine Stellvertreter-Regelung vorgesehen ist.

- *Rolle*: Eine Rolle stellt in Leu eine Berechtigung zur Durchführung einer Aktivität oder den Zugriff auf Daten dar. Hierbei können Rollen sowohl Stellen als auch Personen zugeordnet werden. Eine Einschränkung der Rechte in Abhängigkeit des Objektwertes (z.B. eines Vertragswertes) ist möglich. Rollen werden in Leu in einer strengen Hierarchie angeordnet, wobei eine untergeordnete Rolle die Berechtigungen der übergeordneten Rolle erbt, diese Berechtigungen aber auch einschränken und ergänzen kann.

- *Stelle*: Die Stelle wird in ihrer herkömmlichen Bedeutung verwendet, wobei zwischen Instanzen und Ausführungsstellen ohne Weisungsbefugnis unterschieden wird. Instanzen können nur von einer Person besetzt werden, wohingegen normale Stellen von mehreren besetzt werden dürfen. Einer Stelle können Rollen zugewiesen werden, deren Berechtigungen auf den Stelleninhaber übergehen.

- *Projekt*: Eine Projekt ist in Leu nicht wie zu erwarten wäre eine Organisationseinheit, sondern eine Verwaltungseinheit „thematisch zusammenhängender Workflow-Modelle“[**?**]. Organisationseinheiten wie z.B. Abteilungen sind vorgesehen.

3.1.4 Diplomarbeit Nr. 1763 an der Universität Stuttgart

Die Diplomarbeit Nr. 1763 entstand wie diese Arbeit im Zuge desselben Teilprojekts des SFB 467 an der Universität Stuttgart. Thema dieser Arbeit ist die *Entwicklung eines flexiblen Organisationsmanagements für eine kooperative Ablaufsteuerung*. Hierbei wurde ein Metamodell der Organisationsstruktur erstellt, das in Abb. 3.4.

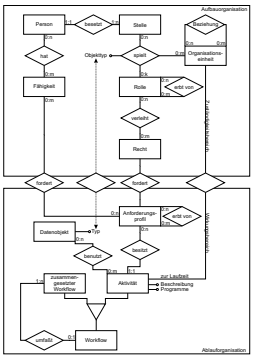

Abbildung 3.4: Organisationsstruktur-Metamodell Diplomarbeit Nr. 1763, Universität Stuttgart nach [**?**]

Folgende Modellierungskonstrukte wurden verwendet:

- *Person*: Eine Person hat Fähigkeiten, wobei ausdrücklich darauf hingewiesen wird, daß hier nicht nur die Fähigkeiten anzugeben sind, die eine Person für die Besetzung ihrer Stelle benötigt, sondern auch sonstige Fähigkeiten, die über das Stellenprofil hinausgehen.

- *Rolle*: Eine Rolle kann mehreren Stellen zugewiesen werden. Sie beinhaltet eine Menge von Rechten. Neu ist die mögliche Einschränkung der Rolle auf eine oder mehrere Organisationseinheiten. Dadurch wird erreicht, daß die Rechte eines Stelleninhabers nur für die spezifizierten Organisationseinheiten gültig sind. Außerdem erfolgt eine objektbezogene Rechtevergabe, d.h. Rechte können ebenfalls auf bestimmte Objekte beschränkt werden. Es besteht ferner die Möglichkeit, Rollen und somit die an diese gebundenen Rechte zu vererben.

- *Organisationseinheit*: Organisationseinheiten gruppieren mehrere Stellen. Sie können in einer Hierarchie angeordnet werden, wobei hierbei eine Organisationseinheit nicht nur *einer* weiteren unterstellt sein muß, sondern auch mehrere übergeordnete Einheiten haben kann. Dadurch sind auch Mehrliniensysteme wie die Matrixorganisation modellierbar.

- *Fähigkeit*: Einem Akteur werden explizit Fähigkeiten zugewiesen. Diese Möglichkeit ist bei den meisten kommerziellen Systemen nicht gegeben (vgl. die obigen Ausführungen). Der Nutzen dieser expliziten Fähigkeiten-Modellierung wird später bei der Besprechung der Zuweisung von Aufgaben an Aufgabenträger deutlich werden (siehe Abs. 3.3.5).

3.1.5 Workflow Management Coalition

Die Modellierung der Organisationsstruktur wird im Standard der Workflow Management Coalition nur am Rande erwähnt. Dieser Standard enthält auch die Abbildung 3.5, die als einziges Element der Aufbauorganisation die Rolle nennt. Die Rolle wird wie folgt definiert:

> A group of participants exhibiting a specific set of attributes, qualifications and/or skills. [**?**]

Diese Definition stimmt mit der oben aufgeführten überein. Das Modell einer Organisation wird als eine Menge von Organisationseinheiten und ihrer gegenseitigen Beziehungen beschrieben.

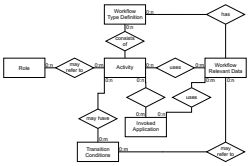

Abbildung 3.5: Prozeß-Definitions-Metamodell nach [**?**]

3.1.6 MOBILE (Forschung Universität Erlangen)

MOBILE ist eine Workflow-Management-System, das im Zuge der Forschungsarbeiten an der Universität Erlangen entwickelt wurde. Bemerkenswert ist dabei die saubere Strukturierung nach den einzelnen Aspekten aus Abs. 2.2.2, wie sie von Jablonski vorgeschlagen wurden. Die Organisationsstrukturverwaltung ist dabei innerhalb des organisatorischen Aspekts zu finden. Bereits Abb. 2.11 auf Seite 23 zeigt ein Metamodell der Organisationsstruktur. Als eine konsequente Fortführung dieses Modells kann das Metamodell gelten, das Christoph Bußler im Rahmen seiner Dissertation entworfen hat, und das in Abb. 3.6 zu sehen ist. Aus Gründen der Übersichtlichkeit fehlen in dieser Abbildung die folgenden Elemente, die jeder Entität aus Abb. 3.6 mehrfach zugeordnet werden können:

- Typattribut
- Instanzattribut
- Organisationstypoperation
- Organisationsinstanzoperation
- Organisationstypkonsistenzregel
- Organisationsinstanzkonsistenzregel

Auffällig ist das Fehlen organisatorischer Elemente wie Stelle, Rolle oder Organisationseinheit. Diese Elemente werden erst durch die Definition eines konkreten Organisationsstrukturtyps eingeführt. Bußler führt in [**?**] exemplarisch die Definition eines Strukturtyps *FlowMark* durch, der die Entitäten, Relationen und Attribute des FlowMark-Modells aus Abs. 3.1.2 nachbildet. Hier ein kleiner Ausschnitt dieser Definition:

Abbildung 3.6: Organisationsstruktur-Metamodell von MOBILE nach [**?**]

```
Organisationsstrukturtyp("FlowMark")
2  Agenttyp("Person", "FlowMark")
   Nichtagenttyp("Role", "FlowMark")
4  Organisationsbeziehungstyp("Related_role", "Person", "FlowMark",
                              "Role", "FlowMark")
```

Listing 3.1: Definition eines Organisationsstrukturtyps nach [**?**]

Definiert wird ein Agenttyp mit dem Namen `Person`, der dem Organisationsstrukturtyp `FlowMark` zugeordnet wird (Zeile 2), sowie eine Nicht-Agenttyp namens `Role`, der ebenfalls dem FlowMark-Strukturtyp zuzuordnen ist (Zeile 3). Diese beiden Typen werden durch die Relation `Related_Role` in Beziehung gesetzt. Dies entspricht den Begriffen und Beziehungen aus Abs. 3.1.2, in dem das FlowMark-Metaschema erläutert wurde.

Bereits durch dieses kurze Beispiel wird die Mächtigkeit dieses Ansatzes deutlich. Eine ausführliche Diskussion hierzu ist in Abs. 3.2 zu finden.

3.2 Diskussion der Ansätze zur Organisationsstrukturmodellierung

Bevor die verschiedenen Ansätze verglichen werden können, ist es sinnvoll, das Vergleichskriterium festzulegen. Bußler macht hierzu in [**?**] folgende Vorschläge:

- Geltungsbereich der Organisationstruktur: Teilen sich alle Workflows ein globale Organisationsstruktur, oder können Workflows verschiedene Sichten auf die Organisationsstruktur haben?
- Art der Organisationsstrukturverwaltung: Ausdrucksmächtigkeit, Erweiterbarkeit
- Dynamik der Organisationsstrukturverwaltung: Sind Änderungen zur Laufzeit möglich? Ist eine Möglichkeit zur Versionierung vorgesehen?
- Eigenständigkeit und Offenheit der Organisationsstrukturverwaltung: Sind Funktionen für den autorisierten Zugriff auf die Organisationsstruktur vorgesehen?

Besonders wichtig erscheint Punkt Nummer zwei, denn er hat unmittelbare Auswirkungen auf die Aufbauorganisation des Unternehmens. Dieser Punkt kann anhand der bereits vorgestellten Metamodelle näher beleuchtet werden, wobei das Modell der Workflow Management Coalition zu vage ist, um darauf eine Diskussion aufzubauen. Es wird daher nicht weiter betrachtet.

Alle vorgestellten kommerziellen Systeme beschränken sich auf streng hierarchische Organisationsformen. Sowohl WorkParty als auch FlowMark bieten nicht die Möglichkeit, eine Organisationseinheit mehreren anderen zu unterstellen. Leu läßt ebenfalls nur strenge Hierarchien

zwischen Leitungsstellen zu, verzichtet zugleich ganz auf die Möglichkeit, Stellen zu Organisationseinheiten zusammenzufassen. Somit sind diese Ansätze auf Einliniensysteme beschränkt. Eine Matrixorganisation ist z.B. nicht möglich. Das bedeutet bereits eine zumindest sehr starke Einschränkung, wenn nicht gar die Unbrauchbarkeit für die praktische Verwendbarkeit in Unternehmen mit nicht streng hierarchischen Organisationsformen.

In der Diplomarbeit Nr. 1763 wurde dieses Defizit erkannt und behoben. Organisationeinheiten können in nahezu beliebigen Beziehungen stehen. Trotzdem schränkt auch dieses Metamodell bereits zu stark ein. Ein Beispiel ist die Rechtevergabe. Rechte sind in diesem Modell an Stellen gebunden, die wiederum von Personen besetzt werden und somit auf diese übergehen. Ohne Zweifel ist dieses Vorgehen sinnvoll, denn die Stellenbeschreibung enthält, wie bereits in Abs. 2.1.2.1 beschrieben wurde, eine Menge von Rechten. Was aber, wenn bestimmte Rechte einer Person direkt zugeordnet werden sollen? Zum Beispiel ist es denkbar, einer Person aufgrund einer Vollmacht (Prokura) Rechte zuzusprechen. Außerdem kann einer Person, die keine Stelle innerhalb des Unternehmens begleitet, gar kein Recht zugesprochen werden. Man mag einwänden, daß eine solche Person nicht modelliert werden muß, da sie ja nicht Teil der Aufbauorganisation des Unternehmens ist. Trotzdem bleibt die Einschränkung, und wenn die Modellierung einer solchen Person und ihrer Rechte von Nutzen ist, dann hat sie auch ihre Berechtigung.

Betrachtet man die bisher besprochenen Modelle, so fällt auf, daß alle Einschränkungen bezüglich ihrer Ausdrucksmächtigkeit haben. Theoretisch läßt sich ein fehlendes Konstrukt scheinbar leicht ergänzen. Diese Ergänzung ist aber nur vom Entwickler des entsprechenden Workflow-Management-Systems durchzuführen, der Benutzer bzw. Modellierer muß später mit den Konstrukten leben, die ihm der Hersteller zugesteht. Die Ausdrucksmächtigkeit steht und fällt also mit den angebotenen Modellierungskonstrukten. Alleine die Menge an solchen Konstrukten ist dabei kein Garant für eine größere Allgemeingültigkeit. Man könnte theoretisch alle Konstrukte der bisher besprochenen Systeme anbieten und trotzdem blieben noch viele Formen der Aufbauorganisation übrig, die nicht modelliert werden könnten. Das liegt vor allem darin begründet, daß die verwendeten Konstrukte bereits zu sehr auf konkrete Organisationsformen zugeschnitten sind. Eine freie Definition ist somit nicht möglich. Bereits Bußler erkannte diesen Zusammenhang. Für ihn stellen die bisherigen Modelle keine richtigen Metamodelle dar, denn sie seien nicht anwendungsgebietneutral, da sie bereits auf Meta-Ebene Fachtermini festlegen würden und daher nicht allgemein verwendbar seien. Das Metamodell nach der Definition von Bußler, welches in Abs. 3.1.6 vorgestellt wurde, überkommt diesen Mangel, indem es nur wirkliche Meta-Begriffe zu Modellierung anbietet. Die Ausdrucksmächtigkeit wird dadurch demonstriert, daß andere „Metamodelle“ (z.B. das FlowMark-Modell) mit Hilfe des Bußler'schen Metamodelles nachgebildet werden. Diese „Metamodelle“ können daher als Ausprägung des Metamodelles nach Bußler gesehen werden. Damit könnte man das Bußler'sche Metamodell als ein Meta-Metamodell bezeichnen (diese Auffassung wird z.B. in [**?**] vertreten, wobei hier das Modell nach Jablonski aus Abb. 2.11 als Meta-Metamodell bezeichnet wird). Man kann die anderen Metamodelle aber auch als Spezialisierungen des Bußler'schen Metamodells ansehen, und wie das eben bei Spezialisten ist, taugen diese Metamodelle nicht für alle, sondern nur für einige wenige Fälle. Allgemein hat ein Modell nach der Modelltheorie von Stachowiak folgende Merkmale:

- *Abbildungsmerkmal*: Das Modell bildet ein Orginal ab (dieses kann fiktiv oder real sein).
- *Verkürzungsmerkmal*: Im allgemeinen unterdrückt ein Modell Attribute des Originals und betrachtet nur die für die Fragestellung relevanten Merkmale.
- *Pragmatisches Merkmal*: Ein Modell ist in der Lage, unter bestimmten Fragestellungen das Original zu ersetzen.

Die fehlenden Möglichkeiten der Metamodelle verhindern bereits im ersten Punkt die Abbildung wichtiger Eigenschaften des Unternehmens in ein Organisationsmodell. Davon ist auch das pragmatische Merkmal (Punkt 3) betroffen, denn wenn wichtige Informationen nicht im Modell vorhanden sind, kann das Modell nicht dafür verwendet werden, bestimmte Fragestellungen zu klären. Es ist daher wichtig zu erkennen, daß der Ansatz nach Bußler eine *freie* Definition der Organisationsstruktur zuläßt, mit der alle wichtigen Merkmale der Organisation abgebildet werden können, wohingegen die anderen Ansätze sich bereits durch die vorgegebenen Konstrukte nur für wenige Formen eignen.

An dieser Stelle soll auch noch erwähnt werden, daß auch Kritik an einem solchen freien Ansatz zu finden ist. Zum Beispiel wird in [**?**] folgendes zum Metamodell von Jablonski geschrieben:

> Diese Mächtigkeit ist gleichzeitig aber auch der Pferdefuß des Ansatzes bei der praktischen Umsetzung. Wie flexibel ein mit MOBILE beschriebenes Metamodell wird, ist nicht durch das Verfahren garantiert, sondern hängt allein davon ab, welche Gedanken zur Flexibilität sich der Modellierer gemacht hat. Es können natürlich flexible, dynamische Unternehmens- und Ablaufstrukturen mit diesen Mitteln gestaltet werden, aber ebensogut kann dabei ein traditioneller, ineffektiver und unbeweglicher Aufbau herauskommen. In der Praxis ist damit zu rechnen, daß das Metamodell auf der Grundlage der vorhandenen Strukturen entworfen wird und damit nur zur Modellierung des Status quo taugt, während die Flexibilität auf der Strecke bleibt. [**?**]

Diese Kritik ist natürlich zumindest teilweise berechtigt, denn ein flexibles Werkzeug zur Modellierung *alleine* reicht nicht aus. Es muß auch entsprechend eingesetzt werden. Die aktuellen Systeme schließen eine frei Definition aber von vornherein aus. Entscheidet sich ein Unternehmen für ein bestimmtes Workflow-Management-System, so muß es seine Organisation an das System anpassen, was natürlich gerade beim Business Reengineering der falsche Weg ist, denn hier gilt das Motto *Structure follows process* und nicht *Structure follows technology* (siehe auch Abs. 1.3.3). Das System hat sich an die Organisationsform anzupassen, die aufgrund des Prozesses die ideale ist und die eine effiziente Arbeitsweise ermöglicht.

3.3 Ansätze zur Zuweisung von Aufgabenträgern

Die Zuweisung von Aufgaben an Aufgabenträger erfolgt durch Referenzierung entsprechender Elemente der Organisationsstruktur, wobei grundsätzlich zwei Arten der Zuweisung unterschieden werden können:

- *Direkte* Zuweisung (Abb. 3.7a): Akteure[1] werden direkt benannt (z.B. „Mitarbeiter Müller und Schmidt können Aktivität XY ausführen.“).
- *Indirekte* Zuweisung (Abb. 3.7b): Es erfolgt keine direkte Benennung von Akteuren, sondern die Angabe von Kriterien, die Akteure zu erfüllen haben, um gewisse Aufgaben auszuführen. Zum Beispiel kann festgelegt werden, daß eine Aktivität von allen Sachbearbeitern einer bestimmten Abteilung bearbeitet werden kann, wobei die Auflösung hin zu den konkreten Mitarbeitern erst zur Laufzeit erfolgt, d.h. die entsprechenden Mitarbeiter werden aufgrund ihrer Zugehörigkeit zu einer bestimmten Organisationseinheit und Stelle bestimmt.

[1] Obwohl hier meist von menschlichen Aufgabenträgern ausgegangen wird, können die meisten Erkenntnisse auch auf nicht-menschliche Aufgabenträger (Server-Prozesse) übertragen werden

Prominentestes Beispiel für die indirekte Zuweisung, wie sie heute bei vielen Workflow-Management-Systemen anzutreffen ist, ist die rollenbasierte Zuweisung von Aufgabenträgern. Hierzu werden Mitarbeitern z.B. aufgrund von bestimmten Fähigkeiten und Kompetenzen Rollen zugeordnet. Rollen stellen dadurch eine Abstraktion vom konkreten Mitarbeitern dar. Einzelne Aktivitäten können von Mitarbeitern ausgeführt werden, die eine bestimmte Rolle einnehmen. Der Vorteil dieser indirekten Referenzierung liegt in der Unabhängigkeit der Aufgabenzuweisung von konkreten Akteuren. War es bei der direkten Zuweisung noch notwendig, Zuweisungsregeln zu ändern, sobald Akteure z.B. die Stelle wechselten oder Akteure hinzukamen bzw. das Unternehmen verlassen haben, so ist diese Änderung bei der indirekten Referenzierung nicht mehr notwendig, da z.B. die referenzierten Rollen von solchen Änderungen unberührt bleiben (was sich ändert ist die Zuordnung von Akteuren zu Rollen). Es wird dadurch ein gewisser Grad an Entkopplung der Ablauf- von der Aufbauorganisation erreicht, wobei später noch deutlich wird, daß diese Entkopplung noch nicht ausreicht und weiter verbessert werden kann.

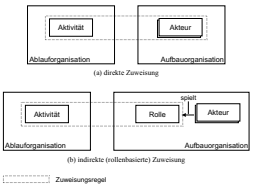

Abbildung 3.7: Direkte und indirekte Zuweisung von Aufgabenträgern

So gut wie kein System bietet heute alleine die direkte Zuweisung an. Hauptsächliche Unterschiede aktueller Systeme finden sich bei den Möglichkeiten zur Selektion einzelner Akteure, wobei diese Möglichkeiten bereits durch die Modellierungskonstrukte der Organisationsstrukturverwaltung beschränkt werden. Wird z.B. kein Modellierungskonstrukt *Stelle* angeboten, so können später auch nicht die Akteure selektiert werden, die eine bestimmte Stelle besetzen.

Die folgende Beschreibung der Workflow-Management-Systeme FlowMark und WorkParty stützt sich hauptsächlich auf die Beschreibungen in [**?**]. Diese Quelle enthält ebenfalls den Vergleich weiterer kommerzieller Systeme, wobei wiederum gelten kann, daß die hier beschriebenen Systeme typisch für die zur Zeit anzutreffenden kommerziellen Systeme sind.

3.3.1 FlowMark

Bei FlowMark wird zwischen spezifischen und dynamischen Zuweisungen unterschieden. Spezifische Zuweisungen beruhen auf einer Liste von Benutzerkennungen, denen Aktivitäten zur Laufzeit zugewiesen werden (direkte Zuweisung). Wurde einem Akteur ein Stellvertreter zugewiesen, so erhält dieser die Zuteilung. Die dynamische Zuweisung kann rollenbasiert erfolgen, wobei zusätzlich bestimmt werden kann, ob der Rolleninhaber oder sein Koordinator die Zuweisung erhalten soll. Außerdem kann die Zuweisung auf die Mitglieder einer Organisationseinheit eingeschränkt werden, wobei zusätzlich die Hierarchie der Organisationeinheiten zur Geltung kommen kann, d.h. es kann spezifiziert werden, daß alle Mitarbeiter von untergeordneten Organisationeinheiten ebenfalls für die Bearbeitung in Frage kommen. Ein weiteres Kriterium für die Zuweisung ist die Zugehörigkeit zu einem bestimmten Level, den ein Mitarbeiter bei FlowMark einnehmen kann (siehe Abs. 3.1.2). Ebenfalls möglich ist die Zuweisung einer Aktivität an den Initiator eines Workflows, sowie die Zuweisung an einen Akteur, der einen bereits beendeten Workflow bearbeitet hat, oder an dessen Manager.

3.3.2 WorkParty

Auch bei WorkParty ist die direkte Benennung von Akteuren möglich. Außerdem ist eine indirekte Referenzierung nach den folgenden Kriterien möglich:

- Organisationeinheit des Akteurs
- Stelle des Akteurs
- Stellentyp der Stelle des Akteurs (Rolle im Sinne von WorkParty; siehe Abs. 3.1.1)
- Kompetenz des Akteurs

3.3.3 Leu

Leu bietet die Möglichkeit der direkten Zuweisung von Aufgaben an Akteure. Ebenfalls möglich ist die rollenbasierte Zuweisung, wobei Rollen Berechtigungen zur Durchführung bestimmter Aktivitäten beinhalten. Wie in [**?**] beschrieben ist es hierbei ebenfalls möglich, diese Berechtigungen auf Objekte einzuschränken.

3.3.4 Workflow Management Coalition

Die Ausführungen innerhalb des Standards der Workflow Management Coalition bezüglich der Zuweisung von Aufgabenträger rechtfertigen eigentlich keinen eigenen Abschnitt. Trotzdem soll hier der Vollständigkeit halber erwähnt werden, daß das Referenz-Modell durch die Zuordnung von Rollen zu Aktivitäten (siehe Abb. 3.5) nahelegt, eine rollenbasierte Zuweisung durchzuführen.

3.3.5 Diplomarbeit Nr. 1763 an der Universität Stuttgart

Die Diplomarbeit Nr. 1763 beschäftigt sich recht ausführlich mit der Zuweisung von Aufgabenträgern zu Aktivitäten. Abb. 3.4 zeigt das dabei erstellte Referenzmodell. Einer Aktivität wird hierbei ein sogenanntes Anforderungsprofil zugeordnet. Ein Anforderungsprofil fordert dabei eine Menge von Rechten und Fähigkeiten, die zur Ausführung einer Aktivität notwendig sind. Die Rolle selbst verleiht eine Menge von Rechten, und eine Person hat eine Menge von Fähigkeiten.

Das wirklich Neue an dieser Arbeit ist die Erkenntnis der Anfälligkeit aktueller Systeme für Veränderungen bezüglich der Organisationsstruktur. Idealerweise sollten sich die Veränderungen der Organisationsstruktur nicht auf die Ablauforganisation auswirken und umgekehrt. Der Koppelpunkt von Ablauf- und Aufbauorganisation ist hierbei die Zuweisung von Akteuren zu Aktivitäten, d.h. die Zuweisung eines Elements aus der Aufbauorganisation (dem Akteur) zu einem Element aus der Ablauforganisation (der Aktivität). Die Art und Weise der Zuordnung bestimmt maßgeblich, wie stark sich jeweils Veränderungen der einen Seite auf die andere auswirken. Viele traditionelle Systeme vollziehen die Kopplung durch einfache Referenzierung von Elementen der Organisationsstruktur. Im schlechtesten Fall ist das die direkte Zuweisung von Akteuren. Jegliche Änderung am Mitarbeiterstamm oder jeder Stellenwechsel wirkt sich hierbei direkt auf die Zuordnung aus, d.h. jede Veränderung bleibt nicht auf die Aufbauorganisation beschränkt, sondern erfordert eine Anpassung der Zuordnung von Aktivitäten zu Akteuren. Dieses Problem wurde durch die indirekte Zuweisung z.B. durch rollenbasierte Zuweisungen zumindest zum Teil behoben, denn eine reine Änderung im Mitarbeiterstamm wirkt sich jetzt nur noch auf

die Aufbauorganisation aus, da nie ein Akteur direkt referenziert wird. Doch was geschieht bei Veränderungen der Rollen selbst?

Bei der Modellierung mit Hilfe der Rollen hatte der Entwerfer eine gewisse Menge von Rechten und Fähigkeiten im Hinterkopf, die er einer Rolle zuschreibt. Leider hat er dieses Wissen aber nicht *explizit* modelliert, sondern eben nur *implizit*.[2] Die Zuordnung einer Aktivität zu einer Rolle bedeutet eigentlich die Zuordnung einer Aktivität zu einer Menge von Fähigkeiten und Rechten, die die Aktivität fordert. Ändern sich diese Anforderungen, so müßte theoretisch entweder eine neue Rolle eingeführt werden oder zumindest eine Änderung der Zuordnung von Akteuren zu Rollen erfolgen, denn nach der Änderung wird die zugeordnete Rolle mehr oder weniger Rechte und/oder Fähigkeiten fordern, und nicht jeder Akteur, der vorher die Rolle aufgrund seiner Qualifikationen und Kompetenzen einnehmen konnte, wird dies auch nach der Änderung können. Das heißt, Änderungen der Ablauforganisation wirken sich wieder direkt auf die Aufbauorganisation aus. Umgekehrt können sich Änderungen innerhalb der Aufbauorganisation auch auf die Ablauforganisation auswirken. So macht z.B. eine Verschiebung von Kompetenzen innerhalb der Organisationsstruktur evtl. eine Änderung der Zuordnung von Aktivitäten zu Akteuren notwendig, da nun z.B. andere Rollen, Stellen, o.ä. für die Durchführung einer Aufgabe zuständig sind .

Diesem Defizit wird in [**?**] durch die oben beschriebene *explizite* Modellierung von einerseits angebotenen und andererseits geforderten Rechten und Fähigkeiten begegnet (siehe auch Abb. 3.8).

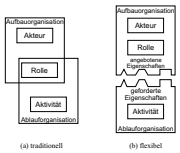

Abbildung 3.8: Kopplung von Aufbau- und Ablauforganisation nach [**?**]

So wirkt sich eine Änderung des Anforderungsprofils, d.h. der Menge der Fähigkeiten und Rechte, die für die Ausführung einer Aktivität benötigt werden, nicht mehr auf die Aufbauorganisation aus, denn das geänderte Anforderungsprofil führt nicht zu einer Änderung der Rechte und Fähigkeiten eines Akteurs. Andererseits führt die Änderung der Aufbauorganisation nicht mehr zu einer Änderung der Ablauforganisation, denn es werden von dieser keine Elemente der Aufbauorganisation referenziert und die geforderten Rechte und Fähigkeiten sind weiterhin vorhanden, wenn auch anders verteilt. Spribille führt darüber hinaus die Bindung von Rollen (und damit von Rechten) an Objekttypen und Organisationseinheiten ein, wobei diese Bindungen bei der Zuweisung ausgenutzt werden können.

3.3.6 MOBILE (Forschung Universität Erlangen)

Policy Resolution Christoph Bußler hat im Rahmen seiner Dissertation [**?**] eine Organisationsverwaltung für das Workflow-Management-System MOBILE, einem Forschungsprototypen der Universität Erlangen, entwickelt. Die Zuweisung von Aufgabenträgern zu Aktivitäten erfolgt hierbei durch *Policy Resolution*, wobei Policy Resolution zum Zwecke der Aufgabenträger-

[2] Teilweise sind die für eine Aktivität notwendigen Kompetenzen und Fähigkeiten z.B. in der Stellenbeschreibung festgehalten, aber eben nicht in einer Form, die das Workflow-Management-System auswerten und somit zur Zuweisung heranziehen kann.

Zuweisung in Workflow-Management-Systemen in [**?**] näher erläutert wird. Hierbei werden zunächst sogenannte Agenten-Profile spezifiziert, die bestimmte Akteure aus der Organisationsstruktur selektieren. Zum Beispiel könnte das Agenten-Profil `manager_of_employee(agent)` den Vorgesetzten eines Mitarbeiters selektieren. Die Policies verbinden dann Workflow-Spezifikationen mit den entsprechenden Agenten-Profilen, d.h. sie legen fest, welche Akteure (bestimmt durch die entsprechenden Agenten-Profile) unter welchen Bedingungen welche Aktivitäten durchführen können. Durch die Möglichkeit der Angabe von Bedingungen können Zuweisungen dabei auch z.B. in Abhängigkeit von workflowrelevanten Daten erfolgen. So kann z.B eine Aufgabe in Abhängigkeit vom Vertragswert verschiedenen Akteuren übertragen werden. Der folgende leicht gekürzte Ausschnitt aus [**?**] zeigt eine Policy, die die Aktivität `sign` eines Reisekostenantrags (**T**ravel **C**laim) in Abhängigkeit vom Betrag entweder dem Manager des Antragstellers (`TC.agent()` liefert den Initiator der Antrags, also den Antragsteller) oder einem höheren Vorgesetzten zuweist.

```
POLICY signing_of_travel_claim
    ENVIRONMENT TC.sign
        /* this policy applies for the sign
        workflow within TC */
    ASSIGNMENT
        if sign.amount() < 1000:
            manager_responsible_for(TC.agent());
        if sign.amount() >= 1000:
            VP();
```

Listing 3.2: Aufgabeträgerzuweisung mit Policy

Die bisher beschriebenen Möglichkeiten des Policy-Resolution-Ansatzes zur Zuweisung von Aufgabenträgern werden von Christoph Bußler in [**?**] weiter verfeinert (z.B. wird eine Möglichkeit zur Behandlung deontischer Konflikte eingeführt). Das oben beschriebene Prinzip bleibt allerdings erhalten, so daß auf weitere Ausführungen verzichtet wird.

Versionierung und Konfigurationsmanagement Im direkten Zusammenhang mit der Zuweisung von Aufgabenträgern steht der Versionierungsansatz aus [**?**]. Dieser Ansatz bezieht sich direkt auf das Problem der Inkonsistenzen, die sich innerhalb eines Workflow-Management-Systems ergeben können, sobald sich das Modell der Organisation ändert.

Dieser Ansatz versucht dem Problem durch eine Verfahren Herr zu werden, das auch in der Softwareentwicklung bereits erfolgreich angewandt wird, um Änderungen an Entwicklungsdokumenten auf eine kontrollierten Art und Weise durchführen zu können. Man bezeichnet dieses Verfahren in der Softwareentwicklung als *Konfigurationsmanagement*. Das Prinzip ist auch für Änderungen an der Organisationsstruktur dasselbe. Den Objekten der Organisation werden jeweils eigene Versionsnummern zugeordnet, die bei Veränderungen erhöht werden. Eine Version eines Objekts stellt somit einen bestimmten Zustand eines Objekts dar, wie er im Laufe der Zeit aufgetreten ist. Handelt es sich bei einem Objekt nicht um ein atomares Objekt, sondern ist das Objekt selbst aus mehreren anderen Objekten zusammengesetzt, so wird auch diesem zusammengesetzten Objekt eine Versionsnummer zugeordnet. Außerdem können für zusammengesetzte Objekte Konfigurationen gebildet werden. Eine Konfiguration besteht aus den enthaltenen Objekten in einer jeweils bestimmten Version. Zuweisungsregeln werden ebenfalls versioniert. Da die gesamte Organisationsstruktur nichts weiter als ein zusammengesetztes Objekt ist, das z.B. aus diversen Organisationseinheiten u.ä. besteht, kann auch eine Konfiguration der gesamten Organisation gebildet werden. Änderungen an einem Objekt werden durch Betrachtung der

Versionenhistorie eines Objekts ersichtlich. Dadurch werden vorgenommene Änderungen nachvollziehbar. Ein Workflow findet immer eine bestimmte Konfiguration der Organisationsstruktur vor, wobei diese Konfiguration natürlich zum entsprechenden Workflow passen sollte, so daß eine Aufgabenträgerzuweisung auch tatsächlich möglich ist. Inwieweit dieser Ansatz für die verschiedenen möglichen Änderungen der Organisationsstruktur anwendbar ist bzw. zu welchem Aufwand er führt, wird in Abs. 3.4 gezeigt.

3.4 Diskussion der Ansätze zur Zuweisung von Aufgabenträgern

Eine Diskussion der vorgestellten Ansätze erfolgt im Hinblick auf die als relevant erachteten Änderungen an der Organisationsstruktur, wie sie in Abs. 1.4.1 unter dem Punkt *Änderung der Organisationsstruktur* aufgelistet sind. Hierbei sind vor allem die Punkte 1,2,3 und 5 zu berücksichtigen, da Punkt 4 aus den bereits genannten Gründen als unkritisch für die flexible Kopplung von Aufbau- und Ablauforganisation gelten kann. Ziel ist es zu zeigen, in wie weit die bestehenden Ansätze in der Lage sind, Änderungen an der Organisationsstruktur zuzulassen, *ohne* das Prozeßmodell zu beeinflussen bzw. Anpassungen daran nötig zu machen. Das Prozeßmodell muß auf irgendeine Art Elemente der Organisationsstruktur referenzieren. Es wird hier davon ausgegangen, daß das durch Zuweisungsregeln geschieht. Erfordert eine Änderung an der Organisationsstruktur eine Anpassung an den Zuweisungsregeln, die die geänderten Elemente enthalten, so bedeutet das einen hohen Aufwand, da u.U. sehr viele Zuweisungsregeln angepaßt werden müssen. Kann diese Anpassung nicht automatisch erfolgen, so wirken sich Änderungen der Organisationsstruktur sehr stark auf die Prozesse aus, was natürlich vermieden werden soll. Im Idealfall sind gar keine Anpassungen – außer natürlich die Änderungen an der Organisationsstruktur selbst – notwendig.

3.4.1 Vergleich

Punkt 1: Änderung der Position eines Mitarbeiters innerhalb des Unternehmens Wie bereits in Abs. 1.4.2 angedeutet, ist dieser Punkt einer der wenigen, die bereits als ausreichend gelöst gelten dürfen. Durch Verzicht auf die direkte Zuweisung von Akteuren wirken sich Änderungen dieses Punktes nicht auf das Prozeßmodell bzw. die Zuweisungsregeln aus. Jedes der hier vorgestellten Workflow-Management-Systeme bietet die Möglichkeit der indirekten, rollenbasierten Zuweisung, die vom konkreten Akteur abstrahiert und die – solange die Rolle unverändert bleibt – eine Zuweisung unabhängig vom Rolleninhaber zuläßt.

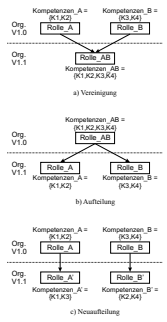

Abbildung 3.9: Formen von Änderungen der Organisationsstruktur

Punkt 2a: Zusammenlegung von Elementen der Organisationsstruktur Abb. 3.9a zeigt eine Zusammenlegung von Rollen. Die vormals getrennten Rollen A und B mit den *impliziten* enthaltenen Kompetenzen 1 und 2 bzw. 3 und 4 werden zu einer neuen Rolle AB mit den Kompetenzen 1,2,3 und 4 vereinigt. Die neue Organisationsstruktur enthält somit keine Akteure mehr, die die Rollen A oder B einnehmen. Der Versionierungsansatz würde hierzu jeweils neue Versionen der geänderten Elemente erstellen, so daß auch eine neue Version der gesamten Organisation entsteht. Um die Anpassung aller vorhandener Zuweisungsregeln mit den Rollen A und B zu vermeiden, kann eine „Umleitung“ aller Anfragen erfolgen, die sich auf die Rollen A oder B beziehen, so daß in diesen Fällen immer automatisch die Rolle AB referenziert wird. Dies ist möglich, da in der Versionenhistorie ein eindeutiger Pfad von Rolle A zu AB und von Rolle B zu AB existiert. Somit ist klar, daß ein Objekt, daß die Rolle AB einnimmt, ein Objekt der Rolle A und B ersetzen kann. Somit ist dieser Fall unkritisch.

Punkt 2b: Aufteilung von Elementen der Organisationsstruktur Dieser Fall ist die Umkehrung von 2a. Aus Rolle AB werden Rollen A und B erzeugt, die jeweils einen Teil der vormals in AB enthaltenen Kompetenzen enthalten (siehe Abb. 3.9b). In diesem Fall bietet der Versionierungsansatz keine Hilfe für die automatische Handhabung, denn es kann nicht gesagt werden, ob eine vormalige Anfrage nach einem Objekt der Rolle AB nun auf A oder B umgeleitet werden soll – zumindest nicht ohne weitere Informationen. Es bleibt in diesem Fall nichts anderes übrig, als alle Zuweisungen manuell zu überprüfen.

Das Problem liegt hier in der *impliziten* Modellierung der mit den Rollen verbundenen Kompetenzen. Dieses Problem wird vom Ansatz aus Abs. 3.3.5 gelöst, indem die zur Durchführung einer Aufgabe notwendigen Rechte und auch Fähigkeiten *explizit* modelliert werden. Eine Zuweisung enthält nun nicht mehr eine Selektion der Form „Finde alle Akteure, die Rolle A einnehmen können.“, sondern „Finde alle Akteure mit Kompetenz 1 und 2.“. Wie Abb. 3.9b zeigt, sind diese Kompetenzen auch nach den Änderung vorhanden. Es ist daher keine Anpassung der Zuweisungen notwendig.

Punkt 3: Neuaufteilung von organisatorischen Elementen Dieser Fall ist in Abb. 3.9c zu sehen. Es kommen hier keine Rollen hinzu oder fallen weg. Da die meisten Metamodelle sich im Großen und Ganzen auf Rollen konzentrieren, ändert sich für das Workflow-Management-System die Organisation scheinbar nicht (Kompetenzen können aufgrund der eingeschränkten Metamodelle ja nicht explizit modelliert werden). Prozesse können somit weiterhin die Elemente referenzieren. Leider wird die Zuweisung trotzdem nicht mehr korrekt sein, denn die gefundenen Akteure haben nun evtl. nicht mehr die für die Durchführung einer Aktivität geforderte Kompetenzen. Dieser Fall ist noch wesentlich kritischer als Fall 2b, denn obwohl die Zuweisung nicht offensichtlich als falsch erkannt wird, weil keine Akteure mehr gefunden werden, liefert sie stillschweigend falsche Ergebnisse.

Der einzige Ansatz, der diesen Fall richtig und ohne großen Aufwand behandelt, ist wieder der Ansatz aus Abs. 3.3.5. Hier gelten wieder die Ausführungen von Punkt 2b.

Punkt 5: Austausch von Organisationseinheiten Die extremste Form des Austausches ganzer Organisationseinheiten ist die Durchführung eines Prozesses in einer anderen Organisation. Auch massive Umstrukturierungen ohne Rücksicht auf alte Elemente führen zu einer praktisch neuen Organisation, aber auch die Verwendung eines bewährten Prozesses in einem anders strukturierten Unternehmensteil fällt hierunter.

Versionierung u.ä. bringt hier keine Vorteile mehr, denn es existieren keine Elemente in alten Versionen, aus denen die neuen hervorgegangen sind. Der Ansatz aus Abs. 3.3.5 kann nur dann sinnvoll eingesetzt werden, wenn die neue Organisation Rechte und Fähigkeiten gleich bezeichnet oder zumindest eine 1:1-Abbildung verschiedener Bezeichner möglich ist. Ansonsten müssen alle Zuweisungen angepaßt werden. Hieraus ergibt sich die Notwendigkeit der Verwendung wohldefinierter Begriffe für die Modellierung der Organisation, die später noch unter dem Begriff der Ontologie eine große Rolle spielen wird.

3.4.2 Bewertung

Bewertet man die verschiedenen Ansätze, so ist ein deutlicher Anstieg der „Immunität“ der Prozeßbeschreibungen gegenüber Änderungen an der Organisationsstruktur in folgender Reihe zu erkennen: Namensreferenzierung → Rollenreferenzierung → Selektion aufgrund von Eigenschaften eines Akteurs. Da Name und Rolle eines Akteurs im Grunde nichts anderes als Eigenschaften sind, subsumiert der dritte Punkt die beiden anderen. Trotzdem muß klargestellt werden, daß eine Selektion von Aufgabenträgern nach Namen und Rollen nicht zu empfehlen ist. Nur die Selektion nach Eigenschaften wie Rechten, Fähigkeit, o.ä. bringt die gewünschte Entkopplung von Ablauf- und Aufbauorganisation. Die Selektion muß also aufgrund von grundlegenden Eigenschaften erfolgen, von denen sichergestellt werden kann, daß sie auch nach Veränderungen an der Organisation noch im Modell anzutreffen sind. Man muß sich dazu fragen, welche Eigenschaften tatsächlich notwendig sind, um eine Arbeit auszuführen. Solche Eigenschaften sind beispielsweise Befugnisse und Fähigkeiten aber auch noch andere wie z.B. der aktuelle Aufenthaltsort oder allgemein die Verfügbarkeit eines Akteurs.

Man könnte nun einwänden, daß diese Modellierung von Eigenschaften einen sehr großen Aufwand bedeutet, nicht zuletzt weil das Modell immer auf dem aktuellen Stand gehalten werden muß. Das ist sicher korrekt. Aber dieser Aufwand muß auch bisher betrieben werden. Schließlich muß für jede Rolle die Menge der Eigenschaften festgelegt werden, die ein Akteur mitbringen muß, um die Rolle einnehmen zu können. Der einzige Unterschied besteht darin, daß diese Eigenschaften bisher oft in einer Form festgehalten werden, die keine direkte Auswertung durch ein Workflow-Management-System zuläßt.

3.5 Zusammenfassung

Es ist angebracht, an dieser Stelle ein Zusammenfassung der Ergebnisse der Analyse des Status quo zu formulieren.

Es wurde festgestellt, daß aktuelle Systeme vor allem im Bereich der Modellierungsmöglichkeiten der Aufbauorganisation wesentliche Defizite aufweisen. Diese Defizite sind auf die angebotenen Modellierungskonstrukte der Systeme zurückzuführen, die oft bereits auf bestimmte Organisationsformen zugeschnitten sind und daher zu stark einschränken. Eine flexible Definition der Aufbauorganisation ist mit solchen Systemen kaum möglich. Eine Lösung dieses Mankos bietet die Bereitstellung eines Metamodells, dessen Konstrukte eine freie Definition zulassen. Dazu müssen diese Konstrukte allgemein genug gefaßt werden.

Die Definition der Aufgabenträgerzuweisung leidet ebenfalls unter den Modellierungsmöglichkeiten aktueller Systeme, denn es können nur Elemente der Organisationsstruktur referenziert werden, die sich mit den Mitteln des Systems beschreiben lassen. Es wurde ebenfalls erkannt, daß für eine Entkopplung von Aufbau- und Ablauforganisation die explizite Modellierung von

Kompetenzen und Fähigkeiten unverzichtbar ist, da ansonsten Änderungen am Modell der Organisation direkt zu Änderungen an den Prozeßbeschreibungen führen. Außerdem muß einer Organisation eine sauber definierte Terminologie zugrundeliegen, auf deren Basis die Modellierung der Organisation erfolgen kann. Nur diese Terminologie gewährleistet die Verwendung von Prozeßbeschreibungen innerhalb verschiedener Organisationsformen.

Kapitel 4

Konzeptgraphen

Das vorige Kapitel stellte zwei grundsätzliche Forderungen auf: Zum einen muß es möglich sein, die Organisationsstruktur möglichst frei zu definieren. Ohne eine solche Möglichkeit wird es in vielen Fällen nicht möglich sein, die für die Durchführung eines Prozesses ideale Organisationsform zu wählen. Zum anderen ist eine *explizite* Modellierung von Kompetenzen und Fähigkeiten oder allgemein Eigenschaften eines Akteurs notwendig, um über diese Eigenschaften die Zuweisung von Aufgabenträgern zu Aktivitäten durchführen zu können, und damit eine weitgehende Entkopplung von Ablauf- und Aufbauorganisation zu erreichen. Für das Beschreibungsmittel der Aufbauorganisation bedeutet dies, daß es sowohl die notwendige Flexibilität und Allgemeingültigkeit, als auch die Möglichkeit zur Modellierung der Eigenschaften eines Akteurs bieten muß. Ein Beschreibungsmittel, das beide Eigenschaften erfüllt und zudem noch eine sehr anschauliche graphische Modellierung der Aufbauorganisation zuläßt, sind die sogenannten Konzeptgraphen, deren Grundlagen zunächst in diesem Kapitel erläutert werden sollen, bevor im nächsten Kapitel der eigentliche Entwurf der Organisationsverwaltung mit Hilfe von Konzeptgraphen erfolgt.

4.1 Einführung und Notationen

Konzeptgraphen wurden 1984 von John F. Sowa in [**?**] eingeführt. Sie stützen sich auf philosophische, psychologische und linguistische Grundlagen, sowie auf Erkenntnisse aus dem Bereich der künstlichen Intelligenz.

Sowa führt Konzeptgraphen auf der Grundlage des menschlichen Wahrnehmungsprozesses ein. Dieser mehrstufige Prozeß führt über die sensorische Wahrnehmung der Umwelt und den Vergleich dieser Eindrücke mit bereits bekannten Eindrücken zu einem Modell der Umwelt in Form eines Konzeptgraphen. Dabei kann ein Konzeptgraph nicht nur gegenständliche Dinge, sondern auch abstrakte Konzepte, wie z.B. den Begriff der Freundschaft o.ä., beschreiben.

Die Ausdrucksmächtigkeit von Konzeptgraphen wird in dieser Arbeit zur wissensbasierten Modellierung der Aufbauorganisation genutzt. Wie in [**?**] beschrieben, resultiert der Wissensbegriff auf der Unterscheidung zwischen unstrukturierten Daten und Wissen. Daten werden durch einen mehrstufigen Prozeß in Wissen transformiert. Zunächst erfolgt eine Selektion von relevanten Daten, die dann in ein einheitliches Format transformiert werden. Durch die Interpretation dieser Daten entsteht das Wissen. Innerhalb dieser Arbeit ist dabei das Wissen über den organisatorischen Aufbau eines Unternehmens von Relevanz, insbesondere das Wissen über die Beziehungen zwischen einzelnen Elementen der Organisation und das Wissen über bestimmte Eigenschaften potentieller Aufgabenträger (Akteure). Zur Repräsentation dieses Wissens werden Konzeptgraphen eingesetzt.

Konzeptgraphen stellen dabei einen Ausschnitt aus einem semantischen Netz dar. Semantische Netze werden dazu verwendet, um Wissen eine strukturierte Form zu geben. Ein semantisches Netz besteht aus Knoten, die semantische Einheiten repräsentieren, und Kanten, die Relationen zwischen diesen Einheiten darstellen. Setzt man die Konzepte eines Konzeptgraphen zu anderen Konzepten, zu Sinneseindrücken, zu Emotionen, usw. in Beziehung, so erhält man ein semantisches Netz.

Aus mathematischer Sicht sind Konzeptgraphen endliche, bipartite und zusammenhängende Graphen, bestehend aus zwei Arten von Knoten: *Konzeptknoten* und *Relationsknoten*. Konzepte und Relationen können über gerichtete Kanten miteinander verbunden sein. Kanten zwischen zwei Konzeptknoten bzw. zwischen zwei Relationsknoten sind nicht zulässig.

Bei der *graphischen Darstellung*[1] eines Konzeptgraphen werden Konzeptknoten als Rechtecke, Relationsknoten als Kreise oder Ellipsen dargestellt. Abb. 4.1 zeigt einen einfachen Konzeptgraphen bestehend aus zwei Konzeptknoten und einer binären Relation. Allgemein sind n-stellige Relationen möglich, wobei die Kanten mit Zahlen beschriftet werden, um die Argumente der Relation zu unterscheiden. Nach [**?**] hat eine n-stellige Relation genau eine ausgehende und $n-1$ eingehende Kanten. Dabei müssen alle eingehenden Kanten eindeutige Nummern aus $\{1,\ldots,n-1\}$ haben. Die ausgehende Kante hat dann die Nummer n.[2] Isolierte Relationen sind nicht möglich, d.h. eine Relation muß mit mindestens einem Konzept verbunden sein, ein Konzeptgraph kann aber aus nur einem Konzeptknoten bestehen. Eine formale Definition eines Konzeptgraphen erfolgt erst in Abs. 4.3, wenn die dafür notwendigen Grundlagen erläutert wurden.

Abbildung 4.1: Elemente eines Konzeptgraphen

Neben der grafischen Darstellung gibt es eine weitere Notation für Konzeptgraphen, die als *lineare Notation* bezeichnet wird. Diese Notation eignet sich besonders für die Eingabe per Tastatur, ist aber bei komplexen Graphen relativ unübersichtlich, besonders wenn diese Zyklen enthalten (siehe unten). In der linearen Notation werden Konzepte durch eckige Klammern und Relationen durch runde Klammern gekennzeichnet. Die gerichteten Kanten werden auch in der linearen Notation durch Pfeile ausgedrückt, die durch Zahlen numeriert werden können, falls z.B. mehrere eingehende Kanten vorhanden sind und daher die Argumente unterschieden werden müssen. Der Graph aus Abb. 4.1 würde in der linearen Notation wie folgt dargestellt, wobei die Kantennumerierung in diesem Fall nicht zwingend notwendig wäre:

[Konzept_A]−1−>(Relation)−2−>[Konzept_B].

Weitere Einzelheiten zur linearen Notation werden an entsprechender Stelle gegeben.

4.2 Konzepte und Relationen

Die Konzeptknoten eines Konzeptgraphen repräsentieren Entitäten, Attribute, Ereignisse, o.ä., Relationen veranschaulichen die Beziehungen zwischen Konzepten. Somit läßt sich z.B. die Aus-

[1] Sowa bezeichnet die grafische Darstellung als „display form“.

[2] Im Prinzip sind die Kantenrichtungen somit überflüssig. Bei binären Relationen werden sie trotzdem oft verwendet, um Kante 1 als eingehende und Kante 2 als ausgehende Kante zu kennzeichnen.

sage „Ein Rad ist Teil eines Autos“ durch den in Abb. 4.2 dargestellten Konzeptgraphen darstellen.

Abbildung 4.2: Konzeptgraph „Ein Rad ist Teil eines Autos.“

Man kann einen Konzeptgraphen `[Konzept_A]`→`(RELATION)`→`[Konzept_B]` lesen, indem man den Satz bildet „Der/die/das *RELATION* eines/einer *Konzept_A* ist ein *Konzept_B*“. Evtl. sind leichte grammatikalische Anpassungen notwendig, insbesondere wenn Relationen nicht durch Substantive, sondern z.B. mit einem Verb bezeichnet werden. Trotzdem wird der Sinn eines Konzeptgraphen durch diese Transformation klar. Das Beispiel aus Abb. 4.2 würde z.B. in den Satz „Das Teil eines Autos ist ein Rad“ übersetzt.

Dieses einfache Beispiel zeigt bereits die leichte Verständlichkeit zumindest einfacher Konzeptgraphen. Oft ist es relativ einfach möglich, einen Konzeptgraphen in einen natürlichsprachlichen Satz umzusetzen, oder umgekehrt aus einem natürlichsprachlichen Satz einen Konzeptgraphen zu bilden. So läßt sich z.B. der Satz „Herr Müller fährt mit seinem Auto nach Stuttgart.“ durch den Konzeptgraphen in Abb. 4.3 darstellen.

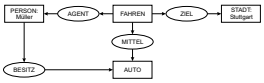

Abbildung 4.3: Konzeptgraph „Herr Müller fährt mit seinem Auto nach Stuttgart.“

4.2.1 Kanonische Konzeptgraphen

Nicht alle Kombinationen von Konzepten und Relationen machen wirklich Sinn. Ein Konzeptgraph kann auch völlig sinnlose Zusammenhänge beschreiben. In Anlehnung an ein Beispiel aus [**?**] könnte z.B. folgende Aussage durch einen Konzeptgraphen formuliert werden:

[SCHLAFEN]→(AGENT)→[IDEE]→(FARBE)→[GRÜN].

Natürlich macht die Aussage „Eine grüne Idee schläft.“ keinen Sinn. Um solche sinnlosen Aussagen von sinnvollen zu unterscheiden, wurde der Begriff der *kanonischen Konzeptgraphen* eingeführt. Kanonische Konzeptgraphen repräsentieren reale oder zumindest sinnvolle Situationen. Es gibt dabei drei grundsätzliche Möglichkeiten, einen kanonischen Konzeptgraphen zu bilden:

- *Wahrnehmung*: Konzeptgraphen, die aufgrund der Wahrnehmung der externen Welt entstehen, sind kanonisch.
- *Kanonische Formationregeln*: Kanonische Konzeptgraphen lassen sich durch Anwendung einer Menge von Regeln aus anderen kanonischen Konzeptgraphen herleiten (siehe Abs. 4.5).
- *Erkenntnis*: Durch einen Erkenntnisprozeß können beliebige Konzeptgraphen für kanonisch erklärt werden.

Bei der Modellierung der Aufbauorganisation wird eine Menge von kanonischen Konzeptgraphen durch den menschlichen Modellierer vorgegeben. Diese Konzeptgraphen entstehen aus dem Bild

des Modellierers, das sich dieser von der Aufbauorganisation bildet. Die erstellten Konzeptgraphen bilden eine kanonische Basis.

4.2.2 Konzeptarten

Abb. 4.2 und Abb. 4.3 zeigen zwei grundsätzliche Arten von Konzepten: *Generische* und *individuelle* Konzepte. Während das generische Konzept `[RAD]` aus Abb. 4.2 für irgendein Rad steht, bezeichnet das individuelle Konzept `[PERSON: Müller]` aus Abb. 4.3 eine individuelle Person, nämlich die Person mit dem Namen „Müller“.

4.2.2.1 Generische Konzepte

Formal besteht ein Konzept aus zwei Teilen: Einem *Konzepttyp* (z.B. dem Typ „RAD“, siehe auch Abs. 4.2.3) und einem *Referenten*. Generische Konzepte werden durch einen *generischen Marker* in Form eines Sterns * gekennzeichnet, der aber oft einfach weggelassen wird. Somit handelt es sich sowohl bei `[Rad: *]` als auch bei `[Rad]` um generische Konzepte. Generische Konzepte (genauer: generische Marker) können als eine Art Variable gesehen werden, die für ein nicht näher bestimmtes Individuum steht. Diesen Variablen können auch Namen gegeben werden, z.B. `[Rad: *x]`. Somit kann das nicht näher benannte Individuum referenziert werden, was z.B. in der linearen Notation sinnvoll sein kann, um Zyklen darzustellen, die ansonsten in linearer Form nicht darstellbar wären. Abb. 4.3 ließe sich z.B. wie folgt in linearer Form darstellen, wobei die Variable `*x` in Zeile 3 und 5 dasselbe, nicht näher benannte Auto kennzeichnet. Durch die Einführung dieser Variable ist es möglich, den Zyklus aufzubrechen, und somit eine lineare Darstellung zu ermöglichen:

```
  [FAHREN] −
2     (AGENT)→[PERSON: Müller] −
          (BESITZ)→[AUTO: ∗x],
4     (ZIEL)→[STADT: Stuttgart]
      (MITTEL)→[AUTO: ∗x].
```

Ein einzelnes Minus-Symbol „-“ am Ende einer Zeile ermöglicht die Fortführung mehrerer verbundener Relationen in weiteren Zeilen (die Relationen „AGENT“, „ZIEL“ und „MITTEL“ sind z.B. alle mit dem Konzept „FAHREN“ verbunden). Ein Komma beendet diese Fortführung. Damit bilden Minus-Symbole und Kommata eine Möglichkeit zur Klammerung, wobei Minus-Symbole den öffnenden und Kommata den schließenden Klammern entsprechen. Der Punkt „.“ zeigt schließlich das Ende des Konzeptgraphen an. Der Punkt schließt auch implizit die erste öffnende Klammer, d.h. es ist hier nicht notwendig „ ,.“ zu schreiben. Die Einrückung dient nur der Veranschaulichung, hat aber ansonsten keinerlei syntaktische Bedeutung.

4.2.2.2 Individuelle Konzepte

Individuelle Konzepte haben als Referenten einen *individuellen Marker*, der durch # und eine eindeutige Nummer gekennzeichnet wird. So handelt es sich bei `[PERSON: #1234]` nicht um irgendeine (generische) Person, sondern um eine ganz spezielle. Hierbei könnte man `#1234` als eine Art Seriennummer betrachten, die nur ein Mal vergeben wird und ein Individuum eindeutig kennzeichnet. `#1234` ist also ein Stellvertreter (engl. Surrogate) für ein spezielles Individuum. Betrachtet man den generischen Marker als eine Variable z.B. im Sinne einer Programmiersprache,

so stellt ein individueller Marker eine Konstante dar. Anstatt einen individuellen Marker anzugeben, wird auch oft einfach der Name eines Individuums verwendet, was zumindest für den Menschen leichter lesbar ist. Das Beispiel aus Abb. 4.3 enthält z.B. das Konzept `[PERSON: Müller]`. Im Prinzip handelt es sich bei diesem Konzept um eine Abkürzung für den unteren Graphen aus Abb. 4.4. Man bezeichnet diese Abkürzung nach [**?**] auch als *Namenskontraktion*. Existiert im Kontext nur eine Person mit dem Namen „Müller“, so ist diese Bezeichnung ebenfalls eindeutig. Innerhalb dieser Arbeit wird ein Name als eindeutig angesehen, sofern nicht ausdrücklich etwas anderes gesagt wird (theoretisch könnte ein „Name“ z.B. auch eine eindeutige Personalnummer sein, so daß diese Annahme keine Einschränkung bedeutet).

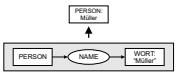

Abbildung 4.4: Namenskontraktion

Folgende Hilfsfunktion wird eingeführt, um den Referenten eines Konzepts zu bestimmen: Die Funktion $referent$ bildet ein generisches Konzept auf seinen individuellen Marker und ein generisches Konzept auf den generischen Marker * ab. Zum Beispiel ist $referent($`[RAD: *]`$) =$ `*` und $referent($`[PERSON: #1234]`$) =$ `#1234`.

Maße Ähnlich wie Namen können auch Maße behandelt werden. So sind z.B. 1 inch und 2.54 cm zwei Namen für dieselbe Länge. Maße werden durch das Symbol @ gekennzeichnet, so daß sich z.B. die „Länge 100 m“ wie in Abb. 4.5 oben beschreiben läßt, wobei der untere Teil von Abb. 4.5 die ausführliche Schreibweise darstellt. Dasselbe Prinzip läßt sich auch auf einen Geldbetrag o.ä. anwenden (z.B. `[BETRAG: @100 DM]`). Innerhalb dieser Arbeit wird nur die kurze Schreibweise verwendet.

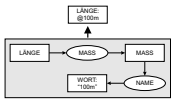

Abbildung 4.5: Maße

Mengen Sowa nennt in [**?**] ebenfalls die Möglichkeit, Mengen als Referenten eines Konzepts zu verwenden. So läßt sich z.B. der Satz „Herr Schmidt ist Leiter der Abteilungen Verkauf und Fertigung.“ wie folgt ausdrücken:

[PERSON: Schmidt]←(LEITER)←[ABTEILUNG:{Verkauf,Fertigung}].

Doch Mengen sind nicht immer leicht verständlich, wie folgendes Beispiel zeigt:

[PERSON: {Meyer,Kunze}]←(VORGESETZTER)←[PERSON:{Schmidt,Müller}].

Wie ist diese Aussage zu interpretieren? Eine mögliche Interpretation wäre zu sagen, *sowohl* Meyer *als auch* Kunze sind Vorgesetzte von Schmidt *und* Müller, d.h. sowohl Schmidt als auch Müller haben zwei Vorgesetzte, nämlich Meyer und Kunze. Leider ist dies nicht die einzige mögliche Interpretation. Eine weiter Möglichkeit wäre z.B., *ein* Element der ersten Menge mit *einem* Element der zweiten Menge in Beziehung zu setzen. So könnte z.B. Meyer der Vorgesetzte

von Schmidt und Kunze der Vorgesetzte von Müller sein oder Meyer der Vorgesetzte von Schmidt und Müller und Kunze nur ein weiterer Vorgesetzter von Müller. Ohne zusätzlich Angaben ist die Semantik des Mengenkonstrukts nicht eindeutig. In obigem Beispiel wäre es daher angebracht, einfach zwei Konzeptgraphen zu bilden:

[PERSON: Meyer]←(VORGESETZTER)←[PERSON: Schmidt].
[PERSON: Kunze]←(VORGESETZTER)←[PERSON: Müller].

Zumindest bei endlichen Mengen, bei denen die eingeschlossenen Individuen bekannt sind, ist dies ohne Probleme möglich. Um dagegen Sätze wie „Herr Meyer ist der Vorgesetzte einer Menge von Personen.“ auszudrücken, ist ein Mengenkonstrukt nicht verzichtbar:

[PERSON: Meyer]←(VORGESETZTER)←[PERSON: {∗}].

Dabei handelt es sich bei `{*}` um eine generische Menge (hier eine beliebige Menge von Personen). Solche Aussagen über nicht näher bestimmte Mengen von Personen sind für diese Arbeit nicht von besonderer Relevanz. Es wird daher auf das – auch in der sonstigen Literatur nicht unumstrittene – Mengenkonstrukt für Konzeptgraphen verzichtet.

4.2.3 Konzepttypen und Konzepttyphierarchien

Konzepttypen bestehen zunächst aus einem Bezeichner für den Konzepttyp, z.B. dem Typ „Person“ in obigem Beispiel, das das Konzept `[PERSON: Müller]` enthält oder dem Typ „RAD“ in Abb. 4.2. Konzepttypen sind den Klassen in der objektorientierten Programmierung vergleichbar. Sie bestimmen die Art eines Individuums, während der Referent (siehe Abs. 4.2.2.1 und Abs. 4.2.2.2) das Individuum bestimmt, das somit eine Instanz dieser Klasse im Sinne der objektorientierten Programmierung darstellt. Zum Beispiel kann man das Individuum mit dem Namen „Müller“ als eine Instanz der Klasse „Person“ ansehen. Welche Instanzen zu einem Typ gehören, wird durch die *Denotation* des Typs bestimmt:

Definition 1 (Denotation eines Typs) *Sei t ein Typbezeichner. Die Denotation δt eines Typs ist die Menge aller Entitäten, die Instanzen des Typs t sind.*

Die Funktion *type* bildet Konzepte und auch Relationen auf ihren Typbezeichner ab. Somit gilt z.B. *type*(`[Person: Müller]`) = `Person` und *type*(`(leitet)`) = `leitet`.

Konzepttypdefinitionen Zur Definition eines Konzepttyps können *Typdefinitionen* verwendet werden. Eine einfache Typdefinition für einen Abteilungsleiter könnte z.B. wie folgt aussehen:

type ABTEILUNGSLEITER(x) **is**
[PERSON: ∗x]←(LEITER)←[ABTEILUNG].

Nach dieser Definition ist ein Abteilungsleiter eine Person, die als Leiter einer Abteilung auftritt. Herr Müller als ein Individuum dieses Typs ließe sich durch folgendes Konstrukt definieren:

individual ABTEILUNGSLEITER(Müller) **is**
[PERSON: Müller]←(LEITER)←[ABTEILUNG: Verkauf].

Hierbei wurde nicht nur die Variable x mit Müller besetzt, sondern auch das Konzept `[ABTEILUNG]` durch ein spezielleres Konzept, nämlich das Konzept `[ABTEILUNG: Verkauf]`, ersetzt. Das ist aber theoretisch nicht zwingend erforderlich (aber sinnvoll, denn wenn bereits

bekannt ist, daß Herr Müller ein Abteilungsleiter ist, dann sollte auch die entsprechende Abteilung angegeben werden). Sowa vergleicht die Definition eines Individuums mit dem Eintrag eines Tupels in die Tabelle einer Datenbank. Generische Konzepte entsprechen hierbei dem NULL-Wert.

Durch *Typ-Expansion* läßt sich nun das Konzept `[ABTEILUNGSLEITER: Müller]` in folgenden Graphen expandieren:

[PERSON: Müller]←(LEITER)←[ABTEILUNG: Verkauf].

Die umgekehrte Operation wird als *Typ-Kontraktion* bezeichnet.

Konzepttyphierarchien Ähnlich den Klassenhierarchien bei der objektorientierten Programmierung lassen sich Typen in eine Hierarchie einordnen. Eine *Typhierarchie* definiert hierbei eine partielle Ordnung $\leq$ über der Menge der Typbezeichner:

Definition 2 (Konzepttyphierarchie) *Eine Konzepttyphierarchie H_C ist ein Tupel $(T_C, \leq_C, \top_C, \bot_C)$ mit*

- *T_C: eine Menge von Konzepttypen*
- *$\leq_C \in T_C \times T_C$: eine Halbordnung*
- *$\top_C \in T_C$ und $\forall x \in T_C : x \leq_C \top_C$*
- *$\bot_C \in T_C$ und $\forall x \in T_C : x \geq_C \bot_C$*

Die Elemente $\top$ und $\bot$ werden auch als das *universelle* bzw. *absurde* Element bezeichnet.[3]

s, t und u seien Typbezeichner aus der Menge T. Dann gelten folgende Begriffe:

Definition 3 (Sub-/Supertyp) *Falls $s \leq t$ gilt, dann ist s ein* Subtyp *von t und umgekehrt t ein* Supertyp *von s.*

Definition 4 (echter Sub-/Supertyp) *Falls $s \leq t$ und $s \neq t$ gilt, dann ist s ein* echter Subtyp *von t ($s < t$) und t ein* echter Supertyp *von s ($t > s$).*

Definition 5 (gemeinsamer Supertyp) *Falls $s \leq u$ und $t \leq u$ gilt, dann ist u ein* gemeinsamer Supertyp *von s und t.*

Definition 6 (gemeinsamer Subtyp) *Falls $u \leq s$ und $u \leq t$ gilt, dann ist u ein* gemeinsamer Subtyp *von s und t.*

Ein Individuum eines bestimmten Typs ist dabei gleichzeitig ein Individuum seiner Supertypen, d.h. es gilt das *Principle of Substituability*, das besagt, daß eine Instanz des Subtyps überall dort stehen kann, wo auch eine Instanz des Supertyps stehen kann. In der objektorientierten Programmierung wird dies als Subtyp- oder Inklusions-Polymorphismus bezeichnet.

Die Konzepttyphierarchie bildet einen Verband[4]:

[3] Innerhalb der Konzepttyphierarchie wird $\top$ auch teilweise als „Etwas“ (engl. Something) und $\bot$ als „Nichts“ (engl. Nothing) bezeichnet (vgl. [**?**]).

[4] Eine Menge M und eine Halbordnung $\leq$ werden als Verband (engl. Lattice) bezeichnet, falls zu jeder zweielementigen Teilmenge ein Infimum (eine größte untere Schranke) und ein Supremum (eine kleinste obere Schranke) existiert.

Definition 7 (minimaler gemeinsamer Supertyp) *Je ein Paar s und t hat einen* minimalen gemeinsamen Supertyp $u = s \sqcap t$*, so daß gilt:* $\forall x \in T : x \geq s \wedge x \geq t \rightarrow x \geq u$

Definition 8 (maximaler gemeinsamer Subtyp) *Je ein Paar s und t hat einen* maximalen gemeinsamen Subtyp $u = s \sqcup t$*, so daß gilt:* $\forall x \in T : x \leq s \wedge x \leq t \rightarrow x \leq u$

Zu beachten ist, daß aufgrund dieser Konstruktion auch Mehrfachvererbung möglich ist. Abb. 4.6 zeigt einen Ausschnitt aus einer möglichen Konzepttyphierarchie, wobei die Pfeile jeweils vom Sub- zum Supertyp zeigen (Generalisierung). Da die Verbindungen zu den Typen $\top$ und $\bot$ auch

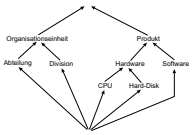

Abbildung 4.6: Konzepttyphierarchie

bereits kleine Typhierarchien evtl. sehr unübersichtlich aussehen lassen, werden diese Verbindungen im weiteren Verlauf dieser Arbeit weggelassen. Alle Typen, die keine Verbindungen zu einem Supertyp haben, sind implizit mit $\top$ verbunden. Umgekehrt sind alle Typen, die keine Verbindung zu einem Subtyp haben, implizit mit $\bot$ verbunden.

Zur Bestimmung, ob ein Individuum einem bestimmten Typ zuzurechnen ist, wird die Konformitätsrelation verwendet:

Definition 9 (Konformitätsrelation) *Sei* T_C *eine Menge von Konzepttypen,* I *eine Menge von Individuen und* $\mathbf{B} = \{true, false\}$*. Die Konformitätsrelation* $conf \in (T_C \times I) \rightarrow \mathbf{B}$ *bestimmt, ob ein Individuum* $i \in I$ *der Denotation* δt *eines Typs* $t \in T_C$ *zuzurechnen ist, d.h. ob* i *eine Instanz des Typs* t *ist. Es gilt:*

$$conf(t,i) = \begin{cases} true & \text{falls } i \text{ eine Instanz von } t \text{ ist} \\ false & \text{sonst} \end{cases}$$

$conf(x,y) = z$ *wird auch als* $x :: y = z$ *abgekürzt. Gilt* $t :: i = true$*, dann ist das Individuum* i konform *zum Typ* t*. Folgende Bedingungen gelten:*

- *Sei* `[t:i]` *ein Konzept. Dann muß gelten* $t :: i = true$*.*
- *Falls* $s \leq t$ *und* $s :: i = true$ *gilt, dann gilt auch* $t :: i = true$*.*
- $\forall i \in I : \ \top :: i = true$
- $\forall i \in I : \ \bot :: i = false$
- $\forall t \in T \setminus \bot : \ t :: * = true$

4.2.4 Relationstypen und Relationstyphierarchien

Relationstyphierarchien Wie Konzepte, so haben auch Relationen einen zugeordneten (Relations-)Typ, der ebenfalls in eine Typhierarchie eingeordnet wird. Da eine n_1-stellige Relation nur mit einer n_2-stelligen Relation vergleichbar ist, falls $n_1 = n_2$ gilt, existieren keine ausgezeichneten Elemente $\top$ und $\bot$ mehr, sondern vielmehr für jedes $n \in \mathbf{N}$ ein eigenes Top- und Bottom-Element:

Definition 10 (Relationstyphierarchie) *Eine Relationstyphierarchie H_R ist ein Tupel $(T_R, \leq_R, \{\top_{R_1}, \ldots, \top_{R_n}\}, \{\bot_{R_1}, \ldots, \bot_{R_n}\})$ mit*

- *T_R: eine Menge von Relationstypen*
- *$\leq_R \in T_R \times T_R$: eine Halbordnung*
- *Sei $n \in \mathbf{N}$ und $n \geq 1$. Dann gilt: $\top_{R_n} \in R^n$ und $\forall x \in R^n : x \leq_R \top_{R_n}$*
- *Sei $n \in \mathbf{N}$ und $n \geq 1$. Dann gilt: $\bot_{R_n} \in R^n$ und $\forall x \in R^n : x \geq_R \bot_{R_n}$*

Ansonsten gelten die Definitionen aus Abs. 4.2.3 auch für die Relationstyphierarchie.

Relationstypdefinitionen Auch für Relationen kann eine (Relations-)Typ-Definition angegeben werden, wie folgendes einfaches Beispiel für eine binäre Relation zeigt:

relation LEITER(x,y) **is**
[PERSON: ∗x]←(LINK)←[ORGANISATIONSEINHEIT: ∗y].

`(LINK)` bezeichnet hierbei eine allgemeine binäre Relation ($\top$ in der Relationstyphierarchie). Der Relationstyp `NO-LINK` steht für $\bot$ in der Relationstyphierarchie. Die Typdefinition ist in etwa der Signatur einer Prozedur vergleichbar, d.h. sie definiert den Namen und die Art und Anzahl der Argumente der Relation. Eine weitere Relationstypdefinition für eine einstellige Relation könnte wie folgt aussehen:

relation VERFÜGBAR(x) **is**
[WARE: ∗x]←(ENTHÄLT)←[LAGER].

Die Relation „VERFÜGBAR“ hat als Argument ein Konzept vom Typ „WARE“ und besagt, daß eine Ware verfügbar ist, wenn ein Lager die Ware enthält. Gegeben sei folgender Konzeptgraph:

(VERFÜGBAR)→[WARE: Kühlschrank].

Durch *Relationsexpansion* läßt sich hieraus unter Anwendung der Typdefinition für „verfügbar“ der folgende Graph bilden:

[WARE: Kühlschrank]←(ENTHÄLT)←[LAGER].

Die umgekehrte Operation wird als *Relationskontraktion* bezeichnet.

4.3 Formale Definition

Da nun die notwendigen Grundlagen eingeführt wurden, kann an dieser Stelle eine formale Definition eines Konzeptgraphen erfolgen. Es ist dabei nicht sinnvoll, einen Konzeptgraphen

isoliert zu betrachten, sondern den Kontext zu berücksichtigen, der durch die Konzept- und Relationstyphierarchie und die entsprechenden Definitionen gegeben ist.

Definition 11 (Konzeptgraph) *Gegeben:*

- *eine Konzepttyphierarchie* $H_C = (T_C, \leq_C, \top_C, \bot_C)$
- *eine Relationstyphierarchie* $H_R = (T_R, \leq_R, \{\top_{R1}, \ldots, \top_{Rn}\}, \{\bot_{R1}, \ldots, \bot_{Rn}\})$
- *eine Konformitätsrelation* $conf$
- *eine Menge von individuellen Markern* I

Ein Konzeptgraph G *ist ein Graph* (C, R, E) *mit*

- C*: eine endlich Menge von Konzeptknoten mit* $|C| \geq 1$
- R*: eine endlich Menge von Relationsknoten*
- $E \subseteq C \times R \times \mathbf{N}$ *eine Menge von Kanten. Eine Kante* $e = (c, r, n) \in E$ *verbindet den Konzeptknoten* c *mit dem Relationsknoten* r *durch eine Kanten mit der Nummer* n*. Dabei ist* $n > 0$*. Falls eine Kante* $e_i = (c, r, n_i)$ *mit* $n_i > 1$ *existiert, dann existiert auch eine Kante* $e_j = (c, r, n_j)$ *mit* $n_j = n_i - 1$ *(d.h. die Kanten einer Relation sind eindeutig von 1 bis* n *durchnumeriert). Es gelte* $r^n = c$*, falls* (c, r, n) *in* E *enthalten ist (d.h.* r^n *liefert das* n*-te Argument einer Relation). Außerdem wird folgende Schreibweise eingeführt, um die Stelligkeit einer Relation* r *darzustellen:* $|r| = |\{(c, r', n) \in E \mid r' = r\}|$

$type(c)$ *bildet wie gehabt ein* $c \in C$ *auf einen Typ* T_C *und ein* $r \in R$ *auf einen Typ in* T_R *ab.* $referent$ *bildet ein* $c \in C$ *auf ein* $i \in I \cup \{*\}$ *ab. Dann gelten für* G *außerdem die folgenden Bedingungen:*

- G *ist zusammenhängend*
- *Sei* t_r^n *der Konzepttyp für das* n*-te Argument der Relationstypdefinition der Relation* r*.* $\forall r \in R : (\forall n \in \{1, \ldots, |r|\} : type(r^n) \leq_C t_r^n)$
- $\forall c \in C : conf(type(c), referent(c))$

4.4 Logische Grundlagen

Ein wesentlicher Vorteil der Konzeptgraphen ist ihre klar definierte Semantik durch die Abbildung eines Konzeptgraphen auf eine Formel der Prädikatenlogik erster Stufe durch den ϕ-Operator.

Definition 12 (ϕ-Operator) *Sei* $G = (C = \{c_1, \ldots, c_m\}, R = \{r_1, \ldots, r_n\}, E)$ *ein Konzeptgraph.* $C_{gen} \subseteq C$ *sei die Menge der generischen Konzept und* $C_{ind} \subseteq C$ *die Menge der individuellen Konzepte von* G*. Dann ist* $\phi(G)$ *die prädikatenlogische Formel, die nach den folgenden Konstruktionsregeln gebildet wird:*

1. *Für jedes generische Konzept* $c_i \in C_{gen}$ *konstruiere eine Variable* x_{c_i}*, so daß zwei verschiedene generische Konzepte* c_i *und* c_j *mit* $i \neq j$ *Variablen* x_{c_i} *und* x_{c_j} *mit verschiedenen Namen haben. Konstruiere ein einstelliges Prädikat* P_{c_i} *mit dem Namen* $type(c_i)$ *und dem Argument* x_{c_i}*.*

2. *Für jedes individuelle Konzept $c_i \in C_{ind}$ konstruiere eine Konstante k_{c_i} mit dem Namen $referent(c_i)$. Konstruiere ein einstelliges Prädikat P_{c_i} mit dem Namen $type(c_i)$ und dem Argument k_{c_i}.*

3. *Für jede n-stellige Relation $r_i \in R$ konstruiere ein Prädikat P_{r_i} mit dem Namen $type(r_i)$ und den Argumenten $a_1, \ldots, a_n$ mit $a_i = x_{c_j}$, falls $r^i = c_j \in C_{gen}$ (d.h. das i-te Argument von r_i entspricht dem generischen Konzept c_j), bzw. $a_i = k_{c_j}$, falls $r^i = c_j \in C_{ind}$ (d.h. das i-te Argument entspricht dem individuellen Konzept c_j).*

4. *$\phi(G)$ ist die Formel, die durch die konjunktive Verknüpfung aller erstellten Prädikate entsteht, wobei alle Variablen existentiell gebunden werden.*

Diese Abbildung soll anhand eines Beispiels erläutert werden. Gegeben sei folgender Konzeptgraph G:

[PERSON:Schmidt] −
(VORGESETZTER)←[PERSON:Meyer]
(LEITER)←[ABTEILUNG].

Der Operator ϕ bildet diesen Graphen auf eine prädikatenlogische Formel ab:

$$\begin{aligned}\phi(G) \quad = \quad & \exists x(PERSON(Meyer) \wedge PERSON(Schmidt) \wedge ABTEILUNG(x) \wedge \\ & VORGESETZTER(Meyer, Schmidt) \wedge LEITER(x, Schmidt))\end{aligned}$$

Alle Individuen, wie z.B. das Individuum „Meyer“, werden auf Konstanten abgebildet. Für generische Konzepte werden Variablen eingeführt, die existentiell gebunden werden. Die Namen der Prädikate entsprechen den Typbezeichnern der Konzept- und Relationstypen, und alle Prädikate werden durch Konjunktionen verknüpft.

Logische Grundlagen der Typhierarchien Auch für Typhierarchien existiert eine Abbildung auf Formel der Prädikatenlogik erster Stufe. Der Ω-Operator bildet hierbei eine Konzepttyp- und eine Relationstyphierarchie auf eine Menge von Formel ab.

Definition 13 (Ω-Operator) *Gegeben sei eine Konzepttyphierarchie $H_C = (T_C, \leq_C, \top_C, \bot_C)$ und eine Relationstyphierarchie $H_R = (T_R, \leq_R, \{\top_{R_1}, \ldots, \top_{R_n}\}, \{\bot_{R_1}, \ldots, \bot_{R_n}\})$. Dann ist $\Omega(H_C, H_R)$ die Menge von prädikatenlogischen Formeln, die nach folgenden Konstruktionsregeln gebildet wird:*

1. *Für jedes Paar $s, t \in T_C$ mit $s <_C t$ konstruiere eine Formel $F_{s,t} = \forall x(P_s(x) \rightarrow P_t(x))$, wobei P_s und P_t jeweils einstellige Prädikate mit den Namen des Typen s bzw. t sind.*

2. *Für jedes Paar $u, v \in T_R$ mit $u <_R v$ und $|u| = |v| = n$ konstruiere eine Formel $F_{u,v} = \forall x_1, \ldots, \forall x_n(P_u(x_1, \ldots, x_n) \rightarrow P_v(x_1, \ldots, x_n))$, wobei P_u und P_v jeweils n-stellige Prädikate mit den Namen des Typs von u bzw. v sind.*

3. *$\Omega(H_C, H_R)$ ist die Menge aller konstruierten Formeln. $\Omega(H_C, H_R)_\wedge$ ist die konjunktive Verknüpfung aller konstruierten Formeln.*

Die Menge $\Omega(H_C, H_R)$ mit der Konzepttyphierarchie aus Abb. 4.6 enthält u.a. die Formeln $\forall x(CPU(x) \rightarrow Hardware(x))$ und $\forall x(Division(x) \rightarrow Organisationseinheit(x))$.

4.5 Operationen auf Konzeptgraphen

4.5.1 Kanonische Formationsregeln

Bereits in Abs.4.2 wurden die kanonischen Formationsregeln angesprochen. Mit Hilfe dieser Regeln ist die Herleitung kanonischer Graphen auf Grundlage bereits existierender kanonischer Graphen möglich. Die kanonischen Formationsregeln bestehen aus den vier Operationen Kopieren, Restriktion, Verbinden und Vereinfachen, die auf Konzeptgraphen ausgeführt werden können.

Kopieren (Copy) Die Kopieroperation kopiert einen Konzeptgraphen 1:1. Das heißt, alle Konzepte und Relationen, sowie die Verbindungen zwischen Konzepten und Relationen, werden exakt in die Kopie übernommen.

Restriktion (Restrict) Bei der Restriktion werden ein oder mehrere Konzepte durch speziellere Konzepte ersetzt. Diese *Spezialisierung* kann auf zwei Arten erfolgen:

- Ein generisches Konzept c wird durch ein individuelles Konzept ersetzt, d.h. der generische Marker * wird durch einen individuellen Marker i mit $type(c) :: i = true$ ersetzt. Das eingesetzte Individuum muß also konform zum Typ des Konzepts sein.
- Der Typ $t_1 = type(c)$ eines Konzepts c wird durch einen Typ t_2 ersetzt, so daß gilt $t_2 \leq t_1$, d.h. t_2 ist ein Subtyp von t_1. Dabei muß gelten $t_2 :: referent(c) = true$.

Abb. 4.7 zeigt ein Beispiel für eine Restriktion, in der beide Fälle auftreten (es wird davon ausgegangen, daß für die Typen „PERSON“ und „MANAGER“ gilt MANAGER $\leq$ PERSON).

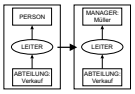

Abbildung 4.7: Restriktion

Verbinden (Join) Beim Verbinden werden zwei identische Konzepte c_1 und c_2 miteinander verschmolzen, d.h. ein Konzept wird gelöscht, und alle Kanten, die mit diesem Konzept verbunden waren, werden mit dem identischen Konzept verbunden. Diese Operation ist nur erlaubt, wenn $type(c_1) = type(c_2)$ und $referent(c_1) = referent(c_2)$ gilt. Um dies zu erreichen, können dem Verbinden Restriktionen vorausgehen. Die Konzepte c_1 und c_2 können aus verschiedenen oder demselben Konzeptgraphen stammen. Abb. 4.8 zeigt ein Beispiel für diese Operation.

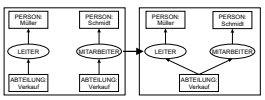

Abbildung 4.8: Verbinden

Vereinfachen (Simplify) Hat ein Konzeptgraph zwei identische Relationen, d.h. zwei Relationen r_1 und r_2, für die gilt

- $type(r_1) = type(r_2)$
- $\forall i \in \mathbf{N}$ mit $1 \leq i \leq |r_1| = |r_2|$ gilt $r_1^i = r_2^i$

dann kann eine Relation mit all ihren Kanten gelöscht werden. Abb. 4.9 zeigt ein Beispiel für das Vereinfachen eines Konzeptgraphen.

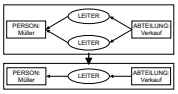

Abbildung 4.9: Vereinfachen

4.5.2 Generalisierung und Spezialisierung

Spezialisierung Die kanonischen Formationsregeln sind Spezialisierungsregeln, d.h. ihre Anwendung auf einen Graphen G bringt einen spezielleren Graphen H hervor.

Definition 14 (Spezialisierung) *Eine Sequenz von Anwendungen der kanonischen Formationsregeln auf einen Konzeptgraphen G. Der resultierende Konzeptgraph H wird als* Spezialisierung *von G bezeichnet ($H \leq G$).*

Der aus der Spezialisierung hervorgegangene Graph H repräsentiert wieder eine *mögliche* Aussage. Es ist dabei wichtig zu erkennen, daß diese Spezialisierungsregeln *keine* Inferenzregeln sind, d.h. es gilt im allgemeinen *nicht* $\Omega(H_C, H_R) \cup \phi(G) \models \phi(H)$[5], wie die folgenden einfachen Gegenbeispiele zeigen.

Gegeben sei folgender Konzeptgraph G:

G = [PERSON]→(VORGESETZTER)→[PERSON: Schmidt].

Durch Anwendung der Restriktionsregel und durch Spezialisierung der Konzepts `[PERSON]` zum Konzept `[PERSON: Meyer]` entsteht dabei folgender Graph H:

H = [PERSON: Meyer]→(VORGESETZTER)→[PERSON: Schmidt].

Die Abbildung auf prädikatenlogische Formeln liefert:

$$\phi(G) = \exists x(PERSON(x) \wedge PERSON(Schmidt) \wedge VORGESETZTER(x, Schmidt))$$

$$\phi(H) = PERSON(Meyer) \wedge PERSON(Schmidt) \wedge VORGESETZTER(Meyer, Schmidt)$$

Betrachtet man die Konzepte `[PERSON]` und `[PERSON: Meyer]`, so ist leicht einzusehen, daß $\phi(G)$ erfüllt sein kann und trotzdem $\phi(H)$ nicht erfüllt sein muß. Dieser Fall tritt ein, wenn keine Person $x = Meyer$ existiert, deren Vorgesetzter Schmidt ist (G macht keinerlei Aussagen darüber, wessen Vorgesetzter Schmidt ist).

Ebenso ist die Vereinigungsregel *keine* Inferenzregel, wie folgendes Beispiel zeigt. Gegeben seien wieder zwei Konzeptgraphen G_1 und G_2:

[5]Man schreibt $\{F_1, \ldots, F_n\} \models G$, wenn G eine Folgerung aus den Formeln F_i ist, d.h. wenn jedes Modell von $\{F_1, \ldots, F_n\}$ ebenfalls ein Modell von G ist.

G_1 = [PERSON]←(VORGESETZTER)←[PERSON: Meyer].
G_2 = [PERSON]←(VORGESETZTER)←[PERSON: Kunze].

Nach der Vereinigung auf dem Konzept `[PERSON]` ergibt sich der Konzeptgraph H:

H = [PERSON] –
(VORGESETZTER)←[PERSON: Meyer]
(VORGESETZTER)←[PERSON: Kunze].

Auch hier gilt *nicht* $\Omega(H_C, H_R) \cup \phi(G_1) \cup \phi(G_2) \models \phi(H)$. Es existieren zwar Personen, die die Vorgesetzten von Meyer und Kunze sind, aber es ist nicht gesagt, daß Meyer und Kunze *denselben* Vorgesetzten haben (die existentiell gebundenen Variablen aus $\phi(G_1)$ und $\phi(G_2)$ können für unterschiedliche Individuen stehen).

Unkritisch bezüglich dem Wahrheitsgehalt einer Aussage sind dagegen die Kopier- und Vereinfachungsregel. Für die Kopierregel gilt trivialerweise $\phi(G) \rightarrow \phi(H)$, da $G = H$ gilt. Bei der Vereinfachungsregel gilt die Folgerung, da $P(x_1, \ldots, x_n) \wedge P(x_1, \ldots, x_n) \rightarrow P(x_1, \ldots, x_n)$ gilt (P wäre das Prädikat, das aus der doppelt vorhandenen Relation des Ursprungsgraphen konstruiert wurde und das nach der Vereinfachung einmal gelöscht wird).

Generalisierung Umgekehrt sind Generalisierungsregeln, also quasi die Umkehrung der Spezialisierungsregeln, Inferenzregeln:

Satz 1 (Generalisierungsregeln sind Inferenzregeln) *Seien G und H Konzeptgraphen, und es gelte $H \leq G$, d.h. H ist eine Spezialisierung von G oder anders ausgedrückt, G geht aus H durch die Anwendung einer Sequenz von Generalisierungsregeln hervor. H_C sei eine Konzepttyphierarchie und H_R eine Relationstyphierarchie, die jeweils die in G und H verwendeten Konzept- und Relationstypen enthalten. Dann gilt $\Omega(H_C, H_R) \cup \phi(H) \models \phi(G)$.*

Beweis: siehe [**?**]. ■

Somit gilt in den obigen Beispielen immer $\Omega(H_C, H_R) \cup \phi(H) \models \phi(G)$.

4.6 Erweiterte Konzeptgraphen

Die bisher besprochenen Konzeptgraphen werden innerhalb dieser Arbeit als *einfache Konzeptgraphen* bezeichnet. Sowa beschreibt in [**?**] Erweiterungen dieser einfachen Konzeptgraphen, so daß diese die gleiche Mächtigkeit wie die Prädikatenlogik erster Stufe erlangen. Diese Erweiterungen basieren auf den existentiellen Graphen von Charles Sanders Peirce, die dieser selbst als „the logic of the future“ bezeichnete. Betrachtet man die Konstruktionsregeln aus Abs. 4.4 näher, so fällt auf, daß nur die Konjunktion und der Existenz-Quantor verwendet werden. Fügt man die Negation hinzu, so läßt sich auch der All-Quantor und die Disjunktion ausdrücken ($\neg(\exists x(F)) \equiv \forall x(\neg F)$ und $\neg(F \wedge B) \equiv \neg F \vee \neg B$). Sowa führt daher die Möglichkeit ein, ganze Konzeptgraphen als Referent eines Konzepts (einer Proposition, d.h. eines Konzepts mit dem Typ `PROPOSITION`) zu verwenden, wobei die „eingeschlossenen“ Konzeptgraphen im Kontext des umschließenden Konzepts stehen. Konzeptgraphen lassen sich somit ineinander schachteln. Außerdem wird eine einstellige Relation `NEG` eingeführt, die für die Negation steht. Somit läßt sich nun z.B. auch der Satz „Keine Person ist ihr eigener Vorgesetzter.“ durch den Konzeptgraphen G aus Abb. 4.10 ausdrücken.

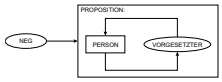

Abbildung 4.10: Konzeptgraph: „Keine Person ist ihr eigener Vorgesetzter."

Die Umsetzung in die Prädikatenlogik liefert:

$$\phi(G) = \neg(\exists x(PERSON(x) \wedge VORGESETZTER(x, x)))$$

Sowa beschreibt in [**?**] ebenfalls eine Menge von Inferenzregeln auf erweiterten Konzeptgraphen. Diese Regeln werden in dieser Arbeit nicht verwendet, so daß der interessierte Leser auf diese Quelle verwiesen wird.

Nachteile der erweiterten Konzeptgraphen Erweiterte Konzeptgraphen bieten alle Möglichkeiten der Prädikatenlogik erster Stufe. Diese Mächtigkeit wird allerdings durch ihre erhöhte Komplexität im Vergleich zu den einfachen Konzeptgraphen erkauft. Einer der größten Vorteile der einfachen Konzeptgraphen ist ihre Anschaulichkeit und das nahezu intuitive Verständnis dieser Graphen. Vergleicht man Konzeptgraphen mit der „herkömmlichen" Logik, so mag man auch hier einen Vorteil in der grafischen Notation sehen. So schreibt z.B. Sowa:

> Standard logic is not widely known, even among computer programmers. For novices, the graph notation is easier to learn and read than the linear notation. [**?**]

Trotzdem ist es sicher legitim, sich zu fragen, ob die volle Mächtigkeit der Prädikatenlogik wirklich von der Aufgabenstellung gefordert wird. Auf diese Frage läßt sich keine allgemeingültige Antwort geben. Ein geschulter Mathematiker oder Informatiker mit fundierten Theoriekenntnissen mag die Möglichkeiten zu schätzen wissen, die ihm die erweiterten Konzeptgraphen im Vergleich zu den einfachen Graphen bieten. Für andere sind die Möglichkeiten der einfachen Konzeptgraphen bereits ausreichend. Innerhalb dieser Arbeit wird daher zunächst versucht, mit den Mitteln der einfachen Konzeptgraphen auszukommen. Wo nötig wird auf die Möglichkeiten hingewiesen, die sich durch die Anwendung der erweiterten Konzeptgraphen ergeben würden.

4.7 Ontologien

Konzeptgraphen können nur sinnvoll verwendet werden, wenn eine Übereinkunft über die Bedeutung der verwendeten Konzepte und Relationen vorliegt. Fehlen Definitionen für die verwendeten Konzepte und Relationen, so kommt es zu Mißverständnissen aufgrund des unterschiedlichen Verständnisses verschiedener Personen. Um diese Mißverständnisse und alle daraus resultierenden Probleme zu vermeiden, führt man eine *Ontologie* ein.

Der Begriff der Ontologie kann verschieden definierte werden (siehe [**?**]). In [**?**] wird eine Ontologie als eine „explizite Konzeptionalisierung" definiert. Unter einer Konzeptionalisierung ist dabei folgendes zu verstehen:

> Broadly, a conceptualisation is a world view; it corresponds to a way of thinking about some domain. It can be seen as a set of informal rules that constrain the structure of a piece of reality. It is typically conceived and/or expressed as a set of concepts (e.g. entities, attributes, processes), their definitions and their inter-relationships. [**?**]

Ebenfalls den Begriff der Konzeptionalisierung verwendet die Definition aus [**?**], die auf einen Eintrag in der Shared Re-usable Knowledgebases Mailing-Liste zurückgeht:

> Ontologies are agreements about shared conceptualizations. Shared conceptualizations include conceptual frameworks for modeling domain knowledge, content specific protocols for communication among inter-operating agents; and agreements about the representation of particular domain theories. In the knowledge sharing context, ontologies are specified in the form of definitions of representational vocabulary. A very simple case would be a type hierarchy, specifying classes and their subsumption relationships. Relational database schemata also serve as ontologies by specifying the relations that can exist in some shared database and the integrity constraints that must hold for them. [**?**]

Überträgt man diese allgemeinen Definitionen des Ontologie-Begriffes auf Konzeptgraphen, so ist hier unter Ontologie die Definition von Konzept- und Relationstypen sowie die Einordnung dieser Typen in die entsprechenden Typhierarchien zu verstehen.

Kapitel 5

Entwurf der Organisationsverwaltung unter Verwendung von Konzeptgraphen

In diesem und den folgenden Kapiteln wird ein wissensbasierter Ansatz zur Realisierung der Organisationsverwaltung auf Basis von Konzeptgraphen vorgestellt. Unter Verwendung der Erkenntnisse der vorangehenden Kapitel erfolgt in diesem Kapitel der Entwurf der Organisationsverwaltung. Die einzelnen Komponenten der Organisationsverwaltung, nämlich Organisationsstrukturverwaltung, Zuweisungsregelverwaltung und Synchronisationsregelverwaltung, wurden bereits vorgestellt. Betrachtet werden hier nur die ersten beiden Komponenten, da sich für die letzte Komponente keine Änderungen durch den neuen Ansatz ergeben oder anders ausgedrückt die bisherigen Konzepte auf diesen Ansatz übertragen werden können.

5.1 Anforderungsanalyse

Betrachtet man die Defizite aktueller Systeme, die in Kapitel 3 herausgearbeitet wurden, so ergeben sich die folgenden Anforderungen für den neuen Ansatz der Organisationsverwaltung:

Die freie Definition der Organisationsstruktur soll möglich sein, d.h. die Modellierungskonstrukte sollen nicht bereits auf einen bestimmten Organisationsstruktur-Typ ausgerichtet sein. Es soll dabei möglich sein, beliebige Eigenschaften eines Akteurs sowie beliebige Beziehungen zwischen den Elementen der Organisation zu definieren. Änderungen an der Organisationsstruktur sollen in einfacher Art und Weise und ohne großen Aufwand möglich sein.

Es ist eine Modellierung anzustreben, die eine möglichst große Entkopplung von Ablauf- und Aufbauorganisation ermöglicht. Dies ist der Kernpunkt dieser Arbeit. Änderungen am Organisationsmodell sollen möglichst keine Anpassung der Prozeßdefinitionen bzw. der Zuweisungsregeln zur Folge haben.

Die Selektion von Aufgabenträgern soll aufgrund der im Organisationsmodell festgehaltenen Eigenschaften erfolgen können. Unter Eigenschaften sind dabei nicht nur Rollen, sondern jegliche Art von explizit modellierten Eigenschaften wie z.B. Fähigkeiten, Rechte, u.ä. zu verstehen.

Bezüglich der Zuweisung von Aufgaben zu Aufgabenträgern wird eine weitere Anforderung festgelegt, die so bisher nicht in aktuellen Systemen anzutreffen ist. Sollte bei der Zuweisung festgestellt werden, daß kein Akteur existiert, der alle für die Erfüllung einer Aufgabe notwendigen

Eigenschaften besitzt, so soll der Akteur gefunden werden, der die geforderten Eigenschaften am besten – wenn auch nicht vollständig – erfüllt (unscharfe Selektion von Aufgabenträgern).

Im Bereich der nichtfunktionalen Eigenschaften ist zunächst die Leistungsfähigkeit (engl. Performance) zu nennen. Hier muß sich der neue Ansatz mit aktuellen Systemen messen lassen, die auf Datenbanken beruhen. Im Bereich Performance können vor allem relationale Datenbanksysteme dank umfangreichster Forschungen (auch) auf diesem Gebiet als ausgereift gelten. Unter Performance-Gesichtspunkten ist vor allem die zeitkritische Zuweisung von Aufgabenträgern zu beachten. In entsprechend großen Systemen können sehr viele solcher Zuweisungen in kurzer Zeit anfallen. Eine Zuweisungsregel sollte daher innerhalb weniger Sekunden ausgeführt werden können.

Eine weitere nichtfunktionale Eigenschaft ist die Benutzerfreundlichkeit. Diese hängt sehr stark von der Benutzungsschnittstelle (engl. **G**raphical **U**ser **I**nterface) ab. Die Berücksichtigung ergonomischer Grundsätze, wie sie z.B. in [**?**] beschrieben werden, ist hierbei mit entscheidend für die Akzeptanz der Anwendung durch den Benutzer. Die Benutzungschnittstelle zur Eingabe und Verwaltung von Konzeptgraphen der Organisationsverwaltung ist Gegenstand einer separaten Diplomarbeit, die parallel zu dieser durchgeführt wird. Diese Arbeit beschäftigt sich daher nur marginal mit der Benutzungsschnittstelle und verweist wo nötig auf [**?**].

Um den neuen Ansatz in einem realen Umfeld testen zu können, soll eine Integration der Organisationsverwaltung in das Workflow-Management-System Changengine der Firma Hewlett-Packard erfolgen.

5.2 Ablauf der Aufgabenträgerzuweisung und Überblick über die involvierten Komponenten

Die Zuweisung von Aufgabenträgern zu Aktivitäten erfolgt durch die Abarbeitung einer Zuweisungsregel. Changengine ruft die Zuweisungsregelverwaltung mit dem Bezeichner der gewünschten Zuweisungsregel und einer Menge workflowrelevanter Daten auf, die für die Zuweisung benötigt werden, sobald ein Aufgabenträger einer Aktion zugewiesen werden soll (zu Details zur Integration der Zuweisungsregelverwaltung in Changengine siehe Abs. 5.4). Die Abarbeitung einer Zuweisungsregel kann in drei Schritte unterteilt werden:

1. Auswahl einer Selektionsregel, d.h. einer Regel, die Akteure mit bestimmten Eigenschaften aus der Organisationsstruktur selektiert. Hierzu können workflowrelevante Daten herangezogen werden. Ein typisches Beispiel aus dem Bankwesen ist die Genehmigung eines Kredits. Überschreitet der Kreditwert einen bestimmten Betrag, so kann gefordert werden, daß der Abteilungsleiter die Genehmigung unterzeichnet. Ansonsten kann diese Aktivität ein einfacher Kreditsachbearbeiter durchführen. Bei diesem Beispiel wäre die Kreditsumme das workflowrelevante Datum. Allgemein ist es notwendig, eine Selektionsregel zu wählen, die diejenigen Akteure selektiert, die die notwendigen Eigenschaften für die Durchführung einer Aufgabe haben.

2. Selektion von Akteuren aufgrund der geforderten Eigenschaften. Wie in Punkt 1 beschrieben, werden diese Eigenschaften durch die gewählte Selektionsregel spezifiziert. Es ist hier ganz bewußt von Eigenschaften und nicht von Rollen die Rede. Herkömmliche Systeme selektieren Akteure, die bestimmte Rollen einnehmen (siehe Abs. 3.3). Die Rolle enthält dabei *implizit* die geforderten Eigenschaften. Zum Beispiel könnte die Rolle *Systemprogrammierer* für eine (abstrakte) Person stehen, die Kenntnisse in der Programmiersprache

C besitzt. Was eigentlich gesucht wird, ist nicht die Rolle Systemprogrammierer, sondern eine Person mit C-Kenntnissen. Diese rollenbasierte Selektion hat eine starke Kopplung von Aufbau- und Ablauforganisation zur Folge (vgl. Abs. 3.3.5). Eine wesentliche Anforderung dieser Arbeit ist die Entkopplung von Ablauf- und Aufbauorganisation, so daß Änderungen an der Organisationsstruktur möglichst wenig Auswirkungen auf die Prozeßdefinitionen haben. Diese Entkopplung kann erreicht werden, wenn nicht nach Rollen, sondern nach den geforderten Eigenschaften selektiert wird. Bisher wurden diese Eigenschaften *implizit* durch die Rolle modelliert, nun werden sie *explizit* durch die Zuweisung von Eigenschaften an Akteure modelliert. Diese explizite Modellierung von Eigenschaften ermöglicht erst die Selektion eines Aufgabenträgers aufgrund der geforderten Eigenschaften.

3. Auswahl der Akteure, die die Aufgabe schließlich durchzuführen haben, aus der Menge der selektierten Akteure und Rückgabe des Ergebnisses an Changengine. Die in Punkt 2 durchgeführte Selektion liefert u.U. nicht nur einen Aufgabenträger, sondern eine Menge von n potentiellen Aufgabenträgern, die alle die geforderten Eigenschaften besitzen. Es muß daher eine Auswahl aus dieser Menge erfolgen. In den meisten Systemen wird die Aufgabe einfach in die Work-Lists aller n Mitarbeiter aufgenommen. Sobald ein Mitarbeiter die Aufgabe wirklich bearbeitet, verschwindet sie bei den anderen Mitarbeitern wieder aus deren Work-List. Es sind aber auch wesentlich kompliziertere Verfahren denkbar. In [**?**] werden hierbei verschiedene Kriterien analysiert. Diese Kriterien werden in die Klassen ökonomische Kriterien und Motivationskriterien unterteilt. Ziel ökonomischer Kriterien ist es beispielsweise, eine möglichst schnelle oder ressourcenschonende Durchführung zu erreichen. Berücksichtigt man dagegen Motivationskriterien, so könnte man z.B. die Zuweisung derart durchführen, daß die Mitarbeiter möglichst abwechslungsreiche Aufgaben durchzuführen haben, was die Arbeit interessanter gestaltet und somit die Motivation steigert. Die Auswahl aus der Menge der potentiellen Akteure, die durch die Anwendung einer Selektionsregel bestimmt wurden, kann beim wissensbasierten Ansatz auf dieselbe Art und Weise erfolgen, wie sie in [**?**] beschrieben wird. Diese Verfahren sind weitestgehend unabhängig von diesem Ansatz und müssen daher nicht angepaßt und überdacht werden (evtl. sind bei der Implementierung kleine Anpassungen erforderlich, z.B. um die Menge der Fähigkeiten einer Person aus deren Modellierung durch Konzeptgraphen zu bestimmen; das Verfahren ansich bleibt davon unberührt). Wichtig ist nur, daß der Entwurf es an entsprechender Stelle zuläßt, diese Auswahl aus der selektierten Menge der Akteure zu treffen. Es müssen also entsprechende Einstiegspunkte hierfür vorgesehen werden.

Abb. 5.1 zeigt die einzelnen Schritte bei der Aufgabenträgerzuweisung etwas genauer und liefert gleichzeitig einen ersten Überblick über die involvierten Komponenten.

1. Aufruf der Zuweisungsregelverwaltung durch das Workflow-Management-System: Das Workflow-Management-System ruft die Zuweisungsregelverwaltung auf, sobald ein Aufgabenträger einer Aktivität zuzuweisen ist. Dabei wird als erster Parameter der Bezeichner einer Zuweisungsregel übergeben. Die weiteren Parameter sind workflowrelevante Daten, die zur Zuweisung evtl. benötigt werden.

2. Auswahl einer Zuweisungsregel: Der Resource Executive nimmt die Anfrage des Workflow-Management-Systems entgegen, ermittelt die gewünschte Zuweisungsregel und startet deren Ausführung.

3. Aufruf der Organisationsstrukturverwaltung: Die Abarbeitung der Zuweisungsregel führt zunächst zur Auswahl einer Selektionsregel. Die Selektionsregel bestimmt, welche Akteure

aus der Organisationsstruktur zu selektieren sind. Der Aufruf der Organisationsstrukturverwaltung erfolgt daraufhin mit dem Bezeichner der gewünschten Selektionsregel.

4. Selektion der Aufgabenträger aus der Organisationsstruktur: Der Taskholder-Finder bestimmt mit Hilfe der Selektionsregel die Akteure, die für die Durchführung der Aktivität geeignet sind. Die Selektionsregel enthält dabei die von der Aktivität geforderten Eigenschaften, so daß nur Akteure mit den geforderten Eigenschaften selektiert werden.

5. Rückgabe der ermittelten Akteure an die Zuweisungsregelverwaltung

6. Weitere Auswahl von Akteuren aus der Menge der ermittelten Aufgabenträger: Die Ausführung einer Selektionsregel ermittelt evtl. nicht nur einen Akteur, sondern eine Menge von geeigneten Akteuren. Diese Menge kann aufgrund weiterer Kriterien eingeschränkt werden. Es könnte z.B. festgestellt werden, welcher der gefundenen Akteure zur Zeit die geringste Auslastung aufweist, so daß diesem die anstehende Aktivität zugewiesen werden kann und somit insgesamt die Arbeit gleichmäßig auf die vorhandenen Akteure verteilt wird. Hier ist auch der Einstiegspunkt für die oben angesprochenen ökonomischen und Motivations-Kriterien.

7. Rückgabe der Aufgabenträger an das Workflow-Management-System. Das Workflow-Management-System informiert nun die entsprechenden Aufgabenträger von der durchzuführenden Aktivität.

Die Zuweisungsregelverwaltung wird hier wie bei [**?**] der Organisationsverwaltung zugeschrieben. Im Prinzip stellt die Zuweisungsregelverwaltung das Bindeglied zwischen Prozeß- und Organisationsmodell dar, so daß sie auch zwischen der Organisationsverwaltung und dem Workflow-Management-System angesiedelt werden könnte.

Der Hauptunterschied der vorgestellten Organisationsverwaltung im Vergleich zu einer „herkömmlichen“ Organisationsverwaltung ist in der Organisationsstrukturverwaltungs-Komponente zu finden. In herkömmlichen Systemen wird die Organisationsstruktur z.B. in einer relationalen Datenbank abgelegt und die Selektion erfolgt z.B. durch SQL-Statements o.ä. Der hier vorgestellte Ansatz modelliert die Organisationsstruktur wissensbasiert. Das heißt, die Organisationsstruktur wird in einer Wissensbasis bestehend aus Fakten und Regeln in Form von Konzeptgraphen und den Konzept- und Relationstyphierarchien abgelegt. Die Selektionsregeln bestehen ebenfalls aus Konzeptgraphen und werden daher später als Anfragegraphen bezeichnet.

5.3 Entwurf der Organisationsstrukturverwaltung

Eine grundlegende Entscheidung bezüglich der Organisationsstrukturverwaltung ist die Verwendung von Konzeptgraphen zur wissensbasierten Modellierung der Organisationsstruktur. Damit werden die Modellierungsmöglichkeiten zunächst nur durch die Ausdrucksmächtigkeit der Konzeptgraphen beschränkt. Die zwei wesentlichen Arten von Konstruktionselementen sind hierbei Konzepte und Relationen. Eine vollständig freie Definition der Organisationsstruktur ist jedoch nicht sinnvoll. Es muß daher eine Ontologie in Form einer Konzept- und Relationstyphierarchie eingeführt werden, die eine Basis des gemeinsamen Verständnisses liefert und somit die Grundlage für die eigentliche Modellierung bildet. Eine vollständige und allgemeingültige Ontologie kann in dieser Arbeit nicht erstellt werden, denn die Bedürfnisse und Begriffsdefinitionen von Unternehmen sind teilweise zu unterschiedlich und zu umfangreich. Jeder Vorschlag einer Ontologie, der einen Anspruch auf Vollständigkeit erhebt, würde daher nur eine unnötige Einschränkung

bei der Modellierung bedeuten, so daß der wesentliche Vorteil dieses Ansatzes verlorengehen würde. Vielmehr ist zu erwarten, daß Firmen eigene Definitionen aufstellen, die exakt auf ihre Bedürfnisse zugeschnitten sind. Was jedoch erreicht werden kann, ist die Definition einer Basis von allgemeingültigen Begriffen, auf deren Grundlage beliebige Organisationen modelliert werden können. Wichtig ist dabei, daß diese Basis allgemein genug ist und Erweiterungen zuläßt, um einem Unternehmen die Freiheit zu lassen, ihre eigenen Bedürfnisse in die Ontologie einzubringen (idealerweise geschieht diese Erweiterung durch Subtypbildung bestehender Basistypen).

5.3.1 Basis-Metamodell der Organisationsstrukturverwaltung

Als Basis für weitere Begriffe wird zunächst ein einziger Konzepttyp definiert, nämlich der Typ *Agent*:

type AGENT(x) **is**
[UNIVERSAL: ∗x].

Das resultierende Metamodell ist in Abb. 5.2 zu sehen. Durch die Einführung des Konzepttyps *Agent* werden die Konzepte eines Konzeptgraphen in Agenten und Nicht-Agenten aufgeteilt. Agenten sind alle Konzepte, die dem Typ *Agent* oder evtl. später eingeführten Subtypen entsprechen (z.B. könnte man einen Akteur als einen menschlichen Agenten als Subtyp des Agententyps betrachten). Nicht-Agenten sind alle restlichen Konzepte, wie z.B. Organisationseinheiten, Fähigkeiten, usw. Die entsprechenden Typen dieser Nicht-Agenten werden durch Einbringen neuer Typen in die Ontologie bestimmt. Ebenso lassen sich beliebige Relationen durch Definition entsprechender Relationstypen einbringen. Dieses Metamodell ist daher zunächst als Basis für eigene Erweiterungen zu sehen.

Bereits alleine durch die Definition dieses einen Typs erhält man ein Metamodell, das trotz seiner Einfachheit eine Ausdruckmächtigkeit besitzt, die – mit einer Ausnahme – über alle analysierten Ansätze aus Kapitel 3 hinausgeht. Die meisten dieser Ansätze hatten mehr oder minder starke Einschränkungen bezüglich der Modellierungsmöglichkeiten aufgrund zu spezieller Metamodelle, d.h. die Metamodelle waren bereits auf konkrete Organisationsformen zugeschnitten. Nur das Metamodell nach Bußler (siehe Abs. 3.1.6) kann als ein allgemeingültiges Metamodell angesehen werden, das die Definition beliebiger Organisationsstrukturen zuläßt. Die hier vorgestellte Modellierung durch Konzeptgraphen besitzt eine ähnliche Mächtigkeit, wie der Bußler-Ansatz. Diese Ähnlichkeit kommt nicht von ungefähr, denn das Metamodell von Bußler (Abb. 3.6) und das Metamodell des Konzeptgraphen-Ansatzes (Abb. 5.2) sind durchaus vergleichbar. Agenttypen des Bußler-Metamodells entsprechen dem Agentkonzepttyp (und evtl. später definierten Subtypen dieses Typs), Nicht-Agenttypen können auf die restlichen Konzepttypen abgebildet werden und Organisationsbeziehungstypen entsprechen den Relationstypen. Ein kleines Beispiel soll die Ähnlichkeiten demonstrieren.

Bußler zeigt in [**?**], wie das FlowMark-Meta-Schema durch die von ihm vorgeschlagene Grammatik ausgedrückt werden kann. Hier ein Ausschnitt aus dieser Definition des Organisationsstrukturtyps *FlowMark*:

```
Organisationsstrukturtyp("FlowMark")
  Agenttyp("Person", "FlowMark")
    Instanzattribut("Name", "string", "Person", "FlowMark")
    Instanzattribut("Telephonenumber", "string",
                    "Person", "FlowMark")
  Nichtagenttyp("Role", "FlowMark")
```

```
    Organisationsbeziehungstyp("Related_role", "Person", "FlowMark",
8                              "Role", "FlowMark")
```

Betrachtet man den Agenttyp `Person`, so läßt sich dieser durch die folgende Konzepttypdefinition ausdrücken:

```
type PERSON(x) is
2    [AGENT: *x] -
         (NAME)→[WORT]
4        (TELEPHONENUMBER)→[WORT].
```

Eine Ausprägung dieses Typs könnte wie folgt aussehen:

```
[PERSON: #1234] -
    (NAME)→[WORT: "Hans Meyer"]
    (TELEPHONENUMBER)→[WORT: "0711−123456"].
```

Ebenso läßt sich ein Relationstyp definieren, der dem Organisationsbeziehungstyp *Related_role* entspricht:

```
relation RELATED_ROLE(x,y) is
    [PERSON: *x]→(LINK)→[ROLE: *y].
```

Theoretisch müßte man hierzu auch noch einen Konzepttyp `ROLE` definieren, worauf aber verzichtet wird.

Läßt man zunächst die Möglichkeit außer acht, daß innerhalb eines Organisationsstrukturtyps bei Bußler weitere Dinge wie z.B. Konsistenzregeln oder Operationen auf Typen oder Instanzen definiert werden können, so entspricht der definierte Organisationsstrukturtyp *FlowMark* weitestgehend der Menge aller Konzept- und Relationstypdefinitionen. Kleine Unterschiede ergeben sich z.B. aus der Möglichkeit des Bußler-Ansatzes, Beziehungen zwischen Organisationsbeziehungen zu definieren (die einzige Beziehung, die bei Konzeptgraphen zwischen Relationen definiert werden kann, ist eine Generalisierungsbeziehung) oder der Einschränkung seines Ansatzes auf binäre Relationen, die bei den Relationen eines Konzeptgraphen so nicht gegeben ist. Beide Ansätze unterscheiden sich von den anderen durch ihre Allgemeingültigkeit und Möglichkeit zur freien Definition der Organisationsstruktur.

Das vorgeschlagene Metamodell ist in dieser Form noch nicht direkt zur Modellierung einer Organisation zu verwenden. Es müssen zunächst – wie beim Bußler-Ansatz auch – weitere Begriffe in Form von Konzept- und Relationstypdefinitionen definiert werden, die für das Unternehmen von Relevanz sind. Aufgrund der Allgemeingültigkeit des Basis-Metamodells ist dies aber ohne Probleme durch einfache Subtypbildung bestehender Begriffe möglich. Abs. 5.6 zeigt eine mögliche Erweiterung um Fähigkeiten, Kompetenzen und Organisationseinheiten.

5.3.2 Entwurf der Wissensbasis

Informationen bezüglich der Organisation werden in der organisatorischen Wissensbasis abgelegt. In dieser Wissensbasis werden alle Informationen gespeichert, die für die Selektion eines Akteurs notwendig sind. Einfache Daten, wie z.B. die Adresse oder Telefonnummer eines Akteurs können dabei durchaus weiterhin in herkömmlichen Datenbanken abgelegt werden, falls sie nicht für die reine Selektion benötigt werden.

Die folgenden Abschnitte zeigen den Entwurf dieser Wissensbasis.

5.3.2.1 Fakten

Die Wissensbasis besteht zunächst aus einer Ontologie in Form je einer Konzept- und Relationstyphierarchie und einer Menge von Fakten in Form von Konzeptgraphen.

Definition 15 (Fakt) *Ein Fakt F ist ein Tupel (C, R, E) mit allen Eigenschaften aus Def. 11 (d.h. ein Fakt ist ein Konzeptgraph).*

Jedes Fakt beschreibt eine für die Aufgabenträgerzuweisung wichtige Information. Wichtige Informationen können z.B. die Fähigkeiten und Rechte eines Akteurs, die Unterstellungsverhältnisse, Abteilungszugehörigkeit, o.ä. sein. Wichtig ist dabei, daß nur wahre Aussagen in der Wissensbasis abgelegt werden, d.h. Aussagen, die konsistent mit der Realität sind. Welche Informationen wichtig für die Aufgabenträgerzuweisung sind, muß von Fall zu Fall entschieden werden. Abs. 5.6 beschäftigt sich näher damit, wie speziell Kompetenzen und Fähigkeiten modelliert werden können, um auf deren Basis die Aufgabenträgerzuweisung vorzunehmen.

5.3.2.2 Regeln

Theoretisch würden Fakten bereits ausreichen, um die Organisationsstruktur vollständig zu modellieren. Oft ist es aber sinnvoller und bequemer, bestimmte Sachverhalte durch Regeln auszudrücken. Zum Beispiel könnte die Zuordnung einer Menge von Rechten, die mit einer Rolle verbunden werden, bequem durch eine Regel auf den Rolleninhaber übertragen werden, damit später gezielt nach einer Person gesucht werden kann, die eine Menge von Rechten besitzt, anstatt nach einer Person zu suchen, die eine Rolle einnimmt (was, wie Abs. 3.3.5 gezeigt hat, bei Rollenänderungen negative Folgen haben kann). Eine solche Übertragung von Eigenschaften einer Rolle auf den Rolleninhaber könnte durch folgende Regel ausgedrückt werden: „Wenn ein Akteur X Rolle Y einnimmt und Rolle Y die Eigenschaft Z verleiht, dann hat Akteur X die Eigenschaft Z." (siehe auch Abs. 5.6.4).

Solche Regeln können auch in anderen Situationen sehr hilfreich sein. Zum Beispiel könnte damit folgendes ausgedrückt werden: „Der Leiter einer Abteilung ist der Vorgesetzte aller Angestellten, die für diese Abteilung arbeiten." Eine entsprechendes Mittel, solche Regeln auszudrücken, könnte wiederum helfen, Modellierungsaufwand zu vermeiden und konsistente Modelle der Organisationsstruktur zu erzeugen. Modellierungsaufwand wird vermieden, da z.B. oben die Relation *Vorgesetzter* nicht manuell eingefügt werden müßte. Die konsistente Modellierung wird unterstützt, da Regeln nun ebenfalls explizit modelliert werden können. Bei einer Änderung der Regel muß nun nicht mehr manuell im Modell gesucht werden, welche Relationen aufgrund der Regeländerung zu entfernen bzw. hinzuzufügen sind, wobei immer die Gefahr besteht, daß Relationen übersehen oder vergessen werden, was zu einem inkonsistenten Modell führen kann. Es genügt nun, die Regel zu ändern, und den Rest dem System zu überlassen. Es werden daher Regeln in Form von Konzeptgraphen als weiteres Element der Wissensbasis eingeführt, wie sie auch in ähnlicher Form in [**?**] verwendet werden.

Definition 16 (Regel) *Eine Regel G ist ein Tupel $(C, R, E, \tilde{R}, \tilde{E})$ mit C, R, E wie in Def. 11 (d.h. eine Regel ist ein Konzeptgraph) und*

- *$\tilde{R}$: eine Menge von Relationen (diese entsprechen der Konklusion der Regel)*
- *$\tilde{E} \in C \times \tilde{R} \times \mathbf{N}$*
- *$G' = (C, R \cup \tilde{R}, E \cup \tilde{E})$ ist ebenfalls ein Konzeptgraph nach Def. 11.*

In der graphischen Darstellung werden alle $r \in \tilde{R}$ grau hinterlegt dargestellt. Abb. 5.3 zeigt eine Regel, die dem oben genannten Beispiel „Wenn eine Person X Leiter einer Abteilung Y ist und eine Person Z für diese Abteilung Y arbeitet, dann ist X ein Vorgesetzter von Z." entspricht.

Der ϕ-Operator aus Def. 12 wird wie folgt für Regeln G erweitert:

1. Für alle $c_i \in C$ konstruiere Prädikate P_{c_i} wie in Def. 12 Schritt 1 oder Schritt 2.
2. Für alle Relationen $r_i \in R$ konstruiere Prädikate P_{r_i} wie in Def. 12 Schritt 3.
3. Für alle Relationen $s_i \in \tilde{R}$ konstruiere Prädikate P_{s_i} wie in Def. 12 Schritt 3.
4. $x_1, \ldots, x_m$ seien alle konstruierten Variablen; $P_{c_1}, \ldots, P_{c_n}$ alle Prädikate aus Schritt 1; $P_{r_1}, \ldots, P_{r_k}$ alle Prädikate aus Schritt 2; $P_{s_1}, \ldots, P_{s_l}$ alle Prädikate aus Schritt 3.

$$\phi(G) = \forall x_1 \ldots \forall x_m (P_{c_1} \wedge \cdots \wedge P_{c_n} \wedge P_{r_1} \wedge \cdots \wedge P_{r_k} \rightarrow P_{s_1} \wedge \cdots \wedge P_{s_l})$$

Die Abbildung der Regel G aus Abb. 5.3 auf eine prädikatenlogische Formel ergibt dann beispielsweise

$$\begin{aligned} \phi(G) \quad = \quad & \forall x \forall y \forall z (PERSON(x) \wedge ABTEILUNG(y) \wedge PERSON(z) \wedge \\ & LEITER(y, x) \wedge TEIL(y, z) \rightarrow VORGESETZTER(z, x)) \end{aligned}$$

Die Relation *VORGESETZTER* entspricht also der Konklusion, der restliche Graph der Prämisse der Regel. Zu beachten ist, daß nur Relationen als Konklusion zugelassen sind, da für diesen Fall entscheidbar ist, ob eine Aussage aus einer Menge von Fakten und Regeln folgt. Der allgemeine Fall dagegen, der auch Konzepte innerhalb der Konklusion zuläßt, ist nur semi-entscheidbar (siehe [**?**][**?**]).

Ein kleines Problem tritt bei dieser Art der Abbildung auf, wie das folgende Beispiel zeigt. Gegeben sei folgendes Fakt F und seine Abbildung auf eine Formel $\phi(F)$.

[PERSON: Meyer] −
(TEIL)←[ABTEILUNG: Einkauf]
(LEITER)←[ABTEILUNG: Einkauf].

$$\begin{aligned} \phi(F) \quad = \quad & PERSON(Meyer) \wedge ABTEILUNG(Einkauf) \wedge LEITER(Einkauf, Meyer) \wedge \\ & TEIL(Einkauf, Meyer) \end{aligned}$$

Mit Regel G von oben gilt $\phi(F) \cup \phi(G) \models VORGESETZTER(Meyer, Meyer)$, d.h. Meyer ist sein eigener Vorgesetzter. Mathematisch betrachtet ist die Folgerung natürlich richtig, trotzdem ist sie mit hoher Wahrscheinlichkeit nicht sinnvoll (es ist nicht anzunehmen, daß es gewünscht ist, daß eine Person ihr eigener Vorgesetzter ist). Die Mißinterpretation folgt aus einer zu ungenauen Definition der Unternehmensregel. Richtig ausgedrückt müßte es heißen „Wenn eine Person X Leiter einer Abteilung Y ist und eine Person $Z \neq X$ für diese Abteilung Y arbeitet, dann ist X ein Vorgesetzter von Z." Um diesen Sachverhalt ausdrücken zu können, wird daher ein Mittel benötigt, um die Ungleichheit von Person X und Person Z ausdrücken zu können. Es wird daher davon ausgegangen, daß jedes generische Konzept einer Regel und auch der Fakten für ein anderes Individuum steht, d.h. daß für zwei Konzepte $c_1 =$`[TYP_A: *x]` und $c_2 =$`[TYP_B: *y]` gilt: $c_1 \neq c_2 \iff x \neq y$

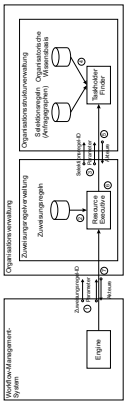

Abbildung 5.1: Ablauf der Aufgabenträgerzuweisung

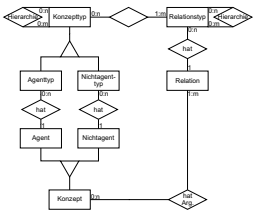

Abbildung 5.2: Basis-Metamodell der Organisationsstrukturverwaltung

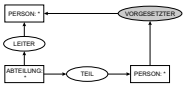

Abbildung 5.3: Regel

Um diesen Sachverhalt durch eine Formel ausdrücken zu können, wird eine Regel auf eine Formel der Prädikatenlogik erster Stufe mit Gleichheit abgebildet. Dazu wird Schritt 4 der ϕ-Operator-Erweiterung von Seite 68 wie folgt abgeändert:

4. $x_1, \ldots, x_m$ seien alle konstruierten Variablen; $P_{c_1}, \ldots, P_{c_n}$ alle Prädikate aus Schritt 1; $P_{r_1}, \ldots, P_{r_k}$ alle Prädikate aus Schritt 2; $P_{s_1}, \ldots, P_{s_l}$ alle Prädikate aus Schritt 3.

$$\begin{aligned}\phi(G) \quad = \quad & \forall x_1 \ldots \forall x_m (P_{c_1} \wedge \cdots \wedge P_{c_n} \wedge P_{r_1} \wedge \cdots \wedge P_{r_k} \wedge \neg(x_1 = x_2) \wedge \cdots \wedge \\ & \neg(x_{m-1} = x_m) \rightarrow P_{s_1} \wedge \cdots \wedge P_{s_l})\end{aligned}$$

Die Abbildung der Regel G aus Abb. 5.3 auf eine prädikatenlogische Formel ergibt nun

$$\begin{aligned}\phi(G) \quad = \quad & \forall x \forall y \forall z (PERSON(x) \wedge ABTEILUNG(y) \wedge PERSON(z) \wedge \\ & LEITER(y, x) \wedge TEIL(y, z) \wedge \neg(x = y) \wedge \neg(x = z) \wedge \neg(y = z) \rightarrow \\ & VORGESETZTER(z, x))\end{aligned}$$

Nun gilt nicht mehr $\phi(F) \cup \phi(G) \models VORGESETZTER(Meyer, Meyer)$, da $Meyer = Meyer$ gilt und sich somit die Prämisse der Regel zu $false$ auswertet. Soll die Folgerung VORGESETZTER(Meyer,Meyer) gelten, so müßte die Regel G so verändert werden, daß die beiden Konzepte `[PERSON: *]` zu einem Konzept verschmolzen werden.

5.3.2.3 Formale Definition der Wissensbasis

Mit den bisher eingeführten Definitionen läßt sich die Wissensbasis wie folgt formal definieren.

Definition 17 (Wissensbasis) *Eine Wissensbasis ist ein Tupel* $(H_C, H_R, \{F_1, \ldots, F_m\}, \{R_1, \ldots, R_n\})$ *mit*

- H_C*: eine Konzepttyphierarchie*
- H_R*: eine Relationstyphierarchie*
- $\{F_1, \ldots, F_m\}$*: eine Menge von Fakten im Sinne von Def. 15*
- $\{R_1, \ldots, R_n\}$*: eine Menge von Regeln im Sinne von Def. 16*

5.3.3 Selektion von Aufgabenträgern

Die Selektion von Aufgabenträgern bedeutet die Bestimmung einer Menge von Akteuren mit bestimmten Eigenschaften. Die Eigenschaften eines Akteurs werden dabei durch die Konzeptgraphen der Organisationsstrukturverwaltung (organisatorische Wissensbasis) definiert. Zwei Möglichkeiten werden hier betrachtet, um Akteure zu selektieren:

- Operationen auf den Konzeptgraphen selbst (Selektion durch Graph-Matching).
- Umsetzung der Konzeptgraphen in ein PROLOG-Programm und Selektion der Akteure durch Stellen einer (PROLOG-)Anfrage.

5.3.3.1 Anfragegraphen

Bevor jedoch eine entsprechende Anfrage gestellt werden kann, muß zunächst eine Selektionsregel definiert werden. Natürlichsprachlich könnte eine solche Anfrage z.B. lauten: „Welche Akteure sprechen Englisch und sind verantwortlich für den Bereich Lebensversicherungen?“ Um die Durchgängigkeit zu waren, ist es sinnvoll, nicht nur die Modellierung der Aufbauorganisation durch Konzeptgraphen vorzunehmen, sondern auch Anfragen durch Konzeptgraphen zu formulieren. Obige Anfragen muß also in einen Konzeptgraphen transformiert werden. Konzeptgraphen, die Anfragen darstellen, werden hier als Anfragegraphen bezeichnet.

Definition 18 (Anfragegraph) *Ein Anfragegraph ist ein Tupel* $(C, R, E, root, P)$.

- C,R *und* E *entsprechen Def. 11, d.h. ein Anfragegraph ist ein Konzeptgraph.*
- $root \in C$*: Wurzelknoten der Anfrage mit* $referent(c) = *$.
- $P \subseteq C \setminus root$*: eine Menge von Parametern mit der Eigenschaft* $\forall c \in P : referent(c) = *$.

root hat die Aufgabe, den zu findenden Akteur zu repräsentieren. Die Parameter stehen für zusätzliche Informationen, die eine erfolgreiche Anfrage zurückliefert. Abb. 5.4 zeigt ein einfaches Beispiel für einen Anfragegraphen. Die Wurzel wird hierbei durch doppelte Umrandung hervorgehoben, die Parameter sind an den benannten Variablen zu erkennen (z.B. `*sprache`).

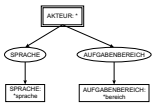

Abbildung 5.4: Anfragegraph

5.3.3.2 Ergebnis einer Anfrage

Läßt man eine Anfrage Q gegen eine Wissensbasis $W = (H_C, H_R, \{F_1, \ldots, F_m\}, \{R_1, \ldots, R_n\})$ laufen, so ist die Anfrage erfolgreich, wenn gilt:

$$\Omega(H_C, H_R) \cup \phi(F_1) \cup \cdots \cup \phi(F_m) \cup \phi(R_1) \cup \cdots \cup \phi(R_n) \models \phi(Q)$$

Gesucht ist allerdings keine Ja/Nein-Antwort, sondern eine Menge von Individuen, die eingesetzt für die generischen Marker des Wurzelknotens und der Parameter eine wahre Aussage ergeben. Sei also Q ein Anfragegraph mit dem Wurzelkonzept *root* und den Parametern $p_1, \ldots, p_n$. Sei außerdem $Q_{[root/i_0][p_1/i_1]\ldots[p_n/i_n]}$ der Konzeptgraph, der aus Q hervorgeht, indem man $referent(root)$ durch den individuellen Marker i_0, $referent(p_1)$ durch den individuellen Marker i_1, usw. ersetzt (dabei muß jeweils $type(c) :: i$ gelten). Dann ist das Resultat der Anfrage die folgende Menge:

$$\{(i_0, \ldots, i_n) \in I^{n+1} \quad | \quad \Omega(H_C, H_R) \cup \{\phi(F_1), \ldots, \phi(F_m)\} \cup \{\phi(R_1), \ldots, \phi(R_n)\} \models Q_{[root/i_0][p_1/i_1]\ldots[p_n/i_n]}\}$$

Das Ergebnis für die Anfrage aus Abb. 5.4 könnte z.B. die Menge {(Meyer, Deutsch, Lebensversicherungen), (Meyer, Englisch, Lebensversicherungen), (Schmidt, Deutsch, EDV)} sein.

Man kann einen Anfragegraph auch als eine Art Schablone ansehen, aus der erst der eigentliche Anfragegraph gebildet wird. Hierzu ersetzt man die generischen Referenten eines oder mehrerer Parameter durch individuelle Marker und erhält dadurch wieder einen Anfragegraph. Nimmt man den Graphen aus Abb. 5.4 als Beispiel, so könnte man das Konzept `[SPRACHE: *sprache]` durch `[SPRACHE: Englisch]` ersetzen. Das Ergebnis der Anfrage wäre dann eine Menge von Tupeln, bestehend aus der Bezeichnungen eines Akteurs und dessen Aufgabenbereich (z.B. {(Meyer,Lebensversicherungen)}).

5.3.4 Selektion durch Graph-Matching

In diesem Abschnitt soll gezeigt werden, wie man durch Operationen auf Konzeptgraphen Aufgabenträger selektieren kann. Die Operation, die zur eigentliche Selektion verwendet wird, wird als Graph-Matching bezeichnet. Diese Operation bildet auf eine bestimmte Art Knoten des Anfragegraphen auf Knoten eines Fakts der Wissensbasis ab. Gelingt das, so wurde ein Match zwischen dem Anfragegraphen und dem Fakt ermittelt und aus der gefundenen Abbildung läßt sich das Ergebnis der Anfrage, d.h. der gesuchte Aufgabenträger und evtl. weitere gesuchte Parameter, ableiten.

Für zwei Konzepte c_i und c_j wird im folgenden $c_i \leq c_j$ geschrieben, falls gilt:

- $type(c_i) \leq type(c_j)$
- $referent(c_j) = *$ oder $referent(c_i) = referent(c_j)$

Für zwei Relationen r_i und r_j wird $r_i \leq r_j$ geschrieben, falls $type(r_i) \leq type(r_j)$ gilt.

Definition 19 (Projektion) *Sei $Q = (C_Q, R_Q, E_Q, root, P)$ ein Anfragegraph und $F = (C_F, R_F, E_F)$ ein Fakt. Dann ist eine Projektion $\Pi_{Q \to F}$ ein Paar von Funktionen $f : C_Q \to C_F$ und $g : R_Q \to R_F$ mit den folgenden Eigenschaften:*

1. *f und g sind totale Funktionen, d.h. jeder Knoten aus Q wird auf Knoten aus F abgebildet.*
2. $\forall c_Q \in C_Q : \; f(c_Q) \leq c_Q$
3. $\forall r_Q \in R_Q : \; g(r_Q) \leq r_Q$
4. $\forall (c_Q, r_Q, n_Q) \in E_Q \; \exists (c_F, r_F, n_F) \in E_F : c_F = f(c_Q) \wedge r_F = g(r_Q) \wedge n_F = n_Q$

$\Pi_{Q \to F}(v) = w$ wird geschrieben, wenn die Projektion den Knoten v des Anfragegraphen auf den Knoten w des Fakts abbildet.

Abb. 5.5 zeigt ein einfaches Beispiel für eine Projektion eines Anfragegraphen auf ein Fakt. Hierbei ist auch zu erkennen, wie sich das Ergebnis der Anfrage aus der gefundenen Projektion ergibt, nämlich aus den individuellen Markern der Konzepte des Fakts, die dem Wurzelknoten und den Parameterknoten des Anfragegraphen zugeordnet sind. Es gilt z.B. $akteur = referent(\Pi_{Q \to F}(root))$, wobei *akteur* der gesuchte Akteur mit den gegebenen Eigenschaften ist (im Beispiel gilt also $akteur = Meyer$). Die Werte der Parameter ergeben sich auf die gleiche Art und Weise aus der Abbildung der Parameter-Konzepte. Im allgemeinen können dabei mehrere Projektionen eines Anfragegraphen auf ein Fakt existieren.

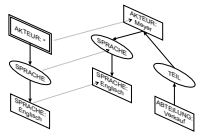

Abbildung 5.5: Beispiel für eine Projektion eines Anfragegraphen auf ein Fakt

5.3.4.1 Korrektheit

Um Projektionen für die Selektion von Aufgabenträgern einsetzen zu können, muß zunächst sichergestellt sein, daß eine Projektion nur korrekte Ergebnisse liefert, d.h. wenn sich ein Anfragegraph Q auf ein Fakt der Wissensbasis projizieren läßt, so muß $\phi(Q)$ logisch aus der Wissensbasis folgern.

Satz 2 *[Korrektheit der Projektion] Gegeben sei ein Anfragegraph Q, ein Fakt F und passende Typhierarchien H_C und H_R. Falls eine Projektion $\Pi_{Q \to F}$ existiert, dann gilt auch $\Omega(H_C, H_R) \cup \phi(F) \models \phi(Q)$.*

Bevor Satz 2 bewiesen wird, wird ein weiterer Satz eingeführt, der den Beweis wesentlich vereinfacht:

Satz 3 *G und H seien Konzeptgraphen. Dann gilt:*

$$\textit{Es existiert } \Pi_{G \to H} \qquad \Longleftrightarrow \qquad H \leq G$$

Beweis: siehe [**?**]

■

Beweis zu Satz 2: Durch Anwendung der Sätze 1 und 3 folgt:

$$\text{Es existiert } \Pi_{Q \to F} \quad \Rightarrow_3 \quad F \leq Q \quad \Rightarrow_1 \quad \Omega(H_C, H_R) \cup \phi(F) \models \phi(Q)$$

■

Somit ist gezeigt, daß eine Projektion korrekte Ergebnisse liefert.

5.3.4.2 Vollständigkeit

Interessant ist aber nicht nur die Korrektheit, sondern auch die Vollständigkeit. Das heißt, wenn sich eine Aussage aus der Wissensbasis herleiten läßt, dann soll auch eine Projektion des Anfragegraphen auf ein Fakt der Wissensbasis existieren. Ein einfaches Gegenbeispiel zeigt allerdings, daß keine Projektion gefunden werden kann, wenn die notwendigen Informationen auf mehrere Fakten verteilt sind. Der Anfragegraph aus Abb. 5.6 kann nicht vollständig abgebildet werden, denn weder F_1 noch F_2 enthält die *vollständige* Information, die für die Beantwortung der Anfrage notwendig wäre. Da sich eine Projektion immer auf genau zwei Graphen bezieht (ein Anfragegraph und *ein* Fakt), kann nie eine Projektion gefunden werden, obwohl die zur Beantwortung notwendige Information in der Wissensbasis enthalten ist, wenn auch verteilt auf mehrere Fakten.

Das Problem kann dadurch behoben werden, daß vor der Selektion, d.h. vor Durchführung der Matching-Operation, eine Sequenz von Join-Operation auf individuellen Konzepten auf den Fakten der Wissensbasis durchgeführt wird.

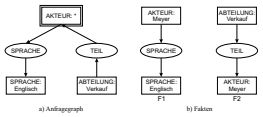

Abbildung 5.6: Verteilte Informationen und Anfragegraphen-Projektion

Definition 20 (Join auf individuellen Konzepten) *Gegeben seien zwei Konzeptgraphen* $G_1 = (C_1, R_1, E_1)$ *und* $G_2 = (C_2, R_2, E_2)$. $c_1 \in C_1$ *und* $c_2 \in C_2$ *seien individuelle Konzepte mit* $referent(c_1) = referent(c_2)$ *und es gilt* $type(c_1) \leq type(c_2)$ *oder* $type(c_2) \leq type(c_1)$. *Ein* Join auf individuellen Konzepten *ausgeführt auf den Graphen* G_1 *und* G_2 *ist eine Operation, die einen Graphen* G_3 *durch folgende Konstruktion erzeugt:*

1. *Sei* t_{sub} *der Konzepttyp mit* $t_{sub} = type(c_1) \sqcup type(c_2)$. *Führe Restriktionen auf* c_1 *und* c_2 *aus, so daß* $type(c_1) = type(c_2) = t_{sub}$ *gilt. Die resultierenden Konzepte seien* c_1' *und* c_2', *die resultierenden Graphen* G_1' *und* G_2'.

2. *Verschmelze* c_1' *und* c_2'. *Der resultierende Graph ist* G_3.

Wir schreiben $G_1 \oplus G_2 = G_3$.

Anmerkungen:

- Die in Schritt 1 beschriebenen Restriktionen können auf jeden Fall durchgeführt werden, denn $referent(c_1) = referent(c_2)$ sind konform zu t_{sub}, da aufgrund der Voraussetzung $type(c_1) \leq type(c_2)$ oder $type(c_2) \leq type(c_1)$ gilt: $t_{sub} = type(c_1)$ oder $t_{sub} = type(c_2)$ (es muß in Schritt 1 also eigentlich nur eine Restriktion durchgeführt werden).

- Das Verschmelzen in Schritt 2 kann durchgeführt werden, da nach den Restriktionen $type(c_1) = type(c_2)$ und nach Voraussetzung $referent(c_1) = referent(c_2)$ gilt.

- Ein Join auf individuellen Konzepten kann auch auf demselben Graphen ($G_1 = G_2$) ausgeführt werden.

- Die Definition beschreibt nur eine Join-Operation auf genau zwei Konzepten. Man kann aber gleichzeitig alle Konzepte c_i und c_j verschmelzen, für die $referent(c_i) = referent(c_j)$ gilt und erhält somit einen Graph G_3, in dem keine zwei individuellen Konzepte identische Referenten haben. Im Prinzip entspricht dies einfach einer Sequenz von Join-Operationen auf jeweils zwei Konzepten mit identischem Referenten, d.h. der unten stehende Beweis gilt ebenso für diesen Fall. Wir nennen diese Operationen einen *maximalen* Join auf individuellen Konzepten und schreiben $G_1 \otimes G_2 = G_3$.

Da sowohl die Restriktion als auch die Verschmelzung Spezialisierungsregeln sind, ist der resultierende Graph G_3 eine Spezialisierung von G_1 und G_2 ($G_3 \leq G_1$ und $G_3 \leq G_1$). Nach Abs. 4.5.2 gilt *im allgemeinen* nicht $\Omega(H_C, H_R) \cup \phi(G_i) \cup \phi(G_j) \models \phi(G_k)$, falls G_k durch die Anwendung von Spezialisierungsregeln aus G_i und G_j hervorgegangen ist. Damit wäre diese Operation unbrauchbar, da sie die Wissensbasis in eine nicht äquivalente Form transformieren würde. Daß dem nicht so ist, d.h. daß die durchgeführte Transformation sehr wohl eine äquivalente Wissensbasis erzeugt, zeigt der Beweis zum folgenden Satz.

Satz 4 *Gegeben: Zwei Konzeptgraphen G_1 und G_2 und die zugehörigen Typhierarchien H_C und H_R. Dann gilt:*

$$\Omega(H_C, H_R)_\wedge \wedge \phi(G_1) \wedge \phi(G_2) \quad \equiv \quad \Omega(H_C, H_R)_\wedge \wedge \phi(G_1 \oplus G_2)$$

Beweis ($\Rightarrow$): Zunächst ist folgendes für Schritt 1 aus Def. 20 zu zeigen:

$$\Omega(H_C, H_R) \cup \phi(G_1) \cup \phi(G_2) \models \Omega(H_C, H_R) \cup \phi(G_1') \cup \phi(G_2')$$

Es gelten die Bezeichnungen aus Def. 20. Falls $type(c_1) = type(c_2) = t_{sub}$ gilt, dann gilt auch $G_1' = G_1$ und $G_2' = G_2$. Die Behauptung ist somit erfüllt.

Sei nun $type(c_1) \neq type(c_2)$. Dann ist t_{sub} unter der Voraussetzung $type(c_1) \leq type(c_2)$ oder $type(c_2) \leq type(c_1)$ gleich $type(c_1)$ oder gleich $type(c_2)$. O.B.d.A. gelte $t_{sub} = type(c_1)$. Dann gilt auch $G_1' = G_1$. Es ist daher nur noch folgendes zu zeigen:

$$\Omega(H_C, H_R) \cup \phi(G_1) \cup \phi(G_2) \models \phi(G_2')$$

$\phi(G_2)$ und $\phi(G_2')$ sind bis auf das Prädikat P_{c_2} identisch, das aus dem Konzept hervorgeht, auf dem die Restriktion durchgeführt wurde. Sei P'_{c_2} das entsprechende Prädikat aus $\phi(G_2')$ nach der Restriktion. P'_{c_2} ist in $\phi(G_1)$ enthalten, da nach der Restriktion $type(c_1) = type(c_2) = t_{sub}$ und nach Voraussetzung $referent(c_1) = referent(c_2)$ gilt. Ein Modell für $\phi(G_1) \wedge \phi(G_2)$ ist daher auch ein Modell für $\phi(G_2')$. q.e.d.

Zu Schritt 2 ist zu zeigen:

$$\Omega(H_C, H_R) \cup \phi(G_1') \cup \phi(G_2') \models \Omega(H_C, H_R) \cup \phi(G_3)$$

$\phi(G_3)$ und $\phi(G_1') \wedge \phi(G_2')$ unterscheiden sich nur dadurch, daß $\phi(G_1') \wedge \phi(G_2')$ das Prädikat P'_{c_2} doppelt enthält. Da $P'_{c_2} \wedge P'_{c_2} \equiv P'_{c_2}$ gilt, ist jedes Modell von $\phi(G_1') \wedge \phi(G_2')$ auch ein Modell von $\phi(G_3)$. q.e.d.

Beweis ($\Leftarrow$): Diese Richtung folgt durch Anwendung des Satzes 1, da die durchgeführten Operationen (in dieser Richtung) nur Generalisierungen sind, und somit $G_3 \leq G_1$ und $G_3 \leq G_2$ gilt. Ein Modell für $\Omega(H_C, H_R) \cup \phi(G_3)$ ist somit auch ein Modell für $\Omega(H_C, H_R) \cup \phi(G_1)$ und für $\Omega(H_C, H_R) \cup \phi(G_2)$. q.e.d.

■

Aus den Fakten aus Abb. 5.6b wird durch Anwendung der vorgestellten Join-Operation der Graph $F_3 = F_1 \otimes F_2$:

F_3 = [AKTEUR: Meyer] −
(SPRACHE) → [SPRACHE: Englisch]
(TEIL) ← [ABTEILUNG: Verkauf].

Eine Wissensbasis kann also durch eine Sequenz von Join-Operationen auf individuellen Konzepten in eine äquivalente Form überführt werden. Seien $\{F_i, \ldots, F_m\}$ die Fakten der Wissensbasis. Dann wird diese Menge durch eine Sequenz von Join-Operationen auf inidviduellen Konzepten in eine zweite Menge von Fakten $\{F_j', \ldots, F_n'\}$ transformiert, so daß für alle individuellen Konzepte $c_1 \in F_i'$ und $c_2 \in F_j'$ mit $c_1 \neq c_2$ gilt: $referent(c_1) \neq referent(c_2)$. Das heißt, kein Paar von individuelle Konzepten hat gleiche Referenten. Folgender Algorithmus führt diese Transformation durch:

```
   // F: Eine Menge von Fakten.
 2 // Rückgabewert: Eine Menge von Fakten, deren Konzepte paarweise verschieden
   //      Referenten besitzen.
 4 procedure join_on_individuals(F) returns set of facts {
       F' ← ∅;
 6     while F ≠ ∅ do
           f = (C_f, R_f, E_f) ∈ F;
 8         F ← F \ f;
           for each g = (C_g, R_g, E_g) ∈ F do
10             for each c_f ∈ C_f do
                   t_{c_f} ← type(c_f);
12                 S_{c_f} ← ∅;
                   for each c_g ∈ C_g do
14                     t ← t_{c_f} ⊔ type(c_g);
                       if (t = t_{c_f} or t = type(c_g)) and
16                         referent(c_f) = referent(c_g) and referent(c_f) ≠ * then
                           t_{c_f} ← t;
18                         S_{c_f} ← S_{c_f} ∪ c_g;
                       fi
20                 od
               od
22             for each c_f ∈ C_f do
                   if S_{c_f} ≠ ∅ then
24                     restrict(c_f, t_{c_f});
                       for each c ∈ S_{c_f} do
26                         restrict(c, t_{c_f});
                       od
28                     f ← join_all_concepts(S_{c_f} ∪ c_f);
                       F ← F \ g;
30                 fi
               od
32         od
           F' ← F' ∪ f;
34     od
       return F';
36 }
```

Listing 5.1: Join auf individuellen Konzepten

In jedem Schritt der while-Schleife ab Z. 6 wird ein Fakt f aus der Menge der ursprünglichen Fakten entfernt. Ab Z. 9 wird versucht, f mit jedem Fakt aus der verbleibenden Faktenmenge auf individuellen Konzepten zu joinen. Hierzu wird für jedes Konzept c_f aus f die Menge S_{c_f} der Konzepte des Fakts g bestimmt, so daß alle Konzept aus S_{c_f} die Voraussetzungen für einen individuellen Join mit c_f erfüllen. Da das mehrere Konzepte aus g sein können, wird außerdem der kleinste Subtyp aller dieser Konzepte bestimmt und in t_{c_f} vermerkt. Nach Beendigung der Schleife aus Z. 10 – Z. 21 ist nun für jedes Konzept $c_f \in f$ bekannt, mit welchen Konzepten aus g es individuell gejoint werden könnte. Mit diesen Informationen kann ab Z. 22 der eigentliche individuelle Join durchgeführt werden. Es erfolgen hier die notwendigen Restriktionen auf allen Konzepten, die gejoint werden sollen (Z. 24, 26), bevor alle diese Konzepte in Z. 28 vereinigt

werden. Konnte mindestens ein Konzept aus f mit einem aus g gejoint werden, so enthält nun der vereinigte Graph (Z. 28) alle Informationen aus (dem ursprünglichen) f und g. Daher muß der Graph g später nicht mehr betrachtet werden und kann in Z. 29 aus F entfernt werden.

Wenn alle Fakten der Wissensbasis zusammen n Konzepte besitzen, dann müssen insgesamt $O(n^2)$ Konzept-Vergleiche durchgeführt werden. Da die Selektion zeitkritisch ist, wird die Transformation nicht bei jeder Selektion durchgeführt. Statt dessen wird die Transformation offline durchgeführt, wenn sich wichtige Änderungen an der Wissensbasis ergeben haben. Die entstandene Wissensbasis wird hier als eine *kompilierte Wissensbasis* bezeichnet. Selektionen (Graph-Matching-Operationen) werden nur auf der kompilierten Wissensbasis ausgeführt, da nur für diese die wichtige Eigenschaft aus Satz 5 gilt (siehe unten).

Nach der obigen Operation gibt es keine individuellen Konzepte mit gleichen Referenten mehr. Kann diese Eigenschaft für alle Fakten der Wissensbasis sichergestellt werden, so gilt der folgende Satz:

Satz 5 (Vollständigkeit der Projektion) *Gegeben:*

- *Eine Wissensbasis* $\Omega(H_C, H_R) \cup \phi(F_1) \cup \cdots \cup \phi(F_n)$.[1] *Es gilt dabei für je zwei individuelle Konzepte* $c_1 \in F_i$ *und* $c_2 \in F_j$ *mit* $c_1 \neq c_2$*:* $referent(c_1) \neq referent(c_2)$ *(d.h. individuelle Konzepte haben paarweise verschiedene Referenten)*
- *Ein Anfragegraph* Q*.*

Dann gilt:

$$\Omega(H_C, H_R) \cup \phi(F_1) \cup \cdots \cup \phi(F_n) \models Q \quad \Rightarrow \quad \textit{Es existiert } \Pi_{Q \to F_i} \textit{ mit } i \in \{1, \ldots, n\}$$

Beweis: siehe [**?**]

■

Es wird nochmals explizit darauf hingewiesen, daß die Vollständigkeit nur dann gilt, wenn alle individuellen Konzepte der Wissensbasis paarweise verschiedene Referenten besitzen, was nach der Ausführung des Algorithmus 5.1 der Fall ist. Ein Beispiel für die Nicht-Vollständigkeit im allgemeinen Fall ist in Abb. 5.10 unten zu sehen (siehe auch Ausführen auf S. 87 ff).

Aufgrund der Korrektheit und Vollständigkeit der Projektion stellt also ein Graph-Matching-Algorithmus, der alle Projektionen eines Anfragegraphen auf die Fakten der Wissensbasis aufzählt, zumindest theoretisch ein geeignetes Mittel zur Selektionen von Aufgabenträgern dar.

5.3.4.3 Komplexität der Matching-Operation

Zur praktischen Anwendbarkeit des Graph-Matching-Ansatzes muß nicht nur die Korrektheit und die Vollständigkeit des Ergebnisses sichergestellt werden, auch die Komplexität dieser Operation muß betrachtet werden. Es genügt dabei, das zugehörige Entscheidungsproblem „Existiert eine Projektion $\Pi_{Q \to F}$ für einen gegebenen Anfragegraphen Q und ein gegebenes Fakt F?“ zu betrachten. Man kann jedem „herkömmlichen“ Graphen G einen Konzeptgraphen zuordnen, indem man für jeden Knoten aus G einen Konzeptknoten und für jede Kante eine binäre Relation konstruiert, die die entsprechenden Konzeptknoten als Argumente besitzt. Alle Konzepte und alle Relationen sind dabei vom selben Typ. Durch diese (polynomielle) Reduktion könnte man nun

[1] Eine Wissensbasis mit Regeln läßt sich in eine äquivalente Wissensbasis transformiert, die nur Fakten enthält (siehe Abs. 5.3.4.6). Daher werden Regeln hier zunächst nicht berücksichtigt.

unter Verwendung der Matching-Operation ein Entscheidungsverfahren für das bekanntermaßen NP-vollständige Graphenmorphismus-Problem konstruieren (siehe [**?**]). Durch die Reduktion des Graphenmorphismus-Problems auf das oben genannte Entscheidungsproblem für die Matching-Operation ist damit die NP-Vollständigkeit der Matching-Operation bewiesen.

Unter der bisher nicht bewiesenen aber sehr wahrscheinlichen Annahme $P \neq NP$ ist somit kein effizienter Algorithmus, d.h. kein Algorithmus mit polynomiellen Zeitaufwand, realisierbar. Da durch die Join-Operation auf individuellen Konzepten, die bei der Kompilierung der Wissensbasis angewandt wird, evtl. sehr große Graphen entstehen, wiegt diese Tatsache um so schwerer, denn die Matching-Operation wird damit auf einen evtl. sehr großen Fakt-Graphen (und auf einen i.a. sehr kleinen Anfragegraphen) ausgeführt.

Zwei Möglichkeiten, um mit einem NP-Problem umzugehen, sind die folgenden Vorgehensweisen:

- Man versucht nicht, das NP-Problem zu lösen, sondern analysiert das Problem auf Sonderfälle, die für die Aufgabenstellung evtl. ausreichend sind und für die P-Algorithmen existieren.
- Man nimmt an, daß sich der exponentielle Zeitaufwand in der Praxis aufgrund „günstiger Umstände“ nicht einstellt. Man setzt also auf die Gutartigkeit des Problems und hofft, daß der „Worst Case“ in der Praxis nicht eintritt.

Bereits durch die Beschränkung auf baumförmige Anfragegraphen läßt sich ein P-Algorithmus erzielen. Der zweite verfolgte Lösungsansatz geht von der Annahme aus, daß die Einschränkungen, die durch die Konzept- und Relationstypen und Referenten entstehen, bereits eine wesentliche Einschränkung der zu betrachtenden Fälle zulassen und somit auch ein NP-Algorithmus in der Praxis anwendbar ist.

5.3.4.4 Ein P-Algorithmus für die Matching-Operation

Wie bereits angedeutet, wird hier nun versucht, einen P-Algorithmus durch Einschränkungen auf den verwendeten Konzeptgraphen „zu erkaufen“. Im allgemeinen werden Anfragegraphen sehr einfach geartet sein, wohingegen Fakten je nach modellierter Organisationsstruktur und angestrebtem Detaillierungsgrad relativ komplex werden können. Es sind kaum Einschränkungen bei den Fakten denkbar, die nicht zu wesentlichen Einschnitten bei den Modellierungsmöglichkeiten führen würden. Daher wird versucht, die Einschränkungen auf die Anfragegraphen zu beschränken.

Ein Algorithmus, der nur geringe Einschränkungen zur Folge hat und trotzdem polynomielle Berechnungszeit garantiert, ist der Algorithmus nach Mugnier und Chein, der in [**?**] beschrieben wird. Dieser Algorithmus kann nur angewandt werden, wenn der Anfragegraph eine Baumform aufweist. Dabei sind allerdings Mehrfachkanten zwischen benachbarten Relationen und Konzepten ausdrücklich erlaubt. Es darf nur keinen Zyklus geben, der sich über mehr als zwei Knoten erstreckt. Die Richtung der Kanten, die durch die Numerierung der Kanten sowieso nur noch proforma besteht, ist hierbei nicht von Belang. Wurde ein Wurzelknoten festgelegt (beim Anfragegraphen bietet sich hierfür der Wurzelknoten der Anfrage an), so ist der Baum eindeutig durch die Vater-Sohn-Beziehungen zwischen den Knoten festgelegt.

Algorithmus Der Algorithmus besteht aus zwei über Kreuz rekursiven Funktionen. Die erste Funktion `match_root` erhält als Eingabeparameter den Wurzelknoten des ersten Graphen (in diesem Fall des Anfragegraphen) und eine Menge von Knoten des zweiten Graphen (des Fakts),

hier Kandidaten genannt, für die überprüft wird, ob es möglich ist, den Wurzelknoten auf einen oder mehrere Kandidaten abzubilden. Durch Aufrufe der Funktion `match_rel`, die für die Abbildung von Relationen zuständig ist und die wiederum `match_root` aufruft, wird schließlich ein Matching für den gesamten Anfragegraphen erzeugt, wobei `match_root` immer Abbildungen für den durch den Wurzelknoten induzierten Teilbaum berechnet. Weitere Details lassen sich am besten an einem einfachen Beispiel erläutern, das nach dem eigentlichen Algorithmus vorgestellt wird. An dieser Stelle wird auch die erzeugte Datenstruktur vorgestellt, aus der sich das Ergebnis ermitteln läßt.

```
// root: Wurzelknoten (ein Konzeptknoten). Jeder Wurzelknoten
//       induziert einen Teilbaum des Anfragegraphen, für den eine
//       Projektion berechnet wird.
// candidates: Eine Menge von Konzeptknoten des Fakts, für die
//       überprüft wird, ob der Wurzelknoten auf einen oder mehrere
//       dieser Knoten abbildbar ist.
// Rückgabewert: Eine Menge M ⊆ candidates, auf die root
//       abgebildet werden kann.
procedure match_root(root, candidates) returns set of concepts {
    candidates ← {c ∈ candidates | c ≤ root};
    successors ← successors(root);
    if candidates = ∅ or successors = ∅ then
        return candidates;
    fi
    for each s ∈ successors do
        E_s ← ∅;
        for each c ∈ candidates do
            W_{c,s} ← {r ∈ neighbours(c) | type(r) ≤ type(s)∧
                                          ∀i ∈ {1,...,|s|} : s^i = root → r^i = c};
            E_s ← E_s ∪ W_{c,s};
        od
        E_s ←match_rel(s, E_s);
        for each c ∈ candidates do
            V_{c,s} ← W_{c,s} ∩ E_s;
        od
        candidates ← {c ∈ candidates | V_{c,s} ≠ ∅};
    od
    return candidates;
}

// r: Ein Relationsknoten des Anfragegraphen.
// candidates: Eine Menge von Relationsknoten des Fakts. Für diese
//       Knoten soll überprüft werden, ob r auf sie abbildbar ist.
// Rückgabewert: Eine Menge M ⊆ candidates von Relationen auf die
//       r abbildbar ist.
procedure match_rel(r, candidates) returns set of relations {
    successors ← successors(r);
    if candidates = ∅ or successors = ∅ then
        return candidates;
    fi
```

```
      for each s ∈ successors do
42        E_s ← ∅;
          for each t ∈ candidates do {
44            W_{t,s} ← {c ∈ neighbours(t) | ∀i ∈ {1,...,|r|} : r^i = s → t^i = c};
              E_s ← E_s ∪ W_{t,s};
46        od
          E_s ←match_root(s, E_s);
48        candidates ← {t ∈ candidates | E_s ∩ W_{t,s} ≠ ∅};
      od
50    return candidates;
  }
```

Listing 5.2: P-Algorithmus für die Matching-Operation nach [**?**]

Erläuterung Der Algorithmus wird anhand eines Beispiels erläutert. Gegeben seien der Anfragegraph Q und das Fakt F aus Abb. 5.7. Für die Konzepte und Relationen gilt die folgende Tabelle.

c_1	$\geq$	d_1, d_5	r_1	$\geq$	s_1
c_2	$\geq$	d_2	r_2	$\geq$	s_2, s_3, s_4
c_3	$\geq$	d_2			
c_4	$\geq$	d_3, d_4, d_5			

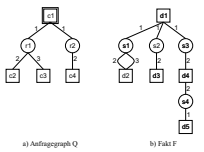

Abbildung 5.7: Selektion mit Baum-Anfrage

Folgende Aufrufe werden im Laufe der Berechnung durchgeführt:

Schritt	Aufruf	Rückgabewert
1.	`match_root`(c_1,$\{d_1, d_2, d_3, d_4, d_5\}$)	$\{d_1\}$
1.1	`match_rel`(r_1,$\{s_1\}$)	$\{s_1\}$
1.1.1	`match_root`(c_2,$\{d_2\}$)	$\{d_2\}$
1.1.2	`match_root`(c_3,$\{d_2\}$)	$\{d_2\}$
1.2	`match_rel`(r_2,$\{s_2, s_3\}$)	$\{s_2, s_3\}$
1.2.1	`match_root`(c_4,$\{d_3, d_4\}$)	$\{d_3, d_4\}$

In Schritt 1 wird die Anfrage gestartet, indem `match_root` der Wurzelknoten der Anfrage und alle Konzeptknoten des Fakts (die Kandidaten) übergeben werden. In Z. 10 werden die Knoten herausgefiltert, die Spezialisierungen des Wurzelknotens darstellen (im Beispiel sind das nur die Knoten d_1 und d_5), denn nur auf diese kann der Wurzelknoten abgebildet werden. In der for-Schleife ab Z. 15 wird nun versucht, alle Nachfolger-Relationen des Wurzelknotens auf Relationsknoten des Fakts abzubilden. Es wird also zunächst versucht, r_1 auf die Relationen s_1,s_2,s_3 und s_4 abzubilden, da diese Relationen Nachfolger von d_1 und d_5 sind. Es werden dabei zuerst in Z. 18 die Relationsknoten herausgefiltert, die vom Typ her auf die Nachfolger-Relation s des

Wurzelknotens passen. Das ist im Beispiel nur der Knoten s_1. Außerdem muß in Z. 19 sichergestellt werden, daß alle Relationen mit den gleichen Kanten mit ihrem Vorgänger (dem Bild des Wurzelknotens) verbunden sind, wie die Nachfolger-Relation im Anfragegraphen. Da r_1 mit der Kante 1 mit c_1 verbunden ist, muß also s_1 ebenfalls mit der Kante 1 mit dem entsprechenden Bild von c_1 also d_1 verbunden sein, was auch der Fall ist. Alle möglichen Bilder der jeweiligen Nachfolger-Relation s werden in der Menge E_s zusammengefaßt. Diese Menge bildet die Kandidaten-Menge für die Abbildung der Nachfolger-Relation, die an die Funktion `match_rel` übergeben wird. `match_rel` filtert alle Relationsknoten heraus, die tatsächlich als Bild von s dienen können (siehe unten). In Z. 24 wird hiermit die Menge $V_{c,s}$ berechnet. Diese Menge enthält alle Relationen des Fakts, auf die s abgebildet werden kann, wenn *root* auf c abgebildet wird. Diese Menge wird in der unten besprochenen Datenstruktur gespeichert, denn sie wird später für die Bildung des eigentlichen Ergebnisses benötigt. In Z. 26 werden alle Kandidaten entfernt, die keine Nachfolger-Relationen besitzen, auf die der Nachfolger von *root* abgebildet werden könnte, denn wenn eine totale Abbildung gesucht wird, dann muß jeder Knoten aus Q abgebildet werden, also natürlich auch die Nachfolger-Relationen von *root*. Im Beispiel wird d_5 aus der Kandidaten-Menge entfernt, denn r_1 kann nicht auf s_4 abgebildet werden. Somit existiert auch keine Projektion, wenn c_1 auf d_5 abgebildet wird. Im nächsten Schleifendurchlauf müssen daher nur noch die Nachfolger von d_1, also s_2 und s_3, berücksichtigt werden.

Die Funktion `match_rel` ist der Funktion `match_root` sehr ähnlich. Falls es sich bei r um ein Blatt handelt oder keine Kandidaten vorhanden sind, ist nichts zu tun (Z. 38,39). Danach wird wieder ab Z. 41 versucht, die Nachfolger von r auf Konzepte aus F abzubilden. In Z. 44 wird dazu wie in Z. 19 überprüft, ob das potentielle Bild des Nachfolgers mit den gleichen Kanten mit dem Bild von r verbunden ist, mit denen auch r mit seinem Nachfolger verbunden ist. Zum Beispiel wird überprüft, ob s_1 mit der Kante 2 mit d_2 verbunden ist, denn r_1 ist auch mit der Kante 2 mit c_2 verbunden. Da das der Fall ist, kann r_1 auf s_1 abgebildet werden, falls auch c_2 auf d_2 abgebildet werden kann. Ob wiederum das der Fall ist, wird durch den Aufruf von `match_root` in Z. 47 festgestellt, denn nur wenn c_2 auf d_2 abgebildet werden kann liefert der Aufruf d_2 im Ergebnis zurück. Es gilt auch hier wieder, daß nur die Kandidaten als Bild dienen können, die es zulassen, daß auch alle Nachfolger-Konzepte des Knotens aus dem Anfragegraphen abgebildet werden können (Z. 48).

Die eigentlichen Projektionen können aus der Datenstruktur abgeleitet werden, die während der Ausführung von `match_root` und `match_rel` erstellt wird. Ein Ausschnitt aus dieser Datenstruktur ist in Abb. 5.8 zu sehen. Zu jedem Konzeptknoten aus dem Anfragegraph wird die Menge der Knoten aus dem Fakt gespeichert, auf die der entsprechende Konzeptknoten abgebildet werden kann. Zum Beispiel kann c_1 nur auf d_1 abgebildet werden. Diese Menge entspricht genau dem Rückgabewert der Funktion `match_root`. Für die Projektion gilt dann $\Pi_{Q\to F}(c_1) = d_1$. Falls c_1 auf d_1 abgebildet wird, so beschreibt die mit dieser Abbildung über einen Zeiger verbundene Liste, wie die Nachfolger von c_1 abgebildet werden können. Da es hier u.U. wieder pro Nachfolger mehrere Möglichkeiten gibt, ist mit jedem Nachfolger eine weitere Liste verbunden. Diese Liste enthält die Menge $V_{c,s}$ die oben beschrieben wurde. Falls c_1 auf d_1 abgebildet wird, kann r_1 nur auf s_1 abgebildet werden. r_2 kann allerdings auf s_2 oder s_3 abgebildet werden. Durch Traversieren dieser Datenstruktur und Kombination der Abbildungen erhält man alle möglichen Abbildungen. Das Beispiel enthält genau zwei Projektionen:

- $\{(c_1 \mapsto d_1), (c_2 \mapsto d_2), (c_3 \mapsto d_2), (c_4 \mapsto d_3), (r_1 \mapsto s_1), (r_2 \mapsto s_2)\}$
- $\{(c_1 \mapsto d_1), (c_2 \mapsto d_2), (c_3 \mapsto d_2), (c_4 \mapsto d_4), (r_1 \mapsto s_1), (r_2 \mapsto s_3)\}$

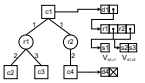

Abbildung 5.8: Datenstruktur Mugnier-Chein-Algorithmus

Einschränkungen Welche Einschränkungen muß man beim Mugnier-Chein-Algorithmus in Kauf nehmen? Alle Anfragegraphen mit Zyklen sind verboten, d.h. daß z.B. ein Anfragegraph, wie der in Abb. 5.9a dargestellte Anfragegraph Q, nicht mehr erlaubt ist, da er einen Zyklus besitzt. Dieser Graph läßt sich allerdings ohne Probleme in einen Graphen Q' transformieren, der keinen Zyklus enthält und trotzdem das richtige Ergebnis liefert. Das wurde in Abb. 5.9b) dadurch erreicht, daß das Konzept `[ABTEILUNG: Verkauf]` aufgetrennt wurde (im Prinzip also die Umkehrung der Join-Operation und somit eine Generalisierung des Anfragegraphen). In diesem Fall war die Einschränkung auf einen zyklenfreien Graphen also gar keine wirkliche Einschränkung. Der transformierte Anfragegraph Q' liefert weiterhin korrekte Ergebnisse, denn es gilt: $\phi(Q) \equiv \phi(Q')$. Im Prinzip handelt es sich bei der Umformung um eine Generalisierung, d.h. es gilt $Q \leq Q'$. Im *allgemeinen* gilt dabei *nicht* $\Omega(H_C, H_R) \cup Q' \models Q$, im speziellen aber schon, denn bei der Aktion, die Q' in Q (rück-)transformiert, handelt es sich um nichts anderes, als um einen Join auf individuellen Konzepten (siehe Seite 74ff), und diese Operation überführt Konzeptgraphen nach Satz 4 in äquivalente Konzeptgraphen. Ersetzt man dagegen das Konzept `[ABTEILUNG: Verkauf]` in Q durch das generische Konzept `[ABTEILUNG: *]`, so ist keine äquivalente Umformung mehr möglich. Jedes Auftrennen des Zykluses würde in $\phi(Q')$ eine neue existentiell gebundene Variable einführen, die in $\phi(Q)$ nicht vorhanden ist (z.B. $\phi(Q') = \exists x \exists y \ldots ABTEILUNG(x) \wedge ABTEILUNG(y) \ldots$). Eine Einschränkung ergibt sich also dann, wenn ein Zyklus existiert, der nur generische Konzepte enthält (jeder andere Zyklus läßt sich an einem individuellen Konzept auftrennen). Ob solche Anfragen in der Praxis sehr häufig anzutreffen sind, kann eher bezweifelt werden. Trotzdem gibt es sicher den einen oder anderen Fall, bei dem sich diese Einschränkung auswirkt. Diverse Workarounds sind allerdings für diese seltenen Fälle denkbar (z.B. ein nachträglicher Vergleich auf Gleichheit; z.B. $x = y$ im obigen Beispiel).

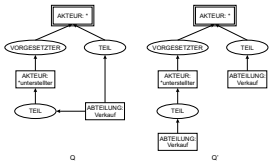

Abbildung 5.9: Transformation in einen baumförmigen Anfragegraphen

Komplexität Der Mugnier-Chein-Algorithmus hat eine Komplexität von $O(|E_Q||E_F|)$ wobei $|E_Q|$ und $|E_F|$ die Anzahl der Kanten des Anfragegraphs bzw. des Fakts ist (Beweis siehe [**?**]). Somit hat die Suche nach einem Akteur in einer Wissensbasis mit n Fakten die folgende Komplexität:

$$O(n|E_Q||E_F|) \tag{5.1}$$

5.3.4.5 Ein NP-Algorithmus für die Matching-Operation

Ein NP-Algorithmus wird hier zunächst unter der Vermutung verwendet, daß sich der exponentielle Worst-Case-Charakter in der Praxis nicht einstellen wird und sich der Algorithmus daher „gutartig“ verhält. Diese Vermutung stützt sich auf die Tatsache, daß in jeder praxisrelevanten Konzept- und Relationstyphierarchie viele zueinander inkompatible Typen anzutreffen sind (d.h. es gilt weder $t_1 \leq t_2$ noch $t_2 \leq t_1$). Konzepte und Relationen dieser Typen scheiden von vornherein für eine Abbildung aus, d.h. ein Konzept oder eine Relation des Typs t_1 könnte unter obiger Annahme nicht auf t_2 abgebildet werden. Außerdem sind zumindest einige, oft sogar der Großteil der Konzepte individuell. Für individuelle Konzepte des Anfragegraphen gibt es nur eine Möglichkeit, diese auf einen Faktknoten abzubilden, denn die kompilierte Wissensbasis besteht aus einer Menge von Fakten, deren Konzepte paarweise unterschiedliche Referenten besitzen (vgl. auch Abs. 5.3.4.2). Damit könnte z.B. das Konzept `[SPRACHE: Englisch]` des Anfragegraphen nur auf das insgesamt nur einmal in der Wissensbasis vorhandenen gleiche Konzept abgebildet werden. Diese Einschränkung wirkt sich allerdings nur sehr stark aus, wenn auch wirklich viele Konzepte des Anfragegraphen individuell sind, was aber in vielen Praxissituationen zu erwarten ist. Insgesamt können daher viele Knoten eines Fakts des Wissensbasis bei der Suche nach einer Projektion ignoriert werden, d.h. die zu betrachtende Knotenmenge wird bereits vor der Matching-Operation stark eingeschränkt. Auf der anderen Seite ist zu erwarten, daß gerade die Fakten der kompilierten Wissensbasis durch die Join-Operation auf individuellen Konzepten sehr groß werden können. Da immer Beispiel konstruierte werden können, die zu einer exponentiellen Laufzeit führen, genau so wie auch Beispiele mit polynomieller Laufzeit konstruiert werden können, kann nur ein Praxistest mit möglichst relevanten Beispielen zeigen, ob ein NP-Algorithmus in der Praxis auch wirklich anwendbar ist (siehe Abs. 6.2).

Algorithmus Der in [**?**] beschriebene Algorithmus nach Ullmann ist nicht speziell auf Konzeptgraphen zugeschnitten. Seine eigentliche Aufgabe ist das Finden von Subgraphen-Isomorphismen, einem Problem, das sich wie folgt formulieren läßt:

Gegeben: Zwei Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$
Frage: Enthält G_2 einen Subgraphen G_2', d.h. eine Menge $V_2' \subseteq V_2$ und eine Menge $E_2' = \{e = (v_1, v_2) \in E_2 \mid v_1 \in V_2' \wedge v_2 \in V_2'\}$, der isomorph zu G_1 ist. Das heißt, gesucht ist eine bijektive Funktion $f : V_1 \rightarrow V_2'$, so daß gilt: $(u, v) \in E_1$ gdw. $(f(u), f(v)) \in E_2'$

Dieser Algorithmus wird leicht abgewandelt bzw. erweitert, so daß er die Besonderheiten eines Konzeptgraphen (Konzept- und Relationstypen, Referenten, Kantennumerierung) beachtet werden und der Algorithmus somit zur Suche nach Projektionen verwendet werden kann. Aufgrund der Bijektivität der Abbildung – einer Eigenschaft, die so nicht für eine Projektion gefordert wurde – bestehen leichte Einschränkungen bezüglich der Anfragegraphen und der Fakten, die aber nicht gravierend sind (siehe unten).

Auch dieses Mal wird zunächst der Algorithmus vorgestellt, bevor eine Erläuterung erfolgt.

```
  // Q: Anfragegraph
2 // F: Fakt
  procedure match(Q = (C_Q, R_Q, E_Q, root, P), F = (C_F, R_F, E_F)) {
4     Cand ← ∅;
      for each c ∈ C_Q do
6         Cand_c ← {x ∈ C_F | x ≤ c};
      od
8     for each r ∈ R_Q do
```

```
        Cand_r ← {x ∈ R_F | x ≤ r ∧ ∀i ∈ {1,...,|r|} : x^i ≤ r^i};
    od
    Unmapped ← C_Q ∪ R_Q;
    Match ← ∅;
    find_next_mapping(Unmapped, Cand, Match);
}

// Unmapped: Noch nicht abgebildete Knoten des Anfragegraphen.
// Cand: Abbildung von Knoten aus Unmapped auf eine Menge von
//       Knoten des Fakts, auf die der jeweilige Knoten abgebildet
//       werden könnte.
// Match: Abbildungen von Knoten des Anfragegraphen auf Knoten
//       des Fakts.
procedure find_next_mapping(Unmapped, Cand, Match) {
    if Unmapped = ∅ then
        save(Match);
        return;
    fi
    src ∈ Unmapped;
    Unmapped ← Unmapped \ src;
    for each img ∈ Cand_src do
        Match_src ← img;
        Cand' = Cand;
        for each n ∈ Unmapped do
            Cand'_n ← Cand'_n \ img;
        od
        if check_mapping(Unmapped, Cand') then
            find_next_mapping(Unmapped, Cand');
        fi
        Match_src ← ∅;
    od
    Unmapped ← Unmapped ∪ {src};
}

// Unmapped: Eine Menge von Knoten des Anfragegraphen, die noch
//       nicht abgebildet wurden.
// Cand: Abbildung von Knoten aus Unmapped auf eine Menge von
//       Knoten des Fakts, auf die der jeweilige Knoten abgebildet
//       werden könnte. Diese (möglichen) Abbildungen werden von
//       dieser Operation angepasst, so dass sie konsistent mit
//       den Abbildungen aus Match sind.
// Match: Abbildung von Knoten des Anfragegraphen auf Knoten
//       des Fakts.
procedure check_mapping(Unmapped, Cand, Match) returns boolean {
    for each s ∈ Unmapped do
        for each i ∈ Cand_s do
            for each (src, img) ∈ Match do
                if s ∈ C_Q and src ∈ R_Q then
                    if not check_consistency((s,i),(src,img)) then
```

```
58                              Cand_s ← Cand_s \ i;
                            fi
60                      fi
                        else if s ∈ R_Q and src ∈ C_Q then
62                          if not check_consistency((src, img), (s, i)) then
                                Cand_s ← Cand_s \ i;
64                          fi
                        fi
66
                    od
68              od
                if Cand_s = ∅ then
70                  return false;
                fi
72          od
            return true;
74  }

76  // (c_src, c_img): Abbildung eines Konzepts aus C_Q auf ein Konzept
    //       aus C_F.
78  // (r_src, r_img): Abbildung einer Relation aus R_Q auf eine Relation
    //       aus R_F.
80  // Rückgabewert: true, wenn c_src und r_src durch Kanten mit den
    //       gleichen Nummern miteinander verbunden sind wie c_img
82  //       und r_img; false sonst.
    procedure check_consistency((c_src, c_img), (r_src, r_img)) returns boolean {
84      return ∀n ∈ {1, ..., |r_src|} : (r^n_src = c_src ∧ r^n_img = c_img) ∨ (r^n_src ≠ c_src ∧ r^n_img ≠ c_img);
    }
```

Listing 5.3: NP-Algorithmus für die Matching-Operation frei nach [**?**]

Erläuterung Der Ullmann-Algorithmus benutzt eine Verfahren, das als Backtracking bezeichnet wird, d.h. er benutzt eine Tiefensuche auf dem Berechnungsbaum. Er geht solange im Berechnungsbaum vorwärts, bis eine Lösung gefunden wurde oder feststeht, daß von dieser Stelle aus keine weiteren Lösungen zu erreichen sind. Dann vollzieht er einen Rückschritt im Berechnungsbaum und verfolgt einen anderen Ast. Jeder Schritt im Berechnungsbaum ist dabei gleichzusetzen mit der erfolgreichen Abbildung eines Knotens des Anfragegraphen.

Die Suche wird durch Aufruf der Operation `match` mit dem Anfragegraphen Q und dem Fakt F als Parameter gestartet. Zunächst wird die Datenstruktur $Cand$ initialisiert. Diese Datenstruktur speichert für jeden Knoten des Anfragegraphen eine Menge von Knoten des Fakts, auf die der jeweilige Knoten abgebildet werden könnte ($Cand_x$ liefert die Menge aller Knoten des Fakts, auf die x abgebildet werden könnte). In Z. 5–7 wird für jeden Konzeptknoten aus Q die Menge der Knoten aus F bestimmt, so daß die Knoten einer Menge jeweils Spezialisierungen des zugehörigen Knotens des Anfragegraphen sind. Die Hoffnung ist hierbei, daß diese Mengen jeweils aus den bereits oben genannten Gründen sehr klein sind. Ebenso werden diese Mengen in den Zeilen 8–10 für jeden Relationsknoten bestimmt, wobei hier nicht nur der Typ der Relation berücksichtigt wird, sondern auch die Argumente der Relation, was die zu betrachtende Knotenmenge zusätzlich reduziert. Auch hier sind möglichst kleine Mengen sehr positiv bezüglich des

späteren Zeitaufwandes. Die Datenstruktur $Match$ besteht aus einer Menge von Knotenabbildungen und repräsentiert somit die Funktionen f und g einer Projektion. Die Menge $Unmapped$ nimmt alle Knoten auf, die bisher noch nicht abgebildet wurden, für die $Match$ also noch nicht definiert ist.

Die Operation `find_next_mapping` versucht, einen Knoten aus $Unmapped$, also einen bisher nicht abgebildeten Knoten des Anfragegraphen, auf einen Knoten des Fakts abzubilden. Ist die Menge $Unmapped$ leer (Test in Z. 23–25), so wurden alle Knoten des Anfragegraphen bereits abgebildet, es wurde also eine Projektion gefunden, und die gefundenen Projektion kann gespeichert werden. Ansonsten wird ein Knoten aus $Unmapped$ ausgewählt (Z. 27) und gleichzeitig aus $Unmapped$ entfernt, denn bei einem späteren rekursiven Aufruf von `find_next_mapping` muß für diesen Knoten keine Abbildung mehr gefunden werden. Ab Z. 29 wird nun versucht, diesen Knoten des Anfragegraphen auf einen Knoten des Fakts abzubilden. Dabei werden alle Knoten durchgegangen, die bei der Initialisierung der Datenstruktur $Cand$ als kompatibel zu diesem Knoten vermerkt wurden. $Cand_{src}$ enthält genau diese Knoten für den Knoten src. In Z. 30 werden der Match um die Abbildung des gewählten Knotens des Anfragegraphen erweitert. Damit dies auch eine korrekte Erweiterung des bisher gefundenen Matches darstellt, darf die Menge $Cand_v$ für jeden Knoten v des Anfragegraphen nur die Knoten des Fakts enthalten, die auch wirklich eine gültige Erweiterung des bisher gefundenen Matchings darstellen. Bei der Initialisierung wurde bereits dafür gesorgt, daß nur Spezialisierungen des Knotens des Anfragegraphens als Bild dieses Knotens in Frage kommen. Diese Bedingung alleine genügt allerdings nicht, denn es muß auch sichergestellt sein, daß jede Kante des Anfragegraphen in gleicher Form auch in der Abbildung (also im Fakt) enthalten ist. Wenn zwei Knoten v_1 und v_2 des Anfragegraphen z.B. durch die Kanten 1 und 3 verbunden sind, so muß das für ihre Abbildung auch gelten, d.h. auch zwischen $\Pi_{Q\to F}(v_1)$ und $\Pi_{Q\to F}(v_2)$ müssen *genau* die Kanten 1 und 3 (nicht mehr und nicht weniger) vorhanden sein. Beim ersten Aufruf von `find_next_mapping` ist das automatisch der Fall (es existieren noch keine Abbildungen). Sobald aber eine Abbildung $src \mapsto img$ dem Match hinzugefügt wurde, muß die Datenstruktur $Cand$ entsprechend angepaßt werden. Zunächst wird die gesamte Datenstruktur in Z. 31 kopiert, da sonst beim Rückschritt aus der Rekursion dafür gesorgt werden müßte, daß die Datenstruktur den gleichen Inhalt wie beim Aufruf besitzt. Dann werden alle Vorkommen von img entfernt (Z. 32–34), denn es darf nur genau ein Knoten des Anfragegraphen auf diesen Knoten des Fakts abgebildet werden.[2] Die weiter unten erläuterte Funktion `check_mapping` stellt nun sicher, daß $Cand'$ nur noch solche Knoten des Fakts enthält, die den Match zu einem weiteren gültigen Match erweitern könnten. Diese Funktion liefert $true$, falls das der Fall ist. In diesem Fall wird durch einen rekursiven Aufruf von `find_next_mapping` versucht, den nächsten Knoten des Anfragegraphen abzubilden. Liefert `check_mapping` dagegen $false$, so kann der Match nicht mehr zu einer Projektion erweitert werden. Man muß diesen Match daher nicht weiter betrachten (also auch nicht versuchen, weitere noch nicht abgebildete Knoten abzubilden), sondern kann sofort eine neue Abbildung für src versuchen. Dazu wird die gerade gefundene Abbildung in Z. 38 zurückgenommen und im nächsten Schleifendurchlauf eine andere getestet.

Die Funktion `check_mapping` ist zum einen dafür zuständig, die Datenstruktur $Cand$ so anzupassen, daß sie für eine gültige Erweiterung des bisher gefundenen Abbildungen dienen kann, und zum anderen sorgt sie dafür, daß möglichst frühzeitig erkannt wird, wann ein Abbildung nicht zu einer Projektion erweitert werden kann. Die Anzahl der durchzuführenden Berechnungen ist dabei sehr stark davon abhängig, wie früh erkannt werden kann, wann sich weitere Schritten nicht mehr lohnen. Nur so besteht die Chance, den Berechnungsbaum möglichst weit oben zu beschneiden, und somit möglichst viele Rechenschritte zu sparen, die sowieso zu keiner

[2] Zur Erinnerung: Der Ullmann-Algorithmus sucht Subgraph-Isomorphismen.

Lösung geführt hätten. Diese Funktion testet für jeden noch nicht abgebildeten Knoten s, ob die jeweils zugehörige Menge $Cand_s$ Knoten des Fakts enthält, auf die s nicht mehr abgebildet werden kann, da eine bereits gefundenen Abbildung aus $Match$ das verhindert. Das heißt, es muß für jede bereits gefundenen Abbildung $src \mapsto img$ aus $Match$ überprüft werden, ob unter dieser Voraussetzung $s \mapsto i$ noch eine gültige Abbildung darstellen würde. Ist das nicht der Fall, so wird der entsprechende Knoten entfernt (Z. 58,63). Die Funktion `check_consistency` führt die eigentliche Überprüfung für ein Paar von Abbildungen durch. Hierbei wird genau die oben genannte Bedingung untersucht, die besagt, daß im Anfragegraphen vorhandene Kanten genau so auch im Zielgraphen (im Fakt) vorhanden sein müssen. In Z. 69–71 wird überprüft, ob es einen noch nicht abgebildeten Knoten des Anfragegraphen gibt, der unter dem bisher gefundenen Match nicht mehr abbildbar ist (seine Kandidatenmemge ist dann leer). Ist dies der Fall, so muß die bisher gefundenen Abbildung nicht mehr weiter untersucht werden, denn bei einer Projektion müssen alle Knoten des Anfragegraphen abgebildet werden. Man spart sich dadurch, wie bereits beschrieben, Rechenschritte, die zu keiner Lösung geführt hätten.

Komplexität C_Q und C_F sei die Menge der Konzepte des Anfragegraphen bzw. die entsprechende Menge des Fakts, R_Q und R_F sind die Relationsmengen dieser beiden Graphen. Um die Kandidatenmengen zu initialisieren sind $O(|C_Q||C_F|+|R_Q||R_F|)$ Schritte notwendig, denn jedes Konzept des Anfragegraphen muß mit jedem des Fakts verglichen werden und für die Relationen gilt das entsprechende, wobei hier zusätzlich die Argumente verglichen werden, wobei man aber davon ausgehen kann, daß meist nur binäre Relationen verwendet werden. Der schlimmste Fall tritt ein, wenn alle Konzepte und alle Relationen den gleichen Typ haben und außerdem der Graph insgesamt stark verbunden ist. Dann müssen in `find_next_mapping` jeweils $O(|C_F|)$ oder $O(|R_F|)$ Abbildungen überprüft werden. Da für jede Projektion diese Funktion $|C_Q|$ mal für Konzeptabbildungen und $|R_Q|$ mal für Relationsabbildungen aufgerufen werden muß, ergeben sich insgesamt $O(|C_F|^{|C_Q|}\ |R_F|^{|R_Q|})$ Aufrufe, wobei jedes Mal die Funktion `check_mapping` aufgerufen wird, die selbst wieder $O((|C_Q|+|R_Q|)^2)$ Schritte benötigt, um die Kandidatenmengen anzupassen. Besteht die Wissensbasis aus n Fakten, so ergibt sich somit die Gesamtkomplexität zu

$$O(n|C_F|^{|C_Q|}\ |R_F|^{|R_Q|}(|C_Q|+|R_Q|)^2) \tag{5.2}$$

Diese Betrachtung ist eher theoretischer Natur. Messungen sollen zeigen, ob dieser exponentielle Zeitaufwand tatsächlich für den konkret vorliegenden Fall realistisch ist (siehe Abs. 6.2.1).

Einschränkungen Außer der bereits angesprochenen u.U. exponentiellen Laufzeit des Algorithmus ist eine weitere Einschränkung anzumerken. Eine Projektion kann durch diesen Algorithmus nur gefunden werden, wenn eine 1:1-Beziehung zwischen den Knoten des Anfragegraphen und denen des Fakts besteht. Formal betrachtet sind die in Def. 19 definierten Funktionen f und g bijektiv, es handelt sich bei einer gefundenen Projektion also um einen Subgraphen-Isomorphismus (ein nach [**?**] NP-vollständiges Problem, daher auch die exponentielle Laufzeit, zumindest für allgemeine Graphen). Um diese spezielle Projektion von der allgemeinen Projektion zu unterscheiden, wird der Begriff der *Iso-Projektion* eingeführt:

Definition 21 (Iso-Projektion) *Eine Iso-Projektion $\Pi_{Q\rightarrowtail F}$ ist eine Projektion nach Def. 19, wobei zusätzlich die Funktionen f und g bijektiv sind.*

Diese 1:1-Beziehung[3] hat gewisse Auswirkungen auf die Anfragegraphen. Betrachtet man das Beispiel aus Abb. 5.10, so kann hier aus folgenden Gründen mit dem Ullmann-Algorithmus keine vollständige Projektion gefunden werden:

- c_2 kann nur auf c_6 *oder* c_7 abgebildet werden. In jedem Fall kann dann entweder für r_1 oder für r_2 keine Abbildung mehr gefunden werden.
- Wenn c_3 auf c_8 abgebildet wird, dann kann nicht c_4 gleichzeitig auch auf c_8 abgebildet werden. Das gleiche gilt für c_3, wenn c_4 auf c_8 abgebildet wird.

Der erste Punkt ist dabei kein spezielles Problem des Ullmann-Algorithmus, sondern ist vielmehr auf die Definition der Projektion selbst zurückzuführen (eine Projektion ist ein Paar von *Funktionen*, und somit kann ein Knoten des Anfragegraphen nur auf *einen* Knoten eines Fakts abgebildet werden). Dieses Problem tritt in der kompilierten Wissensbasis so nicht auf, denn hier haben alle individuellen Konzepte paarweise verschiedene Referenten, was in F bei den Konzepten c_6 und c_7 offensichtlich nicht der Fall ist (zur Erinnerung: Nur wenn alle individuellen Konzepte der Fakten paarweise verschiedene Referenten besitzen, liefert die Projektion ein vollständiges Ergebnis; siehe Abs. 5.3.4.2).

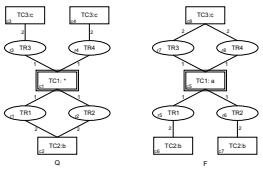

Abbildung 5.10: Bijektivität der Abbildung verhindert Projektion

Der zweite Punkt tritt dagegen nur aufgrund der Bijektivität der Funktion f der Iso-Projektion auf. Um einen zu Q äquivalenten Anfragegraphen zu erhalten, der eine bijektive Abbildung zuläßt, müssen nur die Konzepte c_3 und c_4 miteinander verschmolzen werden. Dies entspricht wieder einem Join auf individuellen Konzepten und liefert daher einen zu Q äquivalenten Anfragegraphen Q'. Allgemein wird daher auch für Anfragegraphen die Forderung erhoben, daß keine zwei verschiedenen individuellen Konzepte c_1 und c_2 eines Anfragegraphen gleiche Referenten besitzen dürfen (diese Forderung gilt *nicht* für baumförmige Anfragegraphen und Graph-Matching mit dem Mugnier-Chein-Algorithmus, denn hier sind individuelle Konzepte mit gleichen Referenten durchaus sinnvoll; siehe Abs. 5.3.4.4 und Abb. 5.9). Dasselbe gilt auch für Relationen. Das heißt, der Anfragegraph darf keine redundanten Relationen enthalten, da diese sonst eine Abbildung verhindern könnten.

5.3.4.6 Graph-Matching und Regeln

Bisher wurden bei der Selektion durch Graph-Matching die Regeln einer Wissensbasis ignoriert. Es wurde zwar die Vollständigkeit der Projektion bewiesen, aber eben nur für Wissensbasen ohne Regeln. Es wurde aber bereits angedeutet, daß sich eine Wissensbasis mit Regeln wie sie in Def. 16 definiert wurden in eine äquivalente Wissensbasis ohne Regeln transformieren läßt. In [**?**] wird ein ähnliches Vorgehen für allgemeine Regeln in Form von Konzeptgraphen vorgeschlagen

[3]Quasi das Gegenteil der 1:1-Beziehung, nämlich die n:m-Beziehung wird auch als *Folding* bezeichnet. Beim Folding können einem Knoten des einen Graphen viele Knoten des anderen Graphen zugeordnet werden und umgekehrt.

und die Korrektheit und Vollständigkeit dieses Verfahrens bewiesen. Wie die angesprochene Transformation in eine regelfreie Form durch Graph-Matching geschehen kann, soll hier nun gezeigt werden.

Prinzip der Transformation Das Prinzip der Transformation läßt sich folgendermaßen beschreiben: Zunächst wird eine Regel ausgewählt und für diese Regel überprüft, ob die Fakten der Wissensbasis die Prämisse der Regel erfüllen. Ist dies der Fall, so kann die Konklusion als neues Fakt in die Wissensbasis aufgenommen werden, es entsteht also eine neue Menge von Fakten. Dieses Vorgehen wird für alle Regeln solange wiederholt, bis die Anwendung einer beliebigen Regel keine neue Information mehr in die Wissensbasis einbringt, d.h. bis alle Informationen explizit in den Fakten der Wissensbasis enthalten sind. Dazu muß natürlich entscheidbar sein, wann dieser Zeitpunkt gekommen ist. Für die in dieser Arbeit verwendeten Regeln ist dies der Falls (siehe unten). Die Regeln können nun ignoriert werden, da die Fakten alle Informationen enthalten.

Algorithmus Zunächst ist eine Regel nach Def. 16 nichts anderes als ein Konzeptgraph. Das heißt, man kann auch die Matching-Operation auf eine Regel R und auf ein Fakt F anwenden und erhält somit eine Menge von Abbildungen der Knoten aus R auf Knoten aus F (die Relationsknoten der Konklusion werden hierbei nicht abgebildet!). Als Beispiel soll die in Abb. 5.11 links dargestellte Regel R und das rechts dargestellte Fakt F (zunächst ohne die hellgrau hinterlegte Relation) dienen. Die Regel hat die folgende Abbildung in die Prädikatenlogik 1. Stufe mit Gleichheit:

$$\begin{aligned}\phi(G) \quad = \quad & \forall x \forall y \forall z (PERSON(x) \wedge ABTEILUNG(y) \wedge PERSON(z) \wedge \\ & LEITER(y,x) \wedge TEIL(y,z) \wedge \neg(x=y) \wedge \neg(x=z) \wedge \neg(y=z) \rightarrow \\ & VORGESETZTER(z,x))\end{aligned}$$

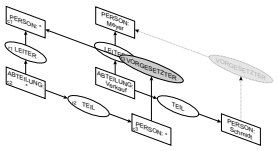

Abbildung 5.11: Regeln und Graph-Matching

Lassen sich nun die Knoten aus R auf F abbilden, die die Prämisse der Implikation darstellen (im Beispiel sind das die Knoten c_1, c_2, c_3, r_1 und r_2 aus R), so können die Relationen der Konklusion in das Fakt zwischen die entsprechenden Konzepte eingefügt werden. Abb. 5.11 zeigt rechts (hinten) das Fakt F' nach Anwendung der Regel, wobei die hellgrau hinterlegte Relation entsprechend der Relation r_3 aus R erstellt und zwischen $\Pi_{R \hookrightarrow F}(c_1)$ und $\Pi_{R \hookrightarrow F}(c_3)$ eingefügt wurde. Es gilt $\Omega(H_C, H_R)_\wedge \wedge \phi(F) \wedge \phi(R) \equiv \Omega(H_C, H_R)_\wedge \wedge \phi(F')$. Das heißt, die Wissensbasis wurde in eine äquivalente Wissensbasis *ohne* Regeln transformiert. In diesem speziellen Fall war das sehr einfach. Eine allgemeine Wissensbasis wird jedoch aus mehreren Fakten und mehreren Regeln bestehen, und es wird i.a. nicht nur eine Projektion von den Knoten der Prämisse einer Regel auf die Knoten des Fakt geben, sondern mehrere. Trotzdem ist die Transformation einer Wissensbasis in eine regelfreie Wissensbasis mit dem folgenden Algorithmus möglich:

```
// F: Die Fakten der Wissensbasis.
// R: Die Regeln der Wissensbasis.
procedure kb_to_kb_without_rules(F, R) {
    do
        boolean kb_changed = false;
        for each fact = (C_fact, R_fact, E_fact) ∈ F do
            for each rule = (C_rule, R_rule, E_rule, R̃_rule, Ẽ_rule) ∈ R do
                M ← find_iso_projections(rule, fact);
                for each Π_{rule↪fact} ∈ M do
                    S ← {r ∈ R̃_rule | ∄s ∈ R_fact : (s ≤ r)∧
                            (∀i ∈ {1, ..., |r|} : s^i = Π_{rule↪fact}(r^i))};
                    if S ≠ ∅ then
                        for each s ∈ S do
                            r ← new relation with type(r) = type(s) and
                                    ∀i ∈ {1, ..., |s|} : r^i = Π_{rule↪fact}(s^i);
                            R_fact = R_fact ∪ {r};
                            E_fact = E_fact ∪ {(c, r', n) | r' = r ∧ c = r^n};
                        od
                        kb_changed ← true;
                    fi
                od
            od
        od
    } while kb_changed;
}
```

Listing 5.4: Transformation in regelfreie Form

Eine Regel sollte nur angewandt werden, wenn sie neue Informationen in ein Fakt einbringt. Kann eine Regel einmal angewandt werden, so kann sie theoretisch beliebig oft angewandt werden, denn die Knoten ihrer Prämisse lassen sich dann immer wieder auf das gleiche Fakt abbilden. Es ist aber leicht ersichtlich, daß nur die erste Anwendung und das erste Einfügen der Relationen der Konklusion wirklich neue Informationen in ein Fakt einbringen. Würde man eine Regel mehrmals anwenden, so würde man Relationen einfügen, die bereits im Fakt vorhanden sind (z.B. würde eine zweite Regelanwendung in Abb. 5.11 die Relation `(VORGESETZTER)` ein zweites Mal zwischen `[PERSON: Schmidt]` und `[PERSON: Meyer]` einfügen, da die gleiche Relation aber schon aus der ersten Regelanwendung vorhanden ist, bringt das zweite Einfügen keine neue Information in das Fakt ein). Die neu eingefügte Relation wäre also redundant. Um dies zu verhindern, wird in Z. 10 und 11 die Menge der Relationen der Konklusion bestimmt, für die gilt, daß sie nicht bereits im Fakt vorhanden sind. Nur diese Relationen werden in Z. 13–18 in das Fakt eingefügt. Daher ist auch sichergestellt, daß der Algorithmus terminiert, denn die Fakten enthalten nur endlich viele Konzepte, zwischen die Relationen eingefügt werden können, und jede Regel besteht aus endlich vielen Konklusions-Relationen. Somit ist auf jeden Fall irgendwann der Zeitpunkt erreicht, an dem jede neue Regelanwendung nur redundante Relationen einfügen würde. Zu diesem Zeitpunkt sind alle implizit durch Regeln ausgedrückten Informationen explizit in den Fakten der Wissensbasis vorhanden.

Da diese Transformation u.U. sehr zeitaufwendig ist, wird sie als weiterer Schritt bei der Kompilierung der Wissensbasis offline durchgeführt. Die Überführung einer Wissensbasis in eine kompilierte Wissensbasis besteht also nun aus zwei Schritten:

1. Sequenz von Join-Operationen auf individuellen Konzepten (Algorithmus 5.1) ergibt Wissensbasis W'.

2. Transformation von W' in regelfreie Form mittels Algorithmus 5.4 ergibt die endgültige kompilierte Wissensbasis W''.

Diese Schritte müssen in dieser Reihenfolge durchgeführt werden, denn sonst können aus bereits genannten Gründen im 2. Schritt evtl. keine Projektionen gefunden werden. Da die Anwendung von Regeln keine Konzepte hinzufügen, muß Schritt 1 nicht nach Ausführung von Schritt 2 wiederholt werden.

Noch eine Anmerkung zur Matching-Operation bei der Überführung in eine regelfreie Form. Es können nur Iso-Projektionen verwendet werden, denn nur diese stellen sicher, daß verschiedene generische Konzepte der Regel auch auf verschiedene Konzepte des Fakts abgebildet werden und somit auch die Bedingungen der Form $\neg(x = y)$ aus $\phi(R)$ erfüllt sind, denn verschiedene individuelle Konzepte haben nach den Join-Operationen auf individuellen Konzepten (Schritt 1 oben) verschiedene Referenten, und somit gilt für diese immer $x \neq y$. Der Algorithmus 5.3 zur Suche nach Iso-Projektionen wurde bereits in Abs. 5.3.4.5 erläutert und kann unverändert verwendet werden.

5.3.4.7 Problem des Graph-Matching-Ansatzes

Es wurde bereits gezeigt, daß der Matching-Ansatz korrekt und auch vollständig ist, falls individuelle Konzepte paarweise verschiedene Referenten besitzen. Trotzdem gibt es eine Einschränkung dieses Ansatzes, die zwar nichts an den zuvor erwähnten Eigenschaften ändert, die aber trotzdem in der Praxis nicht zu vernachlässigen ist. Um diese Einschränkung zu verdeutlichen, wird eine Wissensbasis betrachtet, die nur aus dem in Abb. 5.12b dargestellten Fakt F besteht. Außerdem sei der Anfragegraph Q aus Abb. 5.12a gegeben. Es ist leicht einzusehen, daß es keine

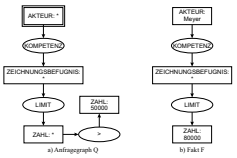

Abbildung 5.12: Matching und Zahlenvergleiche 1

vollständige Abbildung von Q nach F und somit auch keine Projektion geben kann, denn weder die Relation `(>)` noch das Konzept `[ZAHL: 50000]` lassen sich auf ein Konzept aus F abbilden. Trotzdem erscheint es auf den ersten Blick logisch, daß der Akteur Meyer alle von der Anfrage geforderten Eigenschaften besitzt. Akteur Meyer hat eine Zeichnungsbefugnis, die auf 80.000 DM[4] beschränkt ist. Gesucht wird ein Akteur, der eine Zeichnungsbefugnis mit einem Wert größer 50.000 DM besitzt. Meyer scheint daher der gesuchte Akteur zu sein. Bedeutet das nun, daß die Projektion doch nicht vollständig ist, denn offensichtlich enthält F keine individuellen Konzepte mit denselben Referenten, und für diese wurde ja die Vollständigkeit der Projektion bewiesen? Das Problem ist hier, daß einen das intuitive Verständnis täuscht, denn Tatsache ist, daß *nicht*

[4]Es wird davon ausgegangen, daß die Zahlen eines Limits für Geldbeträge stehen.

$\phi(F) \models \phi(Q)$ gilt.[5] Es ergeben sich die folgenden Abbildungen auf prädikatenlogische Formeln:[6]

$$\begin{aligned}\phi(F) &= \exists x(AKTEUR(Meyer) \wedge ZEICHNUNGSBEFUGNIS(x) \wedge \\ &\quad KOMPETENZ(Meyer, x) \wedge LIMIT(x, 80000))\end{aligned}$$

$$\begin{aligned}\phi(Q) &= \exists x \exists y \exists z(AKTEUR(x) \wedge ZEICHNUNGSBEFUGNIS(z) \wedge \\ &\quad KOMPETENZ(Meyer, z) \wedge LIMIT(z, y) \wedge GROESSER(y, 50000))\end{aligned}$$

Man kann sich nun eine zu $\phi(F)$ und $\phi(Q)$ passende Struktur A ausdenken, so daß $\phi(F)$ erfüllt ist, $\phi(Q)$ aber nicht. Man kann dazu z.B. einfach das Relationssymbol $GROESSER$ so durch eine konkrete Relation ersetzen, daß *nicht* $GROESSER(80000, 50000)$ gilt (etwa die Interpretation $I_A(GROESSER) = \{(x, y) \in \mathbf{N}^2 \mid x = y\}$), was nicht gerade intuitiv aber erlaubt ist. Das scheinbare Mißverständnis beruht also darauf, daß $GROESSER$ intuitiv die Semantik der allgemein bekannten Relation $>$ zugeordnet wird. Setzt man aber eine Struktur A' voraus, die diese Interpretation $I_{A'}(GROESSER) = \{(x, y) \in \mathbf{N}^2 \mid x > y\}$ enthält, so müßte man für alle $x, y \in \mathbf{N}$ mit $x > y$ Fakten der Form `[ZAHL: x]`→`(>)`→`[ZAHL: y].` in die Wissensbasis aufnehmen, damit die Matching-Operation auf den beschriebenen einfachen Konzeptgraphen weiterhin angewandt werden kann. Das sind bereits unendlich viele Fakten, und das gleiche müßte man noch für die Relationen $<$, $\leq$, $>$, $\geq$, usw. machen. Leicht einzusehen, daß das nicht möglich ist.

Man könnte sich nun vorstellen, die Relation $>$ mit in ein Konzept des Anfragegraphen aufzunehmen. Zum Beispiel könnte man in Abb. 5.12a das Konzept `[ZAHL: *]` durch `[ZAHL: > 50000]` ersetzen und die Knoten `(>)` und `[ZAHL: 50000]` löschen. Somit könnte man beim Vergleich der beiden Knoten `[ZAHL: > 50000]` und `[ZAHL: 80000]` feststellen, ob diese beiden Konzepte aufeinander abbildbar sind. Die beiden Konzepte werden während der Matching-Operation sowieso verglichen. Man müßte hier noch die Semantik von $>$ und allen anderen gewünschten Relationen mit in den Vergleich einbringen. Auf den ersten Blick erscheint dieser Vergleich ausreichend. Auf den zweiten ergibt sich aber ein weiteres Problem, das dieser Ansatz nicht lösen kann. Abb. 5.13 zeigt hierfür ein Beispiel. Für `(>)` aus Q kann keine Abbildung gefunden wer-

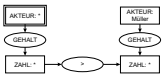

Abbildung 5.13: Matching und Zahlenvergleiche 2

den, denn `(>)` taucht in den Fakten nicht auf. Man kann $>$ nun auch nicht einfach in eines der betroffenen Konzepte aufnehmen. Da die Wissensbasis nur endlich viele Konzepte enthält, die Zahlen darstellen, könnte man theoretisch die entsprechenden Relationen in die Wissensbasis einfügen. Unter Umständen sind das aber sehr viele zusätzliche Knoten, was sich sehr negativ auf die Geschwindigkeit der Matching-Operation auswirken könnte. Aber auch wenn man das vernachlässigt, so erscheint es fragwürdig, ob man einmal den Vergleich in das Konzept aufnehmen und einmal eine Relation verwenden sollte, ganz abgesehen davon, daß man die Semantik der Relation wieder irgendwie in die Matching-Operation einbringen muß.

Man muß aber nicht versuchen, das Problem direkt durch Graph-Matching zu lösen, sondern kann den Vergleich aus dem Graph-Matching herausnehmen. Dazu können die Parameter des Anfragegraphen verwendet werden. Das heißt, der Anfragegraph aus Abb. 5.12a wird durch den

[5] Die Typhierarchien können hier vernachlässigt werden.

[6] Es wird mit Absicht *GROESSER* anstatt $>$ geschrieben, um zwischen diesen beiden Relationen unterscheiden zu können.

Anfragegraphen aus Abb 5.14 ersetzt. Jede Projektion liefert nun nicht nur den Akteur, sondern auch dessen Limit für seine Zeichnungsbefugnis. Dieser Wert kann nun *nach* erfolgtem Matching mit dem gewünschten Limit verglichen werden. Allerdings kann auch diese Lösung nicht als ideal bezeichnet werden.

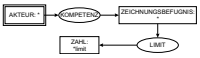

Abbildung 5.14: Anfragegraph ohne Zahlenvergleich

Für dieses Problem scheint es keine besonders elegante Lösung zu geben. Daher wird als „Workaround“ vorgeschlagen, Zahlenvergleiche nicht innerhalb des Graph-Matchings durchzuführen, sondern in diesem Fall Parameter zu verwenden, um die Werte extern vergleichen zu können.

Es stellt sich noch die Frage, wie oft wirklich solche Vergleiche bei der Selektion von Akteuren benötigt werden. Gerade das Beispiel mit der beschränkten Zeichnungsbefugnis würde in der Praxis sicher oft anders gelöst. Man würde mit großer Wahrscheinlichkeit ab einem bestimmten Vertragswert eine zusätzliche Aktivität *Genehmigen* in den Ablauf einfügen, die z.B. nur von einer höhergestellten Position durchgeführt werden dürfte (z.B. vom Vorgesetzten des Sachbearbeiters). Die Abfrage, ob ein Vertragswert größer 50.000 DM vorliegt, würde somit in der Prozeßdefinition erfolgen und nicht in der Selektion von Aufgabenträgern, denn der Vertragswert beeinflußt hier den Prozeßablauf und nicht die Zuweisung.

5.3.5 Selektion durch logische Programmierung

Durch die direkte Abbildbarkeit eines Konzeptgraphen auf eine Formel der Prädikatenlogik erster Stufe bietet es sich an, die Selektion von Aufgabenträgern mit Hilfe einer logischen Programmiersprache durchzuführen. Interessant dürfte hierbei auch der direkte Vergleich mit den Algorithmen des Graph-Matching-Ansatzes aus Abs. 5.3.4 bezüglich des Zeitaufwandes einer Selektion sein (siehe dazu Abs. 6.2.1).

Als logische Programmiersprache wird PROLOG eingesetzt. Die Transformation einer Wissensbasis in ein PROLOG-Programm wird am Beispiel erläutert. Gegeben sei hierzu eine Wissensbasis $W = (H_C, H_R, \{F_1, F_2\}, \{R\})$ mit zwei Fakten und einer Regel und den Typhierarchien aus Abb. 5.15.

Abbildung 5.15: Typhierarchien des PROLOG-Beispiels

Die Fakten seien wie folgt gegeben:

F_1 = [PERSON: Meyer]←(LEITER)←[ABTEILUNG: Verkauf].
F_2 = [ABTEILUNG: Verkauf] −
(TEIL)→[PERSON: Meyer]
(TEIL)→[PERSON: Schmidt].

$$\phi(F_1) = PERSON(Meyer) \wedge ABTEILUNG(Verkauf) \wedge LEITER(Verkauf, Meyer)$$

$$\phi(F_2) = PERSON(Meyer) \wedge PERSON(Schmidt) \wedge ABTEILUNG(Verkauf) \wedge TEIL(Verkauf, Meyer) \wedge TEIL(Verkauf, Schmidt)$$

Die Regel R ist die bereits bekannte Regel aus Abb. 5.3.

5.3.5.1 Transformation der Fakten

Um die Fakten der organisatorischen Wissensbasis in ein PROLOG-Programm einzubringen, müssen diese zunächst in eine Menge sogenannter *Hornklauseln* transformiert werden, da PROLOG auf Hornklausel-Programme beschränkt ist. Eine *Klausel* K ist eine Menge von Literalen $\{L_1, \ldots, L_n\}$, die als Abkürzung für die Disjunktion $(L_1 \vee \ldots \vee L_n)$ steht. Eine *Hornklausel* enthält höchstens ein positives Literal. Eine Klauselmenge $\{K_1, \ldots, K_m\}$ ist eine Abkürzung für $K_1 \wedge \ldots \wedge K_m$, also eine Abkürzung für eine Formel in konjunktiver Normalform (KNF). Ein *Fakt*[7] eines Hornklauselprogramms ist eine Klausel, bestehend aus einem positiven Literal $\{L\}$.

$\phi(F_1) \wedge \phi(F_2)$ liegt bereits in KNF vor und läßt sich daher direkt als PROLOG-Programm formulieren:

```
con_person(meyer).
con_abteilung(verkauf).
rel_leiter(verkauf,meyer).
con_person(schmidt).
rel_teil(verkauf,meyer).
rel_teil(verkauf,schmidt).
```

Bei einem PROLOG-Programm wird eine Klausel durch den Punkt beendet. Das heißt, das obige Programme ist eine Menge einelementiger Klauseln. Die Zeilen 1–3 ergeben sich hierbei aus $\phi(F_1)$, die restlichen Zeilen aus $\phi(F_2)$, wobei das Fakt `con_abteilung(verkauf)` auch in $\phi(F_2)$ enthalten ist. Um Probleme zu vermeiden, die im unwahrscheinlichen aber nicht ganz auszuschließenden Fall auftreten könnten, wenn eine einstellige Relation und ein Konzepttyp mit dem gleichen Bezeichner existieren, werden Konzeptbezeichner durch das Präfix „con_“ und Relationsbezeichner durch das Präfix „rel_“ gekennzeichnet.

Diese Form der Umwandlung funktioniert so nur, wenn alle Konzepte individuell sind. Sobald generische Konzepte in Fakten enthalten sind, kann diese Vorgehensweise nicht mehr direkt verwendet werden, wie das Beispiel des Konzeptgraphen F_3 zeigt:

F_3 = [PERSON: Meyer]→(FÄHIGKEIT)→[PROGRAMMIEREN].

$$\phi(F_3) = \exists x(PERSON(Meyer) \wedge PROGRAMMIEREN(x) \wedge FAEHIGKEIT(Meyer, x))$$

Würde man dieses Fakt nach der oben beschriebenen Art in ein PROLOG-Programm umwandeln, so würde man folgendes Ergebnis erhalten:

```
con_person(meyer).
con_programmieren(X).
rel_faehigkeit(meyer,X).
```

[7] Nicht zu verwechseln mit den gleichnamigen Konzeptgraphen der organisatorischen Wissensbasis.

Probleme treten in Z. 2 und 3 auf. Zeile 2 hat in PROLOG die Bedeutung $\forall x(Programmieren(x))$, was nicht mit $\phi(F_3)$ übereinstimmt (in $\phi(F_3)$ ist x existentiell gebunden). Außerdem steht die Variable X aus Z. 3 in keinerlei Zusammenhang zu dem X aus Z. 2, da es sich um verschiedene Klauseln handelt.

Um dieses Probleme zu umgehen, werden die Formeln $\phi(F_i)$ der Wissenbasis, die aus Fakten mit Variablen hervorgegangen sind, zunächst in erfüllbarkeitsäquivalente Formeln in *Skolemform* überführt. Der folgende Algorithmus führt diese Umwandlung durch:

```
  // F: Die Fakten der Wissensbasis.
2 // Rückgabewert: Eine Menge erfüllbarkeitsäquivalenter Formeln in
  //      Skolemform.
4 procedure skolemisierung(F = {F_1, ..., F_n}) returns set of facts {
      j ← 0;
6     for each F_i ∈ F do
          while F_i contains variable x do
8             // F_i hat die Form ∃xG
              // a_j ist eine Konstante, die bisher in keiner
10            // Formel der Wissensbasis enthalten ist.
              F_i ← G[x/a_j];
12            j ← j + 1;
          od
14    od
      return F;
16 }
```

Listing 5.5: Skolemisierung

Der Algorithmus ersetzt alle existentiell gebundenen Variablen der Fakten durch verschiedene Konstanten und entfernt die existentiellen Quantoren (in Z. 11 wird eine Variable der Formel ersetzt und der zugehörige Existenzquantor entfernt). Die Formel $\phi(F_3)$ würde z.B. in die folgende erfüllbarkeitsäquivalente Formel überführt:

$$\phi(F_3)' = PERSON(Meyer) \wedge PROGRAMMIEREN(a) \wedge FAEHIGKEIT(Meyer, a)$$

a bezeichnet hierbei eine neu eingeführte Konstante. Zu beachten ist, daß durch diese Umwandlung keine Information verlorengeht, denn es wurde nur ein nicht näher spezifiziertes Individuum x durch die Konstante a ersetzt. Hätte man bei der Modellierung bereits gewußt, für welches Individuum x steht, so hätte man dort bereits den konkreten Namen angegeben.

Solche Formeln in Skolemform lassen sich wie oben beschrieben problemlos in Fakten eines PROLOG-Programms transformieren.

5.3.5.2 Transformation der Typhierarchien

$\Omega(H_C, H_R)$ enthält Formeln der Form $\forall x_1 \ldots \forall x_n(P_1(x_1, \ldots, x_n) \rightarrow P_2(x_1, \ldots, x_n))$. Diese Formeln können ohne Probleme in Hornklauseln und somit in PROLOG-Regeln transformiert werden. Jede Generalisierungsbeziehung erzeugt dabei eine *Regel* im Sinne von PROLOG. In PROLOG werden Implikationen der Form $A_1 \wedge \ldots \wedge A_n \rightarrow B$ als Regeln bezeichnet. In KNF geschrieben hat diese Implikation die Form $\neg A_1 \vee \ldots \vee \neg A_n \vee B$. Diese Formel läßt sich als Hornklausel $\{\neg A_1, \ldots, \neg A_n, B\}$ schreiben und somit als Klausel in das PROLOG-Programm

aufnehmen. In PROLOG werden solche Klauseln als `B :- A1,...,An` geschrieben. „`:-`“ kann man sich dabei als den nach links zeigenden Implikationspfeil vorstellen.

Somit läßt sich die Typhierarchie aus Abb. 5.15 folgendermaßen als PROLOG-Programm schreiben („%“ bezeichnet in PROLOG einen Kommentar):

```
% Konzepttyphierarchie
con_universal(X).
con_universal(X) :- con_akteur(X).
con_akteur(X) :- con_person(X).
con_universal(X) :- con_organisationseinheit(X).
con_organisationseinheit(X) :- con_abteilung(X).
% Relationstyphierarchie
rel_universal(X,Y).
rel_universal(X,Y) :- rel_leiter(X,Y).
rel_universal(X,Y) :- rel_vorgesetzter(X,Y).
```

5.3.5.3 Transformation der Regeln

Die vorgeschlagenen Konzeptgraphen-Regeln haben allgemein die Form $\forall x_1 \ldots \forall x_n (T_1(x_1) \wedge \cdots \wedge T_2(x_n) \wedge R_1(\ldots) \wedge \cdots \wedge R_m(\ldots) \wedge \neg(x_1 = x_2) \wedge \cdots \wedge \neg(x_{n-1} = x_n) \rightarrow R_k(\ldots) \wedge \cdots \wedge R_l(\ldots))$, wobei die Prädikate T aus den Typen der Konzepte gebildet wurden und die Prädikate R aus Relationen. Eine Regel mit n Relationen in der Konklusion muß wegen der Beschränkung auf Hornklauseln in n PROLOG-Regeln umgewandelt werden. Die paarweise Ungleichheit der Variablen kann durch das build-in Prädikat `not` ausgedrückt werden.

Die Beispiel-Regel hat die folgende prädikatenlogische Formel und somit die danach dargestellte Umsetzung in eine PROLOG-Regel.

$$\begin{aligned}\phi(G) \;=\; & \forall x \forall y \forall z (PERSON(x) \wedge ABTEILUNG(y) \wedge PERSON(z) \wedge \\ & LEITER(y,x) \wedge TEIL(y,z) \wedge \neg(x = y) \wedge \neg(x = z) \wedge \neg(y = z) \rightarrow \\ & VORGESETZTER(z,x))\end{aligned}$$

```
rel_vorgesetzter(Z,X) :- rel_leiter(Y,X), rel_teil(Y,Z),
    con_person(X), con_abteilung(Y), con_person(Z),
    not(X=Y), not(Y=Z), not(X=Z).
```

5.3.5.4 Anfragen

Transformation in Zielklausel Als Beispiel-Anfrage dient zunächst der Anfragegraph aus Abb. 5.16a. Diese Anfrage ist gleichbedeutend mit der folgenden prädikatenlogischen Formel:

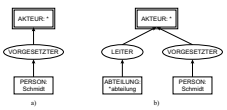

Abbildung 5.16: Anfragegraphen des PROLOG-Beispiels

$$\phi(Q) = \exists x (AKTEUR(x) \wedge PERSON(Schmidt) \wedge VORGESETZTER(Schmidt, x))$$

Natürlichsprachlich steht diese Anfrage für „Existiert ein Akteur X der der Vorgesetzte der Person Schmidt ist?“. PROLOG versucht nun nicht, diese Frage zu beantworten, sondern negiert zunächst die Frage (eine Antwort auf die negierte Frage $\neg\phi(Q)$ liefert genau dann „ ja“ wenn die ursprüngliche Frage „nein“ liefern würde und umgekehrt):

$$\begin{aligned}\neg\phi(Q) &= \neg\exists x(AKTEUR(x) \wedge PERSON(Schmidt) \wedge VORGESETZTER(Schmidt, x)) \\ &\equiv \forall x(\neg AKTEUR(x) \vee \neg PERSON(Schmidt) \vee \neg VORGESETZTER(Schmidt, x))\end{aligned}$$

Aus $\neg\phi(Q)$ kann man nun die *Zielklausel* bilden:

$$K_Z = \{\neg Akteur(x), \neg Person(Schmidt), \neg Vorgesetzter(Schmidt, x)\}$$

Diese Zielklausel wird in PROLOG wie folgt geschrieben:

```
?- con_akteur(x), con_person(schmidt), rel_vorgesetzter(schmidt,X).
```

Ziel ist es nun zu zeigen, daß $\{K_{p1}, \ldots, K_{pn}, K_Z\}$ unerfüllbar ist, wobei $K_{p1}, \ldots, K_{pn}$ die Programmklauseln (Fakten und Regeln) sind. Gelingt dies, dann ist – unter der Annahme, daß $K_{p1}, \ldots, K_{pn}$ wahre Aussagen sind – K_Z unwahr und die ursprüngliche Frage kann mit „ ja“ beantwortet werden, ansonsten mit „nein“. PROLOG versucht dazu durch *Resolution* aus der Zielklausel und den Programmklauseln die leere Klausel $\square$ herzuleiten. Gelingt dies, so ist die Klauselmenge unerfüllbar. Es ist nicht das Ziel dieser Arbeit, die Resolution im Detail zu erklären. Dafür wird auf die Literatur (z.B. [**?**]) verwiesen. Trotzdem ist ein gewisses Grundverständnis wichtig, um *effiziente* Anfragen zu generieren. Die folgenden Ausführungen sind daher teilweise vereinfacht dargestellt.

Anfragegraphen und SLD-Resolution PROLOG benutzt die sogenannte SLD-Resolution (engl. linear resolution with **s**el**e**ction function for **d**efinite clauses). Diese Art der Resolution ist für die Klasse der Hornformeln korrekt und vollständig und ermöglicht außerdem eine sehr effiziente Resolution (daher auch die Einschränkung von PROLOG auf Hornklauseln). Bei der SLD-Resolution bestimmt eine Auswahlfunktion, nach welchem Literal zu resolvieren ist. Bei PROLOG ist das das am weitesten links stehende Literal der Zielklausel. Die Resolution erfolgen nur zwischen Zielklausel und Klauseln der Wissensbasis (Fakten, Regel). Gegeben sei folgende PROLOG-Wissensbasis, die aus dem Fakt aus Abb. 5.17b generiert wurden:

```
con_ct1(a).
con_ct2(b).
con_ct2(c).
rel_rt1(a,b).
rel_rt2(a,c).
```

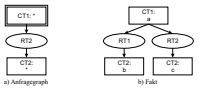

Abbildung 5.17: Graphen des Resolutionsbeispiels

Gestellt wird die aus Abb. 5.17a generierte Anfrage

```
?- con_ct1(X), con_ct2(Y), rel_rt2(X,Y).
```

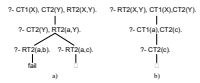

Abbildung 5.18: Berechnungsbaum der Resolution

Der Berechnungsbaum, den PROLOG bei der Anfrage durchläuft, ist in Abb. 5.18a zu sehen. Es wird davon ausgegangen, daß PROLOG eine Tiefensuche verwendet. Zunächst wählt PROLOG das Literal `con_ct1(X)` der Zielklausel (dieses steht am weitesten links) und versucht, dieses mit einer Klausel aus der Wissensbasis zu resolvieren. Dazu sucht PROLOG in der Wissensbasis von oben nach unten und findet die Klausel `con_ct1(a)`. Mit dieser kann `con_ct1(X)` resolviert werden, `X` wird dabei durch `a` ersetzt, so daß sich nun die neue Zielklausel `?- con_ct2(Y), rel_rt2(a,Y).` ergibt. Das nächste Literal, nach dem zu resolvieren ist, ist nun `con_ct2(Y)`. Dieses wird zunächst mit `con_ct2(b)` resolviert und es ergibt sich die Zielklausel `?- rel_rt2(a,b)`. Diese Zielklausel kann nicht weiter resolviert werden, denn die Wissensbasis enthält keine Klausel mit dem positiven Literal `rel_rt2(a,b)`. Daher erfolgt ein Backtracking im Suchbaum und `con_ct2(Y)` wird dieses mal mit `con_ct2(c)` resolviert. Nun läßt sich die neu entstandene Zielklausel `rel_rt2(a,c)` mit der Klausel aus Z. 5 zu □ resolvieren, und die Berechnung ist damit erfolgreich beendet. Es waren inkl. Backtracking 5 Schritte notwendig. Hätte es noch mehr Konzepte des Typs *CT2* mit Referent ungleich *c* gegeben, wären noch viel mehr Rückschritte und erfolglose Versuche nötig gewesen. Es stellt sich daher die Frage, ob es wirklich sinnvoll ist, die Konformität zu den geforderten Konzepttypen vor den Relationen zu testen. Wäre es nicht sinnvoller, zunächst die Relationen zu überprüfen? Eine Relation hat ja auch gewisse Restriktionen auf die Typen ihrer Argumente, so daß eine spätere Konzepttypüberprüfung mit relativ hoher Wahrscheinlichkeit erfolgreich ist. Zum Vergleich wird nun die gleiche Anfrage gestellt, nur werden die Relationen zuerst in der Zielklausel genannt (somit wird auch zuerst nach den Relationen resolviert).

?− rel_rt2(X,Y), con_ct1(X), con_ct2(Y).

Abb. 5.18b zeigt den neuen Berechnungsbaum. `rel_rt2(X,Y)` kann nur mit `rel_rt2(a,c)` resolviert werden. Daher lautet die neue Zielklausel `?- con_ct1(a), con_ct2(c).` `con_ct1(a)` kann sofort mit der Klausel aus Z. 1 resolviert werden und darauf folgt die Resolution von `con_ct2(c)` mit der Klausel aus Z. 3. Es ergibt sich somit bereits nach drei Schritten und ohne Backtracking die leer Klausel. Es ist allgemein anzunehmen ist, daß es sich positiv auf die benötigte Berechnungszeit auswirkt, wenn die Relationen in der Zielklause zuerst genannte werden. Dasselbe gilt auch für die Regeln aus vorigem Abschnitt. Auch hier ist es sinnvoll, zunächst die Prädikate zu nennen, die aus den Relationen der Prämisse der Regel entstanden sind, und dann erst die Konzepttypüberprüfung durchzuführen.

Gefahr von endlosen Rechnungen in PROLOG PROLOG könnte unter ungünstigen Umständen in eine endlose, erfolglose Rechnung geraten. Diese Gefahr besteht nur durch die Konzeptgraphen-Regeln. Fakten sind unkritisch und bei den PROLOG-Regeln, die aus den Typhierarchien entstanden sind, ist die Endlichkeit der Suche durch die Zyklenfreiheit und endliche Tiefe der Hierarchie garantiert. Bereits einfache Regeln können aber unendliche Rechnungen zur Folge haben. Folgendes PROLOG-Programm führt z.B. bei der Anfrage `?- rel_unterstellt(meyer,schmidt)` zu einer unendlichen Rechnung.

```
rel_unterstellt(X,Y) :- vorgesetzt(Y,X).
rel_vorgesetzt(X,Y) :- unterstellt(Y,X).
```

Dieses Programm könnte aus den Regeln „Wenn ein Akteur X Akteur Y unterstellt ist, dann ist Y vorgesetzt zu X.“ und „Wenn ein Akteur X Akteur Y vorgesetzt ist, dann ist Y X unterstellt.“ resultieren. PROLOG wird nun bei der obigen Anfrage immer zwischen diesen zwei Regeln pendeln, vorausgesetzt, es gibt nicht noch zusätzliche Fakten, die sagen, daß Meyer wirklich Schmidt unterstellt ist.

Der Graphen-Ansatz hat dieses Problem nicht, denn er kann ohne Probleme entscheiden, wann es nicht mehr sinnvoll ist, eine Regel anzuwenden (siehe Abs. 5.3.4.6). In PROLOG könnte dies auch über Tricks realisiert werden, doch sinnvoller ist eine Regeldefinition, die endlose Rechnungen verhindert. Es wäre eine sehr interessante Aufgabe, kritische Regeln zu erkennen, die in der Berechnung zu Zyklen führen könnten, allerdings muß aus Zeitgründen darauf verzichtet werden, diese Aufgabe anzugehen.

Ausgabe der Ergebnisse Natürlich genügt eine Ja-Nein-Antwort als Ergebnis einer Anfrage nicht. Es müssen der gefundene Akteur und evtl. vorhandene Parameter zurückgegeben werden. Nach einer erfolgreichen Rechnung ist das Ergebnis in den Variablenersetzungen enthalten, die PROLOG während der Resolution durchgeführt hat. Da eine Anfrage von links nach rechts abgearbeitet wird, können am Schluß „Anweisungen“ angegeben werden, die die entsprechenden Werte ausgeben. Eine Zielklausel für den Anfragegraphen aus Abb. 5.16a könnte nun wie folgt aussehen.

```
?- rel_vorgesetzter(schmidt,X), con_akteur(X), con_person(schmidt),
   write(X), nl, fail.
```

`write(X)` gibt im Falle einer erfolgreichen Rechnung den gefundenen Akteur gefolgt von einem Newline (`nl`) aus. `fail` löst nach erfolgreicher Suche das Backtracking aus, so daß alle Lösungen ermittelt werden und nicht nur eine. Ansonsten würde PROLOG im Berechnungsbaum an der Stelle anhalten, an der die leere Klausel resolviert werden konnte. Durch das fail wird das Backtracking erzwungen und der Berechnungsbaum vollständig durchlaufen.

Im Falle einer Anfrage mit Parametern wie in Abb. 5.16b wird nicht nur der gesuchte Akteur ausgegeben, sondern auch alle Parameter. So wird im Beispiel zusätzlich die Abteilung ausgegeben, deren Leiter der gesuchte Akteur ist.

```
?- rel_leiter(Y,X), rel_vorgesetzter(schmidt,X),
   con_akteur(X), con_person(schmidt), con_abteilung(Y),
   write(X), nl, write(Y), nl, fail.
```

5.3.6 Negierte Selektionen

Nicht immer ist ein Akteur gesucht, der bestimmte Eigenschaften besitzt. Manchmal ist es auch sinnvoll, einen Aufgabenträger zu suchen, der bestimmte Eigenschaften *nicht* besitzt. Der allgemeine Fall ist eine Kombination aus gewünschten und unerwünschten Eigenschaften. Einfache Konzeptgraphen bieten keine Möglichkeit zur Negation, d.h. eine unerwünschte Eigenschaft läßt sich nicht direkt in einem Anfragegraphen ausdrücken. Theoretisch wäre es nun möglich, erweiterte Konzeptgraphen nach Abs. 4.6 zu verwenden. Darauf wird in dieser Arbeit aus den bereits genannten Gründen verzichtet. Statt dessen wird hier nun gezeigt, wie man auch mit einfachen Konzeptgraphen eine entsprechende Selektion von Aufgabenträgern durchführen kann, vorausgesetzt, es besteht die Möglichkeit, Operationen auf Mengen von selektierten Akteuren durchzuführen.

Gesucht sei ein Akteur, der Eigenschaft P_1 besitzt und Eigenschaft P_2 nicht besitzt. Es läßt sich nun ein Anfragegraph Q_1 erstellen, der alle Akteure selektiert, die P_1 besitzen und ein weiterer Anfragegraph Q_2, der alle Akteure selektiert, die P_2 besitzen. Das heißt, die Durchführung einer Selektion mit Q_1 liefert eine Menge S_1 mit Bezeichnern von Akteuren, die P_1 besitzen, und die Selektion mit Q_2 liefert eine Menge S_2 mit Bezeichnern von Akteuren, die P_2 besitzen. Die gewünschten Akteure lassen sich nun durch die Bildung der Differenzmenge von S_1 und S_2 ermitteln. $S = S_1 \setminus S_2$ ist also die gesuchte Menge an Akteuren.

Changengine bietet die Möglichkeit, solche Mengen-Operationen durchzuführen (siehe Abs. 6.1.5), so daß entsprechende Selektionen möglich sind.

5.3.7 Partielle Projektionen

Nach Änderungen an der Organisationsstruktur oder auch beim Ausführen eines Prozesses in einer vollständig anderen Organisationsstruktur als der ursprünglich vorgesehenen, kann die Situation auftreten, daß kein Akteur gefunden werden kann, der die geforderten Eigenschaften vollständig besitzt. In bisherigen Systemen ist es in solchen Situationen üblich, die Zuweisung durch manuellen Eingriff vorzunehmen. Der wissensbasierte Ansatz kann auch in dieser Situation vorteilhaft eingesetzt werden. Zum einen besteht die Möglichkeit, den Administrator des Workflow-Management-Systems bei der Suche nach einem geeigneten Aufgabenträger zu unterstützen, so daß diesem die manuelle Zuweisung erleichtert wird. Zum anderen ist es möglich, die Zuweisung auch in diesen Fällen automatisch vorzunehmen, wenn durch das Workflow-Management-System automatisch festgestellt werden kann, welchem der Akteure trotz fehlender Eigenschaften die Aufgabe zugewiesen werden soll.

Besitzt ein Akteur nicht alle von der Anfrage geforderten Eigenschaften, so bedeutet das zunächst, daß der Anfragegraph nicht vollständig auf Fakten der Wissensbasis abgebildet werden kann, d.h. es bleiben Knoten des Anfragegraphen übrig, die keine Entsprechung im Fakt finden. Anstatt einer totalen Projektion existiert nur noch eine partielle Projektion.

Definition 22 (Partielle Projektion) *Gegeben:*

- *Ein Anfragegraph* $Q = (C_Q, R_Q, E_Q, root, P_Q)$
- *Ein Fakt* $F = (C_F, R_F, E_F)$
- *Ein zusammenhängender Subgraph* $Q' = (C_{Q'}, R_{Q'}, E_{Q'}, root, P_{Q'})$*, der von den Konzeptknoten* $C_{Q'} \subseteq C_Q$ *und den Relationsknoten* $R_{Q'} \subseteq R_Q$ *induziert wird, so daß* Q' *wieder ein Anfragegraph mit der Wurzel root ist, d.h. der Wurzelknoten von* Q *ist ebenfalls der Wurzelknoten von* Q'*.*
- *Ein zusammenhängender Subgraph* $F' = (C_{F'}, R_{F'}, E_{F'})$*, der von den Konzeptknoten* $C_{F'} \subseteq C_F$ *und den Relationsknoten* $R_{F'} \subseteq R_F$ *induziert wird, so daß* F' *wieder ein Fakt ist.*

Eine partielle Projektion $\Pi_{Q \mapsto F}$ *eines Anfragegraphen* Q *auf das Fakt* F *ist ein Paar von Funktionen* $f : C_{Q'} \to C_{F'}$*,* $g : R_{Q'} \to R_{F'}$ *mit den folgenden Eigenschaften:*

1. f *und* g *sind Funktionen, die für alle* $c \in C_{Q'}$ *bzw.* $r \in R_{Q'}$ *definiert sind (d.h. alle Knoten aus* Q' *werden auf Knoten aus* F' *abgebildet).*
2. $\forall c_Q \in C_{Q'} : \; f(c_Q) \leq c_Q$

3. $\forall r_Q \in R_{Q'} : \; g(r_Q) \leq r_Q$
4. $\forall (c_Q, r_Q, n_Q) \in E_{Q'} \; \exists (c_F, r_F, n_F) \in E_{F'} : c_F = f(c_Q) \wedge r_F = g(r_Q) \wedge n_F = n_Q$
5. $r \in R_{Q'} \Rightarrow \forall i \in \{1, \ldots, |r|\} : r^i \in C_{Q'}$. *Das heißt, wenn eine Relation abgebildet wird, dann werden auch* alle *Argumente der Relation abgebildet.*

Es wird ein zusammenhängender Subgraph Q'gefordert, der den Wurzelknoten der Anfrage enthält, um zu garantieren, daß die gefundenen Eigenschaften mit dem gefundenen Akteur in Beziehung stehen. Ein einfaches Beispiel macht deutlich, welch absurde Situation sonst eintreten könnte. Betrachtet man die Anfrage Q und das Fakt F aus Abb. 5.19 und fordert man *nicht* einen zusammenhängenden Subgraphen, so könnte man hier eine partielle Projektion bilden, die c_1 auf d_1 und c_2 auf d_4 abbildet. Das Konzept `[SPRACHE: Englisch]` steht aber in keinem Zusammenhang mit dem gefundenen Akteur Müller.

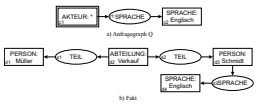

Abbildung 5.19: Partielle Projektion und nicht zusammenhängende Subgraphen

Eigenschaft 5 wird gefordert, da nicht vollständig abgebildete Relationen keinen Sinn mehr ergeben. Will man z.B. die Zeichnungsbefugnis für Urlaubsanträge auf eine bestimmte Abteilung einschränken, so könnte man eine dreistellige Relation `ZEICHNUNGSBEFUGT` bilden, die einen Akteur (den Zeichnungsberechtigten), eine Zeichnungsbefugnis für Urlaubsanträge und eine Abteilung (den Wirkungsbereich nach [**?**]) in Beziehung setzt. Bildet man ein Argument dieser Relation nicht ab, so erhält man z.B. einen Akteur, der zeichnungsbefugt für Urlaubsanträge ist, aber eben nicht in der entsprechenden Abteilung. Oder man erhält einen Akteur, der zwar zeichnungsbefugt in der geforderten Abteilung ist, aber nicht für Urlaubsanträge. In jedem dieser Fälle wäre die gefundene Abbildung der Relation nutzlos.

Intuitiv ist klar, daß nicht irgendeine partielle Projektion gesucht ist, sondern eine, die möglichst viele Knoten des Anfragegraphen abbildet:

Definition 23 (Maximale partielle Projektion) *Eine* maximale partielle Projektion $\Pi_{Q \mapsto F}$ *mit dem Subgraphen Q' ist eine partielle Projektion nach Def. 22, so daß keine partielle Projektion $\Pi'_{Q \mapsto F} \neq \Pi_{Q \mapsto F}$ mit dem Subgraphen Q'' existiert, so daß $C_{Q'} \subset C_{Q''}$ und/oder $R_{Q'} \subset R_{Q''}$ gilt. Das heißt, die maximale partielle Projektion läßt sich durch Hinzunahme von weiteren Knotenabbildungen nicht so erweitern, daß weiterhin alle Eigenschaften der partiellen Projektion erfüllt sind.*

Beispiele für maximale partielle Projektionen sind in Abb. 5.20 zu sehen.

5.3.7.1 Ein P-Algorithmus für die Suche nach maximalen partiellen Projektionen

Ein P-Algorithmus ergibt sich durch eine kleine Modifikation aus dem P-Graph-Matching-Algorithmus für die Suche nach totalen Projektionen aus Abs. 5.3.4.4. Der Algorithmus 5.2 findet in der bisherigen Form nur totale Projektionen, da in Z. 26 alle Kandidaten gelöscht werden, die verhindern, daß der Nachfolger s von *root* abgebildet wird. Das heißt, wenn *root* auf eines dieser Konzepte abgebildet werden würde, so ließe sich der Nachfolger s von *root* nicht mehr abbilden

und somit scheidet diese Abbildung aus, falls eine totale Projektion gefordert wird. Verzichtet man auf die Totalität, so kann man Z. 26 löschen. *root* wird nun auf den Kandidaten abgebildet, obwohl die Nachfolgerelation s unter dieser Abbildung von *root* nicht abgebildet werden kann. Zu beachten sind dabei drei Eigenschaften der gefundenen Abbildung:

- Enthält eine Projektion eine Abbildung für die Relation r, so enthält sie auch Abbildungen für alle Argumente r^i von r. Diese Eigenschaft wird dadurch garantiert, daß in Z. 49 eine Relationen des Fakts aus der Kandidatenmenge gelöscht wird, sobald klar ist, daß mindestens ein Argument nicht abgebildet werden kann.
- Der gefundene Subgraph Q' des Anfragegraphen ist zusammenhängend und enthält eine Abbildung für den Wurzelnoten. Das folgt aus der Arbeitsweise des Algorithmus, der vom Wurzelknoten ausgehend zunächst versucht, Abbildungen für die Nachfolgerknoten zu finden, dann Abbildungen für die Nachfolgerknoten der Nachfolgerknoten, usw. Ein Knoten kann also nur abgebildet werden, wenn auch sein Vorgänger abgebildet wurde.
- Der gefundene Subgraph Q' ist maximal, d.h. die gefundene Projektion kann nicht um weitere Abbildungen erweitert werden, so daß Q' weiterhin zusammenhängend ist und keine unvollständig abgebildeten Relationen enthält.

Daher ist die gefundene Abbildung eine maximale partielle Projektion nach Def. 23. Der Algorithmus ist ebenfalls nur für baumförmige Anfragegraphen anwendbar.

5.3.7.2 Ein NP-Algorithmus für die Suche nach maximalen partiellen Projektionen

Beschränkt man sich nicht auf baumförmige Anfragegraphen, so lassen sich maximale partielle Projektionen durch einen Algorithmus bestimmen, der für die Suchen nach maximalen gemeinsamen Subgraphen (engl. **M**aximal **C**ommon **S**ubgraph, kurz MCS) entworfen wurde. Die Suche nach MCS ist eng mit der Suche nach maximalen partiellen Projektionen verwandt. Bei einem MCS zweier Graphen G und H handelt es sich um einen Graphen, der isomorph zu je einem Subgraphen G' aus G und H' aus H ist. Die Maximalität ergibt sich aus der Nicht-Erweiterbarkeit des Subgraphen. Zwei Unterschiede zur maximalen partiellen Projektion bestehen darin, daß bei einem MCS im allgemeinen nicht gefordert wird, daß dieser zusammenhängend ist und eine Abbildung für einen bestimmten Knoten (z.B. den Wurzelknoten des Anfragegraphen) vorhanden ist. Der erste Unterschied fällt allerdings bereits weg, wenn man nach zusammenhängenden MCS (**C**onnected MCS, kurz CMCS) sucht, wobei das CMCS allerdings genau wie das MCS-Problem NP-vollständig ist. Außerdem wird bei einer partiellen Projektion keine bijektive Abbildung des Subgraphen Q' des Anfragegraphen auf einen Subgraphen des Fakts gefordert. Wie beim Ullmann-Algorithmus zur Suche nach totalen Iso-Projektionen, so ergeben sich auch hier die bereits in Abs. 22 erwähnten Einschränkungen (die wie bereits bekannt keine wirklichen Einschränkungen darstellen), daß Anfragegraphen keine zwei individuelle Konzepte mit demselben Referenten und keine identischen (redundanten) Relationen, d.h. Relationen mit demselben Typ und denselben Argumenten, enthalten dürfen. Wie bei den totalen Projektionen wird auch hier im folgenden von einer (partiellen) Iso-Projektion gesprochen:

Definition 24 (Partielle Iso-Projektion) *Eine* partielle Iso-Projektion $\Pi_{Q \hookrightarrow F}$ *ist eine partielle Projektion, bei der zusätzlich die Funktionen f und g bijektiv sind.*

Die Suche nach MCS hat diverse praktische Anwendungen. So wurde von Koch in [**?**] ein Algorithmus für die Suche nach CMCS vorgeschlagen, um strukturelle Ähnlichkeiten in Protein-Graphen darzustellen. Dieser Algorithmus wird hier in leicht modifizierter Form für die Suche nach maximalen partiellen Projektionen verwendet. Der Koch-Algorithmus basiert auf dem

Bron-Kerbosch-Algorithmus aus [**?**], der alle maximalen Cliquen eines ungerichteten Graphen aufzählt. Da sich das MCS-Problem auf das Cliquen-Problem reduzieren läßt, kann dieser Algorithmus ebenfalls dazu verwendet werden, um alle MCS aufzuzählen. Die Anpassungen in [**?**] optimieren den Bron-Kerbosch-Algorithmus für das CMCS-Problem, wodurch in der Praxis wesentlich kürzere Berechnungszeiten erzielt werden können, sofern nur zusammenhängende Subgraphen von Interesse sind, was bei maximalen partiellen Projektionen der Fall ist.

Algorithmus Der Algorithmus geht in zwei Schritten vor:

1. Aus den Eingabegraphen – hier also dem Anfragegraphen und dem Fakt – wird ein sogenannter Association-Graph gebildet.

2. Mit Hilfe des modifizierten Bron-Kerbosch-Algorithmus werden alle maximalen Cliquen des Association-Graphs aufgezählt. Aus den gefundenen Cliquen lassen sich die CMCS bestimmen.

In [**?**] wurde bereits ein Matching-Algorithmus für Konzeptgraphen vorgeschlagen, der ähnlich vorgeht. Da der Koch-Algorithmus allerdings für *zusammenhängende* Subgraphen optimiert wurde und nur diese für die Suche nach partiellen Projektionen von Relevanz sind, wird auch dieser Algorithmus im folgenden verwendet, da er eine kürzere Laufzeit verspricht.

Die Arbeitsweise wird anhand eines Beispiels erläutert, das auf den Algorithmus folgt.

```
// Q: Anfragegraph
// F: Fakt
procedure match(Q = (C_Q, R_Q, E_Q, root, P), F = (C_F, R_F, E_F)) {
    G = create_association_graph(C_Q, R_Q, C_F, R_F);
    start_search(G, Q, F);
}

// C_Q: Konzeptknoten des Anfragegraphen.
// R_Q: Relationsknoten des Anfragegraphen.
// C_F: Konzeptknoten des Fakts.
// R_F: Relationsknoten des Fakts.
// Rückgabewert: Association-Graph.
procedure create_association_graph(C_Q, R_Q, C_F, R_F)
            returns association graph {
    V_C ← {(c,d) ∈ C_Q × C_F | d ≤ c};
    V_R ← {(r,s) ∈ R_Q × R_F | s ≤ r ∧ ∀i ∈ {1,...,|r|} : s^i ≤ r^i};
    V ← V_C ∪ V_R;
    E_d ← {{(c,d),(e,f)} ∈ V_C × V_C | c ≠ e ∧ d ≠ f};
    E_d ← E_d ∪ {{(c,d),(e,f)} ∈ V_R × V_R | c ≠ e ∧ d ≠ f};
    E_d ← E_d ∪ {{(c,d),(e,f)} ∈ V_C × V_R | ∀i ∈ {1,...,|e|} : e^i ≠ c ∧ f^i ≠ d};
    E_c ← {{(c,d),(e,f)} ∈ V_C × V_R | (∀i ∈ {1,...,|e|} : e^i = c ↔ f^i = d)∧
                                       (∃i ∈ {1,...,|e|} : e^i = c)};
    return (V, E_c, E_d);
}

// G: Association-Graph
// Q: Anfragegraph
```

```
28 // F: Fakt
   procedure start_search(G = (V, E_c, E_d), Q = (C_Q, R_Q, E_Q, root, P),
30                                            F = (C_F, R_F, E_F)) {
       I ← ∅;
32     for each v = (v_1, v_2) ∈ V do
           if v = root then
34             Clique ← {v};
               Cand_visible ← {u ∈ V | {u, v} ∈ E_c ∧ u ∉ I};
36             Cand_forbidden ← {u ∈ V | {u, v} ∈ E_c ∧ u ∈ I};
               Cand_hidden ← {u ∈ V | {u, v} ∈ E_d};
38             find_cliques(Cand_visible, Cand_hidden, Cand_forbidden, I,
                            Clique, G);
40             I ← I ∪ {v};
           fi
42     od
   }
44
   // Cand_visible: Knoten des A–Graphen, die die bisher gefundene
46 //     Clique erweitern können.
   // Cand_visible: Knoten des A–Graphen, die prinzipiell die bisher
48 //     gefundene Clique erweitern könnten, aber einen nicht
   //     zusammenhängenden Subgraphen erzeugen würden.
50 // Cand_forbidden: Knoten des A–Graphen, die die bisher gefundene
   //     Clique erweitern könnten, aber dann zu einer bereits
52 //     gefundenen Clique führen würden.
   // I: Knoten des A–Graphen, die bereits als Startknoten verwendet
54 //     wurden (siehe start_search).
   // G: Association–Graph.
56 procedure find_cliques(Cand_visible, Cand_hidden, Cand_forbidden, I, Clique,
                           G = (V, E_c, E_d)) {
58     if Cand_visible = ∅ then
           if Cand_forbidden = ∅ then
60             save(Clique);
           fi
62         return;
       fi
64     while Cand_visible ≠ ∅ do
           v ∈ Cand_visible;
66         Cand_visible ← Cand_visible \ v;
           Clique ← Clique ∪ {v};
68         N_c ← {u ∈ V | {v, u} ∈ E_c};
           N_d ← {u ∈ V | {v, u} ∈ E_d};
70         N ← N_c ∪ N_d;
           Cand'_visible ← Cand_visible ∩ N;
72         Cand'_forbidden ← Cand_forbidden ∩ N;
           Cand'_hidden ← ∅;
74         for each u ∈ Cand_hidden do
               if u ∈ N_d then
76                 Cand'_hidden ← Cand'_hidden ∪ {u};
```

```
                else if u ∈ N_c then
78                  if u ∈ I then
                        Cand'_forbidden ← Cand'_forbidden ∪ {u};
80                  else
                        Cand'_visible ← Cand'_visible ∪ {u};
82                  fi
                fi
84          od
            find_cliques(Cand_visible, Cand_hidden, Cand_forbidden, I, Clique,
86                       G = (V, E_c, E_d));
            Clique ← Clique \ v;
88          Cand_forbidden ← Cand_forbidden ∪ {v};
        od
90  }
```

Listing 5.6: NP-Algorithmus zur Suche nach maximalen partiellen Projektionen nach [**?**]

Erläuterung Zur Erläuterung des Algorithmus sei die Anfrage aus Abb. 5.20a und das Fakt aus Abb. 5.20b gegeben.

Der erste Schritt des Algorithmus besteht darin, aus diesen zwei Graphen des Association-Graph zu bilden (Prozedur `create_association_graph`). In Z. 15 und 16 werden dazu alle Paare von Knoten des Anfragegraphen und des Fakts ermittelt, die aufgrund des Typs und der Referenten aufeinander abbildbar sind. Diese Paare bilden die Knotenmenge V des Association-Graphen (im folgenden a-Knoten genannt). Zwei Knoten des Association-Graphen werden genau dann durch eine Kante verbunden, wenn sie beide gleichzeitig zu einer Projektion gehören können. Da das MCS-Problem eine Bijektion, d.h. eine 1:1-Beziehung zwischen den Knoten, fordert, können zwei a-Knoten $\{c, d\}$ und $\{e, f\}$ nicht durch eine Kante verbunden werden, wenn $c = e$ oder $d = f$ gilt (siehe Abb. 5.10 und zugehörige Erläuterung). Außerdem kann eine a-Kante nur dann existieren, wenn zwischen den Knoten c und e des Anfragegraphen und den Knoten d und f des Fakts genau die gleichen Kanten (d.h. Kanten mit den gleichen Nummern) verlaufen. Da zwei Konzeptknoten und zwei Relationsknoten nie miteinander verbunden sind, muß diese Bedingung in Z. 18 und 19 nicht explizit überprüft werden. Um nur zusammenhängende Subgraphen zu finden, werden die Kanten des Anfragegraphen in c-Kanten und d-Kanten unterteilt („c“ steht für connected und „d“ für disconnected). Zwischen a-Knoten kann nur dann eine c-Kante verlaufen, wenn die Knoten c und e des Anfragegraphen und die Knoten d und f des Fakts jeweils miteinander verbunden sind. Ansonsten kann nur eine d-Kante zwischen den a-Knoten verlaufen. Die Mengen E_c und E_d enthalten alle c- bzw. d-Kanten. Da nur zwischen Konzept- und Relationsknoten Kanten im Anfragegraphen und im Fakt verlaufen können, müssen diese Knoten in Z. 20–22 speziell betrachtet werden. Verläuft im Anfragegraphen und Fakt gar keine Kante zwischen den Knoten, so handelt es sich um eine d-Kante (Z. 20), verlaufen genau die gleichen Kanten, so handelt es sich um eine c-Kante (Z. 21,22). Der Association-Graph für das Beispiel ist in Abb. 5.21 dargestellt.

Da nun alle a-Knoten, die zur selben Projektion gehören können, durch Kanten miteinander verbunden sind, beschränkt sich die weiter Aufgabe darauf, vollständig verbundene Teilgraphen (Cliquen) des Association-Graphen zu finden und zwar so, daß eine Clique aus möglichst vielen a-Knoten besteht (maximale Clique). Jede gefundene Clique bildet – mit einer kleinen Einschränkung, die weiter unten betrachtet wird – eine maximale partielle Projektion. Das Auffinden der Cliquen erfolgt nun mit dem modifizierten Bron-Kerbosch-Algorithmus, der durch die

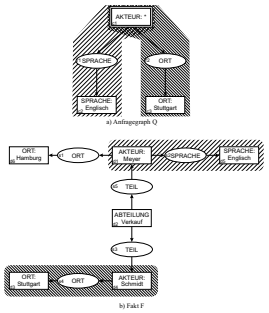

Abbildung 5.20: Beispiel zum Algorithmus für die Suche nach maximalen partiellen Projektionen

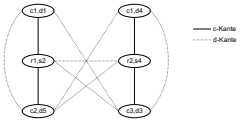

Abbildung 5.21: Association-Graph

Modifikation nach Koch nur solche Cliquen findet, die CMCS bilden.

Auch der Koch-Algorithmus benutzt für die Suche nach Cliquen eine Backtracking-Strategie. In jedem Schritt wird versucht, eine Clique durch Hinzunahme eines weiteren a-Knotens zu erweitern, bis ein Punkt erreicht ist, an dem feststeht, daß sich die Clique nicht mehr erweitern läßt. Dann erfolgt der Rückschritt. Außerdem stellt eine spezielle Überprüfung sicher, daß frühzeitig erkannt wird, wenn die Erweiterung einer Clique zu einer bereits gefundenen Clique führen würde, so daß auch hier der Suchbaum beschnitten werden kann.

Initialisiert wird die Suche durch die Prozedur `start_search`. Als Ausgangspunkt für die Suche nach Cliquen wir ein a-Knoten ausgesucht, der eine Abbildung für den Wurzelknoten der Anfrage darstellt (Test in Z. 33). Es werden daher nur noch Cliquen gefunden, die eine Abbildung für den Wurzelknoten des Anfragegraphen enthalten (somit ist diese geforderte Eigenschaft einer partiellen Projektion automatisch erfüllt). Eine Abbildung ohne einen solchen Knoten ist wertlos, da der Wurzelknoten für den zu findenden Akteur der Anfrage steht. Daher wird dieser a-Knoten in Z. 34 zur Clique hinzugefügt. Die Menge *Clique*, die – wie der Name vermuten läßt – die Knoten einer Clique enthält, besteht zu jedem Zeitpunkt aus einer Menge von vollständig verbundenen a-Knoten, d.h. *Clique* ist zu jeder Zeit eine Clique (allerdings nicht immer eine maximale Clique). Zu Beginn besteht diese Clique nur aus einem Knoten, nämlich der oben genannten Abbildung für den Wurzelknoten des Anfragegraphen. Ziel es ist nun, die Clique durch Hinzunahme weiterer a-Knoten zu erweitern. Dies geschieht durch rekursive Aufrufe der Prozedur `find_cliques`. Dieser Prozedur werden neben dem Association-Graphen und der bisher gefundenen Clique vier weitere Parameter übergeben:

- $Cand_{visible}$ besteht aus einer Menge von a-Knoten, die die bisher gefundene Clique erweitern könnten.
- $Cand_{hidden}$ besteht aus einer Menge von a-Knoten, die die bisher gefundene Clique erweitern könnten, aber bisher nicht verwendet werden können, da nicht sichergestellt werden kann, daß diese Knoten zu einem zusammenhängenden MCS führen.
- $Cand_{forbidden}$ besteht aus einer Menge von a-Knoten, die die bisher gefundene Clique

erweitern könnten, aber dann zu einer bereits gefundenen Clique führen würden.

- I enthält die bisher benutzten Startknoten der Suche.

Initialisiert werden die Kandidatenmengen in Z. 35–37. Da v bisher der einzige Knoten der Clique ist, müssen nur seine Nachbarn im Association-Graphen überprüft werden. Alle Nachbarn, die über c-Kanten mit v verbunden sind und die bisher noch nicht als Initialisierungsknoten gedient haben, können die Clique erweitern und werden daher in Z. 35 zu $Cand_{visible}$ hinzugefügt. Hat ein solcher a-Knoten aber schon als Initialisierungsknoten gedient, so wurden bereits alle Cliquen, die diesen Knoten enthalten, durch frühere Aufrufe von `find_cliques` aufgezählt. Eine Hinzunahme dieses Knotens zur aktuellen Clique würde also nur zu einer bereits gefundenen Clique führen, und daher wird dieser a-Knoten der Menge der verbotenen Knoten hinzugefügt (Z. 36). Alle a-Knoten, die über eine d-Kante mit v verbunden sind, gehören zur Menge der verborgenen Kandidaten $Cand_{hidden}$. Erst wenn der Quellknoten des a-Knotens von einem Quellknoten eines a-Knotens der Clique zu erreichen ist, wird dieser Knoten sichtbar.

Rekursive Aufrufe von `find_cliques` erweitern nun die bisher einelementige Clique. Eine maximale Clique, die einen CMCS darstellt, ist genau dann gefunden, wenn die Mengen $Cand_{visible}$ und $Cand_{forbidden}$ leer sind (Test in Z. 58 und 59), denn dann kann die bisher gefundene Clique nicht mehr um weitere Knoten erweitert werden. Die gefundene Projektion kann daher gespeichert werden. Ist dagegen nur die Menge $Cand_{visible}$ leer, $Cand_{forbidden}$ aber nicht, so könnte die bisher gefundene Clique um Knoten aus $Cand_{forbidden}$ zu einer weiteren, größeren Clique erweitert werden. Da aber bereits alle Cliquen aufgezählt wurden, die Knoten aus $Cand_{forbidden}$ enthalten, würde man eine bereits gefundene Clique doppelt aufzählen. Man kann die Suche für Erweiterungen der bisher gefundenen Clique daher ebenfalls abbrechen. Zu beachten ist, daß die Suche abgebrochen werden kann, obwohl $Cand_{hidden}$ evtl. nicht leer ist. Alle Knoten aus dieser Menge könnten die bisher gefundene Clique zwar erweitern, aber man würde dadurch einen nicht zusammenhängenden Subgraphen erhalten.

Ist die Menge $Cand_{visible}$ nicht leer, so kann die bisher gefundene Clique um a-Knoten aus dieser Menge erweitert werden. Jeder Knoten wird daher der Reihe nach in einer While-Schleife zur Erweiterung der Clique herangezogen. Wurde die Clique um einen Knoten erweitert (Z. 66), so müssen die Kandidatenmengen vor dem rekursiven Aufruf von `find_cliques` angepaßt werden, so daß die oben beschriebenen Bedingungen für diese Mengen weiterhin gelten. In Z. 71 und 72 werden zunächst die Knoten aus den Kandidatenmengen $Cand_{visible}$ und $Cand_{forbidden}$ gelöscht, die nicht adjazent zum neu hinzugefügten Knoten v sind, denn Elemente aus den Kandidatenmengen müssen die bisher gefundene Clique erweitern können, und das können sie nur, wenn sie zu allen Knoten der Clique adjazent sind. Die Knoten der bereits gefundenen Clique sind alle vollständig mit den Knoten der Kandidatenmengen verbunden. Nur für den neu hinzugefügten Knoten muß diese Bedingung noch überprüft werden. Diese Bedingung gilt natürlich auch für die Menge $Cand_{hidden}$, allerdings ist hier zusätzlich zu überprüfen, ob Knoten aus dieser Menge durch das Hinzufügen von v sichtbar geworden sind. Bisher waren die Quellknoten der a-Knoten dieser Menge von keinem Quellknoten der a-Knoten der Clique erreichbar. Falls aber ein Knoten aus $Cand_{hidden}$ zu v über eine c-Kante adjazent ist, so ist der entsprechende Quellknoten nun über den Quellknoten von v erreichbar und kann daher in die Menge der sichtbaren Knoten aufgenommen werden (Z. 81). Dies darf allerdings nur dann erfolgen, wenn der a-Knoten noch nicht als Initialisierungsknoten gedient hat, denn für diesen Knoten wurden bereits alle Cliquen aufgezählt, und man würde nur wieder doppelte Cliquen erhalten. Daher wird der Knoten in diesem Fall in Z. 79 zu den verbotenen Knoten hinzugefügt. Ist ein Knoten, der vormals nicht sichtbar war, mit dem Knoten v nur über eine d-Kante verbunden, so ist er auch weiterhin nicht sichtbar und bleibt daher in C_{hidden} (Z. 76). Der rekursive Aufruf von `find_cliques` versucht

nun, die bisherige Clique zu erweitern. Nach der Rückkehr aus der Rekursion kann der nächste Knoten aus $Cand_{visible}$ zum Erweitern der Clique herangezogen werden. Bevor das jedoch geschieht, wird v in Z. 88 zur Menge der verbotenen Knoten hinzugefügt, denn es kann nach erfolgter Rekursion davon ausgegangen werden, daß alle Erweiterungen der bisher gefundenen Clique, die ja bereits um v erweitert wurde, nun bereits gefunden wurden.

Die einzelnen Aufrufe von `find_cliques` für den Association-Graphen aus Abb. 5.21 sind in folgender Tabelle aufgeführt.

	$Cand_{visible}$	$Cand_{hidden}$	$Cand_{forbidden}$	I	$Clique$
1	$\{r_1, s_2\}$	$\{c_2, d_5\}, \{c_3, d_3\}$	$\emptyset$	$\{c_1, d_1\}$	$\{c_1, d_1\}$
1.1	$\{c_2, d_5\}$	$\{c_3, d_3\}$	$\emptyset$	$\{c_1, d_1\}$	$\{c_1, d_1\}, \{r_1, s_2\}$
1.1.1	$\emptyset$	$\emptyset$	$\emptyset$	$\{c_1, d_1\}$	$\{c_1, d_1\}, \{r_1, s_2\}, \{c_2, d_5\}$
2	$\{r_2, s_4\}$	$\{c_2, d_5\}, \{c_3, d_3\}$	$\emptyset$	$\{c_1, d_1\}$ $\{c_1, d_4\}$	$\{c_1, d_4\}$
2.1	$\{c_3, d_4\}$	$\{c_2, d_5\}$	$\emptyset$	$\{c_1, d_1\}$ $\{c_1, d_4\}$	$\{c_1, d_4\}, \{r_2, s_4\}$
2.1.1	$\emptyset$	$\emptyset$	$\emptyset$	$\{c_1, d_1\}$ $\{c_1, d_4\}$	$\{c_1, d_4\}, \{r_2, s_4\}, \{c_3, d_3\}$

Schritt 1 und 2 ist jeweils der Aufruf aus `start_search` heraus wobei die *Clique* jeweils nur aus der Abbildung des Wurzelknotens besteht. In Schritt 1.1 wird der a-Knoten (r_1,s_2) aus $Cand_{visible}$ entfernt und die Clique damit erweitert. Da (c_2,d_5) aus $Cand_{hidden}$ mit dem Knoten (c_2,d_5) über eine c-Kante verbunden ist, wird er der Menge $Cand_{visible}$ hinzugefügt. Wäre er nur über eine d-Kante verbunden, so würde er in der Menge $Cand_{hidden}$ verbleiben, wäre er gar nicht mit (r_1,s_2) verbunden, so würde er aus der Menge der Kandidaten verschwinden. Eine *neue* Clique ist gefunden, wenn $Cand_{visible}$ und $Cand_{forbidden}$ leer sind (Schritt 1.1.1 und 2.1.1). Wäre z.B. in 1.1.1 $Cand_{forbidden}$ nicht leer, so wäre die gefundene Clique nicht neu. Die gefundenen maximalen partiellen Projektionen sind in Abb. 5.20 zu sehen.

Es wurde angedeutet, daß die gefundenen Cliquen nicht in jedem Fall eine maximalen partielle Projektion darstellen. Dieser Fall tritt auf, wenn eine Relation nicht vollständig abgebildet wurde, d.h. wenn ein Argument r^i der Relation nicht abgebildet wurde (der vorgestellte CMCS-Algorithmus stellt dies nicht sicher). Diesem Problem kann man einfach dadurch begegnen, daß man unvollständig abgebildete Relationen aus der Projektion löscht. Dies kann wiederum dazu führen, daß der Subgraph nicht mehr zusammenhängend ist. Auch dieses Problem läßt sich leicht lösen, indem man vom Wurzelknoten ausgehend eine Art Flooding auf den Knoten des Subgraphen ausführt. Knoten des Subgraphen, die nicht vom Wurzelknoten aus erreichbar sind, können gelöscht werden.

Komplexität Auch beim Koch-Algorithmus tritt wie beim Ullmann-Algorithmus der höchste Aufwand auf, wenn viele Konzepte und Relation den gleichen Typ haben. Dann besteht der in `match` berechnete Association-Graph aus $O(|C_Q||C_F| + |R_Q||R_F|)$ a-Knoten, die mit jeweils $O(|C_Q||C_F| + |R_Q||R_F|)$ a-Knoten verbunden sind. In `find_cliques` kann eine Clique daher um $O(|C_Q||C_F| + |R_Q||R_F|)$ a-Knoten erweitert werden, und da der Berechnungsbaum eine Tiefe von $O(|C_Q| + |R_Q|)$ hat, ergibt sich bei einer Wissensbasis mit n Fakten eine Komplexität von

$$O(n(|C_Q||C_F| + |R_Q||R_F|)^{|C_Q|+|R_Q|}) \tag{5.3}$$

Auch diese Komplexität ist theoretischer Natur. Es wird daher auf die Meßergebnisse aus Abs. 6.2.1 verwiesen.

5.3.7.3 Bewertung von partiellen Projektionen

Die vorgestellten Algorithmen finden meist nicht nur eine partielle Projektion, sondern evtl. sehr viele. Es muß daher eine Möglichkeit geschaffen werden, die gefundenen Projektionen zu bewerten, um den Akteur herauszufiltern, der die geforderten Eigenschaften am besten erfüllt. Es muß also eine Metrik für die Bewertung partieller Projektionen erstellt werden.

An eine Metrik sind nach [**?**] diverse Anforderungen zu stellen:

- *Schärfe*: Eine Metrik sollte *differenziert* sein, d.h. sie sollte feingranular genug sein, um „Unterschiede, die auf andere Weise erkennbar werden, auch in der Bewertung sichtbar zu machen“. Außerdem sollte sie *exakte* Werte liefern (z.B. „2“ statt „ungefähr 2“). Alle vorgestellten Metriken bilden partielle Projektionen auf eine reelle Zahl aus dem Intervall $[0.0, 1.0]$ ab. Diese Abbildung ist exakt und auch differenziert genug, um auch kleine Unterschiede darzustellen.

- *Vergleichbarkeit*: Die Bewertungen von verschiedenen Objekten sollten zumindest vergleichbar sein. Gefordert ist also eine Abbildung auf mindestens eine Ordinalskala. Da die Metrik zur Bewertung verschiedener partieller Projektionen herangezogen werden soll, ist diese Eigenschaft selbstverständlich.

- *Reproduzierbarkeit*: Die Bewertung desselben Objekts unter gleichen Bedingungen sollte das gleiche Resultat liefern. Für Projektionen wird daher gefordert, daß die gleiche Anfrage auf der gleichen Wissensbasis die gleichen Ergebnisse liefert (was durch die vorgestellten Algorithmen gewährleistet ist) und daß gleiche Projektionen gleich bewertet werden.

- *Relevanz*: Das Ergebnis muß auf eine nachvollziehbare Weise mit der Realität korrelieren. Auf diesen Punkt ist bei dem Entwurf der Metrik speziell zu achten.

- *Verfügbarkeit* und *Rentabilität*: Das Ergebnis muß rechtzeitig vorliegen und mit vertretbarem Aufwand zu ermitteln sein. Was „rechtzeitig“ ist, ist natürlich von der Situation und den Prozessen abhängig. Für diese Arbeit wird eine Antwortzeit im Sekundenbereich als ausreichend erachtet. Der zeitkritische Teil wird in jedem Fall nicht die Berechnung der Qualität der Projektion aus einer vorliegenden Projektion, sondern die eigentliche Suche nach der partiellen Projektion selbst, also die Matching-Operation, sein (siehe Abs. 6.2.1).

Der Gegenstand der Bewertung sind partielle Projektionen. Mit $Q' = (C_{Q'}, R_{Q'}, E_{Q'}, root, P_{Q'})$ wird dabei wieder der Subgraph des Anfragegraphen $Q = (C_Q, R_Q, E_Q, root, P_Q)$ bezeichnet, der abgebildet wurde. Eine partielle Projektion ist nur dann sinnvoll zu verwenden, wenn mindestens der Wurzelknoten der Anfrage abgebildet wurde. Wurde nur der Wurzelknoten abgebildet, so soll die Bewertung den Wert 0.0 liefern, denn dann wurde ein Akteur gefunden, der keine der geforderten Eigenschaften besitzt. Die Menge $\{c_1, \ldots, c_m\} = C_Q \setminus root$ bezeichnet die Menge der Konzeptknoten des Anfragegraphen *ohne* den Wurzelknoten; die Menge $\{c'_1, \ldots, c'_n\} = C_{Q'} \setminus root$ die entsprechende Menge des abgebildeten Subgraphen. $R_Q = \{r_1, \ldots, r_k\}$ und $R_{Q'} = \{r'_1, \ldots, r'_l\}$ sind die Relationen des Anfragegraphen bzw. die abgebildeten Relationen des Anfragegraphen. Außerdem kann davon ausgegangen werden, daß der Anfragegraph aus mindestens zwei Knoten besteht, denn sonst wird ein Akteur mit keiner speziellen Eigenschaft gesucht, und somit kann auch nicht sinnvoll nach der Menge der gefundenen Eigenschaften bewertet werden.

Metrik 1 Eine sehr einfaches Maß für die Güte einer partiellen Projektion, das sich nur aus der Anzahl der abgebildeten Knoten des Anfragegraphen und der Gesamtknotenzahl des Anfragegraphen bestimmt, läßt sich wie folgt berechnen:

$$quality(\Pi_{Q \mapsto F}) = \frac{(|C_{Q'}| - 1) + |R_{Q'}|}{(|C_Q| - 1) + |R_Q|} \tag{5.4}$$

Hierbei wird von der Annahme ausgegangen, daß ein Mehr an abgebildeten Knoten auch gleichzeitig eine größere Übereinstimmung mit den geforderten Eigenschaften bedeutet. Diese Metrik läßt sich natürlich noch weiter verfeinern (siehe unten). Der Wurzelknoten wird aus den oben genannten Gründen nicht in die Bewertung mit einbezogen ($|C_{Q'}| - 1$, $|C_Q| - 1$).

Für die Erstellung der Anfragegraphen bedeutet diese Metrik keinerlei zusätzlichen Aufwand.

Metrik 2 Die im vorigen Abschnitt vorgestellte Metrik liefert für alle Fälle gut Ergebnisse, bei denen die Knoten des Anfragegraphen gleiches Gewicht haben, also nicht einzelne Knoten bzw. Eigenschaften als wichtiger als andere erachtet werden. Ist diese Voraussetzung nicht gegeben, so ist die Relevanz des Ergebnisses mehr oder weniger stark eingeschränkt. Man kann sich z.B. eine Anfrage vorstellen, die nach einem Akteur sucht, der sich am Produktionsstandort X befindet und neben anderen Fähigkeiten auch Englischkenntnisse besitzt. Evtl. sind Englischkenntnisse in diesem Fall nur wünschenswert, wohingegen der Aufenthalt am Produktionsstandort ein KO-Kriterium darstellt. Somit wäre ein Akteur, der sich am Ort X befindet und nicht Englisch spricht, einem Akteur vorzuziehen, der zwar Englisch spricht, sich aber nicht am Ort X befindet. Man kann dies in diesem Fall durch entsprechende Gewichtungen der Knoten des Anfragegraphen erreichen. Das heißt, jedem Konzept- und Relationsknoten v wird ein Gewicht $weight(v)$ zugeordnet. Die Güte einer Projektion berechnet sich nun wie folgt:

$$quality(\Pi_{Q \mapsto F}) = \frac{\sum_{i=1}^{m} weight(c'_i) \ + \ \sum_{i=1}^{l} weight(r'_i)}{\sum_{i=1}^{n} weight(c_i) \ + \ \sum_{i=1}^{k} weight(r_i)} \tag{5.5}$$

Ordnet man jedem Knoten das gleiche Gewicht 1.0 zu, so erhält man die Metrik aus Gleichung 5.4. Ist das Gesamtgewicht des Anfragegraphen null, so ist nach dieser Metrik die Güte der Abbildung nicht definiert. Es kann in diesem Fall auch keine sinnvolle Bewertung mehr stattfinden, denn wenn alle Knoten das Gewicht 0 haben, sind alle geforderten Eigenschaften unwichtig, und es kann nicht mehr aufgrund der gefundenen Eigenschaften bewertet werden.

Im Gegensatz zur ersten Metrik ist hier ein gewisser zusätzlich Aufwand bei der Spezifikation der Anfragegraphen zu verzeichnen, der sich aber in Grenzen hält und auch nur betrieben werden muß, wenn die Knoten wirklich unterschiedliche Gewichte haben (ansonsten kann man das Gewicht eine Knotens auf den Default-Wert 1.0 setzen).

Metrik 3 Betrachtet man den Anfragegraphen aus Abb. 5.22, so wird eine Schwachstelle von Metrik 2 deutlich. Angenommen, alle Knoten haben dasselbe Gewicht 1.0 und es kann keine totale Projektion auf ein Fakt der Wissensbasis gefunden werden, d.h. es existiert kein Akteur, der alle Anforderungen erfüllt. Dafür existiert ein Akteur, der Englisch spricht und ein anderer Akteur, der die geforderte Zeichnungsbefugnis für einen bestimmten Vertrag in Irland hat. Es existieren daher zwei partielle Projektionen, mit den Güten

$$quality(\Pi_{Q \mapsto F,1}) = \frac{weight(r_1) + weight(c_2)}{weight(r_1) + weight(c_2) + weight(r_2) + weight(c_3) + weight(c_4)} = \frac{2}{5}$$

$$quality(\Pi_{Q\mapsto F,2}) = \frac{weight(r_2) + weight(c_3) + weight(c_4)}{weight(r_1) + weight(c_2) + weight(r_2) + weight(c_3) + weight(c_4)} = \frac{3}{5}$$

$\Pi_{Q\mapsto F,2}$ scheint daher das bessere Ergebnis zu sein, obwohl beide Akteure jeweils eine Eigenschaft besitzen, die gleich bewertet wird. Somit würde man intuitiv erwarten, daß $weight(\Pi_{Q\mapsto F,1}) = weight(\Pi_{Q\mapsto F,2})$ gilt. Der Grund, warum Metrik 2 hier eine Abweichung

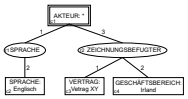

Abbildung 5.22: Beispiel 1 zu Metriken für partielle Projektionen

vom erwarteten Ergebnis liefert, ist folgender: Im Falle einer Nicht-Abbildung einer Relation kann der Fall eintreten, daß auch Argumente der Relation nicht abgebildet werden (wenn z.B. die Relation `(ZEICHNUNGSBEFUGTER)` nicht abgebildet wird, so werden auch die Konzepte `[VERTRAG: Vertrag XY]` und `[GESCHÄFTSBEREICH: Irland]` nicht abgebildet, da gefordert wurde, daß eine Relation nur dann abgebildet wird, wenn alle ihre Argumente abgebildet wurden. Somit bewertet die Metrik zum einen das Fehlen der Relation und zum anderen das Fehlen der Argumente der Relation, obwohl beides hier als eine Einheit gesehen werden müßte, denn die Relation inkl. ihrer Argumente stellt die geforderte Eigenschaft dar. Da die Abbildung einer Relation die Abbildung ihrer Argumente impliziert, kann auf die Bewertung der Konzepte vollständig verzichtet werden. Eine entsprechende Metrik würde wie folgt aussehen:

$$quality(\Pi_{Q\mapsto F}) = \frac{\sum_{i=1}^{l} weight(r'_i)}{\sum_{i=1}^{k} weight(r_i)} \tag{5.6}$$

Es handelt sich hierbei um einen Sonderfall der Metrik 2, bei dem alle Konzeptgewichte auf 0 gesetzt wurden. Die Bewertung der partiellen Projektionen des Beispieles ergibt nun in beiden Fällen den Wert $\frac{1}{2}$.

Einbeziehung der Typhierarchien in die Metrik Die bisherigen Metriken machten keinen Gebrauch von den Informationen, die in der Konzepttyphierarchie vorhanden sind. Theoretisch ist auch eine Metrik denkbar, die diese Informationen in die Bestimmung der Güte einer partiellen Projektion mit einbezieht. Ein Ausschnitt aus einer möglichen Hierarchie ist in Abb. 5.23a zu sehen. Für zwei gegebene Konzepten c_1 und c_2 mit den Typen t_1 und t_2 läßt sich eine *semantische*

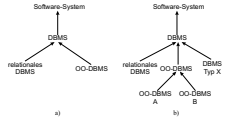

Abbildung 5.23: Metriken und Typhierarchien

Distanz dieser beiden Konzepte berechnen. Hierfür gibt es unterschiedliche Möglichkeiten. In [**?**] wird vorgeschlagen, die semantische Distanz so zu berechnen, daß sie dem kürzesten Pfad in der Typhierarchie entspricht, der t_1 und t_2 verbindet und nicht über $\perp$ führt. In [**?**] wird vorgeschlagen, zunächst den Typ $t_3 = t_1 \sqcap t_2$ zu bestimmen. Die semantische Distanz ergibt sich dann aus der Summe der Distanzen von t_1 zu t_3 und t_2 zu t_3. Die zweite Form der Berechnung wird nun auf ihre Verwendbarkeit für die Bestimmung der Güte einer partiellen Projektion überprüft.

Angenommen, das Konzept c_Q eines Anfragegraphen wird auf das Konzept c_F eines Fakts abgebildet. Nach der Definition der partiellen Projektion gilt $type(c_F) \leq type(c_Q)$. Das Konzept c_Q könnte also z.B. den Typ *DBMS* aus der Hierarchie aus Abb. 5.23a haben und das Konzept c_F den Typ *relationales DBMS*. Die beiden Konzepte haben somit die semantische Dinstanz 1. Ist es hier sinnvoll, überhaupt von einer Distanz zu sprechen? Es wurde z.B. ein Akteur gesucht, der Kenntnisse in einem Datenbank-Management-System (DBMS) hat. Gefunden wurde ein Akteur, der Kenntnisse in einem relationalen DBMS hat. Die gefundene Eigenschaft entspricht zu 100% der gesuchten, denn ein relationales DBMS ist ohne Abstriche ein DBMS, nur daß der Detaillierungsgrad des Fakts höher ist, als der der Anfrage. Nur wenn ein allgemeines DBMS eine Eigenschaft hätte, die ein relationales DBMS nicht hat, könnte man von einer semantischen Distanz sprechen. Dieser Fall tritt aber nur in „unsauberen" Typ- bzw. Klassenhierarchien auf. In sauberen Klassenhierarchien besteht zwischen Ober- und Unterklasse eine uneingeschränkte Ist-ein-Beziehung (in der objektorientierten Programmierung würde man es beispielsweise vermeiden, öffentliche Methoden oder Attribute der Oberklasse in der abgeleiteten Klasse zu verbergen, oder die Programmiersprache selbst würde das bereits verbieten). In diesem Fall bringt also die Betrachtung der semantischen Distanz, wie sie oben vorgeschlagen wurde, kein brauchbares Ergebnis.

Es macht also nur Sinn, von einer semantischen Distanz zwischen c_Q und c_F zu sprechen, wenn der Typ von c_F kein Subtyp von c_Q ist. Daher wird Eigenschaft 2 aus Def. 22 etwas gelockert und wie folgt umdefiniert:

$$\forall c_Q \in C_{Q'} : \; type(f(c_Q)) \sqcap type(c_Q) \neq \top$$

Das heißt, ein Konzept des Anfragegraphen kann nun auch auf ein Konzept des Fakts abgebildet werden, wenn die Typen beider Konzepte einen kleinsten gemeinsamen Supertyp haben, der nicht $\top$ entspricht. Eine Änderung dieser Art ließe sich ohne Problem in die Algorithmen 5.2 und 5.6 einbauen (siehe Z. 10 bzw. Z. 15). Zu beachten ist hierbei aber, daß sich die benötigte Rechenzeit des Koch-Algorithmus wesentlich erhöhen kann, da sich die Anzahl der Knoten des Association-Graphen evtl. stark erhöht.

Es stellt sich allerdings zunächst die Frage, ob nun die nach obiger Anweisung berechnete semantische Distanz sinnvoll verwendet werden kann. Hierzu wieder ein Beispiel: Gesucht ist ein Akteur, mit Kenntnissen in relationalen DBMS. Es existiert aber nur ein Akteur mit Kenntnissen in OO-DBMS. Nach der Umdefinition wird nun auch eine Projektion des Anfragegraphen auf das Fakt gefunden, das die Beschreibung für den Akteur mit den Kenntnissen in OO-DBMS enthält, da beide Typen den gemeinsamen Supertyp *DBMS* haben. Bei der Verwendung der Typhierarchie nach Abb. 5.23a haben die zwei Konzepte c_Q und c_F die semantische Distanz 2, entsprechend der Länge des Pfades vom Typ *relationales DBMS* zum Typ *OO-DBMS*. So weit, so gut. Nun sind aber auch alle Konzepte, die Software-Systeme irgendeiner Art darstellen, kompatibel zu c_Q, da sie einen gemeinsamen Supertyp mit dem Typ *relationales DBMS* von c_Q haben, d.h. es würde für jeden Akteur eine Projektion gefunden, vorausgesetzt, der Akteur hat eine Kenntnis in irgendeinem Software-System. Die semantische Distanz für diese Projektionen wäre allerdings relativ hoch (auf jeden Fall > 2). Man kann dieses Problem dadurch umgehen, daß man bei den Knoten des Anfragegraphen spezifiziert, welche semantische Distanz man notfalls bereit zu akzeptieren ist.[8]

Ein Problem der semantischen Distanz wird allerdings deutlich, wenn man die Hierarchie aus Abb. 5.23b betrachtet. Gesucht ist weiterhin ein Akteur mit Kenntnissen in relationalen DBMS. Allerdings hat man nun die Kenntnisse in OO-DBMS weiter unterteilt, nämlich in Kenntnisse

[8] Ein Vergleich auf die semantische Distanz beim Erstellen des Association-Graphen kann auch die Zahl der a-Knoten, die beim Koch-Algorithmus im Association-Graph erstellt werden, wieder stark reduzieren.

der Systeme der OO-DBMS des Herstellers A und B. Außerdem ist ein weiterer Typ von DBMS hinzugekommen. Es gibt nun weiterhin keinen Akteur, der Kenntnisse in relationalen DBMS hat, dafür Akteure, die Kenntnisse in OO-DBMS des Herstellers A oder Kenntnisse in DBMS des Typs X haben. Führt man nun das Matching durch, so wird ein Akteur mit Kenntnissen in den OO-DBMS des Herstellers A eine höhere semantische Distanz (3) aufweisen, als ein Akteur, der Kenntnisse in den DBMS des Typs X hat (semantische Distanz 2), und das alleine aus der Tatsache heraus, daß man die OO-DBMS detailierter beschrieben hat, als die Systeme des Typs X (in der Hierachie wurde bei OO-DBMS eine weiter Ebene eingefügt, die bei den DBMS des Typs X nicht vorhanden ist). Die höhere semantische Distanz ist eigentlich nicht in der Realität nachzuvollziehen. Daher wird hier eine neue Art der Berechnung der semantische Distanz eines Konzepts c_Q des Anfragegraphen zu einem Konzept c_F eines Fakts vorgeschlagen:

Definition 25 (Semantische Distanz von Anfragegraph-Knoten zu Fakt-Knoten) *Gegeben:*

- *ein Konzept c_Q eines Anfragegraphen*
- *ein Konzept c_F eines Fakts*

Es gelte $t_{super} = type(c_Q) \sqcap type(c_F)$. Dann ist $dist(c_Q, c_F)$ die semantische Distanz *zwischen c_Q und c_F mit*

$$dist(c_Q, c_F) = \begin{cases} \infty & \text{falls } t_{super} = \top \\ 0 & \text{falls } t_{super} = type(c_Q) \\ |\{t \mid t > type(c_Q) \wedge t \leq t_{super}\}| & \text{sonst} \end{cases}$$

Betrachtet man hiermit nochmals das Beispiel von oben, so hat nun ein OO-DBMS des Herstellers A genau die gleiche semantische Distanz zum gesuchten relationalen DBMS, wie ein DBMS des Typs X (beide haben die semantische Distanz 1). Diese semantische Distanz läßt sich auch in der Realität leicht begründen, denn schließlich sind sowohl OO-DBMS des Herstellers A als auch DBMS vom Typ X Datenbank-Systeme, und gesucht wurde ein solches System (wenn auch ein relationales DBMS). Somit ist diese Berechnung der semantischen Distanz gerade für Selektionen sehr viel leichter mit der Realität in Einklang zu bringen, als die zu Beginn vorgeschlagene.

Mir der so definierten semantischen Distanz zweier Konzepte läßt sich nun die semantische Distanz einer partiellen Projektion wie folgt definieren:

Definition 26 (Semantische Distanz einer partiellen Projektion) *Für die* semantische Distanz $dist(\Pi_{Q \mapsto F})$ *einer partiellen Projektion*[9] $\Pi_{Q \mapsto F}$ *mit* $Q = (C, R, E, root, P)$ *und* $\{c_1, \ldots, c_m\} = C_Q \setminus root$ *gilt*

$$dist(\Pi_{Q \mapsto F}) = \sum_{i=1}^{m} dist(c_i, \Pi_{Q \mapsto F}(c_i))$$

Dabei ist es wie bereits erwähnt sinnvoll, für jeden Konzeptknoten des Anfragegraphen eine maximale semantische Distanz anzugeben, die nicht überschritten werden darf. Dazu ist eine entsprechende Kenntnis der Typhierarchie notwendig. Daher muß auch angemerkt werden, daß man durch Änderungen an der Typhierarchie evtl. zu einer Überprüfung der Anfragegraphen gezwungen wird.

[9] Um von einer Distanz sprechen zu können, sind im allgemeinen zwei Punkte notwendig. Strenggenommen wird die semantische Distanz daher zwischen Subgraph Q' und dem Bild von Q' in F bestimmt.

Die so definierte semantische Distanz einer partiellen Projektion läßt sich *alleine* nicht sinnvoll für die Bestimmung der Güte einer Projektion einsetzen. Eine Projektion, die z.B. nur den Wurzelknoten der Anfrage abbildet, hat automatisch die semantische Distanz 0 (was in diesem Fall der beste Wert ist; je höher die semantische Distanz eine partiellen Projektion ist, desto mehr Typabweichungen sind zu verzeichnen). Eine Projektion, die mehrere Knoten abbildet, hat u.U. eine höhere semantische Distanz, ist aber sicherlich als besser zu bewerten. Außerdem werden semantische Distanzen wohl hauptsächlich für Konzeptknoten verwendet, die Kenntnisse/Fähigkeiten darstellen bzw. mit diesen in Verbindung stehen. Viele der Knoten des Anfragegraphen werden eine maximale akzeptable semantische Distanz von 0 (d.h. die bekannte Subtypbeziehung zwischen Anfragegraph-Knoten und Fakt-Knoten) haben. Daher erscheint eine Kombination mit den bereits bekannten Metriken angebracht. Folgende Metrik kombiniert die semantische Distanz einer partiellen Projektion mit Metrik 3:

$$quality(\Pi_{Q\mapsto F}) = \frac{\sum_{i=1}^{l} weight(r'_i)}{\sum_{i=1}^{k} weight(r_i)} \cdot \frac{1}{1 + \alpha \cdot dist(\Pi_{Q\mapsto F})} \tag{5.7}$$

Mit dem Parameter $\alpha \geq 0$ läßt sich dabei bestimmen, wie stark die semantische Distanz die Bewertung beeinflußt. Bei einem Wert von 0 wird die semantische Distanz vollständig ignoriert, und man erhält Metrik 3. Zur Demonstration dieser Metrik werden die Anfrage und die Fakten aus Abb. 5.24 verwendet. Ihnen liegt die Konzepttyphierarchie aus Abb. 5.23b zugrunde. Die

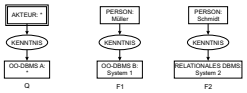

Abbildung 5.24: Beispiel 2 zu Metriken für partielle Projektionen

maximal erlaubte semantische Distanz für das Konzept `[OO-DBMS A: *]` sei 2 und $\alpha = 0.01$. Offensichlicht ist die gefundene partielle Projektion $\Pi_{Q\mapsto F_1}$ die bessere, denn mit ihr wurde eine Akteur gefunden, der zwar nicht ein OO-DBMS des Herstellers A, dafür aber ein OO-DBMS des Herstellers B kennt. Die zweite Projektion $\Pi_{Q\mapsto F_2}$ findet dagegen nur einen Akteur mit Kenntnissen in relationalen DBMS, wobei der Unterschied zwischen zwei OO-DBMS verschiedener Hersteller sicher als geringer angesehen werden kann, als der zwischen einem OO-DBMS und einem relationalen DBMS. Dieser Sachverhalt sollte sich nun auch in der Bewertung der Projektionen widerspiegeln:

$$dist(\Pi_{Q\mapsto F_1}) = \frac{1}{1} \cdot \frac{1}{1 + 0.01 \cdot 1} \approx 0.99$$

$$dist(\Pi_{Q\mapsto F_2}) = \frac{1}{1} \cdot \frac{1}{1 + 0.01 \cdot 2} \approx 0.98$$

Wie man sieht, ist das auch der Fall. Außerdem ist zu sehen, daß bereits kleine α-Werte ausreichend sind, um die semantische Distanz relativ stark in das Ergebnis einfließen zu lassen. Zu große α-Werte führen leicht zu einer Überbewertung der semantischen Distanz. Es ist daher ein sehr feinfühliges Abstimmen dieses Wertes notwendig, um die gewünschten Ergebnisse zu erzielen. Mit sehr kleinen α-Werten kann die semantische Distanz z.B. als „Zünglein an der Waage“ dienen.

5.3.8 Zuweisungsregelverwaltung

Eine Zuweisungsregel wird durch das Workflow-Management-System Changengine aktiviert, sobald eine Aufgabenträgerzuweisung zu erfolgen hat. Dabei können der Zuweisungsregel auch eine

Menge von workflowrelevanten Daten übergeben werden, die z.B. zur Selektion von Aufgabenträgern herangezogen werden können. Bei der Abarbeitung einer Zuweisungsregel werden Akteure mit entsprechenden Eigenschaften selektiert, indem zuerst eine entsprechende Anfrage in Form eines Anfragegraphen erzeugt wird, und dann die eigentliche Selektion durch Graph-Matching bzw. PROLOG-Anfrage stattfindet. Die Selektion wird vom Taskholder-Finder, einer Komponente des Organisationsstrukturverwaltung, durchgeführt (siehe Abs. 5.3.9). Bei der Ausführung einer Zuweisungsregel werden daher entsprechende Aufrufe an diese Komponente abgesetzt.

Changengine enthält bereits eine Zuweisungsregelverwaltung und eine eigene Sprache zur Definition von Zuweisungsregeln. Der Entwurf einer eigenen Komponente ist daher nicht notwendig. Notwendig ist allerdings die Integration der entworfenen Organisationsstrukturverwaltung in Changengine, so daß Selektionen von Aufgabenträgern aus Zuweisungsregeln heraus veranlaßt werden können. Diese Integration wird im Detail in Abs. 5.4 besprochen.

5.3.9 Klassenstruktur

Das System wird auf Grundlage der Notio-Bibliothek ([**?**][**?**]) für Konzeptgraphen entworfen und implementiert. Diese Java-Bibliothek stellt grundlegenden Klassen für die Manipulation von Konzeptgraphen zur Verfügung, die erweitert und ergänzt werden, um die Organisationsverwaltung, wie sie bisher vorgestellt wurde, realisieren zu können. Dieser Abschnitt liefert einen groben Überblick über die Klassenstruktur der entworfenen Organisationsverwaltung und beschreibt die wichtigsten Klassen genauer.

5.3.9.1 Basisklassen

Die grundlegenden Klassen sind in Abb. 5.25 zu sehen. Da die Klassen zur Repräsentation von Konzepten, Relationen, Konzept- und Relationstypen sowie Konzept- und Relationstyphierarchien von Notio nicht ausreichend sind, werden sie durch entsprechende Klassen erweitert (Abb. 5.25a).

Abb. 5.25b zeigt alle Klassen, die Konzeptgraphen darstellen. Im einzelnen sind das Fakten, Regeln und Anfragegraphen. Um die verschiedenen Möglichkeiten zur Selektion von Aufgabenträgern ideal unterstützen zu können, werden mehrere Klassen von Anfragegraphen verwendet. Die allgemeine Klasse `Query` wird verwendet, wenn die Selektion mittels Graph-Matching durch den Ullmann-Algorithmus (totale Projektionen) bzw. den Koch-Algorithmus (maximale partielle Projektionen) erfolgt. Die Klasse `TreeQuery` repräsentiert Anfragegraphen in Baumform, die die Voraussetzung für den Mugnier-Algorithmus ist. Die Klasse `PrologQuery` wird schließlich verwendet, wenn die Selektion mittels PROLOG durchgeführt wird.

Die eigentlichen Matching-Operationen werden von verschiedenen Klassen implementiert. Notio selbst stellt ebenfalls eine Implementierung der Matching-Operation zur Verfügung, die in der `Graph`-Klasse integriert ist. Ohne bereits das Ergebnis der Evaluation der Algorithmen vorwegzunehmen, kann die Notio-Implementierung aber verglichen mit den vorgestellten Algorithmen als ineffizient gelten.

5.3.9.2 Klassen der Wissensbasis

Abb. 5.26 zeigt die Klassen der Wissenbasis. Die Klassen `FileKnowledgebase` und `CompiledFileKnowledgebase` speichern alle Graphen und Typhierarchien in einfachen Dateien (eine Datei pro Graph oder Hierarchie). Die Klasse `CompiledFileKnowledgebase` implementiert

zusätzlich die vorgestellte Kompilation der Wissensbasis (Join auf individuellen Konzepten und Überführung in regelfreie Form), die die Voraussetzung für den Einsatz der Matching-Operation für die Selektion von Aufgabenträgern ist. Die Klasse `CompiledPROLOGKnowledgebase` besitzt die Fähigkeit, eine Wissensbasis in ein PROLOG-Programm zu überführen, so daß PROLOG zur Selektion von Aufgabenträgern verwendet werden kann.

Alle Klassen aus den CORBA-Paketen („from corba“) dienen der Bereitstellung der Funktionen der Wissensbasis über CORBA-Objekte. Da im weiteren Verlauf noch öfter von CORBA die Rede sein wird, erfolgt an dieser Stelle eine kurze Erläuterung von CORBA. Nähere Details zu CORBA finden sich z.B. in [**?**].

Einschub: Common Object Request Broker Architecture (CORBA) Allgemein erfüllt CORBA die bereits auf S. 22 genannten Aufgaben einer Middleware. Kern von CORBA ist der sogenannte Object Request Broker (ORB, siehe Abb. 5.27), der es verteilten Objekten ermöglicht, in heterogenen Umgebungen miteinander zu kommunizieren. Ein Klient kann dabei auf zwei Arten Anfragen an ein Server-Objekt absetzen, nämlich über das Dynamik Invocation Interface oder über die in dieser Arbeit ausschließlich verwendeten Stubs. Diese Stubs werden aus den Schnittstellendefinitionen des Server-Objekts, die in der sogenannten Interface Definition Language (IDL) verfaßt sind, durch einen IDL-Compiler erzeugt. Da die Implementierung in dieser Arbeit mit Java erfolgt, wird hier ein IDL-Compiler verwendet, der Java-Code für die Stubs erzeugt. Außerdem werden durch diesen IDL-Compiler auch die Server-Skeletons erzeugt, über die das Server-Objekt einen Aufruf empfängt und über die es die Ergebnisse des Aufrufs zurückliefert. Diese Skeletons sind in den `ImplBase`-Klassen enthalten, die teilweise in den Klassendiagrammen zu sehen sind.

□

Bei den Klassen der CORBA-Pakete handelt es sich also um schlichte Wrapper-Klassen, die Anfragen an die eigentlichen Klassen zur Bearbeitung weiterleiten und das Ergebnis einer Operation an den Anfragesteller zurückliefern. Da die Benutzungsschnittstellen und andere Klienten nicht notwendigerweise auf demselben Rechner anzutreffen sind, wie die Objekte der Wissensbasis, ermöglichen die Wrapper-Klassen eine Kommunikation über ein Netzwerk (siehe auch Abs. 5.4.2 und Abs. 5.5). Auch die Typhierarchien sind hierbei über CORBA ansprechbar (Abb. 5.26 unten). Die kompletten Schnittstellendefinitionen der Wissensbasis finden sich in Abs. C.1.

5.3.9.3 Klassen der Anfragegraphen-Verwaltung

Abb. 5.28 zeigt die Klassen der Anfragegraphen-Verwaltung (Querybase). Auch die Anfragegraphen werden in einfachen Dateien gespeichert (ein Anfragegraph pro Datei). Es gibt hierbei jeweils eigene Klassen für die Verwaltung „normaler“ Anfragegraphen und die Verwaltung von baumförmigen Anfragegraphen, die für die Selektion mit dem Mugnier-Algorithmus benötigt werden. Letztere enthält zusätzlich Routinen zur Überprüfung der geforderten Baumform, so daß sichergestellt werden kann, daß nur baumförmige Anfragegraphen in die Anfragegraphen-Verwaltung aufgenommen werden.

Wie die Wissensbasis, so bietet auch die Anfragegraphen-Verwaltung ihre Dienste mittels eines CORBA-Wrapper-Objekts an. Die Schnittstellendefinitionen der Anfragegraphen-Verwaltung finden sich in Abs. C.2 dieser Arbeit.

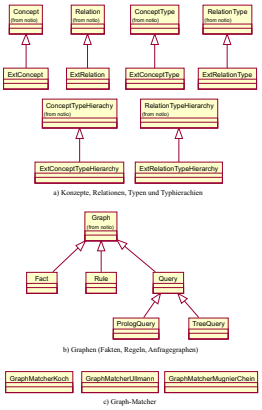

Abbildung 5.25: Basisklassen

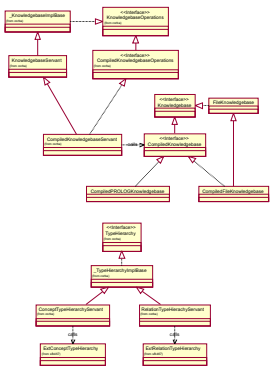

Abbildung 5.26: Klassen der Wissensbasis

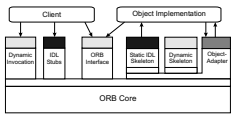

Abbildung 5.27: Object Request Broker nach [**?**]

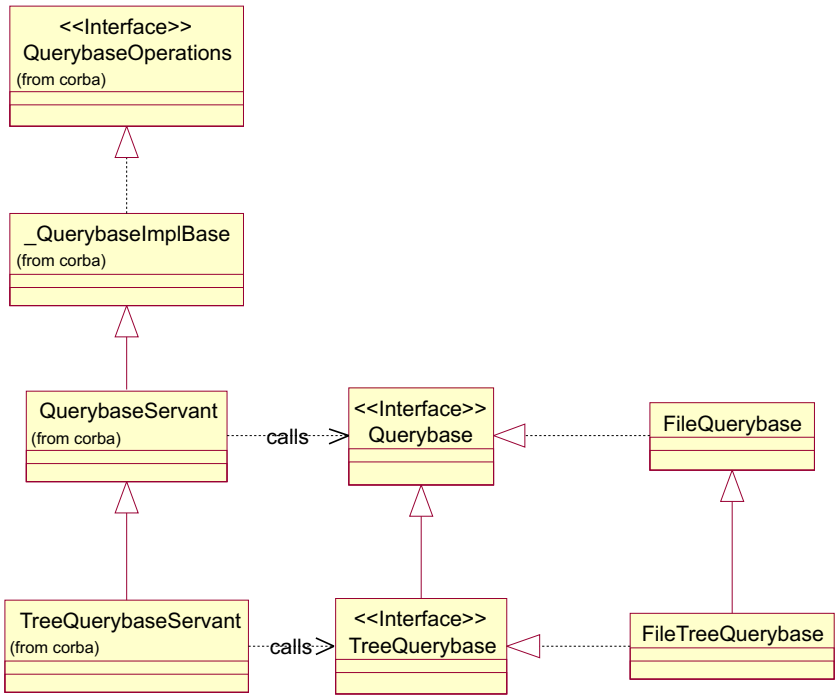

Abbildung 5.28: Klassen der Anfragegraphen-Verwaltung

5.3.9.4 Klassen der Taskholder-Finder-Komponente

Die Taskholder-Finder-Komponente ist für die eigentliche Selektion von Aufgabenträgern zuständig (siehe auch Abs. 5.4.2). Diese Komponente bietet ihren Dienst ebenfalls über eine CORBA-Schnittstelle an (siehe Abs. C.3), so daß Changengine diese Komponente auch ansprechen kann, wenn die Organisationsverwaltung auf einem anderen Rechner als das restliche Workflow-Management-System lokalisiert ist.

5.4 Integration der Organisationverwaltung in das Workflow-Management-System Changengine

Dieser Abschnitt beschreibt, wie die erstellte Organisationsverwaltung in das kommerzielle Workflow-Management-System Changengine der Firma Hewlett-Packard integriert werden kann. Changengine ist ein sehr offenes System, das Erweiterungen relativ leicht zuläßt. Dieser Abschnitt stützt sich dabei auf die Beschreibungen aus [**?**].

5.4.1 Aufgabenträgerzuweisung in Changengine

Der Ablauf der Aufgabenträgerzuweisung ist bei Changengine praktisch identisch zum bereits in Abs. 5.2 vorgestellten Ablauf. Zuweisungsregeln werden bei Changengine als Ressourcen-Regeln (engl. Resource Rule) bezeichnet. Sie legen fest, wie der passende Akteur für eine Aktivität zu bestimmen ist. Ausgeführt werden Regeln vom bereits in Abb. 5.1 dargestellten Resource Executive. Das Ergebnis der Ausführung einer Regel durch den Resource Executive ist eine Menge von Adressen, die den entsprechenden Akteuren zugeordnet sind. Eine Ressourcen-Regel besteht

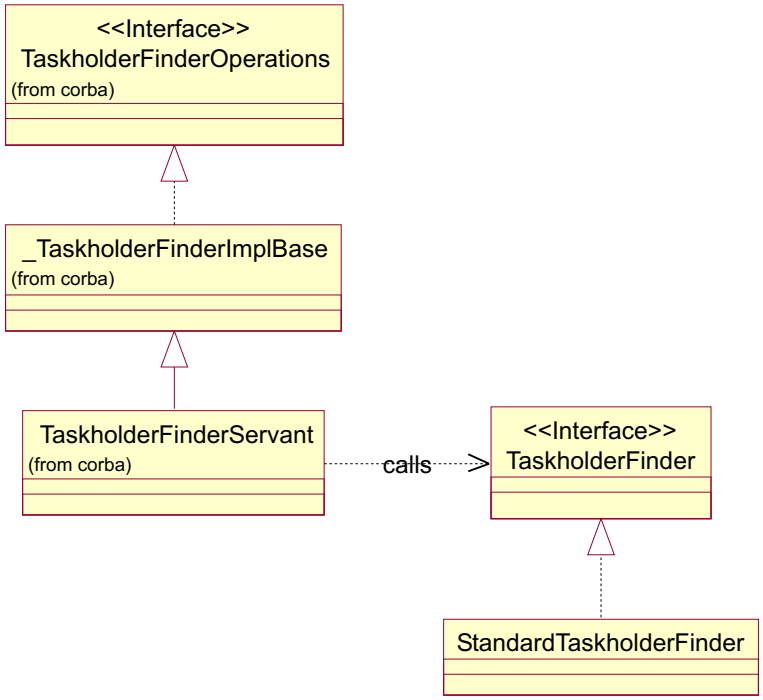

Abbildung 5.29: Klassen der Taskholder-Finder-Komponente

im wesentlichen aus einem oder mehreren Aufrufen von Funktionen sogenannter Ressourcen-Agenten. Folgendes Beispiel zeigt eine Regel mit drei geschachtelten Aufrufen von Funktionen verschiedener Agenten.

```
HEADER
    NAME "mananger_of_initiator"
END_HEADER
RULE
    EMail.Adrs(
        OrgChart.Mngr(
            CP.Get("API_AF_INITIATOR")
        )
    )
END_RULE
```

Listing 5.7: Beispiel einer Ressourcen-Regel des Changengine-Systems

`CP.Get("API_AF_INITIATOR")` ist ein Aufruf des sogenannten Case-Packet-Agents. Mit Hilfe dieses internen Ressourcen-Agenten lassen sich Daten des aktuellen Workflows ermitteln. Auf welche Daten zugegriffen werden kann, wird in der Prozeßdefinition festgelegt. Der hier verwendete Aufruf liefert als Rückgabewert den Initiator des Workflows. `OrgChart.Mngr` ruft den OrgChart-Agenten auf. Dieser ermittelt aus der Organisationsstruktur den Bezeichner des Managers eines Mitarbeiters, in diesem Fall also den Manager des Workflow-Initiators. `EMail.Adrs` ermittelt durch den Aufruf der Funktion `Adrs` des EMail-Agenten die E-Mail-Adresse des Managers. Das Ergebnis dieser Ressourcen-Regel ist also die E-Mail-Adresse des Managers des Workflow-Initiators.

5.4.2 Einbindung der erstellten Organisationsverwaltung

Die externen Ressourcen-Agenten EMail und OrgChart, die im obigen Beispiel verwendet wurde, waren beide fiktiv und dienten nur der Veranschaulichung. Changengine liefert allerdings bereits mehrere externe Ressourcen-Agenten mit, so z.B. einen Agenten zum Zugriff auf eine Datenbank über ODBC[10] oder einen Agenten zum Zugriff auf Verzeichnisse über LDAP[11]. Zur Integration eigener Ressourcen-Agenten bietet Changengine den sogenannten Universal Resource Agent (URA) an. Der URA kann beliebige externe Programme aufrufen, die entsprechende Ergebnisse an den URA zurückliefern, die dieser dann wiederum an den Resource Executive weitergibt.

Der URA wird hier verwendet, um die erstellte Organisationverwaltung an Changengine anzubinden. Abb. 5.30 zeigt, wie diese Einbindung realisiert wird.

1. Die Workflow-Engine ruft den Resource Executive auf und teilt diesem mit, welche Ressourcen-Regel zu verwenden ist.

2. Der Resource Executive startet die Abarbeitung der Ressourcen-Regel.

3. Um Akteure zu selektieren, muß eine Anfrage an die entworfene Organisationsstrukturverwaltung erfolgen. Dazu ruft der Resource Executive eine Funktion des URA auf. Dieser Funktionsaufruf entspricht einer durchzuführenden Selektionsregel, d.h. für jede Selektionsregel der Organisationsstrukturverwaltung (also für jeden Anfragegraphen) wird eine Funktion des URA definiert. Um Changengine mitzuteilen, welche Funktionen und somit welche Selektionsregeln angeboten werden, muß ein entsprechender Eintrag im URA-Description-File vorhanden sein. Ein solcher Eintrag bestimmt, welches externe Programm mit welchen Parametern aufzurufen ist und welche Ergebnisse dieser Aufruf zurückliefert.

4. Das externe Programm, das in diesem Fall aufzurufen ist, ist der Taskholder-Finder-Client. Der Taskholder-Finder bietet seinen Dienst über eine CORBA-Schnittstelle an. Das hat den Vorteil, daß die Organisationsstrukturverwaltung nicht auf demselben Rechner wie Changengine laufen muß. Da der URA allerdings nicht direkt eine Anfrage über CORBA absetzen kann, wird der Taskholder-Finder-Client als Vermittler zwischen Changengine und der Organisationsstrukturverwaltung benötigt. Die Anfrage besteht aus dem Bezeichner eines Anfragegraphen und evtl. aus weiteren Parametern, die als Referenten in die entsprechenden Parameter-Konzepte des Anfragegraphen eingesetzt werden, bevor die eigentliche Selektion stattfindet. Somit kann ein Anfragegraph mit Parametern als eine Art Schablone für eine konkrete Anfrage dienen. Der URA und der Taskholder-Finder-Client kommunizieren über eine bidirektionale Pipe.

5. Der Taskholder-Finder-Client leitet die Anfrage über einen CORBA-Aufruf an den Taskholder-Finder weiter.

6. Der Taskholder-Finder führt die Anfrage mit den gewünschten Parametern aus.

7. Der Taskholder-Finder liefert das Ergebnis zunächst an den Taskholder-Finder-Client zurück.

8. Der Taskholder-Finder-Client leitet das Ergebnis an den URA weiter.

9. Der URA leitet die gefundenen Akteure an den Resource Executive weiter.

[10] Open Database Connectivity
[11] Lightweight Directory Access Protocol

10. Der Resource Executive liefert die gefundenen Akteure an die Workflow-Engine aus.

Folgende Schritte sind nun neben der eigentlichen Modellierung der Organisation und dem Erstellen des Prozeßmodells durchzuführen, bevor eine Aufgabenträgerzuweisung erfolgen kann:

1. Erstellung der von der Zuweisungsregel benötigten Anfragegraphen (Selektionsregeln). Ein Anfragegraph kann dabei von mehreren Zuweisungsregeln verwendet werden (Stichwort Wiederverwendung).
2. Erstellung eines Eintrags für den erstellten Anfragegraphen im URA-Description-File. Hiermit wird Changengine mitgeteilt, welche Anfragen der Taskholder-Finder anbietet und wie diese über den URA aufzurufen sind.
3. Erstellung der Zuweisungsregel (Ressourcen-Regel).

5.5 Integration der Benutzungsoberfläche

Mehrere Objekte sind zu erstellen, bevor eine Aufgabenträgerzuweisung erfolgen kann:

1. Fakten
2. Regeln
3. Konzept- und Relationstyphierarchien
4. Anfragegraphen
5. Zuweisungsregeln
6. URA-Description-File

Der Punkt 6 ist zwar für die vorgestellte Integration in Changengine wichtig, aber hier vernachlässigbar. Zuweisungsregeln lassen sich mit einem einfachen Texteditor erstellen. Wesentlich interessanter für den vorgestellten Ansatz sind die Punkte 1–4. Zwar existiert eine Textform für Konzeptgraphen, diese kann aber als relativ umständlich in der Eingabe durch den Benutzer gelten. Da es sich bei Fakten, Regeln und Anfragegraphen um Graphen handelt, bietet sich geradezu eine grafische Form der Eingabe an. Ebenfalls lassen sich die Typhierarchien leichter mit einer grafischen Benutzungsoberfläche eingeben und ändern. Wie bereits erwähnt, befaßt sich die Diplomarbeit [**?**] mit den Möglichkeiten zur Eingabe von Konzeptgraphen. Die dort erstellte Benutzungsschnittstelle muß in den vorgestellten Entwurf integriert werden. Konkret bedeutet dies, daß die Benutzungsschnittstelle Graphen und Hierarchien an die Organisationsverwaltung übermitteln muß, so daß diese mit ihnen arbeiten kann. Außerdem sind neben dem Einfügen auch Funktionen für das Auslesen und das Löschen von Elementen der Organisationsstrukturverwaltung und der Anfragegraphenverwaltung vorzusehen. Es kann hierbei nicht davon ausgegangen werden, daß die Organisationsstrukturverwaltung auf demselben Rechner wie die Benutzungsschnittstelle ausgeführt wird. Die Wissensbasis und die Anfragegraphenverwaltung werden daher in Form von CORBA-Objekten der Benutzungsschnittstelle zugänglich gemacht, so daß insgesamt eine mehrschichtige Architektur der Organisationsstrukturverwaltung entsteht. Auf der ersten Schicht befindet sich die Benutzungsschnittstelle, auf der zweiten das Taskholder-Finder-Objekt und auf der dritten die Wissensbasis und die Anfragegraphenverwaltung. Abb. 5.31 zeigt nochmals graphisch die Anbindung der Benutzungsschnittstelle.

5.6 Entwurf einer Unternehmens-Ontologie

Es stehen nun alle technischen Mittel zur Verfügung, um eine Organisationsstruktur zu modellieren und Aufgabenträger zu selektieren. Bevor jedoch eine Organisationsstruktur modelliert werden kann, muß zunächst Klarheit über die Begriffe geschaffen werden, mit denen die Organisation beschrieben werden soll. Hierzu ist eine Ontologie zu definieren, auf deren Basis die Modellierung der Organisationsstruktur schließlich stattfindet. Die Auswahl der Begriffe erfolgt hier mit dem Ziel, eine möglichst starke Entkopplung von Ablauf- und Aufbauorganisation zu ermöglichen. Außerdem sollen beliebige Organisationsstrukturen modellierbar sein.

Bevor jedoch eine Ontologie entworfen werden kann, muß zunächst geklärt werden, was genau mit Hilfe der zu entwerfenden Ontologie beschrieben werden soll. Folgende Punkte werden als besonders wichtig erachtet:

- *Struktur der Organisation*: Bereits in Abs. 2.1.2.1 wurde klar, daß in der Praxis viele verschiedenen Organisationsformen anzutreffen sind. Die Ontologie darf hier nicht einschränkend wirken. Es sollen auf jeden Fall verschiedenen Organisationseinheiten, sowie Unterstellungsverhältnisse modellierbar sein.

- *Fähigkeiten*: Jeder Akteur hat eine bestimmte Menge an Fähigkeiten, die aus den bereits bekannten Gründen explizit modelliert werden sollen.

- *Rechte*: Eine Menge von Fähigkeiten alleine genügt oft nicht, um eine Aufgabenträgerzuweisung vorzunehmen. Ein Akteur muß auch die entsprechenden Berechtigungen zur Durchführung einer Aktivität besitzen. Daher müssen die Rechte eines Akteurs ebenfalls explizit modellierbar sein.

Fähigkeiten und Rechte sind zentrale Begriffe eines Unternehmens und ermöglichen, wie bereits Abs. 3.3.5 gezeigt hat, eine weitgehende Entkopplung von Ablauf- und Aufbauorganisation.

Es wird hierbei, wie bereits in der Einleitung zu Abs. 5.3 erwähnt wurde, kein kompletter Satz an Begriffen angestrebt, mit dem alle Eventualitäten abgedeckt werden können. Vielmehr soll eine Ontologie erstellt werden, die eine solide Basis darstellt und in die ohne Schwierigkeiten weitere Begriffe eingefügt werden können, so daß ein Unternehmen ohne Probleme die Ontologie an ihre eigenen, oft sehr unterschiedlichen Bedürfnisse anpassen kann, *ohne* eine komplett neue Ontologie erstellen zu müssen. Idealerweise erfolgt das Einfügen neuer Begriffe durch einfache Subtypbildung bestehender Typen, wobei die Basistypen von den Änderungen unberührt bleiben. Bereits Abs. 5.3.1 hat einen ersten Schritt zur Erstellung einer solchen Ontologie getan, indem der Konzepttyp *Agent* definiert wurde. Alle weiteren Begriffe sind als Vorschlag zu verstehen und erheben keinen Anspruch auf Vollständigkeit.

Bereits die oben vorgestellte Auswahl an zu modellierenden Eigenschaften bietet Grundlage zur Diskussion. Man könnte z.B. einwänden, daß neben Fähigkeiten und Rechten auch die Pflichten bzw. Aufgabenbereiche eines Akteurs zu modellieren sind. Im weiteren wird jedoch wie in [**?**] davon ausgegangen, daß ein ein Akteur, der die Fähigkeiten und das Recht zur Durchführung einer bestimmten Aktivität hat, automatisch als Aufgabenträger in Frage kommt.

In der Literatur wurden bereits diverse Unternehmens-Ontologien vorgeschlagen (siehe z.B. [**?**],[**?**]). Als eine der am weitesten gediehenen Ontologien kann hierbei die Enterprise-Ontologie gelten, an deren Entstehung die Universität Edingburgh maßgeblich beteiligt ist. Ziel des Enterprise-Projekts ist die Verbesserung und wo nötig Ersetzung von Modellierungs-Methoden

durch ein Framework zur Integration von Methoden und Werkzeugen, die zur Unternehmensmodellierung und der Handhabung von Veränderungen geeignet sind. Man erhofft sich dadurch bessere Planungsmöglichkeiten, größere Flexibilität, effektivere Kommunikation und Integration.

Die Enterprise-Ontologie selbst besteht aus mehreren Teilen, die in [**?**] näher beschrieben werden:

- *Meta-Ontologie*: Begriffe, die für die Definition der eigentlichen Ontologie verwendet werden
- *Aktivitäten und Prozesse*: Begriffe, die mit Prozessen und der Planung verbunden sind
- *Organisation*: Begriffe der Unternehmensstukturierung
- *Strategie*: Begriffe des „high level plannings“
- *Marketing*: Verkauf von Produkten, Dienstleistungen

Der Vollständigkeit halber findet sich eine komplette Auflistung aller in der Enterprise-Ontologie definierten Terme in Abs. A.1. Für diese Arbeit sind allerdings nur die Punkte 2 und 3 von Interesse. Wo es sinnvoll erscheint, wird auf die Enterprise-Ontologie Bezug genommen.

5.6.1 Organisationsstruktur

Betrachtet man die Begriffe der Enterprise-Ontolgie (Abs. A.1, Spalte 2), so sind für die Definition der Organisationsstruktur besonders folgende Begriffe interessant:

- *Person*: Personen sind vor allem für die Aufgabenträgerzuweisung von Interesse, da sie Aktivitäten ausführen können.
- *Machine*: Eine Maschine kann ebenso wie eine Person in der Lage sein, eine Aktivität durchzuführen.
- *Organisational Unit*
- *Management Link*: Diese Beziehung zwischen zwei Akteuren definiert das Vorgesetzten-Verhältnis.

Personen und Maschinen sind ungefähr den Mitarbeitern und Server-Prozessen aus Abb. 2.11 vergleichbar. Beide sind in der Lage, Aktivitäten auszuführen. Daher werden in der Ontologie eine Klasse *Agent* und zwei Subklassen *Akteur* und *Maschineller Agent* definiert, wobei unter einem maschinellen Agenten auch ein „Stück Software“ (ein Prozeß) zu verstehen ist.

Die *Organisationseinheit* kann als Basisklasse für die verschiedenen Formen von Organisationeinheiten aus Abs. 2.1.2.1 dienen. Organisationseinheiten sind beispielsweise Abteilungen und Divisionen. Es erscheint sinnvoll, zwischen Organisationseinheiten der Primär- und Sekundärorganisation bzw. zwischen festen und zeitlich beschränkten Organisationseinheiten zu unterscheiden. Ein Projekt ist z.B. eine zeitlich beschränkte Organisationseinheit, wohingegen eine Abteilung der Klasse der festen Organisationseinheiten zuzuordnen ist. Beliebige Typen von Organisationseinheiten lassen sich somit durch einfache Ableitung des Typs *Organisationseinheit* bilden.

Eine typische Relation zwischen Organisationseinheiten aber auch zwischen Organisationseinheiten und Personen ist eine Teil-Ganzes-Beziehung. So kann z.B. eine Abteilung Teil einer Division

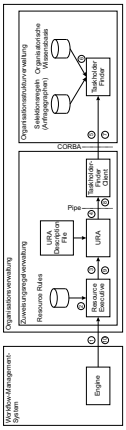

Abbildung 5.30: Integration der Organisationsverwaltung in Changengine

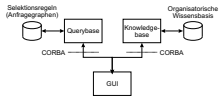

Abbildung 5.31: Integration der Benutzungsschnittstelle

Abbildung 5.32: Ontologie mit Agenten und Organisationseinheiten

und eine Person ein Teil einer Abteilung und/oder eines Projekts sein. Um diese Beziehungen auszudrücken, werden die Relationen *Teil* bzw. *Mitglied* eingeführt.

Das Vorgesetzten-Verhältnis kann einfach über eine Relation *Vorgesetzter* modelliert werden, die zwei Akteure miteinander verbindet. Theoretisch können auch verschiedene Formen von Vorgesetzten mit Weisungsbefugnissen in verschiedenen Bereichen durch mehrere Subtypen der Vorgesetzten-Relation unterschieden werden. Es besteht hierbei auch die Möglichkeit, einem Akteur mehrere Vorgesetzte zuzuweisen. Somit ist es beispielsweise möglich, eine Matrix-Projekt-Organisation (siehe Abs. 2.1.2.2) zu modellieren, bei der eine Person zum einen einem Projektleiter unterstellt ist, zum anderen aber auch einen Linienvorgesetzten hat. Gehört die Person verschiedenen Projekten an, so kann sie auch verschiedene Projektleiter als Vorgesetzte haben. Man könnte hier z.B. eine Relation *Vorgesetzter Linie* und *Vorgesetzter Projekt* einführen, um die verschiedenen Vorgesetztenverhältnisse auszudrücken. Weitere Relationen wie z.B. *Leiter* sind denkbar, um einen Akteur als Leiter einer Organisationseinheit zu definieren, wobei hier aber auch evtl. entsprechende Rollen (siehe unten) verwendet werden können.

Zu beachten ist, daß durch geeignete Definitionen von Vorgesetzten-Verhältnissen und Teil/Ganzes-Beziehungen im vorgestellten Ansatz nicht nur einfache Einlinienorganisationen beschrieben werden können, wie das bei den meisten kommerziellen Systemen der Fall ist (siehe Abs 3.1). Es sind vielmehr beliebige Organisationsformen modellierbar.

5.6.2 Fähigkeiten

Bestimmte Aktivitäten können nur von Akteuren mit den notwendigen Fähigkeiten durchgeführt werden. Anstatt Fähigkeiten implizit Rollen zuzuordnen, und bei der Zuweisung nach Rollen zu suchen, werden hier Fähigkeiten explizit als Eigenschaft eines Akteurs modelliert.

Es gibt mehrere Möglichkeiten zur Modellierung von Fähigkeiten. Eine sehr einfache wäre, einen Konzepttyp *Fähigkeit* und eine Relation *besitzt Fähigkeit* einzuführen, die einen Akteur mit einer Fähigkeit verbindet. Die eigentliche Fähigkeit würde dann als Referent des Fähigkeiten-Konzepts modelliert. Eine Fähigkeiten-Modellierung könnte dann z.B. so aussehen:

[AKTEUR: Meyer]→(BESITZT FÄHIGKEIT) −
→[FÄHIGKEIT: Java−Programmierung].

Durch diese flache Modellierung von Fähigkeiten verschenkt man aber die Möglichkeit, eine Hierarchie von Fähigkeiten zu erzeugen. Zum Beispiel könnte man die Programmierung mit Java als einen Subtyp der objektorientierten Programmierung ansehen und diese Information bei der Suche nach geeigneten Akteuren ausnutzen (die Bestimmung der semantischen Distanz in Abs. 5.3.7.3 für die Bestimmung des besten Matches erfolgt z.B. über die Typhierarchie). Die folgende Fähigkeiten-Modellierung nutzt diese Möglichkeit aus.

In [**?**] wird eine Ontologie speziell für die Modellierung von Fähigkeiten vorgeschlagen. Hier wird zwischen der eigentlichen Fähigkeit, etwas zu tun und den Objekten, mit denen etwas getan wird, unterschieden. Hieraus ergeben sich zwei Hierarchien. Eine Hierarchie von Fähigkeiten und eine Hierarchie von Anwendungsgebieten. Als typische Fähigkeit wird das Entwerfen eines Software-Systems genannt, wobei das Entwerfen die eigentliche Fähigkeit und das Software-System das Anwendungsgebiet ist. Eine Fähigkeit ließe sich somit z.B. als eine dreistellige Relation mit den Parametern Akteur, Tätigkeit und Anwendungsgebiet beschreiben. Es sind aber noch Erweiterungen denkbar. Evtl. ist es manchmal sinnvoll, auch das Werkzeug, mit dem etwas getan wird, mit in die Fähigkeit aufzunehmen („Akteur Meyer hat die Fähigkeit, Tätigkeit X *mit Werkzeug Y* auf Objekt Z durchführen“). Außerdem kann es auch Sinn machen, eine Bewertung

der Fähigkeit vorzunehmen („Akteur Meyer kann *sehr gut* objektorientiert programmieren“). Verschiedene mehrstellige Relationen sind hier nicht sinnvoll. Würde man nur eine allegemeine mehrstellige Relation verwenden, so müßte man bei einzelnen Argumenten evtl. Werte erfinden, denn vielleicht will oder kann man z.B. gar kein Werkzeug angeben. Mehrere mehrstellige Relationen würde eine mehrfache Modellierung ein und derselben Eigenschaft nötig machen, denn da verschiedenstellige Relationen nicht zueinander kompatibel sind und man im voraus nicht weiß, welche Relation mit welcher Stelligkeit zur Suche verwendet wird, müßte man für jede mögliche Relation eine Definition vornehmen („Meyer hat die Fähigkeit, Tätigkeit X auszuführen“, „Meyer hat die Fähigkeit, Tätigkeit X mit Werkzeug Y auszuführen“, usw.). Es wird daher eine andere Form der Modellierung vorgeschlagen.

Zunächst ist eine Fähigkeit eine zweistellige Relation, die einen Akteur mit einer Tätigkeit verbindet, falls der Akteur in der Lage ist (die Fähigkeit besitzt), die Tätigkeit auszuführen. Der Tätigkeit lassen sich nun über weitere Relationen Eigenschaften zuweisen. Abb. 5.33 zeigt ein Beispiel für die Fähigkeiten-Beschreibung „Akteur Müller kann gut mit dem CASE-Tool abc Software-Systeme entwerfen“. Die Relation *OBJ* ordnet der Tätigkeit den Anwendungsbereich bzw. die Objekte zu, auf denen die Tätigkeit ausgeführt werden kann. Die Relation *WKZ* bestimmt das Werkzeug, mit dem der Akteur in der Lage ist, die Aktivität auszuführen. Die Relation *ART* bestimmt schließlich, wie die Tätigkeit vom Akteur ausgeführt werden kann (gut oder schlecht, schnell oder langsam, o.ä.). Diese Art und Weise wird als Konzepttyp angelegt, um auch hier die Vererbungshierarchie nutzen zu können. So kann *sehr gut* ein Subtyp von *gut* sein. Somit ist z.B. `[SEHR GUT]` immer auch passend, wenn nach `[GUT]` gesucht wird (aber natürlich nicht umgekehrt). Es lassen sich nahezu beliebige weitere Eigenschaften mit einer Tätigkeit verbinden. Außerdem ist es auch denkbar, z.B. den Objekten, auf denen die Tätigkeit ausgeführt werden kann, oder den Werkzeugen weitere Eigenschaften zuzuweisen.

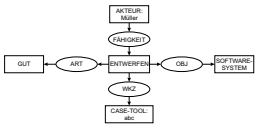

Abbildung 5.33: Fähigkeit

Welche Tätigkeiten für ein Unternehmen von Interesse sind, ist sicher von Fall zu Fall unterschiedlich. Typische Tätigkeiten könnten z.B. sein (wieder ohne Anspruch auf Vollständigkeit): Analysieren, Installieren, Warten, Reparieren, Programmieren, Modellieren, Entwerfen, Bearbeiten, Beschaffen, usw. Weitere Fähigkeiten können z.B. [**?**] entnommen werden. Es wird hier darauf verzichtet, die verschiedenen Tätigkeiten näher zu klassifizieren. Statt dessen wird nur ein gemeinsamer Supertyp *Tätigkeit* in die Ontologie eingefügt, von dem die verschiedenen Tätigkeiten abgeleitet werden können. Somit kann von jedem Unternehmen selbst bestimmt werden, welche Tätigkeiten relevant sind und wie diese evtl. weiter zu untergliedern sind.

Ebenfalls in die Konzepttyphierarchie aufgenommen werden die Objekte und Werkzeuge auf bzw. mit denen eine Tätigkeit durchgeführt werden kann. Auch hier kann an dieser Stelle keine vollständige Liste von Objekten angegeben werden, da diese wieder stark vom Anwendungsgebiet abhängig ist. Einige typische Beispiele sind Abb. 5.34 zu entnehmen. Dokumente können z.B. Gegenstand einer Tätigkeit sein (z.B. das Bearbeiten/Erstellen eines Dokuments). Entwicklungswerkzeuge, Maschinen, Computerprogramme, o.ä. sind typischerweise Werkzeuge (solange das Unternehmen nicht selbst solche Werkzeuge herstellt). Welche Rolle ein Objekt jeweils spielt, ist an der Relation zu erkennen, die das Objekt mit einer Tätigkeit verbindet.

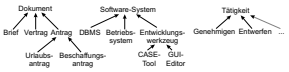

Abbildung 5.34: Beispiele für Objekte und Werkzeuge

5.6.3 Rechte

Wie Fähigkeiten, so lassen sich auch Rechte über Tätigkeiten bestimmen. Eine Tätigkeit stellt eine Befugnis dar, wenn ein Akteur berechtigt ist, die entsprechende Tätigkeit durchzuführen, d.h. dieses Mal wird eine Tätigkeit über die binäre Relation *Befugnis* mit einem Akteur verbunden. Auch hier kann eine Tätigkeit wieder mit weiteren Eigenschaften (z.B. einem Objekt, auf dem die Tätigkeit ausgeführt werden darf) verbunden werden. Ein typisches Beispiel ist die Genehmigung eines Antrages. Als Konzeptgraph könnte die Berechtigung zur Genehmigung eines Antrages wie folgt aussehen:

[AKTEUR: Meyer]→(BEFUGNIS)→[GENEHMIGUNG] −
→(OBJ)→[URLAUBSANTRAG].

Eine weitere Eigenschaft, die sich mit der Genehmigung oder einer Befugnis allgemein verbinden läßt, ist die Zuordnung einer Organisationseinheit. Diese Zuordnung entspricht dem Wirkungsbereich bzw. Zuständigkeitsbereich nach [**?**], d.h. ein Recht läßt sich durch die Relation *Wirkungsbereich* auf eine bestimmte Organisationseinheit einschränken. Dadurch ließe sich im obigen Beispiel z.B. bestimmen, daß Herr Meyer die Berechtigung zur Genehmigung von bestimmten Anträgen nur in einer bestimmten Abteilung hat. Weitere Relationen für Einschränkungen beliebiger anderer Art oder für die genauere Definition der Befugnis können auf dieselbe Art und Weise hinzugefügt werden.

Auch für Rechte läßt sich keine allgemeingültige Liste von Tätigkeiten angeben. Typische Tätigkeiten, die als Befugnisse ausgelegt werden können, sind z.B. Genehmigen, Einsehen, Ändern, Löschen, Ausführen, usw. Dabei kann eine Tätigkeit je nach verbundener Relation durchaus einmal eine Befugnis und das andere Mal eine Fähigkeit darstellen.

5.6.4 Rollen

Rollen können die Zuordnung von Eigenschaften zu einem Akteur wesentlich erleichtern. Einer Rolle können z.B. mehrere Rechte zugeordnet werden, die auf den Rolleninhaber übergehen. Rollen stellen dabei nicht nur eine Erleichterung für den Modellierer der Aufbauorganisation dar, sondern helfen auch, konsistente Modelle zu erzeugen, die mit der Realität übereinstimmen. Müßte man jedem Akteur manuell seine Rechte zuordnen, so könnte man z.B. im Fall eines Stellenwechsels leicht vergessen, einzelne Rechte des betroffenen Mitarbeiters anzupassen. Außerdem ist nicht ohne weiteres klar, welche Rechte einem Akteur aufgrund welcher Bedingungen zugeordnet wurden. Weist man dagegen einer Rolle die Rechte zu, die mit der Stelle verbunden sind und ordnet man diese Rolle dem Stelleninhaber zu, so muß im Falle eines Stellenwechsels nur die Zuordnung zu einer Rolle angepaßt werden. Die mit der Rolle verbundenen Rechte gehen automatisch auf den neuen Rolleninhaber über.

Modellierung mit Rollen Sieht man eine Rolle als eine abstrakte Person an, so ergibt sich eine sehr elegante Form der Modellierung. Ohne bereits an einen konkreten Akteur zu denken, verbindet man mit der Rolle alle Rechte, die man dem späteren Rolleninhaber zuschreiben will.

Hierfür wird die Relation *enthaltene Befugnis* verwendet. Durch geeignete Definition einer Regel gehen diese Rechte später automatisch auf den Rolleninhaber über (dies ist ungefähr der Rollen- und Stellenregel des Workflow-Management-Systems WorkParty vergleichbar, das in Abs. 3.1.1 beschrieben wurde). Hierzu ein Beispiel:

Zunächst wird eine Modellierung mit abstrakten Personen (Rollen) vorgenommen. Das Projekt ABC wird von der abstrakten Person *Projektleiter ABC* geleitet. Die Rolle beinhaltet das Recht, bestimmte Anträge innerhalb dieses Projekts zu genehmigen (Abb. 5.35a). In Abb. 5.35b wird der Akteur Meyer spezifiziert, der die Rolle ABC einnimmt. Durch die Regeln in Abb. 5.35c gehen die Befugnisse auf den Rolleninhaber Meyer über. Die Regel muß dabei nur einmal definiert werden und gilt gleichermaßen für alle Rollen. Zu beachten ist hierbei, daß die Rollenregel einfach eine Relation *Befugnis* zwischen die Tätigkeit *Genehmigung* und den Akteur *Meyer* einfügt. Alle Relationen, die an der Tätigkeit *Genehmigen* hängen, bleiben erhalten.

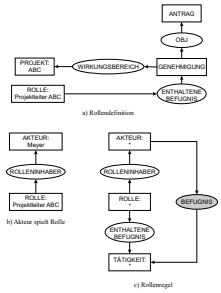

Abbildung 5.35: Beispiel für die Anwendung von Rollen

Aggregation von Rollen Um die Modellierung weiter zu erleichtern, wird eine Möglichkeit geschaffen, Rollen ineinander zu schachteln, d.h. einer Rolle können über die Relation *enthaltene Rolle* mehrere andere Rollen zugeordnet werden. Dabei muß dafür gesorgt werden, daß die Befugnisse, die die enthaltenen Rollen mitbringen, ebenfalls auf den Rolleninhaber übergehen. Dazu dient die Rollenaggregationsregel aus Abb. 5.36. Sie sorgt dafür, daß die Befugnisse der untergeordneten Rollen zunächst auf die jeweils übergeordnete Rolle und schließlich durch die bereits vorgestellte Rollenregel auf den Rolleninhaber selbst übergehen.

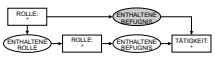

Abbildung 5.36: Rollenaggregationsregel

Die Konzepttyphierarchie wird leicht erweitert, um die hier vorgestellte Modellierung zu ermöglichen. Es werden der Konzepttyp *Rolle* und die Relationen *enthaltene Befugnis* und *enthaltene Rolle* in die Konzept- bzw. Relationstyphierarchie aufgenommen.

5.6.5 Metamodell

Aus den bisher definierten Begriffen läßt sich ein Metamodell ableiten (Abb. 5.37), das beschreibt, welche Elemente zur Modellierung zur Verfügung stehen.

Die beschriebenen Befugnisse eine Akteurs aus Abs. 5.6.3 drücken sich durch die Erlaubnis aus, Tätigkeiten auszuführen (Relation *Befugnis*). Die Fähigkeiten aus Abs. 5.6.2 werden durch die

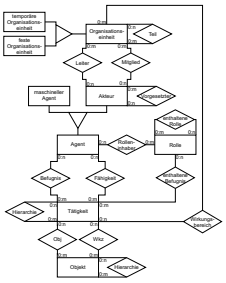

Abbildung 5.37: Metamodell der Organisationsstrukturverwaltung

Zuordnung einer Tätigkeit zu einem Akteur über die Relation *Fähigkeit* definiert. Zu beachten ist, daß Fähigkeiten und Rechte dadurch *explizit* modelliert werden können, so daß später tatsächlich nach einem Akteur gesucht werden kann, der bestimmte Fähigkeiten bzw. Rechte besitzt. Jede Tätigkeit kann nun durch weitere Relationen näher bestimmt werden. So ordnet die Relation *Wkz* einer Tätigkeit das Werkzeug zu, mit dem sie ausgeführt wird, und die Relation *Obj* ordnet der Tätigkeit das Objekt zu, auf dem sie ausgeführt wird (der Anwendungsbereich der Tätigkeit). Außerdem kann einer Tätigkeit ein *Wirkungsbereich* in Form einer Organisationseinheit zugeordnet werden, um z.B. eine Befugnis – die sich ja durch das Recht bestimmt, eine bestimmte Tätigkeit auszuführen – auf eine bestimmte Organisationseinheit einzuschränken. Sowohl Objekte als auch Tätigkeiten können hierarchisch modelliert werden, was ebenfalls bei der Suche nach Akteuren ausgenutzt werden kann (siehe Abs. 5.3.7.3).

Ein Akteur kann mehreren Organisationseinheiten zugeordnet sein (Relation *Mitglied*). Außerdem kann er der *Leiter* von mehreren Organisationseinheiten sein. Organisationseinheiten können durch die Relation *Teil* andere Organisationseinheiten beinhalten. Durch die Relation *Vorgesetzter* können *beliebige* Vorgesetztenverhältnisse modelliert werden.

Die Rollen dienen nur als Hilfsmittel zu Modellierung (siehe Abs. 5.6.4). Sie werden nicht zur Suche nach Akteuren verwendet! Dies geschieht über die explizit modellierten Fähigkeiten und Rechte eines Akteurs. Rollen können eine Menge von Rechten beinhalten (Relation *enthaltene Befugnis*), die auf den Rolleninhaber übergehen. Es besteht die Möglichkeit, Rollen ineinander zu schachteln, um eine weitere Erleichterung bei der Modellierung zu erreichen.

Das vorgestellte Metamodell stellt eine Ableitung des allgemeinen Basis-Metamodells aus Abb. 5.2 dar, das in Abs. 5.3.1 beschrieben wurde. Hierbei ist zu beachten, daß dieses Metamodell um beliebige Begriffe (Relationen, Entitäten) erweitert werden kann, woraus sich die Flexibilität dieses Ansatzes ergibt. Das vorgestellte Metamodell hat also zwei wesentliche Eigenschaften:

- Erweiterbarkeit: Beliebige neue Begriffe und Relationen können definiert werden.
- *Explizite* Modellierung von Fähigkeiten und Rechten und somit die Möglichkeit zur Suche nach Akteuren, die bestimmte Fähigkeiten/Rechte besitzen.

Eine Zusammenfassung inkl. Typdefinitionen der Begriffe der entworfenen Unternehmensontologie findet sich in Abs. B.

5.6.6 Konsistenzbedingungen

Betrachtet man das Metamodell aus Abs. 5.6.5, so fällt auf, daß dieses praktisch keine Einschränkungen bei den Kardinalitäten der Relationen macht. Eine Person kann beispielsweise beliebig

vielen Organisationseinheiten angehören. Diese Freiheit ist zunächst durchaus beabsichtigt, denn nur so können beliebige Situationen abgedeckt werden (vielleicht will man einen Angestellten tatsächlich mehreren Organisationseinheiten zuordnen). Es gibt allerdings auch Fälle, wo diese Freiheit nicht erwünscht ist. Zum Beispiel will man sicher verhindern, daß eine Person ihr eigener Vorgesetzter ist, oder man könnte fordern, daß ein Akteur genau einer festen Organisationseinheit (z.B. einer Abteilung) zugeordnet sein muß. Allgemein kann man Bedingungen formulieren, die innerhalb der Organisation aufgrund von Unternehmensregeln erfüllt sein müssen. Zwei Arten von Regeln erscheinen dabei sehr interessant, nämlich Gebote und Verbote, also Bedingungen, die erfüllt sein müssen, damit eine Unternehmensstruktur als konsistent betrachtet werden kann, und Bedingungen, die nicht erfüllt sein dürfen.

Konzeptgraphen bieten zunächst keine direkte Unterstützung, um solche Konsistenzbedingungen einzuhalten. Im Prinzip kann man z.B. beliebige Relationen setzen, vorausgesetzt, die Argumente der Relation entsprechen den in der Typdefinition der Relation festgelegten Konzepttypen. Allerdings kann man eine Konsistenzbedingung wie jede andere Aussage als Konzeptgraph formulieren. Die Bedingung „ein Akteur darf nicht sein eigener Vorgesetzter sein“ wurde bereits in Abs. 4.6 als ein Beispiel für erweiterte Konzeptgraphen mit Negation vorgestellt (siehe Abb. 4.10). Allerdings ist die Negation nicht unbedingt notwendig, um diese Konsistenzbedingung auszudrücken. Abb. 5.38a zeigt einen Konzeptgraphen, der das Gegenteil der Bedingung ausdrückt, nämlich „es existiert ein Akteur, der sein eigener Vorgesetzter ist“. Man kann nun versuchen, diesen Konzeptgraphen z.B. mittels Graph-Matching auf das Modell der Organisationsstruktur abbzubilden. Gelingt dies, so existiert ein Akteur, der sein eigener Vorgesetzter ist. Da der Graph als *Verbot* gekennzeichnet ist, weiß man, daß es eine Inkonsistenz im Modell gibt und erhält durch die Abbildung auch gleich den betreffenden Akteur, der sein eigener Vorgesetzter ist.

Abbildung 5.38: Konsistenzbedingungen

Man könnte nun denken, ein einfaches Gebot ließe sich auf dieselbe Art durch einfache Konzeptgraphen ausdrücken. Hierzu wird der Graph aus Abb. 5.38b betrachtet, der im folgenden G genannt wird. Das Gebot soll lauten: „Jeder Akteur ist Teil mindestens einer festen Organisationseinheit“. Was Abb. 5.38b ausdrückt ist aber: „Es existiert ein Akteur, der Teil einer festen Organsationseinheit ist“. Formal ausgedrückt müßte in $\phi(G)$ die Variable des Akteurs nicht existentiell, sondern universell gebunden werden:

$$\forall x \exists y (AKTEUR(x) \wedge FESTEOE(y) \wedge MITGLIED(y, x))$$

Dieses sehr einfache Gebot läßt sich bereits nicht mehr durch einfache Konzeptgraphen ausdrücken. Auch die oben verwendete Negation bringt einen hier nicht weiter, denn man erhält nie eine Formel, die bis auf eine evtl. vorangestellte Negation nur Existenzquantoren und Konjunktionen enthält. In diesem Fall wären nun erweiterte Konzeptgraphen gefordert.

Dieses Beispiel zeigt, daß Flexibilität auch ihre Nachteile haben kann, nämlich dann, wenn man die (zu) große Freiheit, die man durch die Modellierung mit Konzeptgraphen zweifelslos erreicht hat, nicht wieder in gewünschtem Maße einschränken kann.

Kapitel 6

Implementierung und Evaluation

Dieses Kapitel beschreibt Details der Implementierung. Außerdem erfolgt eine Evaluation der implementierten Matching-Algorithmen anhand von Laufzeitmessungen, sowie eine Bewertung, inwieweit durch den vorgestellten Entwurf die Ziele bzw. Anforderungen erreicht werden konnten.

6.1 Implementierung

Zur Implementierung des Entwurfs aus Kapitel 5 wird hauptsächlich die Programmiersprache Java (SDK V1.3) verwendet, in der auch die verwendete Notio-Bibliothek implementiert ist, die als Grundlage für die Implementierung dient.

6.1.1 Basisklassen

Die Implementierung der Klassen aus Abb. 5.25a beschränkt sich zum Großteil auf die Ableitung entsprechender Klassen der Notio-Bibliothek und das Hinzufügen einiger Attribute, wie z.B. Attribute für das Gewicht oder die maximale semantische Distanz eines Konzept- und Relationsknotens (diese Attribute werden für die Bewertung der Güte einer partiellen Projektion benötigt, siehe Abs. 5.3.7.3).

Auch die Implementierung der Klassen aus Abb. 5.25b, die verschiedene Konzeptgraphen (Fakten, Regeln, Anfragegraphen) darstellen, stellt kein größeres Problem dar. Sehr viel umfangreicher ist dagegen die Implementierung der verschiedenen Matcher-Klassen aus Abb. 5.25c, die die verschiedenen Matching-Algorithmen implementieren. Da diese Algorithmen bereits im Detail im vorigen Kapitel vorgestellt wurden, wird auf eine weitere Beschreibung verzichtet.

6.1.2 Wissensbasis

Verwaltung der Fakten, Regeln und Typhierarchien Die Wissensbasis verwaltet die Fakten, Regeln und Typhierarchien, wobei jedes einzelne Objekt in einer Datei im XML-Format[1] abgespeichert wird. Zwar bringt Notio bereits die Möglichkeit mit, Konzeptgraphen in diversen Formaten zu speichern und zu lesen, doch bietet keines dieser Formate die Möglichkeit, die zusätzlich notwendigen Attribute der abgeleiteten Klassen aufzunehmen. Daher wird ein selbstdefiniertes Format verwendet.

[1] E**X**tensible **M**arkup **L**anguage

Ein Problem ergibt sich bei der Verwaltung der Typhierarchien. Es soll hier aus Konsistenzgründen nicht möglich sein, Konzept- oder Relationstypen aus der Konzept- bzw. Relationstyphierarchie zu löschen, wenn noch irgendein Fakt oder irgendeine Regel die zu löschenden Typen verwendet. Allerdings werden die Typhierarchien von anderen Objekten verwaltet, als die Fakten und Regeln. Dieses Problem läßt sich durch eine leichte Abwandlung der Observer-Entwurfsmusters lösen.

Einschub: Observer-Entwurfsmuster Allgemein versteht man unter einem Entwurfsmuster (engl. Design Pattern) eine bewährte generische Lösung für ein wiederkehrendes Entwurfsproblem.

Das Observer-Entwurfsmuster definiert eine 1:n-Beziehung zwischen einem Objekt (Subjekt) und seinen Beobachtern (engl. Observers). Immer wenn sich das Subjekt verändert (z.B. seine Daten/Attribute ändert), werden alle Beobachter-Objekte über diese Änderung informiert. Diese können dann auf die Änderung reagieren und z.B. die Anzeige der Daten anpassen, indem sie die veränderten Daten des Subjekts abfragen und neu darstellen.

□

Das Observer-Entwurfsmuster wird zur Lösung des obigen Problems um ein Vetorecht für die Beobachter erweitert, d.h. ein Beobachter, der über eine potentielle Änderung informiert wird, kann Einspruch gegen diese Änderung einlegen und die Änderung somit verhindern. Die Beobachter-Objekte sind in diesem Fall die Verwalter der Fakten und Regeln, die sich als Beobachter bei der Konzept- und Relationstyphierarchie der Wissensbasis anmelden. Immer wenn ein Hierarchie-Objekt eine Aufforderung bekommt, einen Typ zu löschen, werden zunächst alle Beobachter informiert. Nur wenn diese kein Veto gegen die geplante Änderung einlegen – was die Fakten- und Regelverwaltung tun würde, wenn der zu löschende Typ noch gebraucht wird – kann der Typ tatsächlich gelöscht werden.

Bereitstellung der Funktionen über CORBA Wie bereits erwähnt, können die Funktionen der Wissensbasis über CORBA angesprochen werden. Dies wird z.B. von der Benutzungsschnittstelle (Konzeptgraphen-Editor) dazu verwendet, um Daten an die Wissensbasis zu übermitteln und von dieser abzufragen. Außerdem kann über CORBA auch die beschriebene Kompilation der Wissensbasis angestoßen werden.

Um die Funktionen der Wissensbasis überhaupt ansprechen zu können, müssen die entsprechenden Klienten zunächst die Adresse des CORBA-Server-Objekts ermitteln. Dieses Problem stellt sich auch für die weiter unten beschriebenen Komponenten. Es wird daher an dieser Stelle für alle über CORBA ansprechbaren Komponenten der Organisationverwaltung beschrieben, wie deren Adressen ermittelt werden können.

CORBA stellt einen sogenannten Naming-Service zur Verfügung, über den Klienten die für den Zugriff auf die Server-Objekte notwendigen Referenzen ermitteln können. Hierzu trägt sich jedes Server-Objekt unter einem bestimmten Namen[2] in den Naming-Service ein, der natürlich auch den Klienten wohlbekannt sein muß. Außerdem wird dem Naming-Service die Referenz des Server-Objekts mitgeteilt, so daß diese einem Klienten mitgeteilt werden kann, wenn eine Anfrage nach einem Objekt mit einem bestimmten Namen erfolgt. Theoretisch läßt es der Naming-Service auch zu, sogenannte Namenskontexte anzulegen, so daß sich Objekte unter bestimmten Kontexten eintragen können. Auf diese Möglichkeit wird jedoch verzichtet. Statt dessen trägt sich jedes Server-Objekt direkt im Wurzel-Kontext unter einem wohlbekannten Namen ein. Dieser Name wird jedem Klienten (z.B. der Benutzungsoberfläche als Klient der

[2]Genaugenommen besteht ein Name aus zwei Teilen, nämlich „Identifier“ und „Kind“.

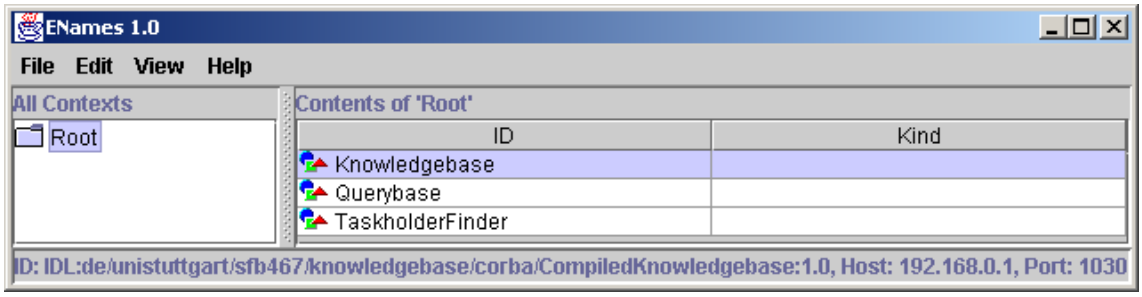

Abbildung 6.1: Naming-Service mit eingetragenen Server-Objekten

Wissenbasis oder dem Taskholder-Finder-Client als Klient des Taskholder-Finders) über seine Konfigurationsdatei mitgeteilt. Diese Konfigurationsdatei enthält auch die IOR[3] des Naming-Service-Objekts, die für den Zugriff auf den Naming-Service benötigt wird, da die Referenz des Naming-Service selbst ja nicht über den Naming-Service ermittelt werden kann. Abb. 6.1 zeigt den Inhalt des Naming-Service, wie er während der Laufzeit der Organisationsverwaltung mit Hilfe eines Naming-Service-Browsers ermittelt wurde. Es sind drei Objekte zu erkennen: *Knowledgebase* (Wissenbasis), *Querybase* (Anfragegraphenverwaltung) und *TaskholderFinder*. Diese Objekte können über CORBA angesprochen werden und jeder Klient kann sich mit Hilfe des Namens des jeweiligen Objekts unter Verwendung des Naming-Service eine Referenz auf das entsprechende Server-Objekt beschaffen.

6.1.3 Anfragegraphen-Verwaltung

Die Anfragegraphen-Verwaltung hat dasselbe Problem, wie die Verwaltungen der Fakten und Regeln. Auch hier darf ein Typ nur dann aus der Typhierarchie gelöscht werden, wenn kein Anfragegraph den zu löschenden Typ mehr referenziert. Daher trägt sich auch die Anfragegraphen-Verwaltung als Beobachter bei der Konzept- und Relationstyphierarchie ein.

Ansonsten speichert auch die Anfragegraphen-Verwaltung die Anfragegraphen in einfachen Dateien in einem speziellen XML-Dateiformat.

6.1.4 Taskholder-Finder

Der Taskholder-Finder-Komponente wird bei ihrer Initialisierung eine Referenz auf eine Wissensbasis und eine Anfragegraphen-Verwaltung übergeben. Da die verschiedenen Anfragegraphen-Verwaltungen dieselbe Schnittstelle implementieren, können diese ohne Probleme gegeneinander ausgetauscht werden. Je nach Klasse der Anfragegraphen-Verwaltung wird zur Selektion von Akteuren Graph-Matching (Ullmann- und Koch-Algorithmus), Graph-Matching mit baumförmigen Anfragegraphen (Mugnier-Chein-Algorithmus) oder PROLOG verwendet. Das Sequenzdiagramm aus Abb. 6.2 zeigt, wie der Taskholder-Finder bei der Selektion eines Akteurs vorgeht und wie er vom Taskholder-Finder-Client aufgerufen wird.

1. Die Selektion wird durch eine Anfrage von Changengine an den Taskholder-Finder-Client gestartet (siehe auch Abs. 6.1.5). Diese Anfrage enthält den Bezeichner eines Anfragegraphen und eine Menge von Parametern. Mit Hilfe dieser Daten kann der Taskholder-Finder-Client mittels CORBA eine entsprechende Anfrage an den Taskholder-Finder stel-

[3] Objektreferenz in Textform

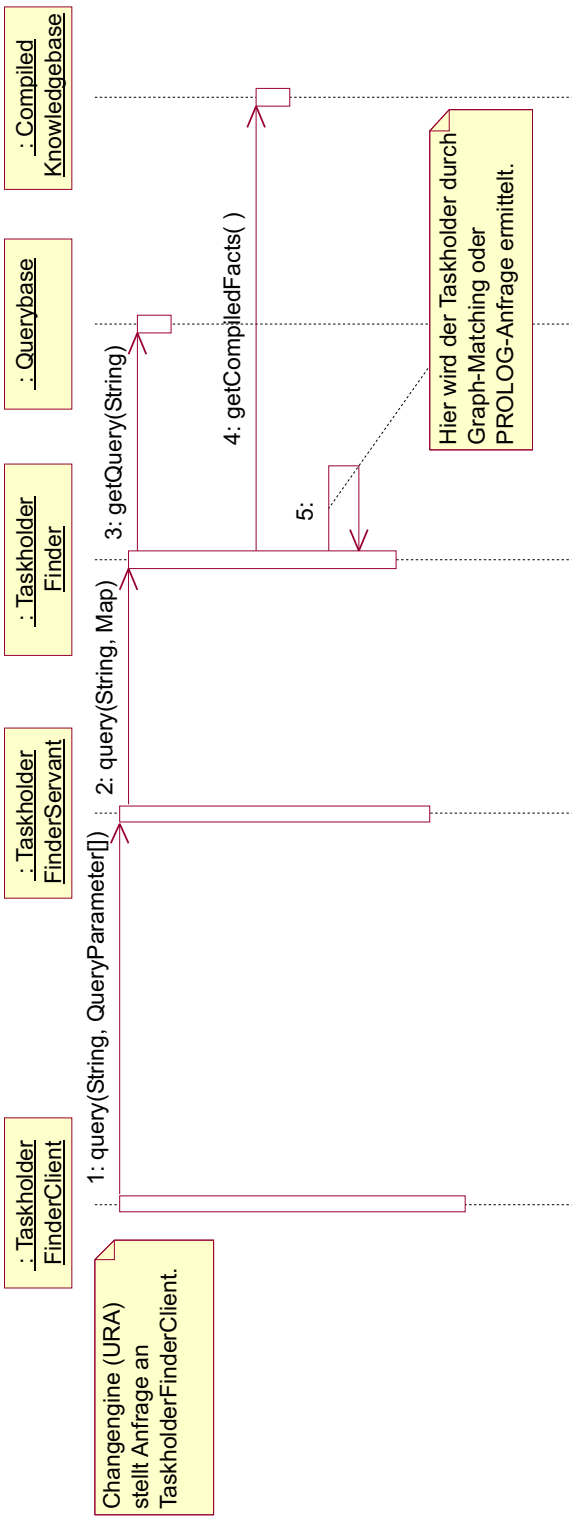

Abbildung 6.2: Sequenzdiagramm: Ablauf der Aufgabenträgerselektion

len. Das `TaskholderFinderServant`-Objekt ist nichts anderes, als der CORBA-Wrapper des Taskholder-Finder-Objekts.

2. Die Anfrage wird von diesem Wrapper an den eigentlichen Taskholder-Finder durchgereicht.

3. Der Taskholder-Finder kann nun aus der Anfragegraphen-Verwaltung den Anfragegraphen ermitteln und diesen mit Hilfe der übergebenen Parameter entsprechend anpassen.

4. Der nächste Schritt ermittelt die Fakten aus der Wissensbasis. Wird die Selektion mit Graph-Matching durchgeführt, so wird eine Menge von Fakten zurückgegeben. Wird die Selektion mit PROLOG durchgeführt, so wird nur der Dateiname des beim Kompiliervorgang erstellten PROLOG-Programms zurückgeliefert.

5. Nun erfolgt die eigentliche Selektion. Entweder werden die passenden Akteure nun durch Graph-Matching des Anfragegraphen auf allen Fakten ermittelt, oder es wird ein externer PROLOG-Compiler aufgerufen. Wird PROLOG verwendet, so wird der Anfragegraph zuvor in eine PROLOG-Anfrage transformiert.

Für PROLOG-Anfragen wird der SWI-PROLOG-Compiler verwendet, der frei unter der General Public License (GPL) verfügbar ist (siehe [**?**]). Theoretisch läßt sich der SWI-Compiler über das Java-PROLOG-Interface (JPL) direkt aus Java heraus ansprechen. Allerdings ergaben sich hierbei Probleme mit der verwendeten Entwicklungsumgebung. SWI-PROLOG wird daher als eigener Prozeß gestartet, wobei die Anfrage als Kommandozeilen-Parameter übergeben wird. Die Ausgabe liest der Taskholder-Finder über eine Pipe mit Hilfe eines kleinen in JavaCC erstellten Parsers. Ein kleiner Nachteil dieses Vorgehens ergibt sich dadurch, daß bei jedem Starten des SWI-Compiler-Prozesses die gesamte PROLOG-Wissensbasis neu kompiliert werden muß, was aber selbst bei großen Wissensbasen sehr schnell geschieht. Die Plattformunabhängigkeit, die eigentlich durch Java gegeben ist, ist durch die Verwendung des SWI-PROLOG-Compilers hinfällig. Sie wäre auch nicht mehr gegeben, wenn JPL verwendet werden würde, da SWI-PROLOG hier über das Java Native Interface (JNI) angesprochen wird. Da sich SWI-PROLOG aber auf nahezu jeder gebräuchlichen Plattform kompilieren läßt und außerdem der Taskholder-Finder dank CORBA auf einem beliebigen Rechner im Netz ausgeführt werden kann, ergibt sich dadurch kein wesentlicher Nachteil.

Die Ergebnisse einer Anfrage werden mitprotokolliert. Das heißt, es wird für jeden Match in einer Protokolldatei festgehalten, wie der Anfragegraph auf ein Fakt abgebildet wurde (welche Knoten des Anfragegraphen auf welche Knoten eines Fakts abgebildet wurden und welche Knoten nicht abgebildet werden konnten). Da alle Matching-Operationen auf der kompilierten Wissensbasis stattfinden, für den Modellierer aber wichtig ist, welche Knoten der ursprünglichen Fakten und Regeln an einem Match beteiligt sind, muß ebenfalls während der Kompilierung der Wissensbasis in der Protokolldatei vermerkt werden, welche Knoten der kompilierten Fakten aus welchen Knoten der ursprünglichen Fakten und Regeln entstanden sind. Hier ein kleiner Ausschnitt aus der Protokolldatei eines Matches:

```
PROJECTION(("Query1"->"N0"), ("Fakt1"->"N0"));
PROJECTION(("Query1"->"N6"), JOINED(("Fakt2"->"N6"),("Fakt3"->"N9")))
PROJECTION(("Query1"->"N1"), CONCLUSION(("Regel1"->"N5")));
```

Jede Angabe in `PROJECTION` steht für eine Knotenabbildung eines Knotens des Anfragegraphen. Zeile 1 zeigt die einfachste mögliche Abbildung. Der Knoten mit der Nummer *N0* des Anfragegraphen mit dem Namen *Query1* wird auf den Knoten *N0* des Fakts *Fakt1* abgebildet.

Etwas komplizierter ist Z. 2. Hier wird der Knoten *N6* des Anfragegraphen auf einen Knoten eines kompilierten Fakts abgebildet, der durch eine Join-Operation auf individuellen Konzepten entstanden ist. Die ursprünglichen Knoten *vor* der Join-Operation sind der Knoten *N6* des ursprünglichen Fakts *Fakt2* und der Knoten *N9* des Fakts *Fakt3*. Zeile 3 zeigt schließlich eine Abbildung auf einen Knoten, der aus einer Regelanwendung entstanden ist. Der Knoten *N1* des Anfragegraphen wird auf den Knoten *N5* der Regel mit dem Namen *Regel1* abgebildet.

Die protokollierten Informationen können später ausgewertet werden. Interessant ist diese Auswertung vor allem für einen Lernvorgang, bei dem versucht wird, die Informationen der Wissensbasis so zu ergänzen, daß Anfragen besser beantwortet werden können. Hierzu sind Informationen sehr hilfreich, die auf Informationslücken in der Wissensbasis schließen lassen. Der Lernprozeß wird im Detail in [**?**] beschrieben. Der interessierte Leser wird daher auf diese Quelle verwiesen.

6.1.5 Taskholder-Finder-Client

Der Taskholder-Finder-Client ist eine relativ kleine Komponente, die nur der Vermittlung von Anfragen des Resource Executive an die Organisationsstrukturverwaltung (genauer den Taskholder-Finder) dient. Es läßt sich an dieser Stelle sehr schön an einem Beispiel zeigen, wie die drei in Abs. 5.4.2 genannten Schritte durchzuführen sind, um die Organisationsstrukturverwaltung über den URA an Changengine anzubinden. Es soll hierzu eine Zuweisungsregel erstellt werden, die als Ergebnis den Leiter eines bestimmten Produktionsbereiches zurückliefert. Es wird davon ausgegangen, daß die Organisationsstruktur (organisatorische Wissensbasis) bereits modelliert wurde.

Schritt 1 – Erstellung der Anfragegraphen Der Anfragegraph *verantwortlicher_produktionsbereich* aus Abb. 6.3 dient der Selektion eines Akteurs, der für einen bestimmten Produktionsbereich verantwortlich ist. Der Produktionsbereich wurde dabei als Parameter ausgelegt. Dieser Parameter wird vor der eigentlichen Selektion mit dem konkreten Produktionsbereich besetzt. Somit kann der erstellte Anfragegraph leicht wiederverwendet werden. Außerdem ist in Abb. 6.3 der Konzeptgraphen-Editor aus [**?**] zu sehen, der neben der Eingabe von Anfragegraphen ebenfalls zur Eingabe der Fakten, Regeln und der Typhierarchien der Wissensbasis dient.

Schritt 2 – Erstellung eines Eintrags im URA-Description-File Der erstellte Eintrag sieht wie folgt aus:

```
[Format Version] 1.0
[Script] taskholderfinderclient
    [Command Line] thf_client.sh
    [Environment]
    [End Environment]
[End Script]
[Function] verantwortlicher_produktionsbereich
    [Use Script] taskholderfinderclient
    [Output Type] StringSequence
    [Parameter] produktionsbereich
        [Type] String
        [Substitute] SINGLE
        [On Exhaustion] BLANK
    [End Parameter]
    [Input Format]
```

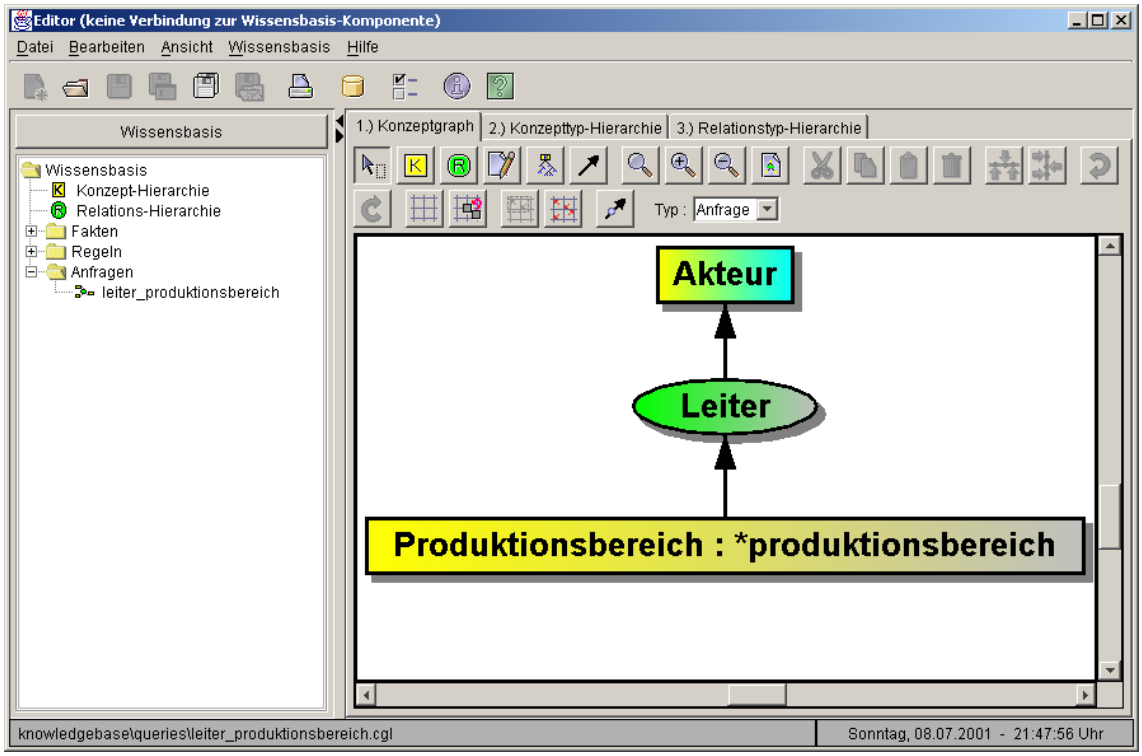

Abbildung 6.3: Konzeptgraphen-Editor mit einem Anfragegraph zum Changengine-Beispiel

```
16          [Header] QUERY(”verantwortlicher_produktionsbereich”) {
            [Body] PARAM(produktionsbereich=”%produktionsbereich%”);
18          [Footer] }
        [End Input Fomat]
20      [Output Format] Plain
            [Footer] END
22      [End Output Format]
    [End Function]
```

Listing 6.1: URA-Description-File mit Aufruf der Organisationsstrukturverwaltung

Interessant ist hierbei vor allem die spezifizierte Funktion `verantwortlicher_produktionsbereich` (Z. 7 ff). Diese Funktion entspricht dem erstellten Anfragegraphen aus Schritt 1. Wird der URA aufgefordert, eine Selektion mit Hilfe dieser Funktion durchzuführen, so ruft er das Skript `taskholderfinderclient` auf (Z. 8). Dieses Skript ist ab Z. 2 definiert. Es ruft den Taskholder-Finder-Client durch das in Z. 3 definierte Kommando auf. `thf_client.sh` ist ein Shell-Skript, das den eigentlichen Aufruf der Java-Komponente enthält.

Dem Taskholder-Finder-Client werden über dessen Standardeingabe i.a. mehrere Parameter übergeben. In Z. 15–19 wird festgelegt, in welchem Format dem Taskholder-Finder-Client diese Parameter zu übergeben sind. Zunächst wird dem Client durch `QUERY("verantwortlicher_produktionsbereich")` mitgeteilt, welche Anfrage auszuführen ist, d.h. welcher Anfragegraph für die Selektion verwendet werden soll. In diesem Fall ist das der in Schritt 1 erstelle Anfragegraph *verantwortlicher_produktionsbereich*. Dann folgen die Parameter der Anfrage. Hier wird der konkrete Produktionsbereich übergeben (Z. 17). Der Wert

dieses Parameters wird dem URA beim Aufruf mitgeteilt. Welche Parameter der Resource Executive dem URA übergibt, steht in Z. 10–14. Der Taskholder-Finder-Client erhält über seine Standard-Eingabe also folgende Anfrage:

```
QUERY(”verantwortlicher_produktionsbereich”) {
    PARAM(produktionsbereich=”produktionsbereich XY”);
}
```

Zur Verarbeitung dieser Anfrage durch den Taskholder-Finder-Client wurde mit JavaCC ein einfacher Parser geschrieben, mit dessen Hilfe der Taskholder-Finder-Client die Daten einer Anfrage extrahiert.

In Z. 9 und Z. 20–22 wird festgelegt, in welchem Format der Taskholder-Finder-Client die Ergebnisse der Anfrage zurückliefert. Die Rückgabe erfolgt über die Standard-Ausgabe des Taskholder-Finder-Client und enthält pro Zeile den Bezeichner eines Akteurs. Die Ausgabe wird durch `END` abgeschlossen. Eine mögliche Ausgabe hat also folgendes Format:

```
Meyer
Schmidt
END
```

Schritt 3 – Erstellung der Zuweisungsregel Nun da Changengine durch das URA-Description-File mitgeteilt wird, welche Anfragen über den URA zur Verfügung stehe, kann die Zuweisungsregel spezifiziert werden:

```
  HEADER
2         NAME ”verantwortlicher_produktionsbereich”
          DESC ”Findet eine Person X mit folgenden Eigenschaften:”
4         DESC ”− X ist verantwortlich für einen bestimmten Produktionsbereich.”
          AUTHOR ”Frank Duerr”
6         VERSION ”X.00.00.00”
          GENERATOR ”emacs”
8         ON_FAILURE DEBUG
          DEFAULT_ERROR DEBUG
10 END_HEADER
  DECLARES
12 END_DECLARES
  RULE
14      EMail.Adr(
            URA.verantwortlicher_produktionsbereich(
16              CP.Get(”produktionsbereich”)
            )
18      )
  END_RULE
```

Listing 6.2: Ressourcen-Regel mit Aufruf der Organisationsstrukturverwaltung über den URA

Der eigentlich Aufruf der Organisationsstrukturverwaltung über den URA erfolgt in Z. 15. Hier wird die in Schritt 2 definierte Funktion des URA aufgerufen, die die entsprechende Selektionsregel aktiviert. Der Produktionsbereich wird mit Hilfe des Case-Packet-Agents (CP) aus den Daten des Workflows bestimmt. Durch Aufruf des E-Mail-Agenten (Z. 14) werden die E-Mail-Adressen der gefundenen Akteure bestimmt (z.B. aus einer entsprechenden Datenbank des

Unternehmens). Diese Adressen werden zurückgegeben. Die Engine kann nun die Akteure über E-Mail über die durchzuführende Aktivität informieren. Eine andere Möglichkeit wäre die Umwandlung in entsprechende Work-List-Adressen gewesen, so daß die Akteure über ihre Work-List informiert worden wären.

In Abs. 5.3.6 wurde erwähnt, daß durch entsprechende Operationen auf Mengen auch negierte Selektionen möglich sind. Diese Mengenoperationen sind im sogenannten Listen-Agenten (engl. List Agent) von Changengine implementiert und können somit auf die Ergebnisse von Selektionen angewandt werden. Hätte man z.B. eine Selektionsregel `selektion1`, die eine Menge von Akteuren mit gewünschten Eigenschaften liefert und eine weitere Selektionsregel `selektion2`, die eine Menge von Akteuren mit unerwünschten Eigenschaften liefert und die jeweils über den URA eingebunden wurden, so könnte man mit Hilfe des Listen-Agenten `List` die endgültige Menge von Akteuren bestimmen:

List . Difference (URA.selektion1(), URA.selektion2())

6.1.6 Testumgebung

Die gewünschte Funktionalität der erstellten Organisationsverwaltung läßt sich nur sehr umständlich testen, wenn die Anfragen durch das Workflow-Management-System Changengine gestellt werden. Daher wurde eine kleine Testumgebung entworfen und implementiert, die als Taskholder-Finder-Client fungiert. Über eine graphische Benutzungsoberfläche (BO) lassen sich Anfragen an den Taskholder-Finder stellen. Die BO bietet dazu Felder zur Eingabe des Bezeichners des gewünschten Anfragegraphen und für die Eingabe der benötigten Parameter (Abb. 6.4 oben). Im unteren Teil der BO werden die Ergebnisse der Anfrage inkl. errechneter Güte dargestellt.

In Abb. 6.4 ist außerdem zu sehen, daß der Taskholder-Finder auch die nicht-spezifizierten Parameter zurückliefert. Es wurde in diesem Beispiel zunächst nach irgendeinem Leiter eines Produktionsbereiches gesucht. Gefunden wurde Herr Meyer, der Leiter des Produktionsbereiches *Kurbelwellenfertigung* ist. Zurückgegeben wird nicht nur Meyer, sondern auch der nicht näher spezifizierte Produktionsbereich. Bei der zweiten Anfrage wurde dagegen der Parameter *produktionsbereich* angegeben (siehe Parametereingabe Abb. 6.4 oben). Da Herr Meyer der Leiter dieses Bereichs ist, wird dieser zurückgegeben.

6.2 Evaluation

6.2.1 Bewertung der Matching-Algorithmen

Wie bereits erwähnt, besitzen die Algorithmen nach Koch und Ullmann theoretisch exponentielle Zeitkomplexität. Trotzdem wurden diese Algorithmen näher betrachtet, da vermutet wird, daß die speziellen Bedingungen, die bei der Modellierung einer Unternehmensstruktur mit Hilfe von Konzeptgraphen anzutreffen sind, in der Praxis oft weit bessere Berechnungszeiten zulassen und somit diese Algorithmen sehr wohl einsetzbar sind (siehe auch Abs. 5.3.4.5). Messungen der für die Matching-Operation benötigten Zeit für Graphen verschiedener Größe sollen zeigen, ob diese Vermutung begründet ist.

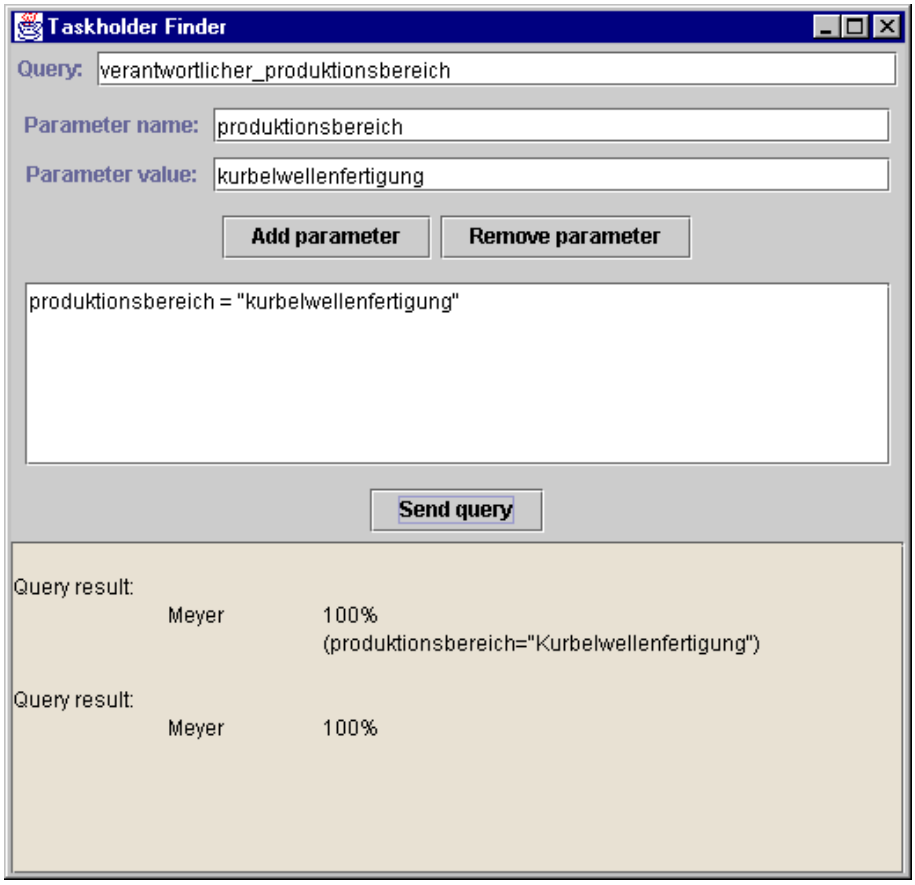

Abbildung 6.4: Testumgebung

6.2.1.1 Testfälle

Der kritische Punkt bei der Laufzeitmessung ist die Auswahl der Testfälle, also der Graphen (Anfragegraph, Fakt) auf denen die Matching-Operation ausgeführt wird. Diese Auswahl entscheidet wesentlich, ob die gemachten Messungen praxisrelevant sind. Es ließen sich ohne Probleme Testfälle konstruieren, die bei den NP-Algorithmen nach Ullmann und Koch praktisch unendliche oder sehr kurze Laufzeiten zur Folge hätten. Ziel ist es aber, ein möglichst realitätsnahes Beispiel zu verwenden, um an diesem die Messungen durchzuführen. Leider stand kein ausreichend großes Praxisbeispiel zur Verfügung, d.h. alle Beispiele hatten so wenige Knoten, daß die Laufzeit der Algorithmen viel zu kurz war, um daraus Schlüsse ziehen zu können. Daher wird versucht, zwar zufällige, aber trotzdem realitätsnahe Beispiele zu erzeugen, an denen die Messungen durchgeführt werden. Für die Wissensbasis gelten folgende Randbedingungen:

- Die Konzepttyphierarchie besteht aus 60 zueinander inkompatiblen Typen, die in einer flachen Hierarchie angeordnet sind, d.h. die Konzepttyphierarchie besitzt 60 Blätter.
- Die Relationstyphierarchie besteht aus 60 zueinander inkompatiblen Typen, die in einer flachen Hierarchie angeordnet sind, d.h. die Relationstyphierarchie besitzt 60 Blätter.
- Es gibt nur binäre Relationen, da diese in der Praxis am häufigsten verwendet werden.
- Jede Relation kann nur Konzepte bestimmten Typs miteinander verbinden.

- Ein Fakt enthält keine redundanten Relationen, d.h. wenn zwischen zwei Konzepten eine Relation vom Typ A vorhanden ist, dann existiert keine zweite Relation vom Typ A, die dieselben Konzepte miteinander verbindet.
- Die Wissensbasis besteht aus einem u.U. sehr großen Fakt. In der Realität wird die kompilierte Wissensbasis zumindest aus sehr wenigen dafür aber u.U. sehr großen Fakten bestehen, wenn das in Kapitel 5 beschriebene Vorgehen gewählt wird.

Zunächst werden die Typhierarchien der Wissensbasis erstellt. Danach wird eine bestimmte Anzahl von Konzeptknoten erstellt. Für jeden Knoten wird zufällig ein Konzepttyp ausgewählt und jeder der Knoten erhält einen anderen Referenten, d.h. ein Fakt besteht nur aus individuellen Konzepten mit paarweise verschiedenen Referenten. In der Praxis sind relativ häufig auch Fakten mit generischen Konzepten anzutreffen. Da allerdings beim Anfragegraph die Möglichkeit besteht, mehrere generische Konzepte einzufügen, die dann wieder auf viele individuelle Konzepte eines Fakts abgebildet werden können, stellt die Beschränkung der Fakten auf individuelle Konzepte keine wirkliche Einschränkung dar.

Im nächsten Schritt werden die Relationen des Fakts erstellt. Dazu wird zunächst ein Relationstyp aus der Typhierarchie bestimmt. Jedem Relationstyp sind zwei Konzepttypen zugeordnet. Mit Hilfe dieser Konzepttypen werden zufällig zwei Konzepte bestimmt, die konform zu diesen Typen sind. Es wird überprüft, ob diese beiden Konzept bereits von einer Relation des gleichen Typs verbunden sind. Ist dies nicht der Fall, wird die Relation eingefügt; ist dies der Fall, werden zufällig zwei neue Konzepte bestimmt, es wird also ein neuer Versuch gestartet. Die maximale Anzahl an Versuchen ist dabei beschränkt (ansonsten würde die Erzeugung der Testfälle bereits einen zu hohen Zeitaufwand beanspruchen). Die maximale Anzahl an einzufügenden Relationen ist ebenfalls beschränkt. Wie sich zeigt, erzeugt dieses Vorgehen Fakten, die in etwa gleich viele Konzept- und Relationsknoten besitzen. In der Praxis ist mit ähnlichen Fakten zu rechnen, wobei eher damit zu rechnen ist, daß ein Fakt etwas mehr Relationen als Konzepte besitzt. Wenn ein Fakt nicht vollständig verbunden ist, werden zusätzlich Relationen eingefügt, die Konzepte beliebigen Typs miteinander verbinden können (die Argumente dieser Relationen sind vom universellen Typ). Im allgemeinen sind nur sehr wenige dieser „universellen Verbindungen“ notwendig, um ein zusammenhängendes Fakt zu erzeugen.

Die so erzeugte Wissensbasis ist sicherlich an der einen oder anderen Stelle verbesserungswürdig. Zum Beispiel könnte man eine tiefere Typhierarchie zugrundelegen. Außerdem sind bisher Konzepte und Relationen von jedem Typ ungefähr gleich häufig vertreten, wohingegen in der Praxis sicher einzelne Typen häufiger anzutreffen sind. Trotzdem kann die so konstruierte Wissensbasis zumindest zeigen, ob ein Algorithmus evtl. völlig ungeeignet ist und welche Algorithmen bessere Laufzeitergebnisse erzielen, als andere.

Zu einem Testfall fehlt noch ein weiteres Element: Der Anfragegraph. Anfragegraphen oder auch Regeln sind typischerweise sehr klein. Es wird daher eine Größe von 10 Konzeptknoten und 9 Relationsknoten angenommen. Da der Mugnier-Algorithmus auf baumförmige Anfragegraphen beschränkt ist und auch in der Praxis meist solche Anfragegraphen verwendet werden, werden alle Anfragegraphen ebenfalls baumförmig angelegt. Ein Anfragegraph wird so erzeugt, daß ein baumförmiger Ausschnitt des erstellten Fakts kopiert wird. Um im Anfragegraph generische Konzepte zu erzeugen, wird aus einigen Knoten dieser Kopie der Referent entfernt (die Wurzel des Anfragegraphen ist dabei auf jeden Fall ein generisches Konzept). Der so erzeugte Graph ist der Anfragegraph. Somit ist auch sichergestellt, daß die Matching-Operation mindestens ein Ergebnis liefern muß, denn der gesuchte Ausschnitt ist nun auf jeden Fall im Fakt vorhanden. Zwei Arten von Anfragegraphen erscheinen besonders interessant: Anfragegraphen mit wenigen generischen Konzepten und Anfragegraphen mit vielen generischen Konzepten. Es ist damit

zu rechnen, daß letztgenannte Anfragegraphen zu einer wesentlich höhere Laufzeit führen. Da auch Regeln meist nur aus generischen Konzepten bestehen, ist dieser Fall besonders für den Ullmann-Algorithmus von Interesse, da nur dieser für die Regelanwendung, wie sie definiert wurde, in Frage kommt.

Die Messungen werden nicht nur mit den NP-Algorithmen nach Ullmann (S. 83 ff) und Koch (S. 102 ff), sondern zum Vergleich auch mit dem P-Algorithmus nach Mugnier, SWI-PROLOG und dem von Notio implementierten Matching-Algorithmus durchgeführt. Es wird darauf verzichtet, die Implementierung des Notio-Algorithmus näher zu analysieren. Da Notio Open-Source ist, wäre das zwar durchaus möglich, allerdings zeigen bereits die Messungen, wo Notio leistungsmäßig einzuordnen ist.

Die Meßwerte, die den hier dargestellten Diagrammen zugrundeliegen, finden sich in Abs. D. Da sich die Algorithmen teilweise erheblich in ihrem Laufzeitverhalten unterscheiden, werden teilweise nicht alle Meßergebnisse in einem Diagramm dargestellt. Es sind daher jeweils die Skalen der Y-Achse zu beachten!

Alle Tests fanden auf einem herkömmlichen PC mit einem Athlon Prozessor mit 800 MHz Taktfrequenz unter Windows NT statt. Die Matching-Algorithmen wurden in Java implementiert, wobei das Java Runtime Environment (JRE) des SDK V1.3 zum Einsatz kam. Jeder einzelne Testlauf wurde 10 mal wiederholt und die Ergebnisse gemittelt, um Schwankungen zu glätten.

6.2.1.2 Messungen 1

Die erste Messung benutzt baumförmige Anfragegraphen mit 3 generischen und 7 individuellen Konzepten sowie 9 Relationen. Es handelt es sich also um einen Anfragegraphen mit relativ wenigen generischen Konzepten.

Abb. 6.5 zeigt das Diagramm für die Algorithmen nach Mugnier, Ullmann und Koch sowie die von Prolog erzielten Zeiten und Abb. 6.6 zeigt die mit dem Notio-Matching-Algorithmus ermittelten Laufzeiten. Zu beachten ist, daß der Notio-Algorithmus zwei starke Ausreißer nach oben aufweist (siehe auch Tab. D.2), weswegen die Skala der Y-Achse in Abb. 6.6 beschränkt wurde, um auch die kleinen Werte noch ausreichend genau darstellen zu können.

Vergleicht man die Werte, so stellt man fest, daß der Algorithmus nach Mugnier und Chein bei weitem die besten Ergebnisse erzielt. Das ist wenig verwunderlich, denn dieser Algorithmus führt (fast) nur die Knotenvergleiche durch, die wirklich zu einem Ergebnis führen, wohingegen die Algorithmen nach Ullmann und Koch zunächst alle Konzepte des Anfragegraphen mit allen Konzepten des Fakts vergleichen müssen, um ihre Tabelle bzw. den Association-Graph zu erzeugen. PROLOG hat einen gewissen Nachteil, da immer zunächst der externe PROLOG-Compiler gestartet werden muß, wofür bereits ca. 80 ms verwendet werden müssen. Zieht man diesen „Offset“ ab, so fällt auf, daß PROLOG und der Ullmann-Algorithmus nahezu identische Werte haben. Beide haben eine deutlich geringere Steigung, als der Koch-Algorithmus. Der Notio-Algorithmus weist bereits bei dem ersten Test teilweise so hohe Laufzeiten auf, daß er sich für eine praktische Anwendung disqualifiziert.

6.2.1.3 Messungen 2

Der zweite Test benutzt baumförmige Anfragegraphen mit 10 generischen Konzepten und 9 Relationen. Im Gegensatz zum ersten Testlauf hat der zweite also wesentlich mehr generische Konzepte, wodurch ein höhere Zeitaufwand zu erwarten ist.

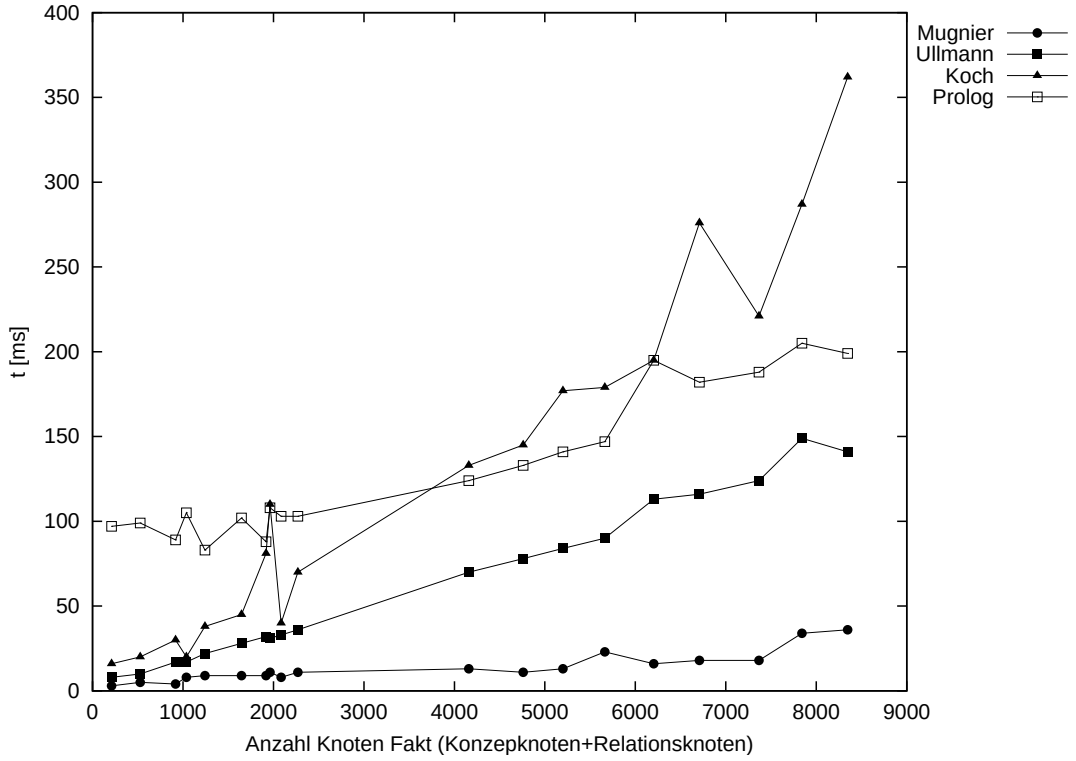

Abbildung 6.5: Meßergebnisse des ersten Laufzeittests Teil 1

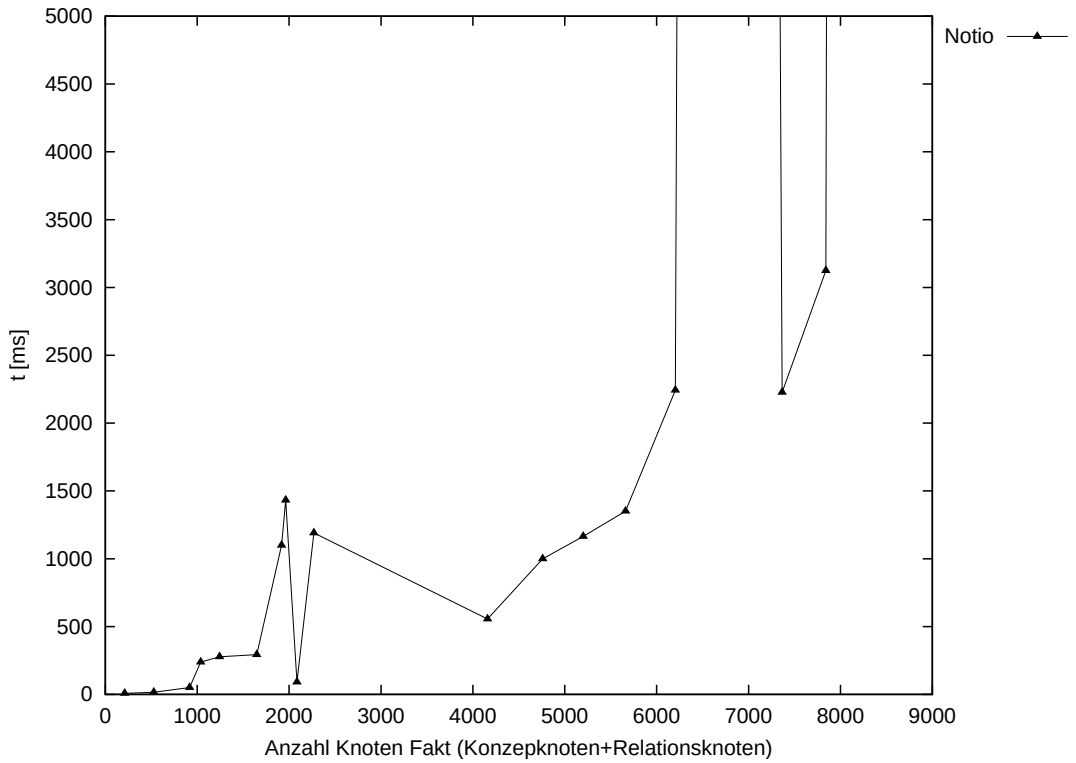

Abbildung 6.6: Meßergebnisse des ersten Laufzeittests Teil 2

Abb. 6.7, 6.8 und 6.9 zeigen die Diagramme des zweiten Testlaufes. Bereits bei sehr kleinen Knotenzahlen sind die Laufzeiten von Notio so hoch, daß die weiteren Tests abgebrochen werden mußten. Bemerkenswert ist, daß sich die Laufzeit des Mugnier-Chein-Algorithmus kaum

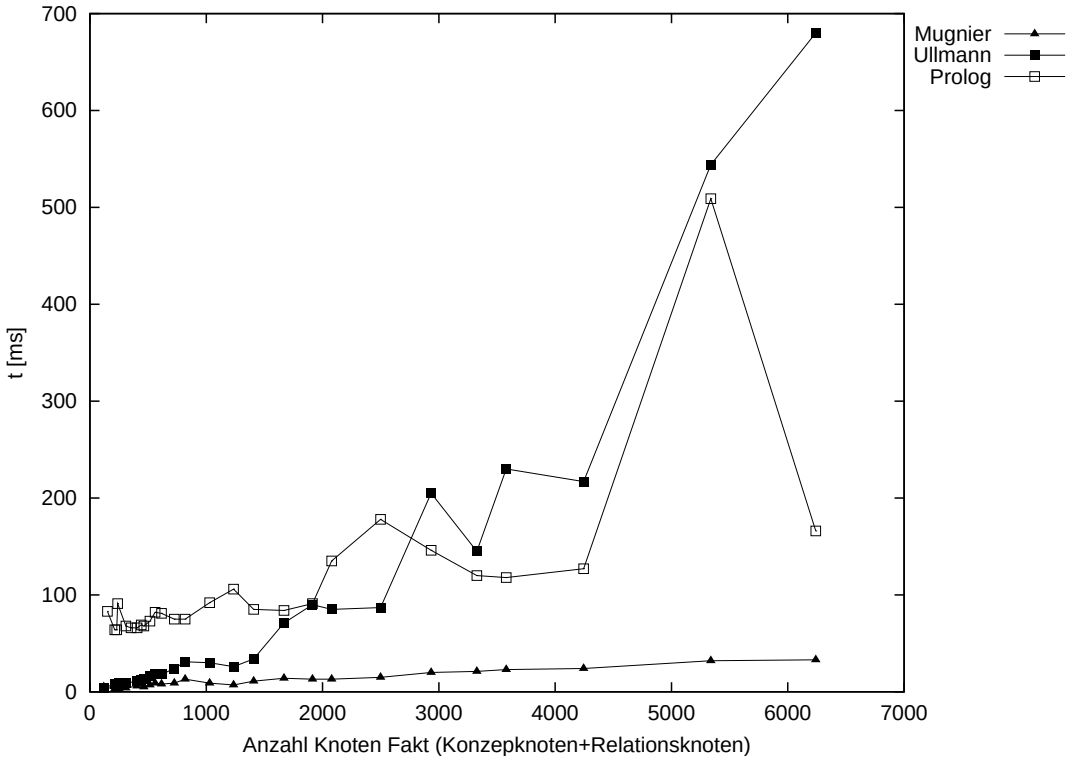

Abbildung 6.7: Meßergebnisse des zweiten Laufzeittests Teil 1

verschlechtert hat, wohingegen die Algorithmen nach Ullmann und Koch merklich – wenn auch nicht dramatisch – längere Laufzeiten aufweisen (ganz zu schweigen vom Notio-Algorithmus).

Aber nicht nur der Mugnier-Algorithmus hat sich kaum gegenüber den ersten Tests verschlechtert, auch PROLOG zeigt sich fast unbeeindruckt von den 7 zusätzlichen generischen Konzepten. Hierbei ist auch interessant, daß, wie in Abs. 5.3.5.4 bereits vermutet wurde, die Reihenfolge der Prädikate in der Zielklausel einen starken Einfluß auf die Rechenzeit hat. Ignoriert man den Vorschlag aus Abs. 5.3.5.4, zuerst die Prädikate in der Zielklausel anzugeben, die aus Relationen des Anfragegraphen entstanden sind, so liegt, wie weitere Messungen ergeben haben, die Rechenzeit bei 500 Knoten bereits bei über einer Minute, im Gegensatz zu ca. 60 ms hier!

Keiner der implementierten NP-Algorithmen (Ullmann, Koch) zeigt eine exponentielle Laufzeit, die theoretisch auftreten könnte. Trotzdem kommen diese Algorithmen bei großen Wissensbasen an ihre Leistungsgrenzen.

6.2.1.4 Fazit der Messungen

Graph-Matching-Algorithmen Es zeigt sich deutlich, daß der Notio-Algorithmus zumindest bei vielen generischen Konzepten nicht für größere Organisationsstrukturen zur Selektion von Aufgabenträgern eingesetzt werden kann. Allerdings muß man fairer Weise sagen, daß die Programmierer der Notio-Bibliothek auch zunächst keine hocheffizienten Algorithmen imple-

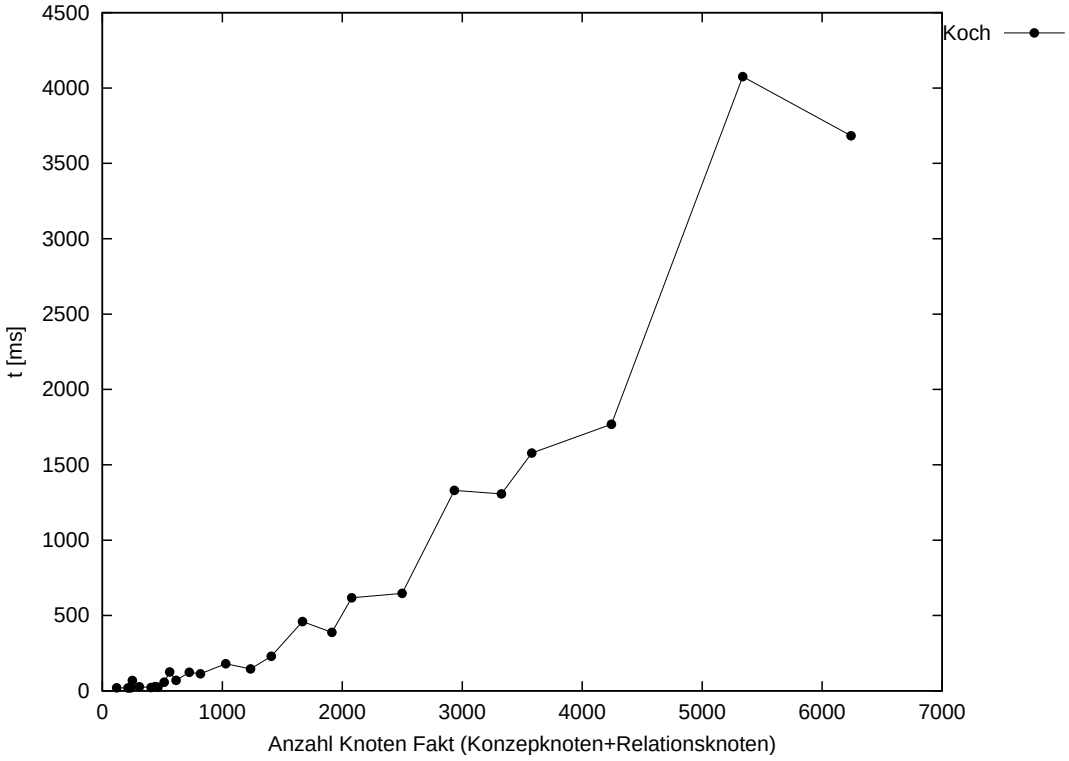

Abbildung 6.8: Meßergebnisse des zweiten Laufzeittests Teil 2

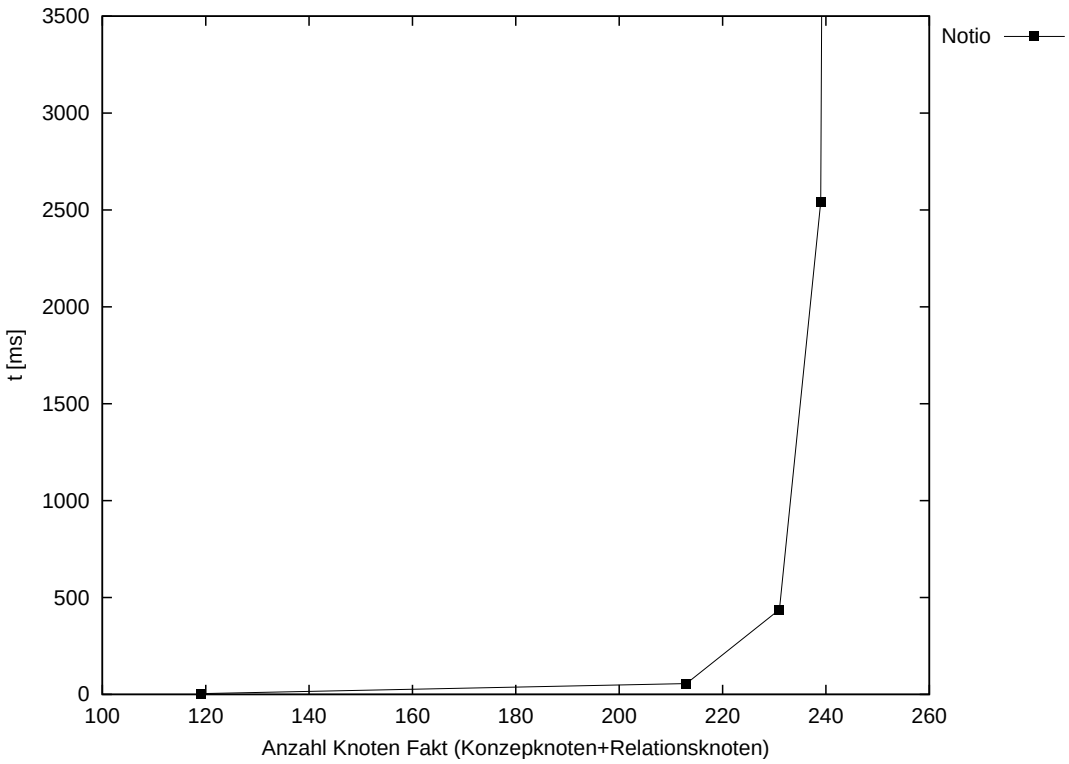

Abbildung 6.9: Meßergebnisse des zweiten Laufzeittests Teil 3

mentieren wollten, sondern ihren Schwerpunkt mehr auf einen sauberen Entwurf ihrer Bibliothek gelegt haben.

Bei den verbleibenden Graph-Matching-Algorithmen muß man berücksichtigen, unter welchen Bedingungen die Algorithmen zum Einsatz kommen. Der Mugnier-Chein-Algorithmus kann sicherlich uneingeschränkt sowohl für totale als auch für partielle Projektionen eingesetzt werden, wenn man die Einschränkung auf baumförmige Anfragegraphen in Kauf nimmt. Will man sich nicht auf baumförmige Anfragegraphen beschränken, so kommen nur der Ullmann- und Koch-Algorithmus in Frage. Der Koch-Algorithmus wurde hier speziell für partielle Projektionen eingeführt. Da eine totale Projektion aber nur einen Sonderfall der partiellen Projektion darstellt, könnte man theoretisch auch den Koch-Algorithmus für die Suche nach totalen Projektionen verwenden. Daß dies nicht sinnvoll ist, zeigen die Meßergebnisse. Der Ullmann-Algorithmus weist hier wesentliche Geschwindigkeitsvorteile gegenüber dem Koch-Algorithmus auf. Es ist daher nur sinnvoll, den Koch-Algorithmus einzusetzen, wenn wirklich partielle Projektionen gesucht werden, d.h. wenn mit dem Ullmann-Algorithmus kein Akteur gefunden werden konnte, der die geforderten Eigenschaften vollständig besitzt. Geht man davon aus, daß dieser Fall nur selten eintritt, so kann man die relativ hohe Laufzeit des Koch-Algorithmus sicherlich in Kauf nehmen. Für sehr große Wissensbasen besteht eigentlich – zumindest für den vorgestellten Ansatz – keine Alternative zum Mugnier-Chein-Algorithmus.

Ein kritischer Punkt beim Graph-Matching-Ansatz ist die Anwendung der Regeln, da hierfür nur der Ullmann-Algorithmus in Frage kommt. Die Regelanwendung wird zwar offline durchgeführt, trotzdem zeigt Abb. 6.7 deutlich, daß zumindest große Wissensbasen diesen Algorithmus an seine Grenzen führen, vor allem, wenn die Regeln oft anzuwenden sind, bis eine regelfreie Form der Wissensbasis erreicht ist.

PROLOG Die Messungen haben gezeigt, daß PROLOG ohne Bedenken auch für große Wissensbasen eingesetzt werden kann. Allerdings bietet PROLOG nach dem vorgestellten Ansatz nicht wie die Graph-Matching-Algorithmen nach Mugnier/Chein und Koch die Möglichkeit, unscharfe Anfragen durchzuführen. Außerdem enthielten die Testfälle keine Konzeptgraphen-Regeln. Bei PROLOG muß daher noch mit Geschwindigkeitseinbußen gerechnet werden, da die Regelanwendung nicht wie beim Graph-Matching-Ansatz offline, sondern zur Zeit der Selektion erfolgt.

6.2.2 Bewertung der erreichten Entkopplung von Aufbau- und Ablauforganisation

Eines der Hauptziele des vorgestellten Entwurfes ist die Entkopplung von Aufbau- und Ablauforganisation, d.h. eine Änderung am Organisationsmodell sollte nach Möglichkeit keinerlei Auswirkungen auf die Prozeßdefinitionen haben und umgekehrt. Eine Bewertung, ob, und wenn ja, bis zu welchem Grad eine Entkopplung erreicht werden konnte, soll an einem Beispiel erfolgen.

6.2.2.1 Auswirkungen von Änderungen

Gegeben sei ein fiktives Versicherungsunternehmen, dessen Organisationsstruktur ausschnittsweise in Abb. 6.10 dargestellt ist. Es handelt sich um eine typische divisionale Organisation (siehe Abs. 7) mit einer Gliederung auf erster Ebene nach Objekten (Lebensversicherungen, Unfallversicherungen, usw.) und den typischen Kernbereichen (Marketing, EDV, usw.). Der vorgestellte Prozeß spielt sich im Bereich *Lebensversicherungen* ab. Diese Abteilung wird von einem

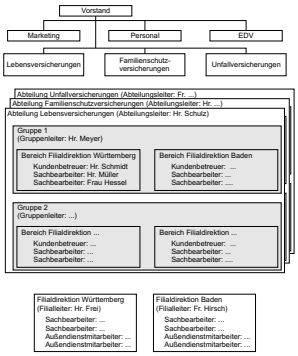

Abbildung 6.10: Organisationsstruktur einer fiktiven Versicherung

Abteilungsleiter geführt, dem mehrere Gruppenleiter unterstellt sind. Zu einer Gruppe gehören mehrere Bereiche. Jeder dieser Bereiche ist für eine bestimmte Filialdirektion zuständig und umfaßt mehrere Sachbearbeiter. Jeder Bereich hat einen ausgezeichneten Sachbearbeiter, der die Rolle des Kundenbetreuers übernimmt. Seine Aufgaben seien zunächst aber nur beratender Natur.

Der Workflow, der näher betrachtet werden soll, ist in Abb. 6.11 dargestellt. Die einzelnen Pro-

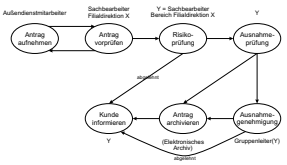

Abbildung 6.11: Ablauf zum Abschluß einer Lebensversicherung

zeßschritte sind mit Ovalen dargestellt, die Akteure jeder Aktivität stehen jeweils über oder unter den Tätigkeiten. Folgende Schritte sind zum Abschluß einer Lebensversicherung durchzuführen:

1. Ein Außendienstmitarbeiter nimmt den Antrag des Kunden auf und übergibt ihn seiner Filialdirektion.

2. Die Filialdirektion führt eine erste Vorprüfung durch um festzustellen, ob der Antrag z.B. vollständig ausgefüllt wurde.

3. Es erfolgt eine Risikoabschätzung. Dieser Schritt wird in der Zentrale von einem Sachbearbeiter durchgeführt, der für die entsprechende Filialdirektion zuständig ist.

4. Sollte der Vertrag spezielle Vereinbarungen enthalten (Sonderkonditionen, o.ä.), so muß der zuständige Gruppenleiter diese Ausnahmen genehmigen.

5. Wurde der Antrag genehmigt, so gilt er als abgeschlossen und kann archiviert werden. Dieser Schritt wird vollautomatisch von einer Anwendung übernommen, die durch das Workflow-Management-System aktiviert wird.

6. Die schriftliche Benachrichtigung des Kunden wird wieder von einem Sachbearbeiter übernommen.

Für die Schritte 4 und 6 wird der Einfachheit halber angenommen, daß sie vom selben Sachbearbeiter wie Schritt 3 durchzuführen sind, d.h. die Aufgabenträgerzuweisung erfolgt hier über die Historie des Workflow-Management-Systems. Näher betrachtet werden nur die Schritte 3 und 4. Für diese Schritte sind zwei Elemente der Organisationsstruktur von Interesse, nämlich der Sachbearbeiter und der Gruppenleiter.

Die Fakten, die einen konkreten Sachbearbeiter und Gruppenleiter beschreiben, sind in Abb. 6.12 zu sehen. In Abs. 5.6.4 wurde vorgeschlagen, wie man Rollen und Regeln verwenden kann, um die Modellierung zu vereinfachen. Für dieses kleine Beispiel wird auf diese Hilfsmittel verzichtet, um nicht vom Wesentlichen abzulenken.

Der Akteur *Müller* stellt einen Sachbearbeiter dar. Er hat die Fähigkeit zur Risikoanalyse von Lebensversicherungen. Es ist anzunehmen, daß Herr Müller zur Aneignung dieser Fähigkeit eine besondere Ausbildung genossen hat. Evtl. hat er auch nur eine Einführung in ein spezielles Softwaresystem erhalten. Dann wäre es sinnvoll, die Fähigkeit entsprechend einzuschränken (z.B. durch die Relation *WKZ*, die ein Werkzeug kennzeichnet, mit dem eine Fähigkeit ausgeführt werden kann; siehe Abs. 5.6.2). Trotzdem ist diese Fähigkeit sicher unabhängig von der Position des Sachbearbeiters und stellte damit wirklich eine Fähigkeit dar. Im Gegensatz zu dieser Fähigkeit handelt es sich bei der Tätigkeit *Bearbeiten* um ein Recht bzw. eine Pflicht des Akteurs. Herr Müller ist berechtigt, Lebensversicherungen zu bearbeiten, die von der Filialdirektion Württemberg aufgenommen wurden. Zunächst stellt sich die Frage, wie genau dieses Recht zu spezifizieren ist. Genügt es, die Befugnis als *Bearbeitung* zu beschreiben, oder sollte man sie lieber als *Risikoprüfung* beschreiben, also genau der einzelnen Tätigkeit, die eigentlich durchzuführen ist? Im Zweifelsfall sind genauere Beschreibungen sicher vorzuziehen, denn nur so können Rechte zur Durchführung verschiedener Tätigkeiten unterschieden und später evtl. auf verschiedene Akteure verteilt werden. Hier spielt auch die Erfahrung des Modellierers eine große Rolle. Interessant ist auch die Relation *Aufnehmende FD*. Diese Relation wurde neu eingeführt, um die Befugnis genau beschreiben zu können. Diese Flexibilität zur Einführung neuer Begriffe bei Bedarf ist ein wesentlicher Vorteil des Konzeptgraphenansatzes.

Abb. 6.12b definiert das Recht des Akteurs *Meyer* (des Gruppenleiters) Lebensversicherungen zu genehmigen, die von der Filialdirektion *Württemberg* entgegengenommen wurden.

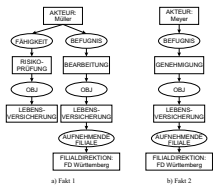

Abbildung 6.12: Fakten des Versicherungsbeispiels

Es werden nun verschiedene Änderungen an der Organisationsstruktur vorgenommen und beobachtet, ob diese Änderungen Auswirkungen auf die Prozeßbeschreibungen haben. Um überhaupt eine sinnvolle Aufgabenträgerzuweisung zu ermögliche, wird davon ausgegangen, daß der Bezeichner der Filialdirektion vom Workflow-Management-System als workflowrelevantes Datum verwendet werden kann. Dieser Wert kann somit automatisch als Parameter `*fd` in die Anfragegraphen aus Abb. 6.13 eingefügt werden. Der Anfragegraph aus Abb. 6.13a wird in Schritt 3 verwendet, um den entsprechenden Sachbearbeiter zu bestimmen, der Anfragegraph aus Abb. 6.13b wird verwendet, um den Gruppenleiter für Schritt 4 zu ermitteln (genauer eine Person, die berechtigt ist, Ausnahmegenehmigungen für Lebensversicherungen auszustellen; nach der Rolle *Gruppenleiter* soll ja gerade nicht gesucht werden).

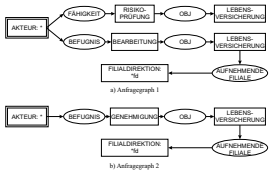

Abbildung 6.13: Anfragegraphen des Versicherungsbeispiels

Änderung 1 – Zusammenlegung der Bereiche Württemberg und Baden Hierzu müssen allen Sachbearbeitern der beiden Bereiche neue Befugnisse für den vormals anderen Bereich zugeordnet werden, was problemlos möglich ist (Müller müßte z.B. zusätzlich die Befugnis für die Filialdirektion Baden zugeordnet werden). Die Anfragegraphen können unverändert übernommen werden und liefern automatisch die richtigen Ergebnisse.

In herkömmlichen Workflow-Management-Systemen würde man mit großer Wahrscheinlichkeit in Punkt 3 nach einem Akteur suchen, der die Rolle *Sachbearbeiter_Bereich_FD_Württemberg* einnimmt. Da dieser Bereich nach der Änderung nicht mehr existiert, laufen alle Anfragen ins Leere. Dieser Punkt entspricht dem Punkt 2a aus Abs. 3.4 und kann daher wie bereits beschrieben mit Versionierung gelöst werden, da es eindeutig ist, welcher neue Bereich dem alten Bereich entspricht (vormals Württemberg und Baden ist nun Baden-Württemberg). Aber bereits mit einer gewöhnlichen Datenbank ließe sich in diesem Fall dieselbe Wirkung wie mit der Konzeptgraphenmodellierung erzielen, denn man müßte nur eine Tabelle anlegen, die jedem Sachbearbeiter die Filialdirektion zuordnet, für die er zuständig ist. Mit einer einfachen Anweisung, wie z.B. `SELECT akteur FROM tabelle WHERE fd='Baden-Württemberg'`, könnte dann der entsprechende Akteur ermittelt werden, immer vorausgesetzt, die Tabelle ist auf dem neuesten Stand. Im Prinzip entspricht der Inhalt einer solche Tabelle zumindest teilweise den Informationen, die in den Fakten der Wissensbasis gespeichert sind.

Änderung 2 – Aufteilung des Bereichs Baden-Württemberg Dies entspricht der Umkehrung von Änderung 1. Dieser Punkt stellt überhaupt kein Problem für den Konzeptgraphen-Ansatz dar, vorausgesetzt es wurde für jeden Akteur modelliert, für welche Filialdirektionen er zuständig ist. Dabei muß die Modellierung vorausschauend erfolgt sein. Wurde bei der Modellierung nicht realisiert, daß eine Änderung dieser Art auftreten könnte, und wurde daher vergessen, die Zuständigkeit für eine Filialdirektion zu modellieren, so hat man auch mit dem Konzeptgraphenansatz ein Problem, da die entsprechende Information nun einfach fehlt. Die Modellierung muß daher immer auch im Hinblick auf wahrscheinliche zukünftige Änderungen erfolgen.

Konventionelle Rollen-Ansätze habe hier, wie Abs. 3.4 Punkt 2b bereits gezeigt hat, auch mit Versionierung keine Chance. Mit der oben beschriebenen Tabelle ist aber auch dieser Fall zu lösen.

Änderung 3 – Entfernung des Gruppenleiters Im Zuge des Lean-Managements könnte man auf die Idee kommen, die Stelle des Gruppenleiters (und somit im Prinzip das ganze Gruppenkonzept) zu streichen und seine Aufgaben auf den Abteilungsleiter und die Kundenberater der Bereiche zu verteilen. Zum Beispiel könnte die Befugnis zur Genehmigung von Ausnahmen auf den Kundenberater übergehen. Die Anweisung `Gruppenleiter(Y)`, die eigentlich den Gruppenleiter eines Sachbearbeiters ermitteln sollte, ist nun wirkungslos. Man könnte nun einwänden, daß man die Zuweisung nach der Art `Vorgesetzter(Y)` hätte gestalten sollen. Da man

den Kundenberater nun als Vorgesetzten der Sachbearbeiter ansehen kann, hätte man der Änderung somit ohne Anpassung von Zuweisungsregeln begegnen können. Dies ist aber nur scheinbar eine Lösung, denn was geschieht dann mit den Kompetenzen, die vom Gruppenleiter nicht auf den Kundenberater, sondern auf den Abteilungsleiter übergegangen sind? Bei näherem Hinsehen entpuppt sich diese Veränderung als Punkt 3 aus Abs. 3.4, der wie bereits gezeigt mit dem Rollenkonzept alleine nicht lösbar ist.

Der Konzeptgraphenansatz hat dagegen keine Probleme mit dieser Änderung. Geänderte Befugnisse werden explizit dem Abteilungsleiter oder Kundenberater zugeordnet. Der Anfragegraph bleibt unverändert, die Änderung bleibt auf das Modell Organisationsstruktur beschränkt.

Änderung 4 – Zusammenlegung aller Bereiche Im Prinzip entspricht dies einer Mehrfachausführung von Punkt 1. Man kann sich aber auch vorstellen, daß man in der Zentrale vollständig darauf verzichten will, Sachbearbeiter bestimmten Filialdirektionen zuzuordnen. Vielleicht hat man festgestellt, daß es eigentlich keinen zwingenden Grund gibt, Anträge von einer Filialdirektion von Sachbearbeitern eines bestimmten Bereichs bearbeiten zu lassen, schließlich sind die Anträge in jeder Filialdirektion gleich (die Filialen selbst dagegen machen durchaus Sinn, denn man will den regional ansässigen Kunden und Außendienstmitarbeitern weiterhin eine Anlaufstelle bieten). Was würde also passieren, wenn man den bisherigen Workflow mit den bisherigen Zuweisungsregeln weiterlaufen lassen würde? Der konventionelle Ansatz scheitert jetzt auch trotz der in Änderung 1 beschriebenen Tabelle, denn was nützt eine Zuordnung von Sachbearbeitern zu Filialdirektionen, wenn man diese Zuteilung aufgeben will. Auch in den Anfragegraphen ist weiterhin die Filialdirektion enthalten, aus den Fakten hat man sie aber entfernt. Somit kann keine totale Projektion des Anfragegraphen auf die Fakten mehr gefunden werden. Trotzdem werden weiterhin Akteure über partielle Projektionen gefunden, denn der Anfragegraph sucht auch nach Akteuren, die die Fähigkeit haben, Risikobeurteilungen für Lebensversicherungen durchzuführen. Diese Fähigkeit hat niemand den Akteuren abgesprochen, und daher ist nun jeder Akteur zwar nicht zu 100% zur Durchführung der Aufgabe geeignet, aber immerhin zu 40%, wenn man Metrik 3 aus Abs. 5.3.7.3 zugrundelegt.

Dieses Beispiel zeigt, was passieren könnte, wenn man einen Workflow auf einer anderen, als der ursprünglich vorgesehenen Organisationsstruktur ohne Änderungen ausführt (Punkt 5, Abs. 3.4). Man wird oft keinen Akteur finden, der die geforderten Eigenschaften vollständig erfüllt, aber mit etwas Glück einen, der zumindest einen Teil der Anforderungen erfüllt. Die Metriken aus Abs. 5.3.7.3 können dann evtl. helfen, den Akteur zu finden, der die Anforderungen am besten erfüllt. Auf Dauer wird man aber nicht mit einem „ganz gut“ passenden Akteur leben wollen. Es muß daher entweder eine Anpassung der Zuweisung oder des Anfragegraphen erfolgen, falls die Annahmen, unter denen der Anfragegraph erstellt wurde, nicht mehr zutreffend sind (wie im obigen Beispiel), oder man muß die Wissensbasis um Einträge erweitern, die bisher einfach nicht modelliert wurden, es muß also ein Lernvorgang stattfinden, der fehlende Einträge in der Wissensbasis ergänzt und somit das Wissen vervollständigt. Der Lernvorgang selbst wird im Detail in [**?**] beschrieben.

Eigentlich gibt es noch einen dritten Fall, der bei partiellen Abbildungen zu berücksichtigen ist, denn es kann tatsächlich kein Akteur im Unternehmen vorhanden sein, der das geforderte Anforderungsprofil vollständig erfüllt. Dieses Problem läßt sich mit einem rein technischen Ansatz allerdings nicht lösen.

6.2.2.2 Fazit

Wie das obige Beispiel zeigt, ist der vorgestellte Ansatz sehr gut geeignet, die Entkopplung von Ablauf- und Aufbauorganisation zu erhöhen. Alle als wichtig erachteten Änderungen der Organisationsstruktur sind ohne Änderungen an den Prozeßdefinitionen möglich, vorausgesetzt, die notwendigen Informationen sind auch in der Wissenbasis vorhanden, und man bemüht sich, nicht nach hergebrachter Weise nach Rollen, sondern nach elementaren Eigenschaften wie Fähigkeiten und Rechten zu selektieren.

Der Konzeptgraphenansatz darf aber auch nicht ganz unkritisch gesehen werden. Es ist teilweise nicht einfach, im voraus zu sagen, welche Eigenschaften wirklich wichtig sind (um so wichtiger ist ein Lernvorgang). Auch der vorgestellte Ansatz bietet noch genügend Raum für Verbesserungen. Wie und welche Eigenschaften zu modellieren sind wird sich teilweise erst beantworten lassen, wenn die nötige Erfahrung vorliegt, die nicht durch einige wenige Beispiele entstehen kann. Einer der größten Vorteile, nämlich die große Flexibilität bei der Modellierung, kann sich leicht ins Gegenteil wandeln, wenn man z.B. anfängt, Fähigkeiten o.ä. auf unterschiedliche Weise zu modellieren. Eine sauber definierte Ontologie kann hier bis zu einem gewissen Grad Vorgaben machen, aber bereits die Aufstellung einer solchen ist keine leichte Aufgabe und erfordert selbst ausführliche Analysen der Unternehmensstruktur. Auch lassen sich teilweise mit „konventionellen“ Mitteln, wie z.B. Datenbanken, zumindest ähnliche Resultate erzielen, vorausgesetzt, man setzt sie richtig ein. Gerade die Suche nach partiellen Abbildungen ist aber ein Merkmal, das den vorgestellten Ansatz zur Aufgabenträgerzuweisung von den bisher verwendeten deutlich abhebt.

6.2.3 Abschließende Bewertung

Dieser Abschnitt liefert eine abschließendes Fazit, inwieweit die Anforderungen aus Abs. 5.1 durch den vorgestellten Ansatz erfüllt werden konnten.

Freie Definition der Organisationsstruktur: Dieser Punkt wurde, wie Abs. 5.3.1 gezeigt hat, vollständig erfüllt, vorausgesetzt, man legt eine Ontologie zugrunde, die keine unnötigen Einschränkungen macht, bzw. frei erweiterbar ist.

Schwieriger ist beim Konzeptgraphenansatz eher der umgekehrte Fall, nämlich das sinnvolle Einschränken der entstandenen Freiheit bei der Modellierung. Konzeptgraphen bieten kein so enges „Korsett“, wie z.B. das vorgegebene Datenbankschema eines Workflow-Management-Systems. Wichtig ist daher eine Möglichkeit, Konsistenzbedingungen zu definieren, die einen bei der Modellierung mit Konzeptgraphen daran hindern, nicht erlaubte Einträge vorzunehmen oder bestimmte Eingaben einfach zu unterlassen, die obligatorisch sind. Abs. 5.6.6 hat dazu gezeigt, wie man Konsistenzbedingungen mit Konzeptgraphen definieren kann. Doch wurde auch deutlich, daß die Mittel, die einfache Konzeptgraphen zur Verfügung stellen, ihre Grenzen haben. Hier müssen weitere Mittel z.B. in Form erweiterter Konzeptgraphen zur Verfügung gestellt werden.

Entkopplung von Ablauf- und Aufbauorganisation: Auch dieser Punkt kann als praktisch vollständig erfüllt gelten, wieder vorausgesetzt, man kopiert mit Konzeptgraphen nicht nur rein rollenbasierte Ansätze, sondern modelliert elementare Eigenschaften eines Akteurs. Unter Umständen resultiert aus einer entsprechenden Modellierung zwar ein relativ hoher Aufwand, aber dafür erhält man Prozeß- und Organisationsmodelle die – soweit das möglich ist – unabhängig voneinander sind. Einschränkend muß gesagt werden, daß es nicht immer leicht ist, im voraus zu bestimmen, welche Eigenschaften tatsächlich zu modellieren sind, um eine ideale Entkopplung zu erhalten. Hier kann ein Lernmechanismus u.U. Erleichterung bei der Modellierung bringen.

Selektion von Aufgabenträgern aufgrund beliebiger Eigenschaften: Eigenschaften, die im Organisationsmodell vorhanden sind, können ohne Einschränkung zur Selektion herangezogen

werden. Allerdings sind Selektionen auf Grundlage von Zahlenvergleichen ($>$, $<$, $\geq$, usw.) ein nicht vollständig gelöstes Problem des verwendeten Graph-Matching-Ansatzes. Außerdem sind negierte Anfragen im bisherigen Ansatz nur über externe Mittel zu bewerkstelligen, liegen also außerhalb des vorgeschlagenen Konzeptgraphenmodells.

Unscharfe Selektion von Aufgabenträgern: Der Graph-Matching-Ansatz bietet die Möglichkeit, Akteure zu selektieren, die nicht alle geforderten Eigenschaften besitzen. Durch diverse Metriken kann eine Bewertung erfolgen, welcher Akteur die Anforderungen am besten erfüllt.

Leistungsfähigkeit (Performance): Diese Anforderung wurde zum Teil erfüllt. Kleine und mittlere Wissensbasen stellen dabei kein Problem dar. Große Wissensbasen bringen dagegen den Graph-Matching-Ansatz an seine Leistungsgrenzen, es sei denn, man nimmt Einschränkungen bei der freien Definition von Aufgabenträger-Selektionen in Kauf (Beschränkung auf baumförmige Anfragegraphen). Die Regelanwendung, wie sie bisher beim Graph-Matching-Ansatz verwendet wird, kann hiervon nicht profitieren. Evtl. sind hier andere Ansätze vorzuziehen (z.B. [**?**]), die aber aus Zeitgründen nicht betrachtet werden konnten.

Erfolgt die Selektion mit Hilfe von PROLOG, so stellen auch große Wissensbasen kein Problem dar. Der vorgestellte Ansatz bietet dann aber keine Möglichkeit mehr, unscharfe Selektionen durchzuführen.

Integration in Changengine: Der vorgestellte Ansatz wurde ohne größere Probleme in das Workflow-Management-System Changengine integriert.

Benutzerfreundlichkeit: Die Benutzungsschnittstelle ist zwar Gegenstand einer anderen Arbeit ([**?**]), trotzdem kann aus eigener Erfahrung gesagt werden, daß eine sehr komfortable visuelle Eingabe und Modellierung der Organisationsstruktur mit Hilfe eines Konzeptgraphen-Editors möglich ist.

Kapitel 7

Zusammenfassung und Ausblick

Es wurde zunächst gezeigt, in welchem Umfeld Unternehmen heute operieren. Dabei wurde festgestellt, daß Faktoren wie kurze Produktlebenszyklen, wachsender Konkurrenzdruck, anspruchsvollere Kunden, sich ändernde Rechtsvorschriften und kurze Entwicklungszeiten für neue Produkte, Unternehmen dazu zwingen, einen hohen Grad an Flexibilität und Wandlungsfähigkeit zu entwickeln, um in diesem turbulenten Umfeld erfolgreich bestehen zu können. Es wurde ferner festgestellt, daß diese geforderte Flexibilität auch Auswirkungen auf die Workflow-Management-Systeme im allgemeinen und auf die Kopplung von Aufbau- und Ablauforganisation innerhalb dieser System im speziellen hat. Es ergab sich die Forderung, eine möglichst große Entkopplung zwischen dem Prozeßmodell und dem Modell der Organisationsstruktur zu erreichen, damit sich Änderungen an einem Modell nicht auf das andere auswirken.

Es folgte eine Erläuterung der betriebswirtschaftlichen Grundlagen, wobei festgestellt wurde, daß in der Praxis eine sehr große Anzahl an verschiedenen Organisationsformen anzutreffen ist. Außerdem wurden die Grundlagen der Workflow-Management-Systeme erläutert und die verschiedenen Aspekte dieser Systeme vorgestellt, wobei der Schwerpunkt auf dem für diese Arbeit zentralen organisatorischen Aspekt lag.

Die anschließende Analyse der Metamodelle verschiedener Workflow-Management-Systeme brachte zu Tage, daß die meisten dieser Modelle bereits wesentliche Einschränkungen bei den Modellierungsmöglichkeiten der Organisationsstruktur eines Unternehmens aufweisen. Nur der analysierte Ansatz nach [**?**] ermöglicht eine freie Definition der Aufbauorganisation. Alle anderen Modelle schränken bereits auf konkrete Organisationsformen ein und ermöglichen somit bereits durch diese Einschränkung keine wirkliche Flexibilität. Der zweite analysierte Punkt betraf die Zuweisung von Aufgaben zu Aufgabenträgern, also die Kopplung von Aufbau- und Ablauforganisation. Anhand verschiedener möglicher Änderungen der Organisationsstruktur wurden festgestellt, daß nur eine eigenschaftenbasierte Aufgabenträgerzuweisung eine wirkliche Entkopplung zuläßt. Bis auf einen Forschungsansatz benutzten alle Systeme zur Aufgabenträgerzuweisung ein Rollenkonzept, bei dem die Analyse aber ergab, daß hiermit keine ausreichende Entkopplung zu erreichen ist, da Änderungen an den Rollen oft eine aufwendige Anpassung aller Zuweisungen nachsichziehen, die diese Rolle referenzieren. Auch der vorgestellte Versionierungsansatz bringt nur in bestimmten Fällen eine Verbesserung und ist daher nicht ausreichend.

Nach einer kurzen Einführung in die Grundlagen der Konzeptgraphen, die das zentrale Beschreibungsmittel für den vorgestellten Entwurf darstellen, konnte mit Hilfe der gewonnenen Erkenntnisse eine Menge von Anforderungen an die zu erstellende Organisationsverwaltung aufgestellt werden. Die Kernpunkte waren hierbei die möglichst freie Definition der Aufbauorganisation und eine möglichst große Entkopplung des Prozeßmodells vom Modell der Organisationsstruk-

tur, sowie eine ausreichend hohe Leistungsfähigkeit im Sinne von kurzen Antwortzeiten bei der Aufgabenträgerzuweisung.

Für den anschließenden Entwurf wurden Konzeptgraphen als zentrales Beschreibungsmittel für das Modell der Organisationsstruktur gewählt, da diese zum einen eine große Flexibilität bei der Modellierung zulassen, aber auch durch die mögliche Abbildung eines Konzeptgraphen auf Formeln der Prädikatenlogik 1. Stufe eine klar definierte Semantik besitzen. Außerdem besteht die Möglichkeit, eine Ontologie in Form von Typdefinitionen und -hierarchien vorzugeben, so daß die Modellierung aufgrund klar definierter Begriffe erfolgen kann. Da bereits in der Analyse die Bedeutung der Allgemeingültigkeit eines Metamodells deutlich geworden war, wurde zunächst ein sehr allgemeines Metamodell entworfen, das zum einen allgemein genug ist, jede Organisationsform zu beschreiben, aber zum anderen auch die Möglichkeit bietet, durch eigene Typdefinitionen eine Anpassung an individuelle Bedürfnisse vorzunehmen. Nachdem die einzelnen Schritte der Aufgabenträgerzuweisung festgehalten wurden, konnte zunächst der Entwurf einer organisatorischen Wissensbasis erfolgen, die das Modell der Aufbauorganisation eines Unternehmens aufnimmt. Drei Hauptbestandteile einer solchen Wissenbasis wurden entworfen: Typhierarchien, Fakten und Regeln. Der zweite Schritt befaßte sich mit den Möglichkeiten zur Selektion von Aufgabenträgern (Akteuren) aus dieser Wissensbasis. Hierfür wurde eine spezielle Form von Konzeptgraphen entworfen, die sogenannten Anfragegraphen. Um mit Hilfe dieser Anfragegraphen eine Selektion von Akteuren durchzuführen, wurde zunächst ein Ansatz verwendet, der auf Graph-Matching beruht. Es wurde gezeigt, daß dieser Ansatz korrekte und vollständige Ergebnisse liefert. Außerdem wurden mehrere Graph-Matching-Algorithmen näher betrachtet, die für die Realisierung des Graph-Matching-Ansatzes in Frage kommen. Des weiteren wurde auch eine Abbildung der Wissensbasis auf Klauseln vorgeschlagen, so daß die Selektion von Akteuren mit der logischen Programmiersprache PROLOG erfolgen kann. Für den Graph-Matching-Ansatz wurde außerdem ein Verfahren zur unscharfe Selektionen von Akteuren entworfen. Zur Bewertung der Güte solcher unscharfen Selektionen wurden verschieden Metriken entworfen. Der Entwurf einer Unternehmensontologie schloß die Entwurfsphase ab. Bei dieser Ontologie wurde großer Wert auf Erweiterbarkeit und Möglichkeiten zur expliziten Modellierung von Fähigkeiten und Befugnissen eines Akteurs gelegt.

Die Implementierung des Entwurfes erfolgte in Java, wobei für die Selektion mittels PROLOG ein externer PROLOG-Compiler verwendet wurde. Die realisierte Organisationsverwaltung wurde in das kommerzielle Workflow-Management-System Changengine integriert. Außerdem erfolgte eine Anbindung an einen Konzeptgraphen-Editor, der parallel zu dieser Arbeit entworfen und realisiert wurde.

Abschließend erfolgte eine Bewertung der Leistungsfähigkeit der implementierten Algorithmen zur Aufgabenträgerselektion mit Hilfe diverser Testfälle. Hierbei stellte sich heraus, daß auch diverse Algorithmen, die theoretisch exponentiellen Aufwand besitzen, zumindest für kleine und mittlere Wissenbasen verwendet werden können, wobei ein implementierter Algorithmus mit garantierter polynomieller Laufzeit jedoch immer die besten Antwortzeiten aufweisen konnte. Dieser Algorithmus hat allerdings eine Einschränkung der Anfragegraphen auf baumförmige Anfragegraphen zur Folge. Als ebenfalls sehr leistungsfähig stellte sich die Selektion mit Hilfe von PROLOG heraus, wobei der hier verfolgte Ansatz allerdings im Gegensatz zum Graph-Matching-Ansatz keine unscharfen Selektionen zuläßt.

Durch ein praktisches Beispiel, an dem diverse Änderungen der Organisationsstruktur vorgenommen wurde, konnte gezeigt werden, daß sich durch den vorgestellte Ansatz deutliche Verbesserungen bei der Entkopplung von Ablauf- und Aufbauorganisation erzielen lassen.

Der vorgestellte Ansatz zur wissensbasierten Modellierung der Organisationsstruktur mit Konzeptgraphen könnte vor allem durch einen Lernalgorithmus profitieren. Diese Erweiterung könnte

wesentlich Erleichterungen für den Modellierer bringen, da es sehr schwierig ist, ein vollständiges Modell der Organisationsstruktur ohne zusätzliche Hilfe zu erstellen.[1] Außerdem wäre es sicherlich interessant, andere Vorgehensweisen als den vorgestellten Graph-Matching-Ansatz für die Selektion von Aufgabenträgern zu betrachten, um deren Antwortzeitverhalten mit dem vorgeschlagenen Ansatz zu vergleichen (z.B. den Ansatz nach [**?**]). Auch die vorgeschlagenen und noch relativ eingeschränkten Regeln in Form von Konzeptgraphen könnten noch zu mächtigeren Regel erweitert werden, um die Modellierung der Organisationsstruktur weiter zu vereinfachen. Da der vorgestellte Ansatz sehr große Freiheiten bei der Modellierung läßt, diese aber nicht immer erwünscht sind, sind außerdem mächtigere Möglichkeiten zur Definition von Konsistenzbedingungen zu schaffen, als die relativ einfachen Möglichkeiten, die bisher hier entworfen wurden und die unter der Beschränkung auf einfache Konzeptgraphen leiden.

[1] Ein entsprechender Ansatz wurde z.B. in [**?**] erarbeitet.

Literaturverzeichnis

[BDS97] Bea, F. X. ; Dichtel, E. ; Schweitzer, M.: *Allgemeine Betriebswirtschaftslehre Bd. 2: Führung*. 7. Lucius & Lucius, 1997

[BGM99] Baget, Jean-François ; Genest, David ; Mugnier, Marie-Laure: Knowledge Acquisition with a Pure Graph-Based Knowledge Representation Model – Application to the Sisyphus-I Case Study. In: *Proceedings of the 12th Workshop on Knowledge Acquisition, Modeling and Management*, 1999

[BJ95] Bussler, Christoph ; Jablonski, Stefan: Policy Resolution for Workflow Management Systems. In: *Proceedings of the 28th Annual Hawaii International Conference on System Sciencies*, 1995, S. 831–840

[BJ96] Bussler, C. ; Jablonski, S.: Database Support for Open Workflow Management Systems / Universität Erlangen – Institut für mathematische Maschinen und Datenverarbeitung. 1996. – Arbeitsbericht Nr. 5

[BK73] Bron, Coen ; Kerbosch, Joep: A457–Finding All Cliques of an Undirected Graph. In: *Communications of the ACM* 16 (1973)

[Bußß98] Bussler, Christoph: *Organisationsverwaltung in Workflow-Management-Systemen*. Deutscher Universitäts-Verlag, 1998

[CS98] Coulondre, Stéphane ; Salvat, Eric: Piece Resolution: Towards Larger Perspectives. In: Mugnier, Marie-Laure (Hrsg.) ; Chein, Michel (Hrsg.): *Proceedings of the 6th International Conference on Conceptual Structures: Theory, Tools and Applications (ICCS-98)* Bd. 1453. Berlin : Springer, August 1998, S. 179–193

[DL88] Davis, S. M. ; Lawrence, P. R.: Matrix-Diamant. In: *Matrix Organisation*. Stuttgart, 1988, S. 17–30

[FBG96] Fox, Mark S. ; Barbuceanu, Mihai ; Gruninger, Michael: An Organisation Ontology for Enterprise Modelling: Preliminary Concepts for Linking Structure and Behaviour. In: *Computers in Industry* 29 (1996), S. 123–134

[GG95] Guarino, N. ; Giaretta, P.: Ontologies and Knowledge Bases: Towards a Terminological Clarification. In: Mars, N. J. I. (Hrsg.): *Towards Very Large Knowledge Bases*. IOS Press, Amsterdam, 1995, S. 25–32

[GHS95] Georgakopoulos, D. ; Hornick, M. ; Sheth, A.: An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. In: *Distributed and Parallel Databases* 3 (1995), S. 119–153

[GJ79] Garey, M. R. ; Johnson, D. S.: *Computers and Intractability – A Guide to the Theory of NP-Completeness*. San Francisco : Freeman, 1979

[GLT87] Garner, B. J. ; Lukose, D. ; Tsui, E.: Parsing Natural Language through Pattern Correlation and Modification. In: *Proceedings of the 7th International Workshop on Expert Systems & their Applications*. Avignon, Mai 1987, S. 1285–1299

[Gsc01] Gschwandtner, Thomas: *Entwicklung eines Editors zur Unterstützung der Organisationsverwaltung in Workflow-Management-Systemen*, Universität Stuttgart, Diplomarbeit, 2001. – Diplomarbeit Nr. 1914

[HC95] Hammer, M. ; Champy, J.: *Business Reengineering: Die Radikalkur für das Unternehmen*. 5. Campus Verlag, 1995

[HP] Hewlett Packard: *HP Changengine Resource Management Guide*. 4.3

[ISO95] Europäisches Komitee für Normung: *Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten, Teil 10: Grundsätze der Dialoggestaltung*. Februar 1995. – Europäische Norm EN ISO 9241-10

[Jab95] Jablonski, Stefan: *Workflow-Management-Systeme: Modellierung und Architektur*. Bonn : Intern. Thomson Publ., 1995

[Kla] Klarmann, Jürgen: A Comprehensive Support for Changes in Organizational Models of Workflow Management Systems. In: *Proceedings of the 4th International Conference on Information Systems Modelling*

[Koc01] Koch, Ina: Fundamental Study: Enumerating all Connected Maximal Common Subgraphs in two Graphs. In: *Theoretical Computer Science* 250 (2001), S. 1–28

[Leh91] Lehner: *Organisationslehre für Wirtschaftsinformatiker*. Hanser Studienbücher, 1991

[Lud97] Ludewig, Jochen. *Skript zur Vorlesung Software Engineering*. April 1997

[MB91] McCarthy, J. C. ; Bluestein, W. M.: The Computing Strategy Report: Workflow's Progress / Forrester Research Inc. Cambridge, Mass., Oktober 1991. – Forschungsbericht

[MC92] Mugnier, M. L. ; Chein, M.: Polynomial Algorithms for Projection and Matching. In: Pfeiffer, Heather D. (Hrsg.) ; Nagle, Timothy E. (Hrsg.): *Proceedings of the 7th Annual Workshop on Conceptual Structures: Theory and Implementation* Bd. 754. Las Cruces, NM : Springer, 1992, S. 239–251

[MLL92] Myaeng, S. H. ; Lopez-Lopez, A.: Conceptual graph matching: a flexible algorithm and experiments. In: *Journal of Experimental Theoretical Artificial Intelligence* 4 (1992), S. 107–126

[Mug92] Mugnier, Marie-Laure: *Contributions algorithmiques pour les graphes d'héritage et les graphes conceptuels*, Université Montpellier II, Doktorarbeit, Oktober 1992

[OMG00] Object Management Group: *The Common Object Request Broker: Architecture and Specification, Minor Revision 2.3.1*. 2000

[Pud96] Puder, Arno: *Typsysteme für die Dienstvermittlung in offenen verteilten Systemen.* Frankfurt am Main, Johann Wolfgang Goethe Universität, Doktorarbeit, 1996

[RM96] Rosemann, Michael ; Mühlen, Michael zur: Der Lösungsbeitrag von Metadatenmodellen beim Vergleich von Workflowmanagementsystemen / Universität Münster, Institut für Wirtschaftsinformatik. 1996. – Arbeitsbericht Nr. 48

[RM97] Rosemann, Michael ; Mühlen, Michael zur: Modellierung der Aufbauorganisation in Workflow-Management-Systemen: Kritische Bestandsaufnahme und Gestaltungsvorschläge / Westfälische Wilhelms-Universität Münster, Institut für Wirtschaftsinformatik. 1997. – Arbeitsbericht Nr. 47

[Sch92] Schöning, Uwe: *Logik für Informatiker.* 3. Bibliographisches Institut Mannheim, 1992 (Informatik 56)

[SL99] Southey, Finnegan ; Linders, James G.: Notio – A Java API for Developing CG Tools. In: *International Conference on Conceptual Structures*, 1999, S. 262–271

[Sla96] Slaghuis, Holger: Der direkte Übergang von BPR zum Workflow mit Leu. In: *Workflowmanagement – State-of-the-Art aus Sicht von Theorie und Praxis*, 1996

[SM96] Salvat, E. ; Mugnier, M.-L.: Sound and Complete Forward and Backward Chainings of Graph Rules. In: *Lecture Notes in Computer Science* 1115 (1996), S. 248–262

[SM99] Stader, J. ; Macintosh, A.: Capability Modelling and Knowledge Management. In: *Applications and Innovations in Expert Systems VII – Proceedings of ES 99 the 19th International Conference of the BCS Specialist Group on Knowledge-Based Systems and Applied Artificial Intelligence*, 1999, S. 33–50

[Sou01] Southey, Finnegan. *Notio Project Page.* `http://backtrack.math.uwaterloo.ca/CG/projects/notio/`. 2001

[Sow84] Sowa, J. F.: *Conceptual Structures: Information Processing in Mind and Machine.* Reading (MA), USA : Addison-Wesley, 1984

[Sow01] Sowa, John F. *Conceptual Graphs: Working Draft für ISO-Standard.* `http://www.bestweb.net/~sowa/cg/cgstand.htm`. 2001

[Spr99] Spribille, Tobias: *Entwicklung eines flexiblen Organisationsmanagements für eine kooperative Ablaufsteuerung*, Universität Stuttgart, Diplomarbeit, 1999. – Diplomarbeit Nr. 1763

[SWI01] *SWI-PROLOG.* `http://www.swi.psy.uva.nl/projects/SWI-Prolog/`. Juli 2001

[TSMB95] Teufel, S. ; Sauter, C. ; Mühlherr, T. ; Bauknecht:, K.: *Computerunterstützung für die Gruppenarbeit.* Bonn : Addison-Wesley, 1995

[UG96] Uschold, Mike ; Grüninger, Michael: Ontologies: Principles, Methods and Applications. In: *Knowledge Engineering Review* 1 (1996), Juni, Nr. 2, S. 93–155

[UKMZ98] Uschold, Mike ; King, Martin ; Moralee, Stuart ; Zorgios, Yannis: The Enterprise Ontology. In: *The Knowledge Engineering Review – Special Issue on Putting Ontologies to Use* 13 (1998)

[Ull76] Ullmann, J. R.: An Algorithm for Subgraph Isomorphism. In: *Journal of the Association for Computing Machinery* 23 (1976), Januar, Nr. 1

[Usc96] Uschold, Mike: Building Ontologies: Towards a Unified Methodology. In: *16th Annual Conf. of the British Computer Society Specialist Group on Expert Systems.* Cambridge, UK, 1996

[WMC95] Workflow Management Coalition: *The Workflow Reference Model.* Januar 1995. – Document Number TC00-1003, Document Status - Issue 1.1

[WMC99] Workflow Management Coalition: *Terminology & Glossary.* Februar 1999. – Document Number WFMC-TC-1011, Document Status - Issue 3.0

[Wöh00] Wöhe, G.: *Einführung in die allgemeine Betriebswirtschaftslehre.* Vahlen, 2000

Abbildungsverzeichnis

2.1 Einliniensystem . . . 16
2.2 Mehrliniensystem . . . 16
2.3 Funktionale Organisation . . . 17
2.4 Divisionale Organisation nach [?] . . . 17
2.5 Stabliniensystem . . . 18
2.6 Matrixorganisation . . . 18
2.7 Matrix-Diamant nach [?] . . . 18
2.8 Reine Projektorganisation (Task Force) . . . 19
2.9 Einfluß-Projektorganisation . . . 19
2.10 Klassifikation von Groupware nach [?] . . . 20
2.11 Beschreibungselemente der Aufbauorganisation nach [?] . . . 23
2.12 Komponenten eines Workflow-Management-Systems frei nach [?] . . . 25

3.1 Organisationsstruktur-Metamodell von WorkParty nach [?] . . . 28
3.2 Organisationsstruktur-Metamodell von FlowMark nach [?] . . . 29
3.3 Organisationsstruktur-Metamodell von Leu nach [?] . . . 30
3.4 Organisationsstruktur-Metamodell Diplomarbeit Nr. 1763, Universität Stuttgart nach [?] . . . 31
3.5 Prozeß-Definitions-Metamodell nach [?] . . . 32
3.6 Organisationsstruktur-Metamodell von MOBILE nach [?] . . . 33
3.7 Direkte und indirekte Zuweisung von Aufgabenträgern . . . 36
3.8 Kopplung von Aufbau- und Ablauforganisation nach [?] . . . 38
3.9 Formen von Änderungen der Organisationsstruktur . . . 40

4.1 Elemente eines Konzeptgraphen . . . 46
4.2 Konzeptgraph „Ein Rad ist Teil eines Autos.“ . . . 47
4.3 Konzeptgraph „Herr Müller fährt mit seinem Auto nach Stuttgart.“ . . . 47
4.4 Namenskontraktion . . . 49
4.5 Maße . . . 49
4.6 Konzepttyphierarchie . . . 52

4.7 Restriktion . . . 56
4.8 Verbinden . . . 56
4.9 Vereinfachen . . . 57
4.10 Konzeptgraph: „Keine Person ist ihr eigener Vorgesetzter.“ . . . 59

5.1 Ablauf der Aufgabenträgerzuweisung . . . 69
5.2 Basis-Metamodell der Organisationsstrukturverwaltung . . . 69
5.3 Regel . . . 69
5.4 Anfragegraph . . . 71
5.5 Beispiel für eine Projektion eines Anfragegraphen auf ein Fakt . . . 73
5.6 Verteilte Informationen und Anfragegraphen-Projektion . . . 74
5.7 Selektion mit Baum-Anfrage . . . 80
5.8 Datenstruktur Mugnier-Chein-Algorithmus . . . 82
5.9 Transformation in einen baumförmigen Anfragegraphen . . . 82
5.10 Bijektivität der Abbildung verhindert Projektion . . . 88
5.11 Regeln und Graph-Matching . . . 89
5.12 Matching und Zahlenvergleiche 1 . . . 91
5.13 Matching und Zahlenvergleiche 2 . . . 92
5.14 Anfragegraph ohne Zahlenvergleich . . . 93
5.15 Typhierarchien des PROLOG-Beispiels . . . 93
5.16 Anfragegraphen des PROLOG-Beispiels . . . 96
5.17 Graphen des Resolutionsbeispiels . . . 97
5.18 Berechnungsbaum der Resolution . . . 98
5.19 Partielle Projektion und nicht zusammenhängende Subgraphen . . . 101
5.20 Beispiel zum Algorithmus für die Suche nach maximalen partiellen Projektionen 106
5.21 Association-Graph . . . 106
5.22 Beispiel 1 zu Metriken für partielle Projektionen . . . 111
5.23 Metriken und Typhierarchien . . . 111
5.24 Beispiel 2 zu Metriken für partielle Projektionen . . . 114
5.25 Basisklassen . . . 117
5.26 Klassen der Wissensbasis . . . 117
5.27 Object Request Broker nach [**?**] . . . 117
5.28 Klassen der Anfragegraphen-Verwaltung . . . 118
5.29 Klassen der Taskholder-Finder-Komponente . . . 119
5.30 Integration der Organisationsverwaltung in Changengine . . . 124
5.31 Integration der Benutzungsschnittstelle . . . 124
5.32 Ontologie mit Agenten und Organisationseinheiten . . . 124

5.33 Fähigkeit 126
5.34 Beispiele für Objekte und Werkzeuge 127
5.35 Beispiel für die Anwendung von Rollen 128
5.36 Rollenaggregationsregel 128
5.37 Metamodell der Organisationsstrukturverwaltung 129
5.38 Konsistenzbedingungen 130

6.1 Naming-Service mit eingetragenen Server-Objekten 133
6.2 Sequenzdiagramm: Ablauf der Aufgabenträgerselektion 134
6.3 Konzeptgraphen-Editor mit einem Anfragegraph zum Changengine-Beispiel . . . 137
6.4 Testumgebung 140
6.5 Meßergebnisse des ersten Laufzeittests Teil 1 143
6.6 Meßergebnisse des ersten Laufzeittests Teil 2 143
6.7 Meßergebnisse des zweiten Laufzeittests Teil 1 144
6.8 Meßergebnisse des zweiten Laufzeittests Teil 2 145
6.9 Meßergebnisse des zweiten Laufzeittests Teil 3 145
6.10 Organisationsstruktur einer fiktiven Versicherung 147
6.11 Ablauf zum Abschluß einer Lebensversicherung 147
6.12 Fakten des Versicherungsbeispiels 148
6.13 Anfragegraphen des Versicherungsbeispiels 149

Index

Ablauforganisation, 13
  Def., 13
Abteilung, 15
Agent, 23
Akteur, 21
Aktivität, 21
Anfragegraph, 71
  Def., 71
Applikation, 22
  freie A., 22
Arbeitsgruppe, 15
Arbeitsliste, 21, 26
Arbeitslisten-Verwalter, 26
Arbeitsteilung, 1, 14
  horizontale A., 14
  vertikale A., 14
Argument, 46
Aufbauorganisation, 14
  Def., 14
  Grundmodelle, 16

Basis-Metamodell, 65
Benutzungsoberfläche
  Integration der B., 121
Bron-Kerbosch-Algorithmus, 103
Build-Time, 25
Business Reengineering, 4

Changengine
  Integration der Organisationsverwaltung in C., 118
Computer Supported Cooperative Work, 20
CORBA, 116

Denotation eines Typs
  Def., 50
Divisionale Organisation, 16

Einliniensystem, 16
Engine, 26
Entwurfsmuster, 132

Fakt, 67, 94
  Def., 67
FlowMark, 36
Folding, 88
Funktionale Organisation, 16
Funktionsspezialisierung, 14

Geschäftsprozeß
  Def., 4
Groupware, 20

Hornklausel, 94

Instanz, 15
Interface Definition Language, 116

Join, 56
  J. auf individuellen Konzepten, Def., 74
  max. J. auf individuellen Konz., 74

Klassenstruktur, 115
Klausel, 94
  Zielklausel, 97
Koch-Algorithmus, 102
Kompetenz, 15
Konfigurationsmanagement, 39
Kongruenzprinzip, 15
Konsistenzbedingungen, 129
Konzept, 46
Konzeptgraph
  Def., 54
  kanonischer K., 47
Konzeptionalisierung, 59
Konzepttyphierarchie
  Def., 51
Kopieren, 56

Lean Management, 7
Leitungsstelle, 15
Leitungssystem, 16
Leu, 37
lineare Notation, 46
Liniensystem mit Querfunktionen, 17

Marker

generischer M., 48
individuelle M., 48
Matrix-Organisation, 18
Mehrliniensystem, 16
Metamodell, 65, 128
Middleware, 22
MOBILE, 38
Mugnier/Chein-Algorithmus, 78

Namenskontraktion, 49
Naming Service, 132
Notio, 115
Notio-Algorithmus, 115

Object Request Broker, 116
Observer, 132
Ontologie, 59
Begriffe der Enterprise-O., 159
Enterprise-O., 122, 159
Unternehmens-O., 122
Organisationseinheit, 14
Organisationsstrukturverwaltung, 24
Organisationsverwaltung, 24
Klassen der O., 115
organisatorische Beziehung, 23
organisatorisches Objekt, 23
ORMS, 28

Phi-Operator, 55
Def., 54
Erweiterung für Regeln, 68
Policy Resolution, 38
Population, 23
Primärorganisation, 14
Programm, 22
adaptierbare P., 22
Legacy P., 22
Projekt
Team, 15
Projektion
Iso-P., 87
Iso-P., Def., 87
maximale partielle P., Def., 101
Metrik für partielle P., 109
partielle Iso-P., Def., 102
partielle P., Def., 100
Projektorganisation
Einfluß-P., 19
Matrix-P., 19
reine P., 19

PROLOG, 93
Prozeß-Instanz, 26

Referent, 48
Regel, 95
Regeln und Graph-Matching, 88
Relation, 46
Relationsexpansion, 53
Relationskontraktion, 53
Relationstyphierarchie
Def., 53
Resolution, 97
SLD, 97
Restriktion, 56
Rolle, 27
Runtime, 25

Sekundärorganisation, 14
Selektion
negierte S., 99
semantische Distanz, 111
Def., 113
Semantisches Netz, 46
Skolemform, 95
Spezialisierung, 56
Stabliniensystem, 17
Stelle, 15
Subtyp
Def., 51
echter S., Def., 51
gemeinsamer S., Def., 51
maximaler gemeinsamer Subtyp, 52
Subworkflow, 22
Supertyp
Def., 51
echter S., Def., 51
gemeinsamer S., Def., 51
minimaler gemeinsamer S., 51
Superworkflow, 22
Synchronisationsregel, 24

Task Force, 19
Team, 15
Tensor-Organisation, 18
Time-to-Market, 3
Toplevel-Workflow, 22
Typ-Definition, 53
Typ-Expansion, 51
Typ-Kontraktion, 51
Typdefinition, 50

Typhierarchie, 51

Versionierung, 10, 39

Wissensbasis
    kompilierte W., 77
Work List, 21
Work List Handler, 26
Workflow, 20
    Ad-hoc-W., 9
    Def., 20
    Subworkflow, 22
    Superworkflow, 22
    Toplevel-Workflow, 22
Workflow Enactment Service, 26
Workflow Management Coalition, 32, 37
Workflow-Management-System
    funktionaler Aspekt, 22
    historischer Aspekt, 24
    informationsbezogener Aspekt, 23
    kausaler Aspekt, 24
    Komponenten eines WMS., 25
    operationaler Aspekt, 22
    organisatorischer Aspekt, 23
    verhaltensbezogener Aspekt, 22
WorkParty, 28, 37

Zentralbereich, 17
Zuordnungsstrategie, 24
Zuweisungsregel, 24
Zuweisungsregelverwaltung, 24

Anhang A

Enterprise-Ontologie

A.1 Begriffe der Enterprise-Ontologie

Activity	Organisation	Strategy	Marketing	Time
Activity	Person	Purpose	Sale	Time Line
Activity Specification	Machine	Hold Purpose	Potential Sale	Time Interval
Execute	Coperation	Intended Purpose	For Sale	Time Point
Executed Activity Specification	Partnership	Purpose Holder	Sale Offer	
T-Begin	Partner	Strategic Purpose	Vendor	
T-End	Legal Entity	Objective	Actual Customer	
Pre-Condition	Organisational Unit	Vision	Potential Customer	
Effect	Manage	Mission	Customer	
Doer	Delegate	Goal	Reseller	
Sub-Activity	Management Link	Help Achieve	Product	
Authority	Legal Ownership	Stategy	Asking Price	
Activity Owner	Non-Legal Ownership	Stragegic Planning	Sale Price	
Event	Ownership	Strategic Action	Market	
Plan	Owner	Decision	Segmentation Variable	
Sub-Plan	Asset	Assumption	Market Segment	

Fortsetzung auf der nächsten Seite

Activity	Organisation	Strategy	Marketing	Time
Planning	Stakeholder	Critical Assumption	Market Research	
Process Specification	Employment Contract	Non-Critical Assumption	Brand	
Capability	Share	Influence Factor	Image	
Skill	Shareholder	Critical Influence Factor	Feature	
Resource		Non-Critical Influence Factor	Need	
Resource Allocation		Critical Success Factor	Market Need	
Resource Substitute		Risk	Promotion Competitor	

Tabelle A.1: Begriffe der Enterprise-Ontologie nach [**?**]

A.2 Meta-Ontologie

Die Meta-Ontologie besteht aus den folgenden Begriffen, die für die Definition der Begriffe aus Tab. A.1 verwendet werden:

- *Entity*
- *Relationship*: Beziehungen zwischen Entitäten. Zum Beispiel wird Sale als eine Beziehung zwischen zwei Legal Entities definiert.
- *Role*: Eine Entität kann in einer Relation eine bestimmte Rolle einnehmen. Zum Beispiel ist eine der Legal Entities in der Sale-Relation der Vendor.
- *Attribut*: Eine Entität kann mit einem oder mehreren Attributen verbunden werden.

Anhang B

Begriffe der entworfenen Unternehmensontologie

Dieses Kapitel gibt eine zusammenfassenden Übersicht über die Begriffe der in Abs. 5.6 entworfenen Unternehmensontologie. Nähere Erläuterungen sind daher Abs. 5.6 zu entnehmen.

UNIVERSAL bezeichnet hier wieder das Element $\top$ der Typhierachie. Zu Beginn ist jeweils angegeben, von welchem Typ sich der Begriff ableitet. Bei Relationen werden zusätzlich die Typen der Argumente und ein Beispiel für die Anwendung der Relation genannt.

B.1 Konzepte

AGENT < UNIVERSAL. Ein Agent ist eine Entität, die in der Lage ist, eine Tätigkeit durchzuführen.

AKTEUR < AGENT. Ein Akteur ist ein menschlicher Agent.

MASCHINELLER AGENT < AGENT. Ein maschineller Agent kann eine Maschine oder eine Applikation sein, die direkt vom Workflow-Management-System angesprochen werden kann, um Tätigkeiten automatisch durchzuführen.

ORGANISATIONSEINHEIT < UNIVERSAL. Eine Organisationseinheit bietet einem Unternehmen die Möglichkeit, das Unternehmen in Subsysteme zu gliedern (siehe Abs. 2.1.2).

FESTE OE < ORGANISATIONSEINHEIT. Der Supertyp aller Organisationseinheiten der Primärorganisation (siehe Abs. 2.1.2.1).

TEMPORÄRE OE < ORGANISATIONSEINHEIT. Der Supertyp aller Organisationseinheiten der Sekundärorganisation (siehe Abs. 2.1.2.1).

ABTEILUNG < FESTE OE. Eine Abteilung ist nur ein Beispiel einer festen Organisationseinheit (siehe Abs. 2.1.2.1). Beliebige weitere können durch Ableitung des Typs *FESTE OE* gebildet werden.

PROJEKT < TEMPORÄRE OE. Ein Projekt ist nur ein Beispiel einer temporären Organisationseinheit (siehe Abs. 2.1.2.1). Beliebige weitere können durch Ableitung des Typs *TEMPORÄRE OE* gebildet werden.

TÄTIGKEIT < UNIVERSAL. Eine Tätigkeit ist eine Aktion, die von einem Akteur innerhalb des Unternehmens ausgeführt werden kann. Die Tätigkeit bildet dabei den Supertyp aller näher spezifizierten Tätigkeiten. Tätigkeiten werden dazu verwendet, um Fähigkeiten und Befugnisse zu modellieren.

GENEHMIGEN < TÄTIGKEIT. Genehmigen ist nur ein Beispiel für eine konkrete Tätigkeit, die ein Akteur innerhalb eines Unternehmens ausführen kann. Weitere Tätigkeiten können durch Ableitung des Typs *TÄTIGKEIT* gebildet werden.

ART < UNIVERSAL. Art und Weise der Durchführung einer Tätigkeit.

GUT < ART. Gut ist nur ein Beispiel für die Art und Weise, wie eine Tätigkeit durchgeführt werden kann.

ROLLE < UNIVERSAL. Eine Rolle stellt eine abstrakte Person dar. Sie beinhaltet i.a. eine Menge von Befugnissen.

OBJEKT < UNIVERSAL. Dieser Typ dient als Supertyp beliebiger Objekte. Ein Objekt kann dabei z.B. eine Maschine oder ein Gerät, aber auch ein Software-System o.ä. sein. Durch entsprechende Relationen (siehe unten) kann ein Objekt als Werkzeug einer Tätigkeit dienen oder auch ein Objekt darstellen, auf dem eine Tätigkeit ausgeführt wird.

B.2 Relationen

Vorweg ein Beispiel, wie die Relationen am besten zu lesen sind:

[ABTEILUNG: Produktion Kfz]←(TEIL)←[DIVISION: Div. Kfz].

„Das Teil einer Division mit dem Namen *Div. Kfz* ist eine Abteilung mit dem Namen *Produktion Kfz*.“[1] Bei generischen Konzepten fällt der Name weg.

TEIL < UNIVERSAL.

1. ORGANISATIONSEINHEIT
2. ORGANISATIONSEINHEIT

Diese Relation ermöglicht die Schachtelung von Organsationseinheiten.

[ABTEILUNG: Produktion Kfz]←(TEIL)←[DIVISION: Div. Kfz].

MITGLIED < UNIVERSAL.

1. ORGANISATIONSEINHEIT
2. AKTEUR

Diese Relation bestimmt die Zuordnung eines Akteurs zu einer Organisationseinheit.

[AKTEUR: Meyer]←(MITGLIED)←[ABTEILUNG: Einkauf].

VORGESETZTER < UNIVERSAL.

[1] Repräsentiert ein Konzept eine Tätigkeit, so ist diese Art des Lesens u.U. weniger geeignet. Die Bedeutung dürfte aber trotzem klar sein.

1. AKTEUR
2. AKTEUR

Diese Relation bestimmmt das Vorgesetztenverhältnis.

[AKTEUR: Meyer]←(VORGESETZTER)←[AKTEUR: Schmidt].

LEITER < UNIVERSAL.

1. ORGANISATIONSEINHEIT
2. AKTEUR

Diese Relation bestimmmt den Leiter einer Organisationseinheit.

[AKTEUR: Meyer]←(LEITER)←[ABTEILUNG: Verkauf].

WKZ < UNIVERSAL.

1. TÄTIGKEIT
2. OBJEKT

Diese Relation bestimmt das Werkzeug, mit dem eine Tätigkeit ausgeführt wird.

[PROGRAMMIEREN]→(WKZ)→[COMPILER: Fast C++].

OBJ < UNIVERSAL.

1. TÄTIGKEIT
2. OBJEKT

Diese Relation bestimmt das Objekt, auf dem eine Tätigkeit ausgeführt wird.

[GENEHMIGEN]→(OBJ)→[URLAUBSANTRAG].

ART < UNIVERSAL.

1. TÄTIGKEIT
2. ART

Diese Relation bestimmt die Art und Weise, wie eine Tätigkeit ausgeführt werden kann.

[PROGRAMMIEREN]→(ART)→[GUT].

FÄHIGKEIT < UNIVERSAL.

1. AGENT
2. TÄTIGKEIT

Diese Relation bestimmt eine Fähigkeit eines Agenten. Die Tätigkeit, die die Fähigkeit darstellt, kann mit den obigen Relationen genauer bestimmt werden.

[AKTEUR: Meyer]→(FÄHIGKEIT)→[OO−PROGRAMMIEREN].

BEFUGNIS < UNIVERSAL.

1. AGENT
2. TÄTIGKEIT

Diese Relation bestimmt eine Befugnis eines Agenten. Die Tätigkeit, die die Befugnis darstellt, kann mit den obigen Relationen genauer bestimmt werden.

[AKTEUR: Meyer]→(BEFUGNIS)→[GENEHMIGEN] −
(OBJ)→[URLAUBSANTRAG].

WIRKUNGSBEREICH < UNIVERSAL.

1. TÄTIGKEIT
2. ORGANISATIONSEINHEIT

Diese Relation bestimmt, in welcher Organisationseinheit eine Tätigkeit ausgeführt wird. Der Wirkungsbereich wird meist bei Tätigkeiten angegeben, die eine Befugnis darstellen, um die Befugnis auf eine bestimme Organisationseinheit einzuschränken.

[AKTEUR: Meyer]→(BEFUGNIS)→[GENEHMIGEN] −
(OBJ)→[URLAUBSANTRAG]
(WIRKUNGSBEREICH)→[ABTEILUNG: Einkauf].

ROLLENINHABER < UNIVERSAL.

1. ROLLE
2. AKTEUR

Diese Relation ordnet einer Rolle einen Akteur zu, der die Rolle spielt.

[AKTEUR: Meyer]←(ROLLENINHABER)←[ROLLE: Rolle X].

ENTHALTENE BEFUGNIS < UNIVERSAL.

1. ROLLE
2. TÄTIGKEIT

Diese Relation bestimmt eine in einer Rolle enthaltene Befugnis. Die Tätigkeit, die die Befugnis darstellt, kann mit den obigen Relationen genauer bestimmt werden.

[ROLLE: Abteilungsleiter]→(ENTHALTENE BEFUGNIS)→[GENEHMIGEN] −
(OBJ)→[URLAUBSANTRAG].

ENTHALTENE ROLLE < UNIVERSAL.

1. ROLLE
2. ROLLE

Diese Relation dient der Schachtelung von Rollen, d.h. mit ihr ist es möglich, einer Rolle weitere enthaltene Rollen und somit auch deren Befugnisse zuzuweisen.

[ROLLE: Rolle A]→(ENTHALTENE ROLLE)→[ROLLE: Rolle B].

Anhang C

CORBA-Schnittstellen der Organisationsverwaltung

Dieser Abschitt zeigt die IDL-Schittstellen aller entworfenen Komponenten.

C.1 Wissensbasis

```
module de {
  module unistuttgart {
    module sfb467 {
      module knowledgebase {
        module corba {
          struct Arc {
            string id;
          };

          typedef sequence<Arc> ArcSeq;

          struct Concept {
            string id;
            string type;
            string nameDesignator;
            boolean allQuantifier;
            long x,y;
          };

          typedef sequence<Concept> ConceptSeq;

          struct Relation {
            string id;
            string type;
            ArcSeq incomingArcs;
            ArcSeq outgoingArcs;
            long x,y;
          };
```

```
30          typedef sequence<Relation> RelationSeq;

32          struct Fact {
              ConceptSeq concepts;
34            RelationSeq relations;
            };
36
            struct Rule {
38            ConceptSeq concepts;
              RelationSeq relations;
40            sequence<string> conclusionRelations;
            };
42
            struct KeyFactPair {
44            string key;
              Fact fact;
46          };

48          struct KeyRulePair {
              string key;
50            Rule rule;
            };
52
            struct Type {
54            string label;
              long x,y;
56          };

58          typedef sequence<KeyFactPair> FactSeq;
            typedef sequence<KeyRulePair> RuleSeq;
60          typedef sequence<string> StringSeq;
            typedef sequence<Type> TypeSeq;
62
            exception TypeInUse {};
64          exception IllegalOperation {};
            exception AlreadyPresent {};
66          exception UnknownType {};
            exception UndefinedNode {};
68          exception NotFound {};

70          interface TypeHierarchy {
              // The label of the universal type.
72            readonly attribute string universalTypeLabel;
              // The label of the absurd type.
74            readonly attribute string absurdTypeLabel;

76            /**
               * Adds a type to the hierarchy. The only supertype of
```

```
78              * this type will be the universal type, the only
                * subtype will be the absurd type.
80              * @param newType The type to be added.
                * @throws AlreadyPresent If the type is already
82              *   present in the hierarchy.
                */
84             void addType(in Type newType)
                 raises(AlreadyPresent);
86             /**
                * Changes the values of a type (e.g. the x and y
88              *   coordinates).
                * @param type The new definition of the type.
90              */
               void changeValues(in Type type)
92               raises(UnknownType);
               /**
94              * Adds a supertype to a type.
                * @param subject The label of the type to which a
96              * supertype will be added.
                * @param superType The label of the new supertype.
98              * @throws UnknownType If one of the specified types
                *   is not part of the hierarchy.
100             * @throws IllegalOperation If the operation would
                *   create an invalid type hierarchy.
102             */
               void addSuperType(in string subject,
104                              in string superType)
                 raises(UnknownType, IllegalOperation);
106            /**
                * Adds a subtype to a type.
108             * @param subject The label of the type to which a
                *   subtype will be added.
110             * @param subType The label of the new subtype.
                * @throws UnknownType If one of the specified types
112             *   is not part of the hierarchy.
                * @throws IllegalOperation If the operation would
114             *   create an invalid type hierarchy.
                */
116            void addSubType(in string subject, in string subType)
                 raises(UnknownType, IllegalOperation);
118            /**
                * Removes a type from the hierarchy.
120             * @param typeLabel Label of the type to be removed.
                * @throws TypeInUse If the type is still in use and
122             *   hence can't be removed.
                */
124            void removeType(in string typeLabel)
                 raises(TypeInUse);
126            /**
```

```
        * Gets all immediate subtypes of a type.
128     * @param subject The label of the type from which
        *   the subtypes are to be retrieved.
130     * @return Immediate subtypes.
        * @throws UnknownType If the specified type is not
132     *   part of the hierarchy.
        */
134   TypeSeq getImmediateSubtypes(in string subject)
        raises(UnknownType);
136   /**
        * Gets all immediate supertypes of a type.
138     * @param subject The label of the type from which
        *   the supertypes are to be retrieved.
140     * @return Immediate supertypes.
        * @throws UnknownType If the specified type is not
142     *   part of the hierarchy.
        */
144   TypeSeq getImmediateSupertypes(in string subject)
        raises(UnknownType);
146   /**
        * Gets a type definition.
148     * @param label The type label.
        * @return The type definition.
150     * @throws UnknownType If the specified type is not
        *   part of the hierachy.
152     */
      Type getTypeByLabel(in string label)
154     raises(UnknownType);
    };
156
    interface Knowledgebase {
158   /**
        * Resets the knowledgebase, i.e. deletes all facts
160     * and rules and clears the type hierarchies.
        * @throws TypeInUse If one of the type hierarchies
162     *   couldn't be purged because some types are still
        *   in use by some conceptual graphs.
164     */
      void reset()
166     raises(TypeInUse);
      /**
168     * Stores the knowledgebase. This may be used for
        * efficient storing (e.g. if you use simple files to
170     * store the hierachies, the facts, and/or rules you
        * may not want a disk operation for every change
172     * especially if you make a lot of changes at the
        * same time.).
174     * @param saveConceptTypeHierarchy Save the concept
        *   type hierarchy.
```

```
176              * @param saveRelationTypeHierarchy Save the relation
                 *   type hierarchy.
178              * @param saveFacts Save all facts.
                 * @param saveRules Save all rules.
180              */
                 void store(in boolean saveConceptTypeHierarchy,
182                         in boolean saveRelationTypeHierarchy,
                            in boolean saveFacts,
184                         in boolean saveRules);
                 /**
186              * Gets the concept type hierarchy.
                 * @return The concept type hierarchy.
188              */
                 TypeHierarchy getConceptTypeHierarchy();
190              /**
                 * Gets the relation type hierarchy.
192              * @return The relation type hierarchy.
                 */
194              TypeHierarchy getRelationTypeHierarchy();
                 /**
196              * Adds a rule to the knowledgebase.
                 * @param key Key of the rule.
198              * @param rule The rule to be added.
                 * @param allowOverwrite An already existing rule
200              *   with the same key will be overwritten, if this
                 *   parameter is set to true.
202              * @throws AlreadyPresent If allowOverwrite is set to
                 *   false and a rule with the specified key is
204              *   already present in the knowledgebase.
                 * @throws UnknownType If one of the concepts or
206              *   relations has a type, that is not part of the
                 *   hierarchy.
208              * @throws UndefinedNode If an undefined node is used.
                 */
210              void addRule(in string key, in Rule rule,
                              in boolean allowOverwrite)
212                raises(AlreadyPresent, UnknownType, UndefinedNode);
                 /**
214              * Adds a fact to the knowledgebase.
                 * @param key Key of the fact.
216              * @param fact The fact to be added.
                 * @param allowOverwrite An already existing fact
218              *   with the same key will be overwritten, if this
                 *   parameter is set to true.
220              * @throws AlreadyPresent If allowOverwrite is set
                 *   to false and a fact with the specified key is
222              *   already present in the knowledgebase.
                 * @throws UnknownType If one of the concepts or
224              *   relations has a type, that is not part of the
```

```
             *   hierarchy.
226          * @throws UndefinedNode If an undefined node is used.
             */
228          void addFact(in string key, in Fact fact,
                          in boolean allowOverwrite)
230            raises(AlreadyPresent, UnknownType, UndefinedNode);
             /**
232          * Gets all facts from the knowledgebase.
             * @return All the facts from the knowledgebase.
234          */
             FactSeq getFacts();
236          /**
             * Gets all rules from the knowledgebase.
238          * @return All the rules from the knowledgebase.
             */
240          RuleSeq getRules();
             /**
242          * Get one fact from the knowledgebase.
             * @param key Key of the fact.
244          * @return The requested fact.
             * @throws NotFound If the requested fact could not
246          *   be found in the knowledgebase.
             */
248          Fact getFact(in string key)
               raises(NotFound);
250          /**
             * Gets one rule from the knowledgebase.
252          * @param key Key of the rule.
             * @return The requested rule.
254          * @throws NotFound If the requested rule could not
             *   be found in the knowledgebase.
256          */
             Rule getRule(in string key)
258            raises(NotFound);
             /**
260          * Removes a fact from the knowledgebase.
             * @param key Key of the fact.
262          */
             void removeFact(in string key);
264          /**
             * Removes a rule from the knowledgebase.
266          * @param key Key of the rule.
             */
268          void removeRule(in string key);
             /**
270          * Gets all identifiers (keys) of the rules.
             * @return Rule keys.
272          */
             StringSeq getRuleKeys();
```

```
274             /**
                 * Gets all identifiers (keys) of the facts.
276              * @return Fact keys.
                 */
278             StringSeq getFactKeys();
              };
280
              interface CompiledKnowledgebase : Knowledgebase {
282             /**
                 * Compiles the knowledgebase.
284              */
                void compileKnowledgebase();
286             /**
                 * Returns the compiled facts.
288              * @return The compiled facts.
                 */
290             FactSeq getCompiledFacts();
              };
292         };
        };
294   };
  };
296 };
```

C.2 Querybase

```
 module de {
2  module unistuttgart {
     module sfb467 {
4      module querybase {
         module corba {
6          struct Arc {
             string id;
8          };

10         typedef sequence<Arc> ArcSeq;

12         struct Concept {
             string id;
14           string type;
             string nameDesignator;
16           boolean allQuantifier;
             long x,y;
18         };

20         typedef sequence<Concept> ConceptSeq;

22         struct Relation {
             string id;
```

```
24            string type;
              ArcSeq incomingArcs;
26            ArcSeq outgoingArcs;
              long x,y;
28          };

30          typedef sequence<Relation> RelationSeq;

32          struct Query {
              string idQueryConcept;
34            ConceptSeq concepts;
              RelationSeq relations;
36          };

38          struct KeyQueryPair {
              string key;
40            Query query;
            };
42
            struct Type {
44            string label;
              long x,y;
46          };

48          typedef sequence<KeyQueryPair> QuerySeq;
            typedef sequence<string> StringSeq;
50          typedef sequence<Type> TypeSeq;

52          exception AlreadyPresent {};
            exception UnknownType {};
54          exception UndefinedNode {};
            exception IllegalQuery {};
56          exception NotFound {};

58          interface QueryBase {
              /**
60             * Resets the query base, i.e. deletes all queries.
               */
62            void reset();
              /**
64             * Stores the query base. This may be used for
               * efficient storing (e.g. if you use simple files
66             * to store the queries you may not want a disk
               * operation for every change especially if you make
68             * a lot of changes at the same time.).
               */
70            void store();
              /**
72             * Adds a query to the query base.
```

```
                * @param key Key of the query.
74              * @param query The query to be added.
                * @param allowOverwrite An already existing query
76              *   with the same key will be overwritten, if this
                *   parameter is set to true.
78              * @throws AlreadyPresent If allowOverwrite is set
                *   to false and a rule with the specified key is
80              *   already present in the knowledgebase.
                * @throws UnknownType If one of the concepts or
82              *   relations has a type, that is not part of the
                *   hierarchy.
84              * @throws UndefinedNode If an undefined node is used.
                * @throws IllegalQuery If this is no legal query,
86              *   e.g. if the field idQueryConcept denotes
                *   a concept that is not part of the query graph.
88              */
               void addQuery(in string key, in Query query,
90                             in boolean allowOverwrite)
                 raises(AlreadyPresent, UnknownType, UndefinedNode,
92                       IllegalQuery);
               /**
94              * Gets all queries from the query base.
                * @return All the queries from the query base.
96              */
               QuerySeq getQueries();
98             /**
                * Get one query from the query base.
100             * @param key Key of the query.
                * @return The requested query.
102             * @throws NotFound If the requested query could not
                *   be found in the query base.
104             */
               Query getQuery(in string key)
106              raises(NotFound);
               /**
108             * Removes a query from the query base.
                * @param key Key of the query.
110             */
               void removeQuery(in string key);
112            /**
                * Gets all identifiers (keys) of the queries.
114             * @return Query keys.
                */
116            StringSeq getQueryKeys();
             };
118        };
       };
120  };
};
```

```
122 };
```

C.3 Taskholder-Finder

```
module de {
  module unistuttgart {
    module sfb467 {
      module taskholderfinder {
        module corba {
          struct Variable {
            string name;
            string value;
          };

          struct QueryResultElement {
            string taskholderID;
            sequence<Variable> variables;
            long relevance;
          };

          typedef sequence<QueryResultElement> QueryResultList;

          struct QueryParameterElement {
            string key;
            string value;
          };

          typedef sequence<QueryParameterElement>
            QueryParameterList;

          exception TaskholderFinderException {};
          exception NoSuchQuery {};
          exception MissingParameter {};

          interface TaskholderFinder {
            /*
             * Searches for taskholders.
             * @param query id Identifier of a query.
             * @param query Parameters Parameters for the query.
             * @throws NoSuchQuery if the specified query can't
             *   be found.
             * @throws TaskholderFinderException if a general
             *   exception occurs.
             * @throws MissingParameter if a query parameter is
             *   missing.
            QueryResultList query(in string queryId,
                                  in QueryParameterList
                                     queryParameters)
              raises (NoSuchQuery, TaskholderFinderException,
```

```
46                                  MissingParameter);
                  };
48              };
            };
50        };
      };
52 };
```

Anhang D

Meßergebnisse der Laufzeittests der Matching-Algorithmen

Laufzeittest 1 (Tab. D.1) wurde mit einem Anfragegraphen mit 3 generischen und 7 individuellen Konzepten und 9 Relationen durchgeführt. Alle Zeitangaben sind in Millisekunden.

Anz. Kon. Fakt	Anz. Rel. Fakt	$t_{Mugnier}$	$t_{Ullmann}$	t_{Koch}	t_{PROLOG}	t_{Notio}
100	110	3	8	16	97	8
250	277	5	10	20	99	16
450	467	4	17	30	89	50
500	537	8	17	20	105	238
600	643	9	22	38	83	277
800	847	9	28	45	102	293
900	1018	9	32	81	88	1100
950	1012	11	31	110	108	1432
1000	1085	8	33	40	103	91
1100	1170	11	36	70	103	1191
2000	2159	13	70	133	124	556
2250	2510	11	78	145	133	999
2500	2700	13	84	177	141	1165
2750	2912	23	90	179	147	1351
3000	3203	16	113	195	195	2243
3250	3459	18	116	276	182	86803
3500	3866	18	124	221	188	2227
3750	4092	34	149	287	205	3124
4000	4347	36	141	362	199	183896

Tabelle D.1: Meßergebnisse des Laufzeittests 1

Laufzeittest 2 (Tab. D.2) wurde mit einem Anfragegraphen mit 10 generischen Konzepten und 9 Relationen durchgeführt. Alle Zeitangaben sind in Millisekunden. Alle mit „–“ gekennzeichneten Tests wurden aufgrund zu langer Laufzeit nicht durchgeführt.

Anz. Kon. Fakt	Anz. Rel. Fakt	$t_{Mugnier}$	$t_{Ullmann}$	t_{Koch}	t_{PROLOG}	t_{Notio}
50	69	5	4	20	83	4
100	113	3	8	19	64	56
110	121	3	8	22	64	438
115	124	3	7	20	91	2541
118	132	3	9	69	66	63455
150	159	4	9	27	68	–
200	206	6	11	22	66	–
215	226	10	12	28	69	–
225	238	5	13	24	68	–
250	266	7	16	58	73	–
275	286	9	18	125	82	–
300	315	8	18	70	81	–
350	375	9	24	123	75	–
400	418	13	31	113	75	–
500	529	9	30	180	92	–
600	635	7	26	146	106	–
700	708	11	34	229	85	–
800	869	14	71	460	84	–
900	1013	13	90	388	91	–
1000	1078	13	85	618	135	–
1200	1299	15	87	647	178	–
1400	1534	20	205	1330	146	–
1600	1727	21	145	1307	120	–
1700	1879	23	230	1578	118	–
2000	2244	24	217	1769	127	–
2500	2838	32	544	4075	509	–
3000	3241	33	680	3683	166	–

Tabelle D.2: Meßergebnisse des Laufzeittests 2

Erklärung

Hiermit versichere ich, diese Arbeit
selbständig verfaßt und nur die
angegebenen Quellen benutzt zu haben.

(Frank Dürr)