Architektur von Fuzzy-Informationssystemen

zur Repräsentation und Verarbeitung unscharfer Daten

Zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften
von der Fakultät für Informatik
der Universität Karlsruhe
genehmigte Dissertation
von
René Witte aus Kassel

Universität Karlsruhe (TH)
Fakultät für Informatik
Institut für Programmstrukturen und Datenorganisation
Am Fasanengarten 5
76128 Karlsruhe

Tag der mündlichen Prüfung: 15. Juli 2002
Erster Gutachter: Prof. Dr. P. C. Lockemann
Zweiter Gutachter: Prof. Dr. W. Menzel

Diese Dissertation kann elektronisch (http://rene-witte.net) sowie in gedruckter Form als Book on Demand (ISBN 3-8311-4149-5) bezogen werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Danksagung

We dangle our three magic letters before the eyes of these predestined victims,
and they swarm to us like moths to an electric light.
They come at a time of life when failure can no longer be repaired easily
and when the wounds it leaves are permanent...
William James, "The Ph.D. Octopus", Harvard Monthly, March 1903

„Wer sind nur diese ganzen Leute? Und warum soll ich das hier lesen?" wird sich schon jeder gefragt haben, der mal ein Buch mit einer Danksagung in den Händen gehalten hat. Warum müssen Autoren immer völlig unbekannten Leuten danken, die außer ihnen offenbar niemand kennt?

Weil es dieses Buch sonst nicht gäbe. Oder zumindest nicht in der vorliegenden Form: Denn ohne die inhaltliche, moralische, oder organisatorische Unterstützung von Freunden und Kollegen läßt sich ein solches Vorhaben nicht bewältigen.

An erster Stelle möchte ich mich hier bei *Dr. Gerd Hillebrand* bedanken, der dieser Arbeit vor allem in ihrer Anfangszeit wesentliche Impulse gab, aber auch im weiteren Verlauf mit seinen Ideen, Anmerkungen und Kritiken immer wieder zur Verblüffung des Autors beitrug.

Bei der Fülle der angesprochenen Themen, insbesondere der Entwicklung der Anwendungsbeispiele, war ich oft auf kompetenten Rat und Hilfe von außen angewiesen. Bei *Wassilios Kazakos* möchte ich mich hier für seine XML-Beratung bedanken. Ohne die unzähligen Teerunden mit *Dr. Ulrike Kölsch* gäbe es die vorgestellten Ideen zum Thema Fuzzy-Software-Reengineering nicht. Und *Prof. Dr. Michael Philippsen* danke ich für seine Hinweise zum vorgestellten Annotations-Entwurfsmuster. Ein besonderer Dank geht an *Prof. Dr. Sabine Bergler* von der Concordia University in Montréal, Kanada, für die vielfältige Unterstützung bei der Entwicklung des Fuzzy-Textanalysebeispieles.

Meinen (ehemaligen) Studenten *Alexander Christoph* und *Hans-Peter Lang* verdanke ich große Teile der Implementierung der vorgestellten Ideen, die sie in ihren Studienarbeiten und während ihrer Arbeit als wissenschaftliche Hilfskräfte entwickelt haben.

Eine unschätzbare Hilfe waren darüber hinaus meine Korrekturleser, die sich die Mühe gemacht haben, eine komplette Vorabversion dieser Arbeit durchzusehen und den heutigen Leser so vor vielen Fehlern, Ungereimtheiten und Unverständlichkeiten bewahrt haben.[1] Mein Dank geht hier an *Dr. Stefan Hänßgen*, *Dr. Ulrike Kölsch*, *Jürgen Laschewski* und *Lars Wetzel*.

Falls Sie die Buchversion dieser Arbeit in Händen halten, können Sie sich zusätzlich an dem von *Tillmann Lübker* entworfenen Umschlag erfreuen, wofür ich mich herzlich bei ihm bedanken möchte.

Das vorliegende Buch ist gleichzeitig meine Dissertation, was ihm einerseits einen Rahmen verleiht, es andererseits aber auch gewissen Erwartungen aussetzt. Meinem Korreferenten, *Prof. Dr. W. Menzel*, möchte ich daher für seine Aufgeschlossenheit, Neugier und Sorgfältigkeit danken, mit der er sich dieser Arbeit angenommen hat. Viele Korrekturen und Präzisierungen, insbesondere des theoretischen Teils, gehen auf seine Anregungen zurück.

Entstanden wäre diese Arbeit aber nicht ohne die langjährige Unterstützung meines Doktorvaters, *Prof. Dr. P. C. Lockemann*. Er hat mich während der gesamten Entstehungsgeschichte dieser Arbeit unterstützt und mir dabei die Möglichkeit gegeben, meinen eigenen Weg zu gehen, wofür ich ihm aufrichtig dankbar bin.

[1] Tatsächlich war dieses Buch bereits vollkommen fehlerfrei, aber da eine solche Singularität zu viel Aufsehen erregt hätte, wurden nachträglich wieder einige Fehler eingefügt.

Teil I

Einführung
Fuzzy-Informationssysteme

Kapitel 1

Einleitung

Facts are stupid things.
Ronald Reagan

Informationssysteme spielen eine zunehmend wichtigere Rolle in allen Bereichen der Gesellschaft. Man spricht daher immer häufiger von dem Übergang in eine Informationsgesellschaft, in der *Information* zu einem eigenständigen Produktionsfaktor und somit Wettbewerbsvorteil wird.

Die Strukturierung und Verwaltung dieser Informationen obliegt dabei dem Bereich der Datenbank- und Informationssystemtechnologie. Ein wichtiges Teilgebiet davon ist die Modellierung eines Ausschnittes aus der realen Welt in Form einer sogenannten Miniwelt und deren nachfolgende Abbildung auf verschiedene Datenmodelle, die sich zur Anwendungsentwicklung und persistenten Speicherung der Informationen in einer Datenbank eignen.

Die Ausdruckskraft dieser Datenmodelle, wie dem relationalen oder objektorientierten Modell, blieb in den vergangenen zwanzig Jahren praktisch unverändert. Fortschritte wurden im wesentlichen auf dem Gebiet der Implementierung von Datenbanksystemen, sowie den Technologien zur Entwicklung von darauf aufbauenden Informationssystemen erzielt.

Durch die Erschließung neuer Anwendungsbereiche ergeben sich aber auch neue Anforderungen an Informationssysteme, die mit den etablierten Technologien nur bedingt befriedigt werden können. Neben den klassischen, einfach strukturierten Datensätzen der ersten Generation, die hauptsächlich früher manuell geführte Informationen digitalisierten, stehen heute neue Anwendungsbereiche, die versuchen, komplexere Informationen wie die Erfahrungen von Mitarbeitern, Expertenwissen, Faustregeln oder Vermutungen zu verwalten. Dazu gehören Informationssysteme, die komplexe, dynamische Sachverhalte abbilden müssen, für die keine exakte Beschreibung mehr möglich ist. Gerade im zur Zeit explosionsartig wachsenden Be-

reich der Internet-basierten Informationssysteme ist es unabdingbar, innerhalb eines Systems mit einer Vielzahl von unvollständigen, inkonsistenten und daher oft widersprüchlichen Informationen aus unterschiedlichsten Quellen umgehen zu können.

Die klassischen Informationssystemtechnologien sind mit der Modellierung und Speicherung solcher sogenannter *imperfekter Daten* überfordert. Als Folge verlagert sich der Aufwand zur Repräsentation und Verarbeitung dieser Informationen vom Bereich der Basistechnologie in die Anwendungsentwicklung, wo für eine spezifische Domäne spezielle Lösungen erarbeitet und implementiert werden müssen. Diese Situation erweist sich jedoch als sehr unbefriedigend, da so, wie in der vor-Datenbankzeit, die Datenmodellierung in die Anwendungen getragen wird, mit allen damit verbundenen Nachteilen. Als Folge steigt die Komplexität der Anwendungsentwicklung, was zu höheren Kosten, längerer Entwicklungszeit und mangelnder Robustheit führt. In vielen Fällen schließt dies die Durchführung eines solchen Projektes sogar von vorneherein aus.

Ursächlich für diese Problematik ist die unzureichende Ausdruckskraft klassischer Datenmodelle, wenn es um Informationen geht, die nur vage oder mit unscharfen Begriffen beschrieben werden können, die nur teilweise bekannt oder inkonsistent zueinander sind.

Im Bereich der Datenbanksysteme hat man schon früh erkannt, daß man einen Mechanismus braucht, mit dem man mit unvollständigen Informationen umgehen kann. Als Lösung wurden sogenannte Nullwerte eingeführt, die zur Offenhaltung von Attributwerten dienen, die nicht exakt benannt werden können. Obwohl bei weitem nicht ausreichend für komplexere Anforderungen, hat sich bis heute kein weiterführender Ansatz im Datenbankbereich etablieren können.

In den letzten Jahren gewann jedoch ein weiterer Ansatz verstärkt die Aufmerksamkeit der Datenbankforscher: die von Lotfi Zadeh bereits 1965 vorgeschlagene Theorie der unscharfen Mengen, die sogenannte *Fuzzy-Theorie*. Eine Eigenschaft dieser Theorie ist die Möglichkeit, mit ihrer Hilfe Informationen, wie sie oben geschildert wurden, in einer für Computer geeigneten Weise zu repräsentieren und zu verarbeiten.

Fast noch interessanter sind jedoch die Erfolge, die mit Hilfe dieser Theorie im Bereich der Steuerungs- und Regelungstechnik erzielt wurden. Beeindruckend an den ersten fuzzy-basierten Regelungssystemen war nicht nur, daß mit ihrer Hilfe eine schwierige Aufgabe gelöst werden konnte, sondern vor allem wie *schnell* sie gelöst wurde und um wieviel *einfacher* und *robuster* die Lösung gegenüber einem vergleichbaren klassischen System war. Innerhalb kürzester Zeit hat sich das sogenannte *Fuzzy Control* vom exotischen Forschungsgebiet zum etablierten, industriell eingesetzten Verfahren gewandelt.

Der große kommerzielle Erfolg des Fuzzy Control hat schon früh das Interesse der Forschergemeinde an dem Gebiet der unscharfen Daten geweckt. Die damit verbundene Hoffnung der Datenbankforscher, im Bereich der Fuzzy-Informationssysteme eine ähnlich erfolgreiche Entwicklung durchlaufen zu können, hat sich jedoch bis

heute nicht erfüllt. Eine genaue Analyse der Ursachen, verbunden mit der Herleitung unseres eigenen Ansatzes, erfolgt in Kapitel 3.

1.1 Fuzzy Systeme

Die Idee, innerhalb eines Systems mit imperfektem Wissen umzugehen, hat sich über den Bereich der Fuzzy-Regelungstechnik hinaus verbreitet. Zadeh formulierte den Begriff des *soft computing*, die Berücksichtigung von Unschärfe und Unsicherheit in komplexen Systemen, folgendermaßen:

> *The real world is pervasively imprecise and uncertain. Precision and certainty carry a cost. The guiding principle of soft computing is: Exploit the tolerance for imprecision, uncertainty, and partial truth to achieve tractability, robustness, and low solution cost.*

Während die traditionelle Sichtweise Unsicherheiten und Inkonsistenzen als etwas Unerwünschtes ansieht, das es möglichst zu vermeiden gilt, porträtiert sie die Idee des Unscharfen Rechnens als eine neue Herausforderung an die Systementwicklung. Gelingt es nämlich, existierende Technologien in natürlicher Weise für den Umgang mit Imperfektion zu erweitern, löst man nicht nur die bestehenden Probleme beim Umgang mit solchen imperfekten Daten — man gewinnt zusätzlich handfeste Vorteile, wie kürzere Entwicklungszeiten oder ein robusteres Endsystem.

Woher sollen nun diese Vorteile kommen? Die klassische Vorgehensweise ist, auftretende ungenaue Sachverhalte durch immer weitere Präzisierung, notfalls unter Ausschluß bestimmter Daten oder Einführung künstlicher Strukturen, auszuschließen. Doch diese Methode stößt schnell an ihre Grenzen. Ein Beispiel für ein Informationssystem, das durch eine Steigerung der Genauigkeit keine besseren Ergebnisse liefern kann, ist die Verwaltung von Zeugenaussagen, die beispielsweise eine Person oder ein Fahrzeug beschreiben. Die zur Verfügung stehenden Informationen bleiben unscharf und möglicherweise inkonsistent, egal wie genau man sie abbildet. Ein System, das in der Lage ist solche unscharfen Informationen zu repräsentieren und trotz auftretender Inkonsistenzen zu verarbeiten, wird bessere, auch exaktere Ergebnisse liefern als ein klassisches System, das versuchen muß, eine nichtvorhandene Präzision abzubilden.

Das Phänomen der imperfekten Daten ist keineswegs auf Randbereiche beschränkt. Moderne Anwendungen werden immer komplexer und müssen mit immer größeren Datenmengen umgehen; dabei ist es nicht mehr möglich, diese kostspielig aufzubereiten, wie im Fall der Auswertung umfangreicher und diverser Daten innerhalb eines Internet-basierten Informationssystems.

Tatsächlich ist damit zu rechnen, daß Systeme in der Zukunft immer öfter mit solchen Unvollkommenheiten konfrontiert werden. Dies begründet sich in einem Zusammenhang zwischen der Komplexität eines Systemes, seiner Präzision und der

Relevanz erzeugter Ergebnisse. Zadeh formulierte diese Wechselwirkung in seinem *Inkompatibilitätsprinzip* (principle of incompatibility):

> *Stated informally, the essence of this principle is that as the complexity of a system increases, our ability to make precise and yet significant statements about its behaviour diminishes until a threshold is reached beyond which precision and significance (or relevance) become almost mutually exclusive characteristics.*

Duboi und Prade formulierten das Inkompatibilitätsprinzip pointiert:

> *The more, the fuzzier.*

Mit der zunehmenden Komplexität von Informationssystemen und deren wachsender gesellschaftlicher und wirtschaftlicher Bedeutung geht auch ein wiedererstarktes Interesse an der Entwicklung tragfähiger, praktisch einsetzbarer Verfahren zum Umgang mit Unsicherheit einher.

Dieses Interesse bildet die Grundlage für das Thema und die Ausrichtung dieser Arbeit.

1.2 Fuzzy-Informationssysteme

Trotz dieser vielversprechenden Möglichkeiten und der potentiell großen Bedeutung von Fuzzy-Anwendungen zeigt sich das Gebiet der Datenbank- und Informationssystemtechnologie schlecht auf die neuen Anforderungen vorbereitet.

Betrachten wir zunächst den Stand der Technik bei der Entwicklung klassischer Informationssysteme. Hier hat sich durch die wachsende Komplexität und die rapide Weiterentwicklung verfügbarer Technologien eine Reihe von Referenzarchitekturen herausgebildet, die die Entwicklung neuer Anwendungen erst wieder handhabbar macht.

Ausgangsbasis für die Entwicklung moderner Systeme sind dabei mehrstufige Client/Server-Architekturen, die sowohl eine Aufgaben-, als auch eine Lastverteilung ermöglichen (siehe Abbildung 1.1).

Damit ein lauffähiges System implementiert werden kann, muß eine solche Architektur mit Technologien gefüllt werden, die eine Realisierung ermöglichen: Programmiersprachen, Bibliotheken, Entwicklungswerkzeuge, Datenbankmanagementsysteme, Ablaufumgebungen. Da man nicht hoffen kann, alle möglichen Anwendungsfälle innerhalb eines Unternehmens oder einer Forschungseinrichtung mit einer einzigen Architektur abzudecken, entwickeln sowohl die Anbieter als auch die Anwender solcher Technologien typischerweise sogenannte *Referenzarchitekturen* (Blueprints), die eine Sammlung von vorgeschriebenen, optionalen und alternativen Architekturbestandteilen enthalten, mittels derer dann die Architektur einer konkreten

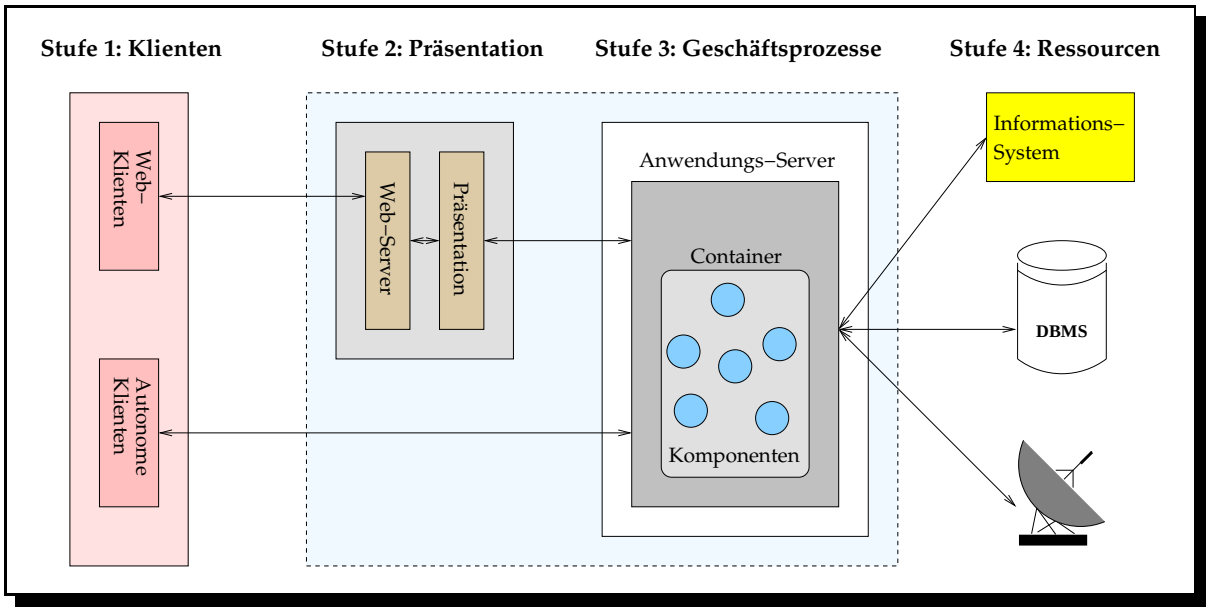


Abbildung 1.1: Mehrstufige Client/Server-Architektur

Anwendung anhand ihrer Bedürfnisse abgeleitet werden kann. Abbildung 1.2 zeigt als Beispiel für eine Referenzarchitektur die von der Firma Sun vorgeschlagene *Java 2 Enterprise Edition (J2EE)*.

Dies gibt den Rahmen vor, in dem die Entwicklung von Fuzzy-Informationssystemen stattfinden muß, denn schließlich wird für die zusätzlichen Möglichkeiten, die die Fuzzy-Technologie bietet, niemand auf etablierte Leistungsmerkmale verzichten wollen.

Doch was hat die Forschung momentan auf dem Gebiet der Fuzzy-Informationssysteme anzubieten?

Bereits früh gab es Vorschläge zur Erweiterung des relationalen Datenmodells um Fuzzy-Mengen, die die Modellierung vager Informationen ermöglichen sollten. Die meisten Arbeiten blieben jedoch theoretisch ausgerichtet und wurden nie für die Entwicklung von Anwendungen eingesetzt. Selbst Modelle, die prototypisch implementiert wurden, haben bis heute nicht Einzug in eines der verfügbaren kommerziellen oder freien Datenbankmanagementsysteme halten können. Angewiesen auf die hohe Stabilität und Leistungsfähigkeit professioneller Datenbanksysteme, sind diese Ergebnisse für den praktischen Einsatz verloren.

Allmählich hat sich zwar in den letzten Jahren ein Schwenk in der Forschungsrichtung hin zu den objektorientierten Technologien vollzogen, die die Grundlage für die Realisierung der meisten modernen Informationssysteme darstellen [YG99]:

> *Object-Oriented Database Management Systems (OODBMS) have been developed to meet the complex data modeling requirements of large scale, data intensive*

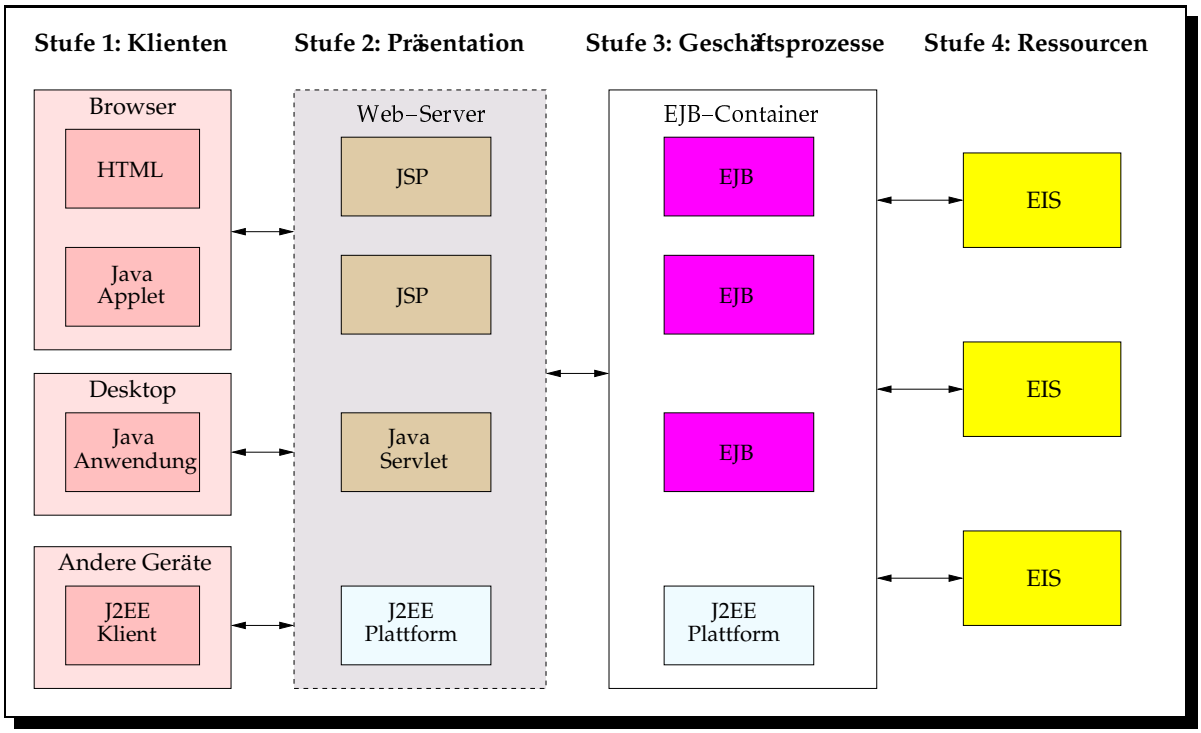


Abbildung 1.2: Beispiel für eine Referenzarchitektur: Java 2 Enterprise Edition (J2EE)

> *applications, such as Office Automation Systems, CAD/CAM, Geographic Information Systems, Multimedia Database Systems. Despite the representational power of the object-oriented paradigm, OODBMS are still ill equipped in dealing with inherently vague, uncertain or imprecise data. However applications require the manipulation and reasoning based on incomplete and imprecise data.*

Doch auch hier bietet sich ein sehr uneinheitliches Bild. Die meisten Arbeiten betrachten nur einzelne theoretische Aspekte der Einbettung von Fuzzy-Mengen in das objektorientierte Datenmodell. Der nächste Schritt, der praktische Einsatz innerhalb einer Anwendung, wird aber fast nirgendwo mehr vollzogen. Motro und Smets [MS97] umschreiben dies diplomatisch:

> *Although there are researchers in information systems who have addressed themselves to issues of uncertainty, as well as researchers in uncertainty modeling who have considered the pragmatic demands and constraints of information systems, to a large extent there has been only limited interaction between these two areas.*

Insgesamt zeigt sich also ein düsteres Bild vom aktuellen Stand der Fuzzy-Informationssysteme: es dominieren isolierte Ansätze, die nur Teilaspekte betrachten und zudem meist keine praktische Umsetzung aufweisen können. Auf die oben beschriebenen Anforderungen moderner Informationssysteme geht kein einziger Ansatz ein, und das obwohl diese Art von Systemen oft explizit als Einsatzgebiet skizziert wird.

1.3 Lösungsansatz

Diese Arbeit definiert *Fuzzy-Informationssysteme* als eine neue, eigenständige Klasse von Systemen und stellt deren Möglichkeiten und Anforderungen in den Mittelpunkt. Damit unterscheiden wir uns grundsätzlich von der Herangehensweise anderer Ansätze: dort stehen nicht die *Anwendungen* im Vordergrund, sondern die theoretischen Erweiterungen verschiedener Modelle, deren Einsatzfähigkeit dann aber fragwürdig bleibt und die somit nur mehr einen Selbstzweck erfüllen.

Wir bieten daher eine neue Sichtweise auf das Gebiet der Fuzzy-Informationssysteme an: nämlich die eines Systementwicklers, der mittels einer Menge verfügbarer Technologien einsatzfähige Anwendungen realisieren möchte.

Die Ausgangsbasis bilden aktuelle Technologien, wie sie heute zur Entwicklung von Informationssystemen eingesetzt werden. Ausgehend von dieser Basis werden die Anforderungen analysiert, die sich bei der Entwicklung von Fuzzy-Informationssystemen stellen, und dazu passende Konzepte entwickelt, die die bestehenden Technologien sukzessive erweitern.

Um diese Anforderungen zu ermitteln, untersuchen wir im nächsten Kapitel bestehende Ansätze auf ihre Einsatzfähigkeit zur Entwicklung von Fuzzy-Informationssystemen. Anhand der ermittelten Schwächen und Einschränkungen werden dann in Kapitel 3 die Anforderungen hergeleitet, die ein Architekturmodell für Fuzzy-Informationssysteme erfüllen muß.

Es zeigt sich, daß Arbeiten auf verschiedenen Gebieten der Theorie und Praxis von Informationssystemen nötig sind, um den Anforderungen gerecht zu werden. Dies spiegelt sich in den einzelnen Teilen dieser Arbeit wider:

Theorie. Die theoretischen Grundlagen von Fuzzy-Informationssystemen werden in Hinblick auf die praktischen Anforderungen entwickelt. Wir zeigen, daß insbesondere im Bereich der *Verarbeitung* unscharfer Daten noch Lücken existieren und erweitern ein auf der Wissensrevision basierendes Verfahren, um die konsistenzerhaltende Verarbeitung imperfekter Informationen zu ermöglichen.

Technologie. Zur Entwicklung lauffähiger Systeme ist die Abbildung der theoretischen Konzepte auf konkrete Technologien notwendig. Wir entwickeln hier die praktische Umsetzung für eine objektorientierte Umgebung, die von aktuellen Technologien wie der Programmiersprache *Java* Gebrauch macht.

Architektur. Wir zeigen, wie die neuen Fuzzy-Technologien in komplexe Systemarchitekturen eingebettet werden können. Dazu gehört insbesondere die Kommunikation und der Datenaustausch mit existierenden nicht-fuzzy Systemen.

Einsatz. Um die Leistungsfähigkeit unseres Ansatzes zu demonstrieren und verschiedene Einsatzmöglichkeiten aufzuzeigen, schließen sich zwei Fallstudien an, die mit den hier entwickelten Konzepten realisiert wurden.

Diese Arbeit bietet damit als erste ihrer Art eine konsequente, durchgängige Vorgehensweise zur Entwicklung von Fuzzy-Informationssystemen an. Sie richtet sich damit ausdrücklich auch an den Praktiker, indem sie angefangen von den theoretischen Grundlagen bis hin zur technischen Implementierung alle notwendigen Aspekte für den Einsatz in realen Systemen betrachtet.

Kapitel 2

Stand der Forschung

"There must be some way out of here," said the joker to the thief,
"There's too much confusion, I can't get no relief."
Bob Dylan, "All Along The Watchtower"

In diesem Abschnitt betrachten wir vorhandene Arbeiten, die für das gesteckte Ziel, die Entwicklung von Fuzzy-Informationssystemen, relevant sind. Im wesentlichen konzentrieren sich diese Arbeiten auf zwei Gebiete: die Definition von Fuzzy-Datenmodellen für Fuzzy-Datenbanksysteme, sowie die Beschreibung von möglichen oder realisierten Fuzzy-Anwendungen.

Um die Literaturauswahl zu strukturieren, wählen wir einen kritischen Ansatz, der die existierenden Arbeiten anhand ihrer Eignung zur Entwicklung von Fuzzy-Informationssystemen bewertet.

Aus Gründen der Übersichtlichkeit enthält dieser Abschnitt keine Referenzen für Grundlagenliteratur, wie für die Definition einer Fuzzy-Menge. Diese erscheinen im weiteren Verlauf der Arbeit direkt bei ihrer Verwendung.

2.1 Kritische Bewertung vorhandener Ansätze

Bereits für ein relatives junges und exotisches Gebiet wie dem der Fuzzy-Datenbanken existiert eine reichhaltige Literatur. In jüngster Zeit erscheinen daher zunehmend Monographien, die zum einen thematisch verwandte Ansätze zusammenfassen, zum anderen aber auch kritisch den Stand und die Zukunft dieses Forschungsgebietes hinterfragen.

Diesen Kritikpunkten werden wir hier nachgehen, um vorhandene Ansätze besser bewerten und einordnen zu können. Im Vordergrund steht dabei immer die in Kapitel 1 beschriebene Entwicklung von Fuzzy-Informationssystemen.

Doch gerade bei der Anwendungsentwicklung existieren offenbar noch große Lücken, wie in [YG99] festgestellt wird:

> *Two decades have passed since the definition of the first data models to handle uncertainty. Despite the increased computational power of current systems, and the large output in fuzzy database research, the penetration of this technology into the market has been slow. This is in sharp contrast to the theory of fuzzy controls where this technology has been rapidly adopted by industry.*

Was aber kann die Ursache dafür sein, daß die potentiell vielversprechenden Fuzzy-Technologien in Bereich der Informationssysteme bisher keine Anwendung gefunden haben?

> *A reason for this has been that the fuzzy database community has concentrated on the development of the theoretical aspects of the technology. This has stunted its adoption in a marketplace primarily concerned with performance. There are two solutions to this problem: 1. The development of data structures optimized for the retrieval of fuzzy data. 2. The application of fuzzy databases to emerging applications in geographical, multimedia, and environmental information systems, data mining and Internet applications.*

Der zitierte Band schlägt neue physische Datenmodelle zur Lösung des ersten Problems vor. Zum zweiten Punkt jedoch wird angemerkt:

> *The research community has hitherto addressed the second issue only in a piece meal fashion. This has not lead to a consistent body of work that clearly demonstrates the advantage of this approach to the database (and application) community at large.*

Im folgenden gehen wir auf einzelne Ansätze ein und überprüfen jeweils, ob die Kritik hinsichtlich der Anwendungsentwicklung berechtigt ist.

2.2 Fuzzy-Datenmodelle und Fuzzy-Datenbanksysteme

Die bereits ältere Monographie [BK95] macht den Fokus früherer Arbeiten auf dem Gebiet der Fuzzy-Datenbanken deutlich. Die Aufsätze teilen sich auf zwei Gebiete auf: die Erweiterung des relationalen Datenmodells um Fuzzy-Mengen, sowie die Erweiterung von Anfragesprachen um Fuzzy-Queries. Zusätzlich wird die Kombination aus beiden Fällen betrachtet. Praktische Anwendungen spielen hier noch keine Rolle, Implementierungen finden nur in Form experimenteller Erweiterungen bestehender Systeme statt.

Auch [Pet96] ist noch auf relationale Modelle ausgerichtet. Zusätzlich werden aber kurz erste Ideen zu objektorientierten Fuzzy-Modellen vorgestellt und mit einem Fuzzy-Query-System für eine relationale Datenbank eine praktische Anwendung.

In den letzten Jahren rücken vermehrt die objektorientierten Datenmodelle in den Vordergrund [Cal97, YG99]. Die Spanne der Arbeiten reicht dabei von der konzeptionellen Modellierung bis hin zu logischen und physischen Datenmodellen.

2.2.1 Konzeptionelle Fuzzy-Modellierung

Da viele Arbeiten die Erweiterung des relationalen Modells um Fuzzy-Mengen betrachtet haben, lag auch eine entsprechende Erweiterung des verbreiteten ER-Modells *(entity-relationship model)* zur konzeptionellen Modellierung nahe. Ein Ergebnis ist FERN *(fuzzy entity-relationship methodology)* [CMR99]. Zusätzlich werden Regeln zur Transformation des Fuzzy-ER-Modells in ein logisches Datenbankschema angegeben. Im einfachsten Fall werden dabei Relationen um ein Attribut zur Speicherung des Fuzzy-Grades erweitert.

In [YG99] wird das IFO Datenmodell um Konzepte zur Repräsentation von Unsicherheit erweitert. Erlaubt wird Unsicherheit in Form von Nullwerten, Alternativwerten und Fuzzy-Werten. Auftreten können diese unsicheren Werte auf Attributebene und bei der Zugehörigkeit von Objekten zu ihren Klassen. Dieses erweiterte ExIFO beziehungsweise ExIFO_2 Modell wird dann mit Hilfe von Transformationsregeln in entweder ein NF^2 oder ein objektorientiertes logisches Schema überführt.

FOOM ist eine um Fuzzy-Mengen erweiterte, auf UML basierende Modellierungstechnik zur Beschreibung unscharfer Anforderungen [LX98]. Attribute können hier Fuzzy-Werte annehmen, Klassenbeziehungen lassen sich mit einem Fuzzy-Wert annotieren und Objekte können mit einem bestimmten Grad zu einer Klasse gehören.

Bewertung Die konzeptionelle Datenmodellierung ist ein wichtiger erster Schritt bei der Entwicklung eines Informationssystems. Die vorgestellten Ansätze ermöglichen dies zwar in unterschiedlichem Maße auch für Fuzzy-Informationssysteme, benutzen dafür jedoch, mit Ausnahme von FOOM, jeweils verschiedene, nicht-standardisierte Modellierungssprachen. Für objektorientierte Systeme ist dies heute die Unified Modeling Language (UML), die explizit Erweiterungsmöglichkeiten anbietet. FOOM verwendet als Grundlage zwar UML, bietet jedoch nur rudimentäre Erweiterungen und geht nicht über die rein graphische Darstellung hinaus; insbesondere wird keine Möglichkeit angeboten, das UML-Modell in eine Implementierung zu überführen. Für den erfolgreichen praktischen Einsatz reichen die vorgestellten Modellierungstechniken daher alleine nicht aus.

2.2.2 Fuzzy Queries

Eine der ersten vorgeschlagenen Erweiterungen von Datenbanksystemen um Fuzzy-Techniken bot die Möglichkeit, unscharfe Anfragen auf scharfen Datenbeständen durchzuführen. Später wurden diese Ansätze ergänzt, um scharfe und unscharfe Anfragen auch auf Fuzzy-Daten zu ermöglichen. Hierfür wurde schließlich eine Reihe geeigneter Zugriffsstrukturen entwickelt, um solche Anfragen effizient durchführen zu können. Beispiele sind der Superimposed Coding Index [Bos94], VLFD Clustering [Buc95] und das Multi Level Grid File (MGLF) [YC99].

Bewertung Fuzzy-Anfragetechniken alleine reichen gleichfalls nicht aus, um komplette Informationssysteme zu realisieren. Mögliche Anwendungsbereiche finden sich jedoch innerhalb von Internet-Suchmaschinen oder bei der fehlertoleranten Volltextsuche.[1] Auch im Rahmen eines kompletten Fuzzy-Informationssystems können die Fuzzy-Abfragetechniken eingesetzt werden, um effiziente Suchverfahren auf Fuzzy-Daten im Rahmen einer Fuzzy-Datenbank zu realisieren.

2.2.3 Objektorientierte Fuzzy-Modelle

Die Arbeiten an objektorientierten Fuzzy Datenmodellen bis etwa Mitte der 90er Jahre sind geprägt von der Suche nach Einsatzmöglichkeiten von Fuzzy-Mengen im objektorientierten Modell [Cro95]. Verschiedene Arbeiten schlagen jeweils verschiedene Einsatzmöglichkeiten vor und betrachten dabei meist einen Teilaspekt genauer. Beispiele sind die Definition von Fuzzy-Attributen, Fuzzy-Objekten oder Fuzzy-Vererbungsbeziehungen.

Nach dieser orientierungssuchenden Phase entstanden die ersten Arbeiten, die konkrete Vorschläge für ein objektorientiertes Fuzzy-Datenmodell machten. Beispiel sind UFO und FOOD, die in den nachfolgenden Abschnitten genauer beschrieben werden.

In jüngster Zeit erschienen außerdem Arbeiten, die eine Konsolidierung der verschiedenen Ansätze vorschlagen, um zu einem einheitlichen Fuzzy-Datenmodell zu gelangen. Diese besprechen wir in Abschnitt 2.2.4.

Das FOOD-Modell

Das *„Fuzzy Object-Oriented Data Model“* (FOOD) [BLP94a, Cal97, YG99, LXHY99] erlaubt die Repräsentation von Unsicherheit auf der Ebene von Attributen, Klassen-

[1] Viele Anwendungen bezeichnen dies bereits heute mit Begriffen wie „Fuzzy-Suche“. Hierbei werden jedoch praktisch nie auf Fuzzy-Mengen basierte Verfahren eingesetzt, sondern vielmehr klassische Techniken wie Abstandsmetriken.

beziehungen und der Vererbungsrelation zwischen Klassen. Attribute können zusätzlich zu scharfen Werten einen unsicheren Wert, repräsentiert durch eine Fuzzy-Menge, annehmen. Unsicherheit auf der Ebene von Klassenbeziehungen wird über einen Fuzzy-Zugehörigkeitsgrad repräsentiert. Analog wird die Zugehörigkeit einer Klasse zu ihrer Oberklasse mittels eines Fuzzy-Grades ausgedrückt. Eine Betrachtung der Verarbeitung von Fuzzy-Attributen durch die Objektmethoden findet nicht statt. Eine Beispielimplementierung, aufbauend auf dem EXODUS System (ESM), wird in [YGA98] vorgestellt. Die Implementierung beschränkt sich auf die Implementierung von Fuzzy-Attributwerten und Fuzzy-Zugehörigkeitsgraden von Objekten zu Klassen; Einsatzbeispiele werden nicht gezeigt.

Bewertung FOOD beschränkt sich auf eine rein strukturelle Behandlung des objektorientierten Datenmodells. Wie die Methoden eines Objektes, ohne die eine Anwendung kaum auskommen dürfte, mit den unscharfen Werten umgehen sollen, wird nicht betrachtet. Dies ist umso kritischer, als sich die entwickelten Konzepte wie unscharfe Klassenbeziehungen nicht einfach in gängige Programmiersprachen einbetten lassen; hierfür müßte unter anderem das Konzept der Substituierbarkeit bei Operationen erweitert werden. Bezeichnenderweise sind die Beispielprogramme dann auch mittels der prozeduralen Sprache C und Unix-Shell-Skripten realisiert und beschränken sich darauf, mit fest vorgegebenen Operationen Fuzzy-Objekte in einer Datenbank zu erzeugen und zu verändern. Der eigentliche Vorteil einer objektorientierten Datenbank, der nahtlose Übergang zwischen den Datenmodellen der Anwendung und des Datenbanksystems, geht dadurch jedoch verloren. Wir unten noch präziser ausgeführt wird, ist für den Praktiker weniger wichtig, ob sich ein Modell irgendwie implementieren läßt, sondern vielmehr wie dies aufbauend auf existierenden Technologien und Standards erfolgen kann. Dieses Ziel verfehlt dieser Ansatz jedoch, weshalb er sich für die Entwicklung realer Anwendungen nicht eignet.

Das UFO-Modell

UFO ist ein weiterer Ansatz zur Einbettung von *„Uncertainty and Fuzziness in an Object-oriented database model“*, von den Autoren beschrieben als eine *„handsome method to cope with the uncertainty and fuzziness that curses information“* [GCV93, Cal97].

Das UFO-Modell ist sehr umfassend und beschreibt Fuzzy-Konzepte auf der Ebene von Attributen, Objekten und Klassen. Bei der Modellierung wird weiterhin unterschieden in unscharfe Informationen (vage Begriffe wie „groß“) und unsichere Informationen (alternative Werte, mögliche Werte, ungenaue und hypothetische Werte). Für die beiden Informationsarten werden dann getrennte Modelle entwickelt, basierend auf Fuzzy-Mengen für den ersten Fall und Possibilitätsverteilungen für den zweiten.

Das Fuzzy-Modell umfaßt Fuzzy-Attribute, -Objekte und -Klassen. Fuzzy-Attribute

sind dabei einfache oder mehrwertige Attribute mit dem neuen Datentyp „Fuzzy-Menge“. Bei Fuzzy-Objekten wird ein Zugehörigkeitsgrad eingeführt, mit dem ein Objekt zu einer oder mehreren Klassen gehört. Fuzzy-Klassen schließlich erweitern das scharfe Vererbungskonzept auf Fuzzy-Vererbungen, dadurch kann eine Klasse mit einem bestimmten Grad zu einer Oberklasse gehören.

Analog werden unsichere Informationen auf Attribut-, Objekt- und Klassen-Ebene eingeführt. Da die Autoren hier an der Modellierung von (unsicheren) Alternativen interessiert sind, wird das Konzept von Stellvertreter-Objekten (*role objects*) eingeführt, die für alle Objekte zugehörige Possibilitätsverteilungen verwalten. Diese werden bei bestimmten Operationen automatisch erzeugt und können für Alternativberechnungen herangezogen werden. Das Konzept der unsicheren Objekte wird über die Einführung einer neuen Oberklasse *uncertain* realisiert, der unsichere Objekte angehören können. Für die Unsicherheit auf Klassen-Ebene wird hier die Idee der „Hypothetischen Modellierung“ vorgestellt, bei der das Schema selbst mit Unsicherheit behaftet ist, allerdings wird diese Idee nicht weiter ausgeführt.

Eine Implementierungsmöglichkeit für UFO wird nicht aufgezeigt, ebenso keine auf dem Modell basierenden Anwendungen.

Bewertung UFO zeigt, wie einfach es ist, das objektorientierte Datenmodell um Fuzzy-Mengen zu erweitern [Cal97]: *„Introducing a fuzzy set semantics at all layers of the OODBMS is obtained by replacing the concept of a “set” with that of a “fuzzy set”, whereever the concept of a set is used in an OODBM and whereever it is meaningful.“*

So läßt sich einfach die Menge aller Unterklassen einer Klasse zu einer Fuzzy-Vererbung und die Menge aller Objekte einer Klasse zu einem Fuzzy-Zugehörigkeitsgrad erweitern. Jetzt muß schließlich „nur noch“ ein passendes Datenbanksystem geschrieben werden, das dieses erweiterte Fuzzy-Modell realisiert, eine zugehörige objektorientierte Fuzzy-Programmiersprache inklusive Übersetzer und Bibliotheken entworfen und implementiert werden, sowie geeignete Modellierungsverfahren für den Entwurf von Fuzzy-Anwendungen entwickelt werden, die Gebrauch von den neuen Möglichkeiten wie Fuzzy-Vererbungsbeziehungen machen. Leider wird die Entwicklung dieser Details dem Leser überlassen.

Obwohl UFO interessante Aspekte bei der Fuzzyifizierung des objektorientierten Datenmodells aufzeigt, bleibt es doch eine theoretisch ausgerichtete Arbeit, die offenbar nicht das Ziel verfolgt, die Entwicklung von Fuzzy-Anwendungen zu ermöglichen. Hierfür fehlt es nicht nur an der technischen Umsetzung alleine, sondern bereits die entwickelten Konzepte wie das der Fuzzy-Vererbung werden nicht weiter hinterfragt, so daß unklar bleibt, ob sie für einen Fuzzy-Anwendungsentwickler tatsächlich Vorteile bringen. Selbst theoretisch unbedenkliche Ideen wie die der „Role Objects“ lassen bei einer praktischen Umsetzung nichtakzeptable Leistungseinbußen in der Laufzeitumgebung befürchten.

2.2.4 Standardisierungsbemühungen

Manche Autoren beklagen bereits die Zersplitterung der Forschung auf dem Gebiet der Fuzzy-Datenbanken, die einen Vergleich der Ergebnisse erschwert oder sogar unmöglich macht. Ein Vorschlag zur Konsolidierung ist die Entwicklung eines standardisierten Fuzzy-Frameworks, in das die verschiedenen Ansätze eingebracht werden können [Cro96, CCV97].

Bewertung Bisher hat sich noch keiner der Vorschläge zu einer Standardisierung des Forschungsgebietes durchsetzen können. Vermutlich ist es beim bisherigen Stand der Forschung noch zu früh für solche Bestrebungen. Schon bei grundlegenden Fragen wie der Formulierung von Fuzzy-Attributen herrscht keine Einigkeit; bei weitergehenden Fragen wie der Möglichkeit von Fuzzy-Klassendefinitionen gehen die Meinungen so weit auseinander, daß eine Vereinheitlichung nur unter dem Ausschluß einzelner Ansätze möglich wird.

Es ist in Anbetracht dessen auch fraglich, ob eine solche Standardisierung überhaupt wünschenswert ist. Sie würde zwar einen Vergleich der Ergebnisse ermöglichen; interessante Ansätze, die nicht in das Standard-Framework passen, gingen dabei jedoch verloren. Da noch keineswegs klar ist, welche Ansätze letztendlich erfolgsversprechend sind, kann dies eine unerwünschte Beschneidung potentiell erfolgreicher Ideen zur Folge haben, was sich dann erst recht negativ auf das Forschungsgebiet auswirkt.

Viel interessanter dürfte es sein, bestehende Ansätze im Wettbewerb beim praktischen Einsatz zu beobachten. Sobald eine Vorgehensweise in der Praxis handfeste, vermittelbare Vorteile anbietet und daraufhin im industriellen Maßstab eingesetzt wird, dürfte sich dies auch auf dem Gebiet der Forschung auswirken.

2.3 Fuzzy-Anwendungen

Neben den theoretisch ausgerichten Arbeiten des letzten Abschnittes entstanden parallel auch Arbeiten, die den praktischen Einsatz von Fuzzy-Technologien in Informationssystemen zum Ziel hatten.

Kennzeichnend für diese Arbeiten ist, daß sie nicht auf ein fertiges, fuzzyfiziertes Datenmodell zurückgreifen, sondern jeweils eigene Lösungen entwickeln, um imperfekte Informationen innerhalb einer konkreten Anwendung zu verwalten.

Im folgenden zeigen wir exemplarisch einige Beispiele.

2.3.1 Entwurfsanwendungen

Boss [Bos96] betrachtet die Domäne der Entwurfsanwendungen im Bereich der Architektur und entwickelt ein objektorientiertes Fuzzy-Datenmodell zur Verwaltung und Verarbeitung von Entwurfsanforderungen. Besondere Merkmale sind die Repräsentationsmöglichkeiten der Fuzzy-Anforderungen, die hierarchisch gegliedert sein können, die Betrachtung der Verarbeitung und Konsistenzprüfung dieser Anforderungen, sowie die lauffähige Implementierung der entwickelten Konzepte aufbauend auf einem existierenden objektorientierten Datenbanksystem.

Bewertung Die Arbeit von Boss ist eine der wenigen, die ein theoretisches Fuzzy-Datenmodell anhand der Anforderungen eines konkreten Anwendungsfalles entwickeln. Zwar wurde das Modell zugeschnitten auf die Bedürfnisse der Entwurfsanwendungen entwickelt, ist aber getrennt von seinem konkreten Einsatz formuliert, so daß es sich zumindest in Teilen wiederverwenden und erweitern läßt. Dagegen abstrahiert die technische Umsetzung nicht genügend von dem Anwendungsfall der Entwurfsanwendungen, so daß sie sich nicht in anderen Gebieten einsetzen läßt. Und schließlich geht diese Arbeit nicht auf die grundsätzlichen Anforderungen von Fuzzy-Anwendungen ein, was eine Voraussetzung für die systematische Entwicklung neuer Konzepte und Technologien darstellt.

2.3.2 Produktentwicklung

Die Arbeit von [ERZD95] verfolgt das Ziel, durch Modellierung unsicherer Informationen eine Parallelisierung von Entwicklungsabläufen und damit kürzere Produktentwicklungszeiten zu erreichen. Modelliert werden die unsicheren Daten mit Hilfe eines Fuzzy-Objektes, das in ein Express-basiertes Datenmodell eingebettet ist.

Ebenfalls die Verkürzung von Produktentwicklungszeiten hat der Ansatz von [Fai99] als Aufgabe. Konkret geht es in dieser Arbeit um die Auswahl geeigneter Komponenten in der Produktentwicklung; demonstriert wird der Ansatz am Beispiel der Auswahl eines passenden Riementypes für ein zu konstruierendes Riemengetriebe. Die unscharfen Eigenschaften der Produkte werden mit Hilfe von Fuzzy-Mengen modelliert. Von Experten gewonnene Grenzwerte für Produktparameter werden dann verwendet, um mit Hilfe eines Fuzzy-Auswahlverfahrens geeignete Produkte aus einer Datenbank auszuwählen.

Bewertung Der erste Ansatz modelliert zwar die imperfekten Informationen, verwendet sie jedoch nicht innerhalb einer realisierten Anwendung. Die Betrachtung der dafür notwendigen Weiterverarbeitung erfolgt daher nicht. Somit kann leider auch kein konkreter Nachweis erbracht werden, daß ein solcher Ansatz tatsächlich kürzere Produktentwicklungszeiten ermöglicht. Interessant ist jedoch die vorgestellte Idee

der „Informationsreife“, die Mitarbeitern den Umgang mit den für sie ungewohnten unsicheren Informationen erleichtern soll.

Der zweite Ansatz dagegen entwickelt zur Validierung eine lauffähige Implementierung. Diese ist jedoch spezialisiert auf das gezeigte Einsatzgebiet: sämtliche Komponenten, angefangen von der Benutzeroberfläche bis hin zur Datenhaltung, wurden eigens für diese Anwendung entwickelt. Die geleistete Arbeit läßt sich daher nicht auf andere Gebiete übertragen; selbst ähnlich gelagerte Anwendungsfälle dürften eine komplette Neuentwicklung aller beteiligten Komponenten erfordern.

2.3.3 Unternehmensplanung

Ein Beispiel aus der Unternehmensplanung ist in [Hau98] ausgeführt. Dabei werden klassische Planungsmodelle wie MODM (Multiple Object Decision Making) oder MADM (Multiple Attribute Decision Making) um Fuzzy-Techniken erweitert (Fuzzy-MODM, Fuzzy-MADM) und mögliche Einsatzgebiete aufgezeigt.

Bewertung Leider findet für die entwickelten Fuzzy-Modelle weder ein theoretischer Vergleich mit den herkömmlichen Modellen noch eine praktische Implementierung mit Beispielrechnungen statt. Demzufolge gibt es auch keine Aussage über die Eignung oder den Vorteil der fuzzyfizierten Ansätze.

2.3.4 Lernprogramme

Die Arbeit von [LGL98] erweitert ein Multimedia Tutoring System um ein Fuzzy-Modell zur Repräsentation des Kenntnisstandes des Benutzers. Damit wird es möglich, eine Lerneinheit abzuschließen, wenn ein bestimmter, unscharfer Kenntnisstand erreicht wird, was auf verschiedenen Wegen möglich ist.

Bewertung Das Beispiel zeigt, wie ein klassisches System durch den gezielten Einsatz von Unschärfe verbessert werden kann. Leider wird nur das theoretische Modell entwickelt; die praktische Umsetzung und Einbettung in existierende Lernsysteme dürfte aber zusätzliche Arbeiten mit signifikantem Aufwand erfordern.

2.3.5 Verkehrsplanung und -steuerung

Ein Anwendungsgebiet, das in jüngster Zeit verstärkt Interesse findet, ist die Modellierung von bewegten Objekten im Raum. Bei diesen Szenarien wird der Zielkonflikt zwischen gewünschter Genauigkeit und erforderlichem Aufwand besonders deutlich. Denn je genauer die aktuelle Position bewegter Objekte bestimmt werden soll, desto höher ist der Aufwand zur Verwaltung und Aktualisierung der Informationen.

Die meisten Arbeiten auf diesem Gebiet konzentrieren sich auf den militärischen Bereich, wie etwa die Erweiterung von ECA-Regeln *(Event/Condition/Action)* für ein aktives fuzzy-relationales Datenbanksystem [SUY99].

Im zivilen Bereich gewinnen dagegen neue, dezentrale Ansätze in der Verkehrsplanung und -steuerung an Interesse, die sich zunehmend der Unsicherheit stellen und versuchen, aus ihr Vorteile für die Systementwicklung und den Anwender zu ziehen [Luk00], [LL00].

Bewertung Dieses Einsatzgebiet ist recht jung, daher befindet sich die Forschung noch im Anfangsstadium. Ein interessanter Unterschied zu den vorher angeführten Einsatzbeispielen liegt jedoch in der bei diesem Szenario auftretenden immensen Datenfülle, die eine Vielzahl intrinsischer Unvollkommenheiten aufweist. Hier kann daher erstmalig ein Anwendungsgebiet vorliegen, in dem klassische informationstechnische Realisierungen aufgrund mangelnder Robustheit und unattraktiv hoher Entwicklungskosten ausgeschlossen sind. Dies stellt eine wirkungsvolle Motivation dar, Mechanismen zum Umgang mit imperfekten Daten bevorzugt in die Betrachtung einzubeziehen.

Die Arbeit von [LL00] geht zum Teil noch darüber hinaus: sie fordert, höhere Robustheit von Systemen im Bereich der Verkehrssteuerung durch gezielte „Verschmierung“ scharfer zu unscharfer Daten zu erreichen. Zusätzlich wird der komplementäre Einsatz verschiedener Repräsentationsverfahren wie Fuzzy-Mengen, Intervallwertige Logiken und Wahrscheinlichkeitstheorie verlangt, um mit der Vielzahl der bei diesem Szenario auftretenden Unsicherheitsphänomene umzugehen. Die praktische Umsetzung befindet sich jedoch noch im Anfangsstadium; die Autoren machen deutlich, daß es an dieser Stelle noch an grundlegenden Forschungsarbeiten fehlt, bis einsatzfähige Anwendungen realisiert werden können, die nach diesem Prinzip arbeiten.

2.4 Zusammenfassung der Kritikpunkte

Jeder der bisher beschriebenen Ansätze verfolgt seine eigene Zielsetzung und versucht, diese in verschiedenem Maße zu erfüllen. Wenn man aber nun die Rolle des Entwicklers von Fuzzy-Anwendungen in den Vordergrund stellt, und die bisher beschriebenen, existierenden Ansätze unter diesem Gesichtspunkt bewertet, ergeben sich eine ganze Reihe von Schwächen und Einschränkungen.

Zusammengefaßt sind diese Kritikpunkte im Einzelnen:

- Die theoretisch ausgerichteten Arbeiten beschränken sich auf Teilaspekte, wie die Formulierung eines formalen Datenmodells. Mit diesen fragmentarischen

Ansätzen lassen sich aber noch keine Anwendungen realisieren. Da die verschiedenen Teilarbeiten auch von unterschiedlichen Voraussetzungen ausgehen und zueinander inkompatibel sind, lassen sie sich nicht untereinander kombinieren, um ein durchgängiges Entwicklungsverfahren zu erhalten.

- In einer Reihe der existierenden Ansätze werden fuzzifizierte Datenmodelle entwickelt, die potentiell zur Modellierung von Fuzzy-Anwendungen geeignet sind. Datenmodelle aber sind kein Selbstzweck, sondern nur für die Modellierung und Realisierung von Systemen sinnvoll einsetzbar. Trotzdem findet eine Validierung dieser Datenmodelle auf ihre Einsatzfähigkeit hin typischerweise nicht statt; somit bleibt unklar, wie geeignet sie für die Entwicklung einer Anwendung wirklich sind.
- Auch in Arbeiten, die zumindest eine prototypische Beispiel-Implementierung vorstellen, wird nicht spezifiziert, für welche Art von Systemen die entwickelten Ansätze eingesetzt werden sollen. Dadurch kann nicht festgestellt werden, welchen Anwendungsbereich die Ansätze haben sollen und ob sie für einen konkreten Anwendungsfall wirklich verwendet werden können.
- Für praktische Anwendungen werden daher jeweils neue, spezialisierte Systeme erstellt, die aber keine allgemeinen, wiederverwendbaren Konzepte anbieten. Der Aufwand zur Erstellung neuer Systeme erhöht sich so enorm. Diese Vorgehensweise ist vergleichbar mit der prä-Datenbankzeit, in der nicht auf ein fertiges Datenbanksystem zurückgegriffen wurde, sondern immer ein eigenes Speichersystem im Rahmen der Anwendungsentwicklung mit realisiert wurde.
- Die Behandlung imperfekten Wissens beschränkt sich auf die strukturelle Modellierung. Aspekte der weiteren Verarbeitung werden (bis auf wenige Ausnahmen) überhaupt nicht behandelt. Die wenigen Modelle, die explizit eine Verarbeitung des imperfekten Wissen behandeln, sind spezialisiert auf einen ganz bestimmten Einsatzbereich. Dazu gehören zum Beispiel verschiedene Fuzzy Controller oder Planungssysteme, die oft auf der Basis von Expertensystemen realisiert wurden. Kritisch ist dies, weil die Verarbeitung direkte Auswirkungen auf die *Konsistenz* von Informationen hat, deren Erhaltung ein zentraler Gegenstand in der Datenbank- und Informationssystemtechnologie darstellt.

In der Literatur stößt man oft auf einen weitereren Kritikpunkt, den wir im nachfolgenden Abschnitt adressieren.

2.5 Der Performance-Mythos

Eine oft in der Literatur vorgetragene Entschuldigung für die mangelnde Akzeptanz vorhandener Fuzzy-Ansätze liegt in der angeblich zu geringen Leistungsfähigkeit [MS97], [YG99]:

> *„First, information systems practitioners are concerned primarily with the* performance *of their systems."*
>
> *„This has stunted its adoption in a marketplace primarily concerned with performance."*

Als Ausweg wird jedesmal die Entwicklung neuer, leistungsfähigerer Speicher- und Zugriffstechniken für Fuzzy-Daten vorgeschlagen.

Man sollte annehmen, daß solche Schlußfolgerungen aus den Erfahrungen mit lauffähigen Implementierungen von Fuzzy-Informationssystemen gezogen wurden, die im praktischen Einsatz eine vorgegebene Leistungsmarke nicht erreichen konnten. Tatsächlich wird dieses Argument jedoch *kein einziges Mal* mit den gemessenen Leistungswerten existierender Fuzzy-Systeme untermauert.

Dies verwundert nicht weiter, ist es doch aufgrund der oben aufgezeigten Schwachpunkte *gar nicht möglich*, mit den existierenden Ansätzen eine ablauffähige Fuzzy-Anwendung zu entwickeln, die dann möglicherweise Leistungsdefizite aufweisen könnte.

Somit ist auch die immer neue Entwicklung besserer Speicher- und Indexstrukturen ein hoffnungsloses Unterfangen, solange die Anforderungen echter Anwendungen noch überhaupt nicht verstanden sind.

Von einem „Performance"-Defizit kann lediglich in Bezug auf den gesamten Entwicklungsprozeß einer Fuzzy-Anwendung gesprochen werden, die mangels durchgehender Unterstützung auf aufwendige, anwendungsspezifische Spezialanfertigungen angewiesen ist.

Dies ist letztendlich auch der wirkliche Grund, warum heute der Fuzzy-Ansatz für einen potentiell interessierten Entwickler noch immer so unattraktiv erscheint.

2.6 Fazit

Arbeiten, die die Entwicklung von Fuzzy-Informationssystemen vom Entwurf bis hin zur Implementierung betrachten, gibt es nicht.

Im Bereich der theoretischen Ansätze wird eine Reihe von Aspekten isoliert betrachtet, die sich bis heute aber noch nicht zu einer geschlossenen Methodik zusammenfügen konnten. Praktisch ausgerichtete Arbeiten, die ein lauffähiges Zielsystem implementieren wollen, entwickeln daher ausnahmslos eigene, auf das jeweilige Problem zugeschnittene Ansätze, die sich aber nicht auf andere Bereiche übertragen lassen.

Im Bereich der Softwaretechnologie, wo die Besonderheiten von Fuzzy-Informationssystemen im Bereich des Entwurfs und der Implementierung berücksichtigt werden müssen, gibt es noch keine Arbeiten.

Die existierenden Ansätze liefern jedoch entscheidende Vorarbeiten, ohne die eine umfassendere Betrachtung, wie sie hier durchgeführt wird, nicht möglich wäre. Insbesondere aus der Arbeit von Boss [Bos96] greifen wir viele Ideen und Ansätze auf, die hier verallgemeinert und weiterentwickelt werden.

Kapitel 3

Zielsetzung und Anforderungsanalyse

"Don't come back till you have him!"
the Tick-Tock Man said, very quietly, very sincerely, extremely dangerously.

They used dogs. They used probes. They used cardioplate crossoffs. They used teepers. They used bribery. They used stiktytes. They used intimidation. They used torment. They used torture. They used finks. They used cops. They used search&seizure. They used fallaron. They used betterment incentives. They used fingerprints. They used the Bertillon system. They used cunning. They used guile. They used treachery. They used Raoul Mitgong, but he wasn't much help. They used applied physics. They used techniques of criminology. And what the hell: they caught him.
Harlan Ellison, "'Repent, Harlequin!' Said the Tick-Tock Man"

In diesem Kapitel präzisieren wir das Ziel dieser Arbeit und stellen dafür eine Reihe von Anforderungen auf, die ein Lösungsansatz erfüllen muß. Diese Anforderungen werden anhand von Beobachtungen und Erfahrungen mit existierenden Arbeiten abgeleitet und in den nachfolgenden Kapiteln bei konkreten Entwurfsentscheidungen herangezogen, um bei Alternativentwürfen eine gewählte Vorgehensweise zu begründen.

3.1 Ziel der Arbeit

Wir präzisieren zunächst das Ziel dieser Arbeit:

> **Das Ziel der Arbeit ist die Entwicklung eines Architekturmodells, das sich zur Konzeption und Realisierung von Fuzzy-Informationssystemen eignet.**

In dieser Arbeit soll also *nicht* ein einzelnes Anwendungsbeispiel betrachtet und mit Hilfe von Fuzzy-Technologien umgesetzt werden. Vielmehr soll ein allgemeines *Architekturmodell* entwickelt werden, das sich dann für die Entwicklung eines konkreten Informationssystems heranziehen läßt.

Mit dem Begriff *Architekturmodell*, oft auch *Blueprint* genannt, bezeichnen wir eine Referenzarchitektur, mittels derer ein Informationssystem realisiert werden kann. Da verschiedene Informationssysteme unterschiedliche Anforderungen haben, enthält ein solches Modell optionale Teile, die nicht zwingend verwendet werden müssen, und Varianten, von denen meist nur eine bei einer Implementierung zum Tragen kommt.

Unter *Informationssystem* verstehen wir ein komplettes, ablauffähiges, anwenderorientiertes, informationsverarbeitendes System. Datenbanksysteme sind nur ein optionaler Teil eines solchen Informationssystems, sie stellen alleine noch kein ablauffähiges System dar.

Unsere *Zielgruppe* sind Informationssysteme, wie sie heutzutage im kommerziellen oder wissenschaftlichen Bereich entwickelt werden, mit einem Schwerpunkt auf Internet/Intranet-basierten Systemarchitekturen. Wie in Kapitel 1 geschildert, können diese Systeme in besonders hohem Maße von dem Einsatz der Fuzzy-Technologie profitieren, womit wir den potentiellen Nutzen und die Verbreitung unseres Ansatz maximieren.

Unterstützt werden soll sowohl die *Konzeption* als auch die technische *Realisierung* eines solchen Informationssystems. Da der Einsatz eines neuen Verfahrens wie der Fuzzy-Technologie Auswirkungen auf den gesamten Entwicklungsprozeß hat, müssen auch alle dazugehörigen Teilaspekte, wie die Modellierung einer Anwendung, ihre Implementierung in einer Programmiersprache und deren Einbettung in ein Laufzeitsystem, berücksichtigt werden.

Der Einsatz der Fuzzy-Technologie schließlich erfolgt nicht als Selbstzweck. Je nach Einsatzgebiet kann es eine Reihe von Zielen geben, die durch ihren Einsatz erreicht werden soll, wie bessere Modellierbarkeit der Anwendungsdomäne, schnellere Realisierbarkeit oder ein robusteres Laufzeitsystem. Wichtig ist, daß das gewählte Architekturmodell keine dieser Möglichkeiten von vornherein ausschließt.

3.2 Anforderungen an den eigenen Ansatz

Im folgenden leiten wir die Anforderungen ab, die wir an ein Architekturmodell für Fuzzy-Informationssysteme stellen.

Jeder der nachfolgenden Abschnitte ist einheitlich gegliedert: nach einer Beschreibung des relevanten Konzeptes wird eine These aufgestellt, aus der dann die Anforderung abgeleitet wird. Jede der Anforderungen bekommt einen eigenen Namen

(zur leichten Erkennung gesetzt in KAPITÄLCHEN), anhand dessen sie in den nachfolgenden Kapiteln referenziert wird. Insbesondere bei Entwurfsentscheidungen wird immer wieder auf die beschriebenen Anforderungen zurückgegriffen, um eine konkrete Entscheidung zu motivieren.

Dabei sind die Anforderungen partiell sortiert: die ersten Anforderungen beschreiben grundsätzliche Eigenschaften, während nachfolgende zunehmend auf Teilaspekte fokussieren.

3.2.1 MODERNE ARCHITEKTUR

Einer der genannten Kritikpunkte war die Ziellosigkeit, mit der existierende Ansätze formuliert sind, ohne ihren angepeilten Einsatzbereich zu definieren. Die Anforderungen und Besonderheiten des gewählten Einsatzbereiches sind jedoch entscheidend für die Entwicklung eines solchen Ansatzes.

Wir definieren daher zuerst die Ziel-Architektur sowie die Basis-Technologien, die von unserem Ansatz abgedeckt werden sollen.

Bereits mehrfach wurde erwähnt, für welche Art von Systemen Fuzzy-Technologien Vorteile bringen sollen: komplexe Systeme, wie Internet/Intranet-basierte Informationssysteme, die mit großen Datenbeständen arbeiten, viel mit Benutzern oder anderen Systemen interagieren und dabei robust mit Unschärfen und Inkonsistenzen umgehen sollen.

Yager beschreibt dies in [Cal97] folgendermaßen:

> *With the rapid development of the internet, a medium based upon the confluence of computing and communication technologies, the stage is set for a decade marked by an explosive expansion of an information based culture throughout the world. [...] In order to realize this promise we need methodologies for the intelligent representation and manipulation of information. Fuzzy sets clearly provides a tool that can help in this task.*

Wenn man also erreichen will, daß Fuzzy-Konzepte in solche Systeme Einzug halten, müssen auch die Technologien unterstützt werden, mit denen diese Systeme realisiert werden.

Heute sind dies vor allem mehrstufige *(multi-tier)* Client/Server-Architekturen, wie sie in Abbildung 1.1 auf Seite 7 skizziert sind. Die einzelnen Stufen sind dabei von Technologien bevölkert, die mit Stichworten wie Java-Applets, (D)HTML/XML, Application Server, EJBs, TP-Monitore, EJB-Komponenten, Servlets, Message Queues und Datenbankmanagementsysteme beschrieben werden und auf die wir in Teil V noch genauer eingehen werden.

Von den wenigen Ansätzen, die eine Implementierung vorstellen, verwendet [Cal97] beispielsweise eine Erweiterung des EXODUS-Speichermanagers. Wie ließe sich dies

nun in das geschilderte Umfeld einpassen, um die Lösung innerhalb eines Projektes zu verwenden? Gar nicht — eine komplette Neuentwicklung wäre nötig. Andere Ansätze, wie das auf Basis des *Austin Kyoto Common Lisp/Encore query algebra* implementierte FOODB-System sind nicht besser geeignet, um moderne Web-Anwendungen zu realisieren.

Es gibt sogar Anhaltspunkte, daß solche ad-hoc Implementierungen den praktischen Einsatz eher behindert als gefördert haben [MS97]:

> *Practitioners are also concerned with* compatibility. *This dictates that capabilities for managing uncertainty and imprecision should be offered as strict extensions of existing standards. Additionally, database practitioners have often been dissatisfied with various* idiosyncratic implementations *of such capabilities [...].* ***This may have had a chilling effect on further implementations.***

Denn die Gemeinde der Datenbank- und Informationssystemforscher ist im Gegensatz zu Modelltheoretikern ingenieurwissenschaftlich ausgerichtet; für sie steht die Einhaltung etablierter Standards und Vorgehensweisen im Vordergrund. Gerade diese werden jedoch von prototypischen Implementierungen, bei denen nur die grundsätzliche Machbarkeit im Vordergrund stehen, verletzt. Die unerwünschte Folge ist, daß Ergebnisse der theoretischen „Community“ nicht von der Gemeinde der Praktiker akzeptiert und eingesetzt werden — und das nicht aus inhaltlichen Gründen, sondern nur weil sie nicht der erwarteten Form genügen.

Die Hauptanforderung muß daher lauten:

> **Eine Fuzzy-Architektur muß die für die angepeilten Zielsysteme relevanten Technologien und Standards unterstützen und in geeigneter Weise erweitern.**

Bei allen vorgenommenen Änderungen existierender Standards und Technologien achten wir daher darauf, diese als orthogonale Erweiterungen auszuführen. Als Ausgangspunkt für die Implementierung verwenden wir heute in der Praxis eingesetzte Architekturmodelle (Blueprints) wie das der *Java 2 Enterprise Edition* (J2EE).

Viele existierende Ansätze stellen sich auf den Standpunkt, daß dies nur Implementierungsdetails sind, die in einer wissenschaftlichen Arbeit nicht berücksichtigt werden müssen. Aber dann darf sich die Forschergemeinde auch nicht darüber beklagen, daß ein mühselig entwickelter Ansatz keine Anwendung findet, wenn die im Vorfeld selbst gewählte Zielgruppe bei der technischen Umsetzung plötzlich ignoriert wird.

3.2.2 DURCHGÄNGIGES VERFAHREN

In Kapitel 2 wurde bereits eine Reihe existierender Ansätze zur Modellierung von Fuzzy-Daten und zur Realisierung von Fuzzy-Informationssystemen vorgestellt. Die-

se beschränken sich jedoch allesamt auf Teilaspekte, zum Beispiel auf die logische Datenmodellierung von Fuzzy-Informationen. Darüber hinaus sind die verschiedenen Ansätze untereinander inkompatibel, da sie jeweils von völlig anderen Voraussetzungen und Zielsetzungen ausgehen und unterschiedliche Notationen und Repräsentationsverfahren wählen.

Um aber ein tatsächlich einsetzbares Fuzzy-Informationssystem realisieren zu können, ist es notwendig, alle Bereiche der Systementwicklung, von der Modellierung bis hin zur Implementierung, abzudecken.

Betrachten wir noch einmal einen Teil des in Abschnitt 2.1 schon angeführten Zitats aus [YG99]:

> *The research community has hitherto addressed the second issue* [application development] *only in a piece meal fashion. This has not lead to a consistent body of work that clearly demonstrates the advantage of this approach to the database (and application) community at large.*

Das heißt, die existierenden fragmentarischen, untereinander inkompatiblen Ansätze helfen nicht weiter, wenn ein konkretes System realisiert werden soll. Die Folge ist, daß man auf eine völlige Neuentwicklung angewiesen ist, die — wie oben gezeigt — dann typischerweise für eine einzelne Anwendungsdomäne erfolgt und weder wiederverwertbare Konzepte verwendet noch selbst welche anbietet. Der Entwicklungsaufwand für eine Fuzzy-Lösung steigt dadurch enorm; ein Vorteil der fuzzy-basierten Vorgehensweise ist im Vorfeld nicht abschätzbar, ganz im Gegensatz zu einer klassischen Realisierung mit kalkulierbarem Aufwand.

Daraus ergibt sich die zweite Anforderung:

> **Ein Verfahren zur Entwicklung von Fuzzy-Informationssystemen darf nicht auf einen Teilaspekt beschränkt sein, sondern muß, ausgehend vom Entwurf bis hin zur Implementierung, ein durchgängig anwendbares Verfahren bieten.**

Insbesondere müssen die Teilschritte der Modellierung einer Anwendung, der Umsetzung des Modells in eine konkrete Programmiersprache sowie der Implementierung aufbauend auf einem Laufzeitsystem von dem Ansatz abgedeckt sein.

3.2.3 UNSCHÄRFE, EFFIZIENZ & EFFEKTIVITÄT

Die Motivation der meisten Ansätze, sich mit fuzzy-basierten Datenmodellen zu beschäftigen, läßt sich exemplarisch mit dem folgenden Zitat [Che98] wiedergeben:

> *That is, as uncertainty and imprecision is part of the reality, it should be modeled as part of the system.*

Wenn es aber um die Realisierung echter, einsetzbarer Informationssysteme geht, ist dies nur eine Illusion.

Das Auftreten von Imperfektionen in der Realität reicht als Motivation alleine nicht aus, um sie auch in Anwendungen zu modellieren: Wie in den vergangenen Jahrzehnten demonstriert, lassen sich Anwendungen sehr wohl auch ohne Techniken zur Repräsentation der domänenspezifischen Unschärfe oder Unsicherheit realisieren. Dies ist typischerweise mit einem erhöhten Aufwand zur Modellierung der Anwendungsdomäne verbunden, um durch hinreichende Präzisierung die tatsächlich unscharfen Sachverhalte in scharfe Strukturen abzubilden.

Betrachten wir dagegen noch einmal die ursprüngliche Formulierung von Zadeh zur Idee des „Soft Computing“:

> *The real world is pervasively imprecise and uncertain. Precision and certainty carry a cost. The guiding principle of soft computing is: Exploit the tolerance for imprecision, uncertainty, and partial truth to achieve tractability, robustness, and low solution cost.*

Man sieht, daß hier keineswegs die Modellierung der Unsicherheit als Selbstzweck vorgeschlagen wird. Vielmehr erwartet man einen *Vorteil*, der sich in anderen Entwicklungsparametern wie Zeitaufwand oder Modellierungskomplexität zeigt. Obwohl in der Literatur oft zitiert, findet aber eine tatsächliche Überprüfung der erwünschten Vorteile, wie Robustheit oder Kostenersparnis, nie statt.

Daraus ergibt sich die nächste Anforderung:

> **Die Entwicklung von Fuzzy-Informationssystemen ist kein Selbstzweck. Das Architekturmodell muß es ermöglichen, Fuzzy-Anwendungen zu realisieren, die gegenüber klassischen Verfahren erkennbare Vorteile bringen. Solche Vorteile können eine schnellere Realisierbarkeit, ein robusteres Endsystem oder geringere Entwicklungskosten darstellen.**

Ein Ansatz zur Modellierung unscharfer Informationen kann noch so formal und vollständig sein; wenn er die oben diskutierten Vorteile für einen Projektleiter, Systemarchitekten oder Anwendungsentwickler nicht bietet, wird er — zu Recht! — nie eingesetzt werden. Diese Arbeit soll jedoch genau dies ermöglichen.

3.2.4 PARTIELLE UNSCHÄRFE

Für welche Art von Systemen eignet sich der Fuzzy-Modellierungsansatz? Der allgemeine Konsens ist, daß in der Realität vorkommende Unschärfen durch Fuzzy-Technologien repräsentiert werden sollen. Aber für welche Anwendungen trifft dies

zu? Und, noch wichtiger, wo bringt es überhaupt einen *Vorteil*, die Unschärfe sichtbar zu machen?

Die im letzten Kapitel beschriebenen, existierenden Ansätze zeichnen sich dadurch aus, daß dieser Aspekt völlig ignoriert wird. Weder existiert eine Vorstellung, für welche Klasse von Informationssystemen die Fuzzy-Techniken genutzt werden sollen, noch findet eine Begründung statt, warum ein entwickelter Ansatz vorteilhaft für die Realisierung einer solchen Anwendung sein soll.

Hieraus ergibt sich als erstes die Leitfrage:

> *Für welche Klasse von Informationssystemen soll ein Fuzzy-Ansatz eingesetzt werden?*

Diese Frage wird in den existierende Ansätzen nicht gestellt — und dadurch auch nicht beantwortet. Allenfalls läßt sich rückwärts aus den Entwurfsentscheidungen schließen, an welche Art von Systemen die Entwickler gedacht haben könnten. Bei konkret realisierten Fuzzy-Systemen schließlich stand nur das Anwendungsbeispiel im Vordergrund, diese Arbeiten lassen sich nicht auf andere Anwendungsdomänen übertragen.

In dieser Arbeit richten wir das Augenmerk auf Informationssysteme, die Unschärfe nicht als Hauptbestandteil haben, sondern sie nur punktuell einsetzen, um dadurch gezielt Vorteile gegenüber klassischen, ausschließlich scharfen Informationssystemen zu erzielen. Wir begreifen Imperfektion also als eine von vielen Eigenschaften, die Daten innerhalb eines Systems aufweisen können, gleichberechtigt neben anderen Eigenschaften wie Persistenz oder Sicherheit.

Die nächste Anforderung lautet daher:

> **Unterstützung einer neuen Klasse von Systemen: keine „100%-Fuzzy-Systeme“, sondern normale Informationssysteme, die selektiv um Fuzzy-Technologien erweitert werden**

Dieser gezielte, punktuelle Einsatz von Unschärfe bedeutet insbesondere:

- typischerweise nur in wenigen Teilen des Systems,
- scharf eingegrenzt, zum Beispiel innerhalb eines Objektes ein einzelnes Attribut und
- orthogonal zu anderen System-Eigenschaften.

Tatsächlich beanspruchen viele existierende Ansätze für sich, auch den nicht-fuzzy Fall abzudecken. Dies ist zwar formal richtig, aber irreführend, da die scharfen Daten lediglich als Spezialfall der fuzzy Daten aufgefaßt werden und mit den gleichen aufwendigen Mitteln bearbeitet werden müssen, die sonst nur für Fuzzy-Daten notwendig sind.

3.2.5 OBJEKTORIENTIERTES SYSTEM

Entsprechend der Anforderung DURCHGÄNGIGES VERFAHREN muß ein geeignetes Datenmodell ausgewählt werden, das die Modellierung, Einbettung, Implementierung und den Ablauf sowie die Speicherung von Fuzzy-Daten innerhalb eines Informationssystems ermöglicht.

Da sich unsere Arbeit an die Entwickler von Informationssystemen richtet, müssen wir auch ein Datenmodell unterstützen, das zur Realisierung aktueller Systeme mehrheitlich verwendet wird. Für die überwiegende Anzahl der heute (Stand 2002) realisierten Systeme ist dies das objektorientierte Datenmodell, sowohl im kommerziellen wie auch im wissenschaftlichen Bereich.

Aufgrund seiner Ausdruckskraft eignet sich das objektorientierte Modell sehr gut, um imperfekte Informationen auf verschiedenen Ebenen zu behandeln [YG99]:

> *The superior modeling power of the semantic and object oriented data models make it possible to represent uncertainty and impreciseness at many additional levels:*
>
> 1. *The attribute value level*
> 2. *The relationship between an object and its class*
> 3. *The class hierarchy*
> 4. *The linkage (or relevance of an attribute) to the object*
>
> *This flexibility makes these semantically richer data models ideal vehicles for implementing the vague enterprise. However there has been very little work done on this aspect — with most efforts concentrating on the data aspect of these models.*

Sofern es um den Aspekt der persistenten Speicherung der Daten geht, ist diese Arbeit Datenbank-Agnostisch. Persistenz betrachten wir als orthogonale Systemeigenschaft; alle aktuellen Technologien zur Implementierung von Informationssystemen verfügen über eine abstrakte Datenbankschnittstelle, die eine persistente Speicherung unabhängig vom gewählten Datenbanksystem ermöglicht. Das Komponentenmodell EJB (Enterprise JavaBeans) beispielsweise ermöglicht es, Java-Objekte transparent in relationalen Datenbanken abzulegen.

Unabhängig davon, ob die reine Speicherungstechnik relational oder objektorientiert ist, läßt sich ein System also allein mit objektorientierten Methoden entwickeln. Bei der Einbettung von Fuzzy-Techniken in das Objektmodell gibt es jedoch noch viel zu tun [YG99]:

> *. . .the flexibility of these newly emerging database models, especially object-oriented and deductive object-oriented paradigm, makes it the true base for future fuzzy databases. Further work is very much needed on this area.*

Denn die Mehrzahl der Arbeiten behandelt immer noch ausschließlich das relationale Datenmodell.

Die Anforderung lautet daher:

> **Das entwickelte Architekturmodell muß mit dem objektorientierten Datenmodell kompatibel sein und es in geeigneter Weise erweitern.**

Man beachte, daß dies nicht ausschließt, daß die entwickelten Konzepte zumindest teilweise auch im Zusammenhang mit anderen Datenmodellen einsetzbar sind. Überdies muß untersucht werden, ob sich nach der durchgeführten Erweiterung des Objektmodells die oben beschriebene Abbildung auf eine relationale Datenbank gleichermaßen durchführen läßt, da diesen in der Praxis eine große Bedeutung zukommt.

3.2.6 VERARBEITUNG UNSCHARFER DATEN

Zusammen mit der Erfüllung der letzten Anforderung OBJEKTORIENTIERTES SYSTEM scheinen bereits alle Voraussetzungen gegeben zu sein, um ein Fuzzy-Informationssystem realisieren zu können. Betrachtet man jedoch die wenigen existierenden Ansätze, stellt man fest, daß sie nie für die Realisierung eines Fuzzy-Informationssystems verwendet wurden.

Weisbrod [Wei98] hält resigniert fest:

> *Although the use of fuzzy knowledge is most promising when dealing with problems that are too complex to be modeled precisely, nowadays we do not find such applications.*

Oder, mit anderen Worten: gerade für die Anwendungsfälle, für die Fuzzy-Technologien besonders geeignet sein sollen, werden sie tatsächlich gar nicht eingesetzt.

Dies liegt vor allem daran, daß die existierenden Ansätze typischerweise aus der Sicht von Datenbank-Forschern entwickelt wurden. Der Fokus liegt dabei rein auf der persistenten Speicherung von Fuzzy-Daten, entweder durch Betrachtung eines erweiterten Datenmodells, oder, wo praktische Aspekte berücksichtigt werden, auf der Erweiterung eines Datenbank-Managementsystems.

Damit ist die Aufgabe des Datenbank-Forschers erfüllt. Für den Anwendungsentwickler jedoch ist die persistente Datenspeicherung nur ein Teilaspekt eines Informationssystems. Die Speicherung von Fuzzy-Daten erfolgt nicht zweckfrei, sondern mit dem Ziel der Verwendung innerhalb eines Informationssystems. Dort findet eine Weiterverarbeitung statt, Berechnungen, Umwandlungen, und die Ergebnisse schließlich werden an Klienten oder nachgelagerte Systeme weitergeleitet.

Genau dazu ist es aber nötig, die Fuzzy-Daten mit den Methoden eines Objektes bearbeiten zu können. In [YG99] wird zu diesem Punkt angemerkt:

> *. . .expecting fuzziness in object methods in a database is not very promising at this level. However this should be a real concern for future designs.*

Angesichts dessen ist es nicht verwunderlich, daß die vorhandenen Ansätze für Fuzzy-Datenmodelle keine Anwendung finden: der wichtige Aspekt der Verarbeitung wird konsequent ignoriert. Die Verarbeitung von Attributen durch Methoden ist ein wesentlicher Bestandteil des objektorientierten Datenmodells; nur eine Erweiterung der Attribute um Fuzzy-Aspekte anzubieten, ohne eine Betrachtung der Verarbeitung dieser Fuzzy-Attribute durch ihre Methoden durchzuführen, kann nicht erfolgreich sein.

Aus dieser Beobachtung ergibt sich die nächste Anforderung:

> **Die Erweiterung des strukturellen Teils des objektorientierten Datenmodells alleine ist nicht ausreichend. Die Verarbeitung der Fuzzy-Daten durch die Objektmethoden muß ebenfalls unterstützt werden.**

Natürlich sind die konkreten Verarbeitungsschritte anwendungsspezifisch. Trotzdem lassen sich grundlegende Operationen anbieten, die für die Entwicklung einer Anwendung genutzt werden können.

3.2.7 INTEROPERABILITÄT

Die Anforderungen UNSCHÄRFE, EFFIZIENZ & EFFEKTIVITÄT (Seite 29) und PARTIELLE UNSCHÄRFE (Seite 30) zusammen zeigen, daß es notwendig ist, die Zusammenarbeit von fuzzy und nicht-fuzzy Systemteilen zu betrachten. Denn nur ein eingegrenzter Teil eines Informationssystems soll mit Fuzzy-Daten umgehen, nämlich dort, wo es für das Gesamtsystem Vorteile bringt. Folglich wird der überwiegende Teil eines Systems weiter mit klassischen, scharfen Daten arbeiten.

Dazu kommt, daß für die Entwicklung eines lauffähigen Systems viele weitere Komponenten notwendig sind: Entwicklungswerkzeuge, Übersetzer, Datenbanksysteme, Versionsverwaltungssysteme, Transaktionsmonitore, Sicherheitsverfahren, Bibliotheken für graphische Oberflächen, Ablaufumgebungen (Lastkontrolle, Ausfallsicherheit, Systemüberwachung, Clustering), Applikations-Server, Verzeichnisdienste etc.

Hinzu kommen oft existierende Systeme, mit denen eine Anwendung kommunizieren muß und die auch weiterhin nur scharfe Daten liefern oder empfangen werden — Klienten (zum Beispiel ein Web-Browser oder ein Testsystem) und Dienstgeber (Server), Datenanalysesysteme (Stichworte Data Mining und Data Warehousing) und Back-End-Systeme (zum Beispiel Enterprise Resource Planning). Mit diesen muß ein Ansatz kompatibel sein — wären Änderungen an all diesen Komponenten erforderlich, verletzt dies nicht nur die Anforderung an die Effizienz, sondern macht die Realisierung eines solchen Systems schlicht unmöglich.

Wir stellen daher die Anforderung auf:

Die neuen Fuzzy-Systeme müssen mit existierenden Systemen zusammenarbeiten können, ohne daß aufwendige Schnittstellenänderungen oder komplette Neuentwicklungen notwendig werden.

Daraus ergibt sich eine Reihe von Teilanforderungen:

- Teile einer Anwendung, die nicht mit Fuzzy-Daten umgehen können, müssen transparent mit scharfen Daten versorgt werden. Beispielsweise muß es im Falle eines Fuzzy-Attributes für einen Systemteil, der nicht mit unscharfen Daten umgehen kann, weiterhin möglich sein, wie gewohnt ein scharfes Attribut auszulesen (was natürlich mit Informationsverlust verbunden ist).
- Die Fuzzy-Technologie muß so eingebettet werden, daß sie transparent für die Systemsoftware (Betriebssysteme, Datenbankmanagementsysteme, Übersetzer), eingesetzte Middleware (Applikations-Server, Transaktionsmonitore) und Entwicklungswerkzeuge (Entwicklungsumgebungen, Versionierungssysteme, Debugger) ist.
- Externe Anwendungen und Prozesse, von denen klassische, scharfe Daten gelesen oder an die diese gesendet werden, müssen eingebunden werden können, ohne daß Änderungen an diesen Systemen notwendig werden. Es kann zwar durchaus sinnvoll sein, diese Systeme zu erweitern, um von den neuen Möglichkeiten zu profitieren, die eine Übermittlung von Fuzzy-Ergebnissen vorgelagerter Systemen bringen kann — es darf jedoch keine zwingende Voraussetzung sein.

Existierende Ansätze, die eine Spracherweiterung fordern (was eine Änderung des Übersetzers erfordert), einen eigenen Speichermanager einsetzen (was mit kommerziellen Systemen nicht machbar ist) oder eigene Notationen einführen (was inkompatibel mit Standard-Entwicklungswerkzeugen ist), scheitern bereits an dieser Anforderung.

3.2.8 KONSISTENZERHALTUNG

Wie oben gezeigt, gibt es praktisch noch keine Anwendungen, die auf der Basis von allgemeinen, fuzzyfizierten Modellen realisiert wurden. Eine Hauptursache haben wir als Grund dafür bereits ausgemacht und mittels der Anforderung VERARBEITUNG UNSCHARFER DATEN (Seite 33) beschrieben.

Betrachten wir jedoch die existierenden Ansätze, die sich bereits mit dem Aspekt der Verarbeitung unscharfer Daten beschäftigen, zeigt sich ein weiteres Problem: der Verlust der Konsistenz bei der Verarbeitung vieler, potentiell widersprüchlicher Informationen. Ausgedrückt mit den Mitteln der Fuzzy-Theorie kann dies schnell zu

sinnleeren („alles ist möglich“) oder absurden („nichts ist möglich“) Ergebnissen führen.

Genau solche Inkonsistenzen treten jedoch bei den betrachteten Anwendungsgebieten auf, beispielsweise bei Webanwendungen oder der Verarbeitung natürlichsprachiger Texte. Mehr noch, sie sind einer der Gründe, warum klassische Verfahren nicht ausreichen und stattdessen Fuzzy-Technologien überhaupt erst eingesetzt werden sollen. Gerade die Erhaltung einer definierten *Konsistenz* ist jedoch eine der Hauptaufgaben von Datenbank- und Informationssystemen.

Die entwickelten Verarbeitungsverfahren müssen also mit Inkonsistenzen umgehen können. Einen typischen Lösungsansatz beschreibt [Wei96] für den Bereich der Fuzzy-Regelungstechnik: Weisbrod unterscheidet dabei zwischen dem exakten und dem mitteilsamen Experten. Je nach vorliegendem Fall wird dann entweder das possibilistische oder evidenzgestützte Schließen eingesetzt.

Aber was passiert, wenn innerhalb einer Anwendung Experten beider Couleur zu Wort kommen? Wer entscheidet, welcher Fall für jede Aussage vorliegt, wenn große Datenbestände aus unterschiedlichen Quellen verarbeitet werden? Wenn die Aussagen nicht von Experten kommen, sondern von Expertensystemen? Von Heuristiken oder ungenauen Meßsensoren? Man sieht bereits, daß dieser Ansatz für den praktischen Einsatz alleine nicht ausreicht.

Hinzu kommt, daß Ansätze dieser Art trotz aller Sorgfalt das Grundproblem nicht lösen: tritt dennoch eine Inkonsistenz auf, erhält man auch hier wieder eine nutzlose Aussage der Form „alles ist möglich“ oder „nichts ist möglich“.

Eine weitere Anforderung muß daher lauten:

> **Kontrollierte Inkonsistenzen sind ein gewünschter Bestandteil von Fuzzy-Informationssystemen. Sie müssen von den entwickelten Verarbeitungsverfahren berücksichtigt werden, damit keine sinnleeren Ergebnisse auftreten. Diese Verarbeitungsmethoden dürfen dabei nicht von Annahmen über Eigenschaften der Informationsquellen abhängig sein.**

Gesucht ist daher eine Möglichkeit, den Grad der Konsistenz/Inkonsistenz zu steuern und dem Fuzzy-Ansatz entsprechend Werte zwischen völliger Inkonsistenz und hundertprozentiger Konsistenz zuzulassen.

Anforderung	Theorie	Technologie	Architektur
MODERNE ARCHITEKTUR		✓	✓
DURCHGÄNGIGES VERFAHREN	✓	✓	✓
UNSCHÄRFE, EFFIZIENZ & EFFEKTIVITÄT	✓	✓	✓
PARTIELLE UNSCHÄRFE		✓	✓
OBJEKTORIENTIERTES SYSTEM		✓	
VERARBEITUNG UNSCHARFER DATEN	✓	✓	✓
INTEROPERABILITÄT		✓	✓
KONSISTENZERHALTUNG	✓		

Tabelle 3.1: Anforderungen und davon betroffene Arbeitsgebiete

3.3 Zusammenfassung der Anforderungen

Tabelle 3.1 faßt die Anforderungen kurz zusammen und zeigt, welches der Gebiete *Theorie*, *Technologie* und *Architektur* jeweils von einer Anforderung betroffen ist. Diese Gebiete bilden die folgenden Teile dieser Arbeit.

Im folgenden liegt der Schwerpunkt dieser Arbeit in der Entwicklung und Evaluierung eines Ansatzes, der den oben beschriebenen Anforderungen gerecht wird. Um einen besseren Überblick zu ermöglichen, zeigen wir im nächsten Kapitel den Aufbau und die Gliederung dieser Arbeit.

Kapitel 4

Aufbau der Arbeit

Delay not, Caesar. Read it instantly.
Shakespeare, "Julius Caesar"

In diesem Kapitel geben wir einen Überblick über die Bestandteile, die für die Entwicklung von Fuzzy-Informationssystemen notwendig sind. Dies soll dem Leser eine bessere Einordnung der in den nachfolgenden Kapiteln entwickelten Konzepte in den Gesamtrahmen ermöglichen.

Abbildung 4.1 auf der nächsten Seite zeigt die Gliederung der Arbeit im graphischen Überblick. Der Hauptteil der Arbeit, das Architekturmodell für Fuzzy-Informationssysteme, unterteilt sich dabei in die Teile III bis V.

Im einzelnen haben die Teile den folgenden Inhalt:

Teil II: Grundlagen führt die Grundlagen aus der Fuzzy-Theorie und die bekannten Konzepte zur Repräsentation unscharfer Informationen ein, auf denen nachfolgend aufgebaut wird.

Teil III: Theorie enthält das formale Modell, das die theoretische Grundlage für die Entwicklung von Fuzzy-Informationssystemen darstellt. Dazu gehört die Modellierung komplexer unsicherer Informationen sowie deren konsistenzerhaltende Verarbeitung.

Teil IV: Technologie beschreibt die technischen Grundlagen für die Entwicklung von Fuzzy-Informationssystemen. Hier wird insbesondere der Aspekt betrachtet, wie das entwickelte theoretische Fuzzy-Modell in ein objektorientiertes Datenmodell eingebettet werden kann. Es erfolgt die Definition des formalen objektorientierten Fuzzy-Modells; anschließend werden Möglichkeiten zu seiner technischen Realisierung betrachtet und eine ausgewählte Variante umgesetzt.

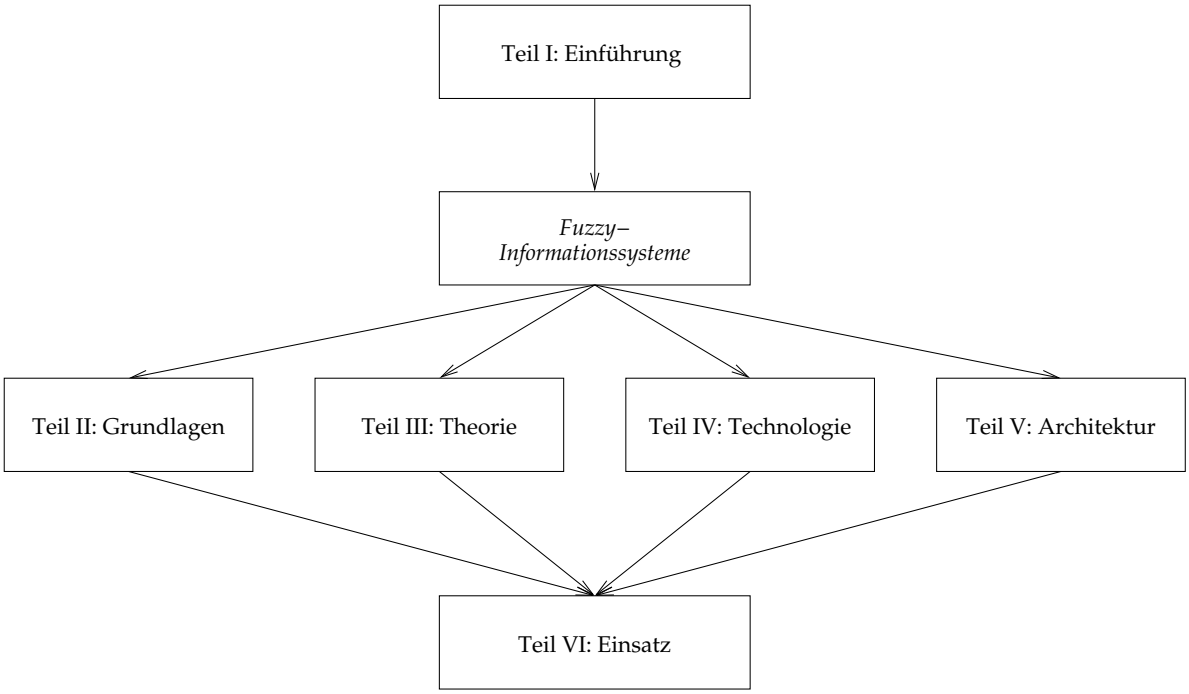


Abbildung 4.1: Gliederung der Arbeit

Teil V: Architektur geht auf den Aufbau und die Architektur von Fuzzy-Informationssystemen ein. Es wird gezeigt, wie eine klassische Referenzarchitektur um Fuzzy-Komponenten erweitert werden kann und wie dabei die Interoperabilität mit existierenden scharfen Systemen sichergestellt werden kann.

Teil VI: Einsatz schließlich dient der Validierung der hier entwickelten Konzepte. Es werden zwei Fuzzy-Anwendungen entwickelt, die die praktische Einsatzfähigkeit unseres Ansatzes demonstrieren.

Ein Resümee mit einer Zusammenfassung der geleisteten Arbeit, sowie ein Ausblick auf weiterführende Fragestellungen schließen die Arbeit ab.

Zunächst legen wir im nächsten Teil jedoch die nötigen Grundlagen, die für das Verständnis der neuen Konzepte erforderlich sind. Wie für jeden Teil zeigen wir vorab seinen Inhalt im graphischen Überblick (siehe Abbildung 4.2 auf der nächsten Seite).

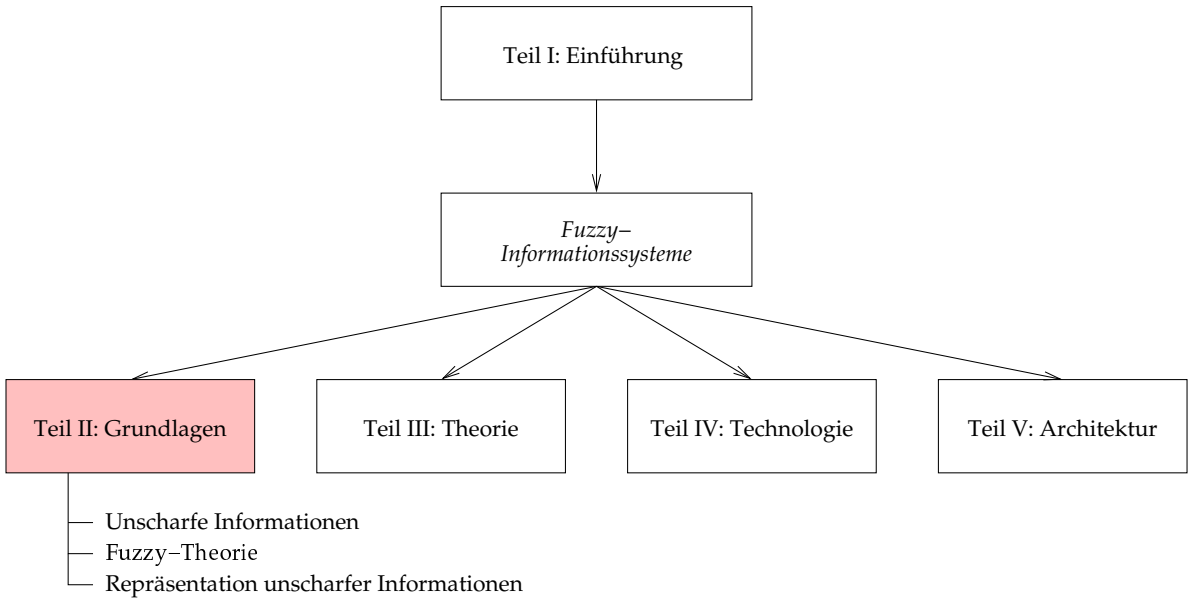


Abbildung 4.2: Gliederung Teil II

Teil II

Grundlagen von Fuzzy-Informationssystemen

Kapitel 5

Unscharfe Informationen

The universe, they said, depended for its operation on the balance of four forces which they identified as charm, persuasion, uncertainty and bloody-mindedness.
Terry Pratchett, "The Light Fantastic"

Das Phänomen der unscharfen, ungenauen oder widersprüchlichen Informationen ist nicht auf das Gebiet der Informatik beschränkt. Bereits in der antiken Philosophie wurde das Problem der Unschärfe in Paradoxien thematisiert; später tauchte es in der sprachanalytischen Philosophie und bei kognitionspsychologischen Untersuchungen wieder auf.

Ein berühmtes Beispiel aus der Antike ist das *Sorites*-Paradoxon[1] des Eubulides aus Milet: Nimmt man von einem Sandhaufen ein Sandkorn weg, bleibt es immer noch ein Sandhaufen. Daran ändert sich auch nichts, wenn man weitere Sandkörner wegnimmt. Irgendwann aber besteht er aus nur noch einem Sandkorn, und nimmt man auch dieses weg, bleibt nichts mehr übrig. Was ist mit dem Sandhaufen passiert?

Oder das *Schiff des Theseus*: Nachdem Theseus den Minotaurus getötet hatte und nach Athen zurückgekehrt war, bewahrten die dankbaren Athener sein Schiff auf. Allmählich aber begannen die Schiffsplanken zu verrotten und wurden durch neue ersetzt. Trotzdem war es immer noch das Schiff des Theseus. Irgendwann hatten die Athener jedoch alle Planken des Schiffes ersetzt. War es nun ein anderes Schiff? Und seit welchem Zeitpunkt?

[1] Abgeleitet von dem griechischen $\sigma\omega\rho\iota\tau\eta\sigma$. Dieses Paradoxon findet sich auch in anderen Formulierungen: eine Variante ist, daß ein Sandkorn noch keinen Sandhaufen ausmacht, und auch das Hinzufügen eines weiteren Sandkorns läßt noch keinen Sandhaufen entstehen — aber irgendwann hat man doch plötzlich einen Sandhaufen. Eine andere Version des Argumentes besagt, daß ein fallender Kornhaufen kein Geräusch machen kann, da der Haufen aus lauter Körnern besteht, die einzeln genommen lautlos zu Boden fallen. Die oben vorgestellte Fassung findet sich bei Eukleides von Megara.

Sowie das *Wangsche* Paradoxon: Wenn die Zahl x klein ist, ist $x+1$ auch klein. Wenn nun 1 eine kleine Zahl ist, ist auch eine Billion eine kleine Zahl, und Unendlich ist auch klein.

Das Problem aller dieser Paradoxa liegt darin, daß der Ausgangspunkt zweifelsfrei richtig ist, in der Schlußkette aber kein einzelner Schritt identifiziert werden kann, der zu dem falschen Ergebnis führt. Doch die Schlußfolgerungen sind offensichtlich falsch — würden null Sandkörner bereits einen Sandhaufen ausmachen, müßten in der Welt überall Sandhaufen herumliegen, was aber nicht der Alltagserfahrung entspricht.

Der Lösungsansatz liegt darin, graduelle Unterteilungen zwischen den beiden Wahrheitswerten *wahr* und *falsch* einzuführen. Der Begriff „Sandhaufen“ wird so zu einem *vagen* Konzept; es entsteht ein gleitender Übergang zwischen den beiden Extremen „ganz sicher ein Sandhaufen“ und „absolut kein Sandhaufen“.

Neben der Vagheit existieren noch weitere Arten imperfekter Informationen. Für die vorliegende Arbeit ist insbesondere die Unterteilung in *unsicheres* und *unscharfes* Wissen wichtig:[2]

Unsicheres Wissen. Wissen wird als unsicher bezeichnet, wenn nicht entschieden werden kann, ob es wahr oder falsch ist. Ein Beispiel für eine unsichere Aussage ist: „Das Sparschwein enthält mehr als 50,- EUR“. In diesem Beispiel läßt sich die Unsicherheit, ob die Aussage wahr oder falsch ist, durch weitere Informationen (Knacken des Sparschweines) beseitigen.

Unscharfes Wissen. Dieses auch als *vage* bezeichnete Wissen entsteht bei der Anwendung von Kategorien, für deren Verwendung keine scharfe Grenze festgelegt ist, so daß sich nicht genau über Zugehörigkeit entscheiden läßt. So existiert beispielsweise keine scharfe Grenze für die Körpergröße eines Menschen, mit der er noch unter die Kategorie *groß* fällt. Andere Beispiele sind die Kategorien *schnelle Autos*, *große Hitze* oder *schöne Frauen*.

Oft wird zusätzlich noch *ungenaues* Wissen unterschieden, das zum Beispiel bei Messungen physikalischer Größen entsteht, die ja nicht beliebig genau bestimmt werden können. Einen guten Überblick über die Historie imperfekter Informationen und deren Kategorisierung geben die Bände [Bie97] und [MS97].

Wichtig ist diese Unterteilung, da man für verschiedene Arten imperfekter Informationen unterschiedliche Logiken benötigt, um sie formal beschreiben zu können. Während sich *ungenaues Wissen*, ausgedrückt als Menge von Alternativen („die Haarfarbe ist rot oder schwarz“) oder Intervallen („die Rechengeschwindigkeit liegt zwischen 200 und 300 MIPS“) noch mit den Mitteln der klassischen Aussagenlogik beschreiben läßt, benötigt man für vage Begriffe andere Formalismen.

[2] Der *Uncertainty Thesaurus* in [MS97] unterscheidet 32 verschiedene Arten imperfekter Informationen von *Ambiguous* und *Amphibologic* über *Confused*, *Incoherent* und *Nonsensical* bis hin zu *Undecidable* und *Vague*.

Für die Modellierung von Unschärfen hat Zadeh die Theorie der Fuzzy-Mengen entwickelt [Zad65]. Sie kann auch, wie später gezeigt wurde, zur Modellierung von ungenauem und unsicherem Wissen eingesetzt werden [YOTN87].

Die Wissensarten unscharf, ungenau und unsicher werden in manchen Quellen unter dem Begriff *imperfektes Wissen* zusammengefaßt. Dies wird in der Literatur jedoch nicht einheitlich gehandhabt, oft dient „unscharfes Wissen“ als Oberbegriff.[3] Wir verwenden aus stilistischen Gründen beide Alternativen, machen bei „unscharfem Wissen“ jedoch deutlich, ob die allgemeine oder spezielle Bedeutung gemeint ist, wenn es sich nicht aus dem Zusammenhang ergibt.

[3] Als weiterer Oberbegriff wird gelegentlich auch „unvollkommenes Wissen“ vorgeschlagen. Viele lehnen dies jedoch mit dem Hinweis ab, daß unsicheres und vages Wissen genauso vollkommen ist wie scharfes Wissen, nur von einer anderen Art: *„Wenn ich Einem sage »Halte Dich ungefähr hier auf!«— kann denn diese Erklärung nicht vollkommen funktionieren? Und kann jede andere nicht auch versagen?“* (Wittgenstein, „Philosophische Untersuchungen“, 1953).

Kapitel 6

Fuzzy-Theorie

The key elements in human thinking
are not numbers but labels of fuzzy sets.
Lotfi Zadeh

In diesem Kapitel werden die für diese Arbeit nötigen Grundlagen aus dem Gebiet der Fuzzy-Theorie vorgestellt. Ziel ist es, alle verwendeten Begriffe zu definieren und zu erklären, damit diese Arbeit ohne Vorkenntnisse auf dem Gebiet der Fuzzy-Mengen gelesen werden kann. Darüber hinaus dient es der Einführung der hier verwendeten Notation, die in der Literatur nicht einheitlich gehandhabt wird.

Bei den Definitionen lehnen wir uns im wesentlichen an die Standardliteratur [KF88, KGK95] an.

6.1 Fuzzy-Mengen

Zuerst definieren wir den Begriff der Fuzzy-Menge. Die Grundidee der Fuzzy-Theorie ist, die für die Definition einer Menge verwendete charakteristische Funktion in ihrem Wertebereich zu ändern. Während bei klassischen Mengen die Bildmenge auf die Werte 0 (Element ist nicht in der Menge enthalten) und 1 (Element ist in der Menge enthalten) beschränkt ist, wird sie bei Fuzzy-Mengen auf ein kontinuierliches Intervall erweitert:

Definition 6.1.1 (Fuzzy-Menge) *Eine Fuzzy-Menge μ von Ω ist eine Funktion von der Referenzmenge Ω in das Einheitsintervall:*

$$\mu : \Omega \to [0, 1].$$

Die Menge aller Fuzzy-Mengen auf Ω sei mit $F(\Omega)$ bezeichnet.

Die charakteristische Funktion μ wird dabei als *Zugehörigkeitsfunktion* bezeichnet, ihre Werte heißen *Zugehörigkeitswerte*. Wie allgemein üblich, setzen wir die Fuzzy-Mengen $\mu_A, \mu_B, \mu_C, \ldots$ abgekürzt mit $A, B, C, \ldots$ gleich.

Mit dieser Definition lassen sich nun die klassischen, scharfen Mengen als spezielle Fuzzy-Mengen begreifen, bei denen alle in der Menge enthaltenen Elemente den Zugehörigkeitswert 1, und die anderen entsprechend den Wert 0 aufweisen.

Enthält eine Fuzzy-Menge mindestens ein Element mit dem Zugehörigkeitswert 1, bezeichnet man sie als *normalisierte* Fuzzy-Menge:

Definition 6.1.2 (Normalisierte Fuzzy-Menge) *Eine Fuzzy-Menge A erfüllt die Normalisiertheitseigenschaft, wenn gilt:*

$$\exists \omega \in \Omega : \mu_A(\omega) = 1.$$

Eine wichtige Beziehung zwischen Fuzzy-Mengen ist die der *Spezifizität:*

Definition 6.1.3 (Spezifizität) *Eine Fuzzy-Menge A ist mindestens so spezifisch wie eine Fuzzy-Menge B, geschrieben $A \preceq B$, wenn gilt:*

$$\forall \omega \in \Omega : \mu_A(\omega) \leq \mu_B(\omega).$$

Gilt zusätzlich $\mu_A \neq \mu_B$, so nennt man A spezifischer als B und schreibt $A \prec B$.

Man sieht, daß die unspezifischste Fuzzy-Menge gerade die Menge mit $\mu(\omega) = 1 \, \forall \omega \in \Omega$ darstellt. Die Spezifizität läßt sich auch als Untermengenbegriff interpretieren, dann gilt für die Fuzzy-Mengen A und B, daß A genau dann eine Untermenge von B ist ($A \subseteq B$), wenn A mindestens so spezifisch wie B ist ($A \preceq B$).

Die nächsten drei Definitionen ermöglichen drei grundlegende Mengenoperationen auch für Fuzzy-Mengen: den Schnitt, die Vereinigung und die Komplementbildung:

Definition 6.1.4 (Schnitt) *Der Schnitt zweier Fuzzy-Mengen A und B, geschrieben $A \cap B$, ist definiert als:*

$$A \cap B = \mu_{A \cap B} \stackrel{\text{def}}{=} \min\{\mu_A, \mu_B\}.$$

Definition 6.1.5 (Vereinigung) *Die Vereinigung zweier Fuzzy-Mengen A und B, geschrieben $A \cup B$, ist definiert als:*

$$A \cup B = \mu_{A \cup B} \stackrel{\text{def}}{=} \max\{\mu_A, \mu_B\}.$$

Definition 6.1.6 (Komplement) *Das Komplement einer Fuzzy-Menge A, geschrieben $\overline{A}$, ist definiert als:*

$$\overline{A} = \mu_{\overline{A}} \stackrel{\text{def}}{=} 1 - \mu_A.$$

Diese Definitionen entsprechen der ursprünglich von Zadeh vorgenommenen Erweiterung der klassischen Mengenoperationen; man nennt sie auch die *Standardoperationen*. Den auf diesen Operationen aufbauenden Zweig der Fuzzy-Theorie bezeichnet man üblicherweise als *Possibilitätstheorie* (possibility theorie) [KF88].

Dies sind jedoch nicht die einzigen Möglichkeiten, die klassischen Mengenoperationen auf Fuzzy-Mengen zu übertragen. Verallgemeinert gesprochen, müssen solche Operationen die Eigenschaften einer t-Norm beziehungsweise einer t-Conorm erfüllen:

Definition 6.1.7 (t-Norm) *Eine Funktion $\top : [0,1]^2 \to [0,1]$ heißt t-Norm, wenn für alle $a, b, c \in [0,1]$ gilt:*

$$\begin{aligned}
&(i) && \top(a,1) = a && \text{(neutrales Element)},\\
&(ii) && \top(a,b) = \top(b,a) && \text{(Kommutativität)},\\
&(iii) && a \leq b \Rightarrow \top(a,c) \leq \top(b,c) && \text{(Monotonie)},\\
&(iv) && \top(a,\top(b,c)) = \top(\top(a,b),c) && \text{(Assoziativität)}.
\end{aligned}$$

Definition 6.1.8 (t-Conorm) *Eine Funktion $\bot : [0,1]^2 \to [0,1]$ heißt t-Conorm, wenn für alle $a, b, c \in [0,1]$ gilt:*

$$\begin{aligned}
&(i) && \bot(a,0) = a && \text{(neutrales Element)},\\
&(ii) && \bot(a,b) = \bot(b,a) && \text{(Kommutativität)},\\
&(iii) && a \leq b \Rightarrow \bot(a,c) \leq \bot(b,c) && \text{(Monotonie)},\\
&(iv) && \bot(a,\bot(b,c)) = \bot(\bot(a,b),c) && \text{(Assoziativität)}.
\end{aligned}$$

Das Operationenpaar $(\top, \bot) = (\min, \max)$ ist dabei die einzige Möglichkeit einer Definition, die die Gesetze der Absorption ($\mu \cap \mu = \mu$) und der Idempotenz ($\mu \cup \mu = \mu$) erhält [KGK95].[1] Darüber hinaus gelten bei dieser Definition die Distributivgesetze

$$\begin{aligned}
&\mu_1 \cap (\mu_2 \cup \mu_3) = (\mu_1 \cap \mu_2) \cup (\mu_1 \cap \mu_3) \qquad \text{und}\\
&\mu_1 \cup (\mu_2 \cap \mu_3) = (\mu_1 \cup \mu_2) \cap (\mu_1 \cup \mu_3),
\end{aligned}$$

was wichtig für unser im nächsten Teil definiertes Repräsentationsmodell ist.

Nicht immer benötigt man das gesamte Einheitsintervall als Wertebereich. Gerade wenn menschliches Alltags- oder Erfahrungswissen mit Fuzzy-Mengen festgehalten

[1] Man beachte, daß das Operationenpaar $(\min, \max)$ nicht die Kontradiktionsgesetze ($\mu \cup \overline{\mu} = \Omega$ und $\mu \cap \overline{\mu} = \emptyset$) erfüllt. Es existiert eine bis heute andauernde Debatte über die Gültigkeit dieser Gesetze in Bezug auf Fuzzy-Mengen. Bertrand Russel beobachtete bereits 1923: *„All traditional logic habitually assumes that precise symbols are being employed. It is therefore not applicable to this terrestrial life, but only to an imaginary celestial one. The law of excluded middle is true when precise symbols are employed but it is not true when symbols are vague, as, in fact, all symbols are."* („Vagueness", Australian Journal of Philosophy, Volume I, 1923).

werden soll, erweist sich die beliebig feine Unterteilung des [0,1]-Intervalls als hinderlich — ein Anwender kann keinen semantischen Unterschied zwischen Zugehörigkeitswerten wie 0,75643 und 0,75644 ausmachen. Für diesen Fall ist es sinnvoll, anstelle des Intervalls $[0, 1]$ einen anderen, typischerweise endlichen Wertevorrat der Zugehörigkeitsfunktion zu fordern. Formal muß es sich hierbei um einen sogenannten beschränkten Verband handeln: eine partiell geordnete Menge L, in der je zwei Elemente x, y eine kleinste obere Schranke $x \sqcup y$ (Supremum) und eine größte untere Schranke $x \sqcap y$ (Infimum) besitzen und in der überdies ein kleinstes Element $l_{\min}$ und ein größtes Element $l_{\max}$ existiert. Wird ein solcher Verband als Wertebereich der Zugehörigkeitsfunktion gewählt, so spricht man von einer L-Fuzzy-Menge:

Definition 6.1.9 (L-Fuzzy-Menge) *Gegeben sei ein Verband $(L, \sqcap, \sqcup)$ mit einem kleinsten Element $l_{\min}$ und einem größten Element $l_{\max}$. Eine L-Fuzzy-Menge η von Ω ist eine Funktion von der Referenzmenge Ω in die Menge L:*

$$\eta : \Omega \to L.$$

Die Menge aller L-Fuzzy-Mengen auf Ω sei mit $L(\Omega)$ bezeichnet.

Insbesondere ist jede endliche Teilmenge des $[0, 1]$-Intervalls ein beschränkter Verband. In der Praxis verwendet man jedoch typischerweise sprechende Bezeichnungen für die Elemente von L, wie das folgende Beispiel zeigt:

Beispiel (L-Fuzzy-Menge) Eine mögliche Definition von L mit fünf unterschiedlichen Zugehörigkeitsgraden ist:

$$\begin{array}{rcll} l_{\max} & = & \text{sicher} & \\ & > & \text{gut möglich} & \\ & > & \text{vorstellbar} & \\ & > & \text{unwahrscheinlich} & \\ & > & \text{unmöglich} & = \quad l_{\min}. \end{array}$$

Diese Einteilung ist von ihrer Granularität typisch für eine, die in praktischen Anwendungen Verwendung findet. Eine dagegen mehr akademische Definition stellt die im englischen Sprachraum bekannte Sherman-Kent-Skala dar, die insgesamt 19 Abstufungen von den Werten *impossible* bis *certain* definiert.

6.2 Repräsentation von Fuzzy-Mengen

Wir wenden uns nun der Frage zu, wie Fuzzy-Mengen repräsentiert werden können. Dieser Aspekt wird im weiteren Verlauf der Arbeit wichtig, wenn die Speicherung

und Verarbeitung von Fuzzy-Mengen innerhalb von Datenbanken und Informationssystemen betrachtet wird.

Im folgenden werden zwei wesentliche Darstellungsformen erläutert: die *vertikale Repräsentation*, bei der die Definition einer Fuzzy-Menge durch geeignete Angabe ihrer charakteristischen Funktion erfolgt; sowie die *horizontale Repräsentation*, die mit sogenannten α-Schnitten arbeitet.

6.2.1 Vertikale Repräsentation

Die oben aufgeführte Definition einer Fuzzy-Menge (6.1.1) verwendet den Begriff der charakteristischen Funktion. Für solche Zugehörigkeitsfunktionen gibt es keine Einschränkungen hinsichtlich Grundmenge oder Funktionsverlauf.

Die Definition einer Zugehörigkeitsfunktion kann auf unterschiedlichem Weg erfolgen, möglich sind zum Beispiel eine explizite Auflistung aller Argument-Werte-Paare oder die Angabe einer Funktionsgleichung.

Für viele Anwendungsfälle reicht es jedoch aus, mit einer Reihe vordefinierter, parametrisierter Funktionen zu arbeiten, so daß hier nur die Art der Funktion sowie die benötigten Parameter zur Definition einer Fuzzy-Menge angegeben werden müssen. Im folgenden zeigen wir eine Reihe solcher Funktionen, die häufig in der Praxis verwendet werden.

Bei dreiecksförmigen Funktionen existiert genau ein Punkt mit dem Funktionswert 1:

Definition 6.2.1 (Parametrisierte Dreiecksfunktion) *Eine parametrisierte Dreiecksfunktion $f_{dreieck}$ mit der Funktionsvariablen x und den Parametern m, α, β, mit $\alpha, \beta > 0$, ist definiert durch:*

$$f_{dreieck}(x, m, \alpha, \beta) \stackrel{\text{def}}{=} \begin{cases} 0 & \textbf{wenn } (x \leq m - \alpha) \vee (x \geq m + \beta) \\ \dfrac{x - (m - \alpha)}{\alpha} & \textbf{wenn } (m - \alpha) < x \leq m \\ 1 - \dfrac{x - m}{\beta} & \textbf{wenn } m < x < (m + \beta). \end{cases}$$

Möchte man einen plateauartigen Funktionsverlauf erhalten, greift man zu einer Trapezfunktion:

Definition 6.2.2 (Parametrisierte Trapezfunktion) *Eine parametrisierte Trapezfunktion f_{trapez} mit der Funktionsvariablen x und den Parametern m_1, m_2, α, β, mit $\alpha, \beta > 0$, ist*

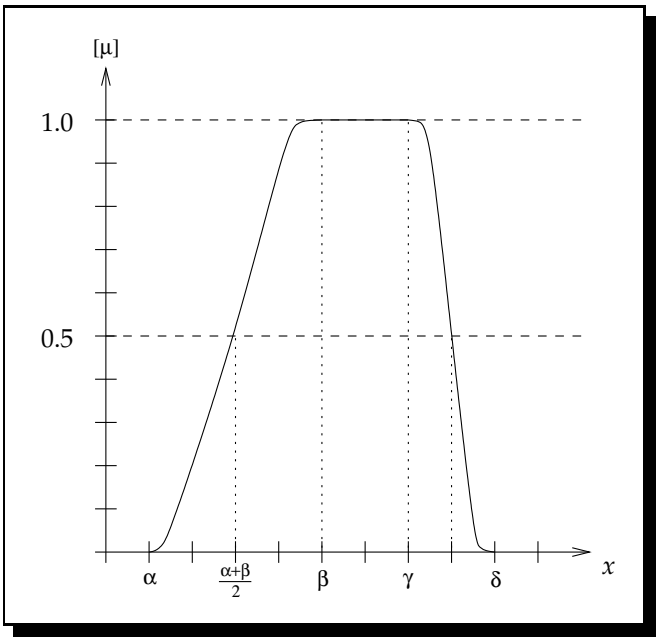


Abbildung 6.1: Parametrisierte s/z-Funktion zur Beschreibung einer Fuzzy-Menge

definiert durch:

$$f_{trapez}(x, m_1, m_2, \alpha, \beta) \stackrel{\text{def}}{=} \begin{cases} 0 & \textbf{wenn } (x \leq m_1 - \alpha) \vee (x \geq m_2 + \beta) \\ \dfrac{x - (m_1 - \alpha)}{\alpha} & \textbf{wenn } (m_1 - \alpha) < x < m_1 \\ 1 & \textbf{wenn } m_1 \leq x \leq m_2 \\ 1 - \dfrac{x - m_2}{\beta} & \textbf{wenn } m_2 < x < (m_2 + \beta). \end{cases}$$

Sowohl die Trapez- als auch die Dreiecksfunktion zeichnen sich durch einen scharfen Übergang an den Wendepunkten aus. Wünscht man einen gleitenden Übergang im Kurvenverlauf, kann eine s/z-Funktion verwendet werden (siehe Abbildung 6.1):

Definition 6.2.3 (s/z-Funktion) *Eine s-Funktion mit der Funktionsvariablen x und den Parametern α, γ, mit $\alpha < \gamma$ ist definiert durch:*

$$s(x, \alpha, \gamma) \stackrel{\text{def}}{=} \begin{cases} 0 & \textbf{wenn } x \leq \alpha \\ 2 \cdot \left(\dfrac{x - \alpha}{\gamma - \alpha}\right)^2 & \textbf{wenn } \alpha < x \leq \frac{\alpha+\gamma}{2} \\ 1 - 2 \cdot \left(\dfrac{x - \gamma}{\gamma - \alpha}\right)^2 & \textbf{wenn } \frac{\alpha+\gamma}{2} < x \leq \gamma \\ 1 & \textbf{sonst} \end{cases}$$

Eine z-Funktion mit der Funktionsvariablen x und den Parametern α, γ, mit $\alpha < \gamma$ ist dann definiert durch:

$$z(x, \alpha, \gamma) \stackrel{\text{def}}{=} 1 - s(x, \alpha, \gamma)$$

Damit können wir nun eine s/z-Funktion mit der Funktionsvariablen x und den Parametern $\alpha, \beta, \gamma, \delta$, mit $\alpha < \beta \leq \gamma < \delta$, definieren:

$$s/z(x, \alpha, \beta, \gamma, \delta) \stackrel{\text{def}}{=} \begin{cases} s(x, \alpha, \beta) & \textbf{wenn } x \leq \beta \\ z(x, \gamma, \delta) & \textbf{sonst.} \end{cases}$$

Als weitere Darstellungsmöglichkeit für Fuzzy-Mengen betrachten wir im folgenden die sogenannte *horizontale Repräsentation*, die sich besonders für die Verwendung innerhalb eines Informationssystems eignet.

6.2.2 Horizontale Repräsentation

Bei der horizontalen Repräsentation einer Fuzzy-Menge wird eine Reihe neuer Mengen gebildet, die sogenannten α-Schnitte. Jeder α-Schnitt enthält dabei die Elemente der Referenzmenge, die mindestens mit dem Zugehörigkeitsgrad α in der Fuzzy-Menge enthalten sind:

Definition 6.2.4 (α-Schnitt) *Es sei $\mu \in F(\Omega)$ und $\alpha \in [0, 1]$. Dann ist die Menge*

$$[\mu]_\alpha \stackrel{\text{def}}{=} \{\omega \in \Omega | \mu(\omega) \geq \alpha\}$$

der α-Schnitt von μ. Der strenge α-Schnitt ist durch die Menge

$$[\mu]_{>\alpha} \stackrel{\text{def}}{=} \{\omega \in \Omega | \mu(\omega) > \alpha\}$$

definiert.

Veranschaulicht entspricht der α-Schnitt einer Fuzzy-Menge A der Projektion des Funktionsgraphen von μ_A auf und oberhalb der Geraden $y = \alpha$ auf die x-Achse.

Beispiel (α-Schnitt) Ein Beispiel ist in Abbildung 6.2 auf der nächsten Seite gezeigt. Bei der dreiecksförmigen Fuzzy-Menge μ entspricht der α-Schnitt dem Bereich auf der x-Achse, für die die Werte von μ größer oder gleich α sind.

Zwei spezielle α-Schnitte sind der *Kern* und die *Basis* einer Fuzzy-Menge:

Definition 6.2.5 (Kern und Basis) *Der Kern μ_1 einer Fuzzy-Menge μ ist der α-Schnitt $[\mu]_\alpha$ mit $\alpha = 1$:*

$$\mu_1 \stackrel{\text{def}}{=} \{\omega \in \Omega | \mu(\omega) = 1\}$$

Die Basis $\mu_{>0}$ einer Fuzzy-Menge ist der strenge α-Schnitt $[\mu]_{>\alpha}$ mit $\alpha = 0$:

$$\mu_{>0} \stackrel{\text{def}}{=} \{\omega \in \Omega | \mu(\omega) > 0\}.$$

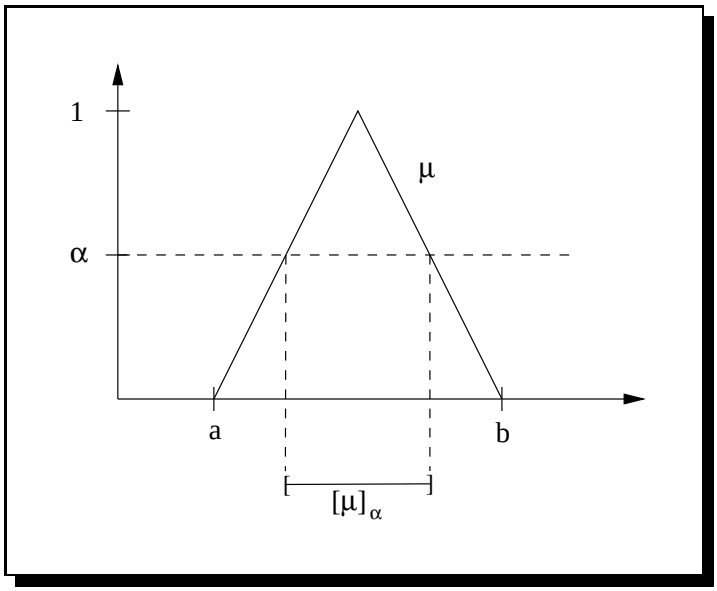


Abbildung 6.2: α-Schnitt $[\mu]_\alpha$ einer Fuzzy-Menge μ

Es gelten die folgenden Eigenschaften für α-Schnitte [KGK95]:

Satz 6.2.6 (α-Schnitte) *Für $\mu \in F(\Omega), \alpha \in [0,1]$ und $\beta \in (0,1]$ gilt:*

$$\begin{aligned} &(a) \quad [\mu]_0 = \Omega, \\ &(b) \quad \alpha < \beta \Rightarrow [\mu]_\alpha \supseteq [\mu]_\beta, \\ &(c) \quad \bigcap_{\alpha:\alpha<\beta} [\mu]_\alpha = [\mu]_\beta. \end{aligned}$$

Tatsächlich läßt sich eine Fuzzy-Menge anhand ihrer α-Schnitte vollständig rekonstruieren, wie der folgende *Repräsentationssatz* [KGK95] zeigt:

Satz 6.2.7 (Repräsentationssatz) *Es sei $\mu \in F(\Omega)$. Dann ist*

$$\mu(\omega) = \sup\{\alpha \in [0,1] | \omega \in [\mu]_\alpha\}.$$

Trägt man die einzelnen α-Schnitte $[\mu]_\alpha$ mit $y = \alpha$ auf, läßt sich die Fuzzy-Menge geometrisch als obere Einhüllende ihrer α-Schnitte rekonstruieren.

Die horizontale Repräsentation von Fuzzy-Mengen hat vor allem für die informationstechnische Verarbeitung große Bedeutung erlangt, da sich jede Fuzzy-Menge beliebig genau durch eine endliche Anzahl ihrer α-Schnitte beschreiben läßt. Genauer gesagt gilt folgendes Lemma:

Lemma 6.2.8 *Zu jeder Fuzzy-Menge $\mu \in F(\Omega)$ und jedem $\varepsilon > 0$ gibt es eine endliche Teilmenge $\{\alpha_0, \ldots, \alpha_N\} \subseteq [0,1]$ mit*

$$|\mu(\omega) - \max\{\alpha_i | 0 \leq i \leq N \wedge \omega \in [\mu]_{\alpha_i}\}| \leq \varepsilon.$$

Beweis. *Wähle $\alpha_i = \varepsilon \cdot i$ für $0 \leq i \leq N := \lceil \frac{1}{\varepsilon} \rceil$. Für jedes $\omega \in \Omega$ gilt dann*

$$\mu(\omega) \in [\mu]_{\alpha_i} \Leftrightarrow \mu(\omega) \geq \alpha_i \Leftrightarrow 0 \leq i \leq \left\lfloor \frac{\mu(\omega)}{\varepsilon} \right\rfloor$$

also gilt

$$\max\{\alpha_i | 0 \leq i \leq N \wedge \omega \in [\mu]_{\alpha_i}\} = \varepsilon \cdot \left\lfloor \frac{\mu(\omega)}{\varepsilon} \right\rfloor$$

und somit

$$|\mu(\omega) - \max\{\alpha_i | 0 \leq i \leq N \wedge \omega \in [\mu]_{\alpha_i}\}| \leq \varepsilon.$$

■

Für praktische Anwendungen werden wir, wie unter Definition 6.1.9 auf Seite 52 bereits angedeutet, einen endlichen beschränkten Verband L verwenden und eine L-Fuzzy-Menge $\mu \in L(\Omega)$ durch ihre α-Schnitte $A_\alpha := [\mu]_\alpha$ für $\alpha \in L$ repräsentieren. Das entstehende Mengensystem A_α erfüllt, wie man leicht sieht, die Konsistenzbedingungen

(a) $A_{\alpha_{\min}} = \Omega$ $(\alpha_{\min} := l_{\min})$,
(b) $\alpha < \beta \Rightarrow A_\alpha \supseteq A_\beta$ (Monotonie).

Umgekehrt induziert jedes Mengensystem $\{A_\alpha\}_{\alpha \in L}$, welches (a) und (b) erfüllt, eine L-Fuzzy-Menge $\mu \in L(\Omega)$ vermöge

(c) $\mu_A(\omega) = \sup\{\alpha \in L | \omega \in A_\alpha\}$

(die rechte Menge ist wegen (a) nicht leer), und für diese L-Fuzzy-Menge gilt

(d) $[\mu_A]_\alpha = A_\alpha \qquad \forall \alpha \in L.$

Damit haben wir die wesentlichen Repräsentationsmöglichkeiten für Fuzzy-Mengen beschrieben. Im folgenden betrachten wir den Übergang von einer Fuzzy-Menge zu einer scharfen Menge, *Defuzzifizierung* genannt.

6.3 Defuzzifizierung

Innerhalb einer Anwendung kann es notwendig werden, einen unscharfen Wert in Form einer Fuzzy-Menge in einen scharfen Wert umzuwandeln. Dies ist beispielsweise der Fall, wenn Ergebnisse an ein nachgelagertes System weitergeleitet werden müssen, das nicht mit unscharfen Daten umgehen kann, wenn einem Benutzer ein scharfes Ergebnis präsentiert werden soll, oder wenn Server- oder Bibliotheksfunktionen benutzt werden müssen, die nur mit scharfen Parametern arbeiten.

Diese Umwandlung einer Fuzzy-Menge in einen einzelnen scharfen Wert oder eine klassische Menge wird als *Defuzzifizierung* (defuzzification) bezeichnet. Die dafür notwendigen Verfahren sind insbesondere im Bereich der Fuzzy-Regelungstechnik verbreitet, wo nach unscharfen Berechnungen ein scharfer Stellwert zur Steuerung abgeleitet werden muß.

Eine einfache Möglichkeit zur Defuzzifizierung ist die sogenannte *Maximum-Methode:*

Definition 6.3.1 (Maximum-Defuzzifizierung) *Es sei $\mu \in F(\Omega)$. Die scharfe Menge M_μ wird gebildet, indem alle Punkte mit dem höchsten Zugehörigkeitsgrad auswählt werden:*

$$M_\mu = \{\omega \in \Omega | \mu(\omega) = \max_{\omega' \in \Omega}(\mu(\omega'))\}.$$

Ein solches Maximum existiert nicht immer, insbesondere nicht bei Fuzzy-Mengen über unendlich großen Grundmengen. Für den praktischen Einsatz spielt dies jedoch keine Rolle, da dort nur Fuzzy-Mengen auf endlichen Domänen zum Einsatz kommen, für die immer mindestens ein Maximum existiert. Ist die Fuzzy-Menge leer, gilt also $\mu(\omega) = 0 \, \forall \omega \in \Omega$, ist auch ihre Defuzzifizierung leer. Für Fuzzy-Mengen, die in Form von α-Schnitten repräsentiert sind, läßt sich die zu einem μ gehörige defuzzifizierte Menge M_μ leicht bestimmen, denn sie entspricht gerade dem α-Schnitt $[\mu]_\alpha$ mit dem größten $\alpha > 0$, für das $[\mu]_\alpha$ nichtleer ist.

Man beachte aber, daß diese Ergebnismenge mehr als ein Element enthalten kann. Dies kann durchaus erwünscht sein, etwa wenn die Fuzzy-Menge ein mehrwertiges Attribut repräsentiert. Soll dagegen ein einzelner Wert, wie zur Entscheidungsfindung, bestimmt werden, muß aus dieser Menge in einem weiteren Schritt ein geeigneter Wert ausgewählt werden. Aus der Regelungstechnik bekannte Verfahren sind zum Beispiel die Wahl des kleinsten *(left of maximum, LOM)* oder größten *(right of maximum, ROM)* Wertes [Bie97]. Darüber hinaus gibt es aufwendigere Verfahren, wie eine Mittelwertbildung oder die Schwerpunktmethode. Diese machen jedoch nur Sinn, wenn die Grundmenge einen Vektorraum aufspannt, was bei Fuzzy-Informationssystemen in vielen Fällen nicht gegeben ist. So ließe sich zwar eine Fuzzy-Menge für die Körpergröße einer Person mit der Schwerpunktmethode defuzzifizieren, nicht aber eine, deren Grundmenge mögliche Variablentypen (int, float, bool, ...) in einem Quellcodeanalysesystem repräsentiert. Auch ist die zugrundeliegende Idee dieser Verfahren, etwa nur möglichst sanfte Änderungen einer Ventilsteuerung durchzuführen, typischerweise nicht auf Fuzzy-Informationssysteme übertragbar. Und selbst in den Fällen, wo der Einsatz von Verfahren wie Mittelwert- oder Schwerpunktbildung möglich ist, kann es zu einer Reihe unerwünschter Effekte kommen, wie dem *Problem des verhungernden Esels.*[2]

[2] Der Esel verhungert, weil er sich nicht entscheiden kann, zu welchem der beiden gleichweit von ihm entfernten Heuhaufen er gehen soll. Auf Fuzzy-Mengen übertragen bedeutet dies, daß durch die angesprochenen Verfahren ein Wert ausgewählt werden kann, der zwar genau zwischen zwei Maxima liegt, aber selbst nur einen Zugehörigkeitsgrad von 0 aufweist, also nicht als Ergebnis in Frage kommt.

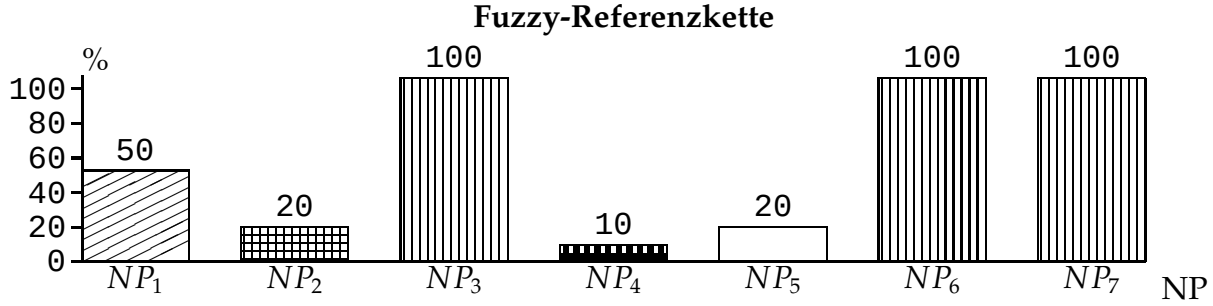


Abbildung 6.3: Beispiel-Fuzzy-Menge zur Defuzzifizierung

Die oben vorgestellte Maximum-Methode, wenn erforderlich kombiniert mit der Auswahl des kleinsten Elements (LOM), hat sich im Kontext von Fuzzy-Informationssystemen als ein generell gut geeignetes und effizient realisierbares Verfahren herausgestellt. In Einzelfällen kann es jedoch durchaus sinnvoll sein, für eine konkrete Anwendung eines der anderen bekannten Defuzzifizierungsverfahren, wie sie etwa in [KGK95, Bie97] vorgestellt werden, zu verwenden.

Beispiel (Maximum-Defuzzifizierung) Innerhalb eines Fuzzy-Textanalysesystems, das später noch im Detail beschrieben wird, repräsentiert eine Fuzzy-Menge eine sogenannte Referenzkette von Nominalphrasen (NPs). Dabei gibt der Zugehörigkeitsgrad an, mit welcher Sicherheit eine konkrete Nominalphrase in der jeweiligen Kette enthalten ist. Ein Beispiel für eine solche Referenzkette zeigt Abbildung 6.3. Zur Ausgabe eines scharfen Ergebnisses muß die Fuzzy-Referenzkette defuzzifiziert werden. Für das gezeigte Beispiel liefert die Maximum-Methode die klassische Menge $\{\mathrm{NP}_3, \mathrm{NP}_6, \mathrm{NP}_7\}$. Da Referenzketten mehrwertige Attribute sind, muß hieraus kein einzelnes Ergebnis ausgewählt werden, das Ergebnis läßt sich also direkt weiterverwenden.

Kapitel 7

Repräsentation unscharfer Informationen durch Fuzzy-Mengen

'It's dark,' he said.
'Yes,' said Ford Prefect, 'it's dark.'
Douglas Adams, "The Hitch Hiker's Guide to the Galaxy"

Bis jetzt haben wir die Definition und Repräsentation von Fuzzy-Mengen betrachtet, ohne genauere Angaben über ihre Interpretation, also ihre Semantik, zu machen. In diesem Kapitel werden wir daher darauf eingehen, wie die in Kapitel 5 beschriebenen Arten imperfekter Daten mit Hilfe von Fuzzy-Mengen ausgedrückt werden können.

7.1 Interpretation von Fuzzy-Mengen

Ein häufig in der Literatur angeführtes Beispiel zur Motivation von Fuzzy-Mengen ist die Definition des Konzeptes „große Person". Bei diesem Begriff handelt es sich um eine natürlichsprachige Kategorisierung, die nicht über scharfe Grenzwerte verfügt. Die Definition einer klassischen Menge, wie in Abbildung 7.1 auf der nächsten Seite gezeigt, ist daher intuitiv unplausibel, da auch nur ein Millimeter unterhalb des definierten Schwellwertes von 1,80 m eine Person in keinem Fall als groß gilt, ab Erreichen des Schwellwertes jedoch uneingeschränkt mit dem Konzept „groß" kompatibel ist.

Durch das Ziehen einer solchen scharfen Grenze läßt sich Vagheit immer beseitigen—man kann bereits erahnen, welche Auswirkungen dies auf Informationssysteme hat, die gezwungen sind, mit einer solchen Repräsentationsform zu arbeiten. Weder sind die innerhalb von klassischen, scharfen Systemen gespeicherten Daten intuitiv, noch

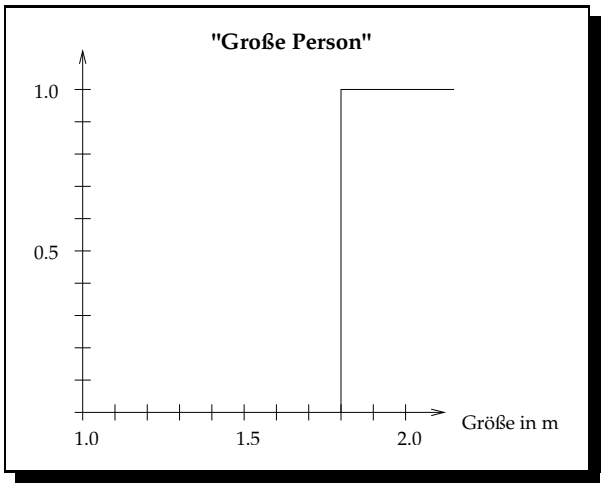


Abbildung 7.1: Scharfe Menge für das Konzept „große Person“

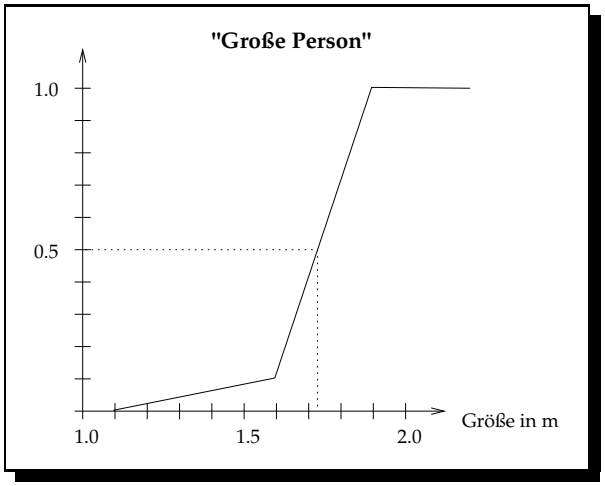


Abbildung 7.2: Fuzzy-Menge für das Konzept „große Person“

stellen sie eine korrekte Beschreibung des repräsentierten Sachverhaltes dar, was zu massiven Informationsverlusten und Bedeutungsverschiebungen führen kann. Insbesondere bei einer Weiterverarbeitung solcher Informationen muß ein hoher Aufwand getrieben werden, damit es durch die erforderliche scharfe Grenzziehung nicht zu willkürlichen und sachlich falschen Ergebnissen kommt. Während sich das Konzept „groß“ vielleicht noch nicht unmittelbar als systemkritisch aufdrängt, ist das bei anderen Begriffen schon näher liegend: eine Bank hat großes Interesse an einer korrekten Beschreibung der *Kreditwürdigkeit* einer Person, ein Umweltinformationssystem muß glaubwürdige Daten über die *Umweltbelastung* einer Region vermitteln, ebenso ein Verkehrsleitsystem über mögliche *Staugefahr*, Business-to-Business Marktplätze brauchen Informationen über die *Zuverlässigkeit* von Geschäftspartnern, Elektronische Bibliotheken über die *Relevanz* aufgespürter Textstellen.

Im Fall der „großen Person“ handelt es sich um ein *vages* Konzept. Dies läßt sich elegant mittels einer Fuzzy-Menge ausdrücken, wie in Abbildung 7.2 gezeigt wird.

In diesem Fall gibt der Zugehörigkeitsgrad an, wie kompatibel ein Wert aus der Grundmenge (hier die Körpergröße) mit dem Begriff „große Person“ ist. Es werden also Abstufungen zwischen den beiden Extremen 0% und 100% ermöglicht. Eine Person mit 1,72 m Größe wird so zu einem Grad von ungefähr 0,5 mit dem unscharfen Begriff „große Person“ kompatibel.

Ein anderer Fall liegt vor, wenn eine *unsichere* Information modelliert werden soll. Folgendes Beispiel möge dies illustrieren: ich bekomme ein Sparschwein geschenkt, mit dem Hinweis, es enthalte „viel Geld“. Der genaue Betrag wird mir jedoch nicht genannt. Aufgrund meiner Kenntnis des Schenkenden und meiner allgemeinen Lebenserfahrung im Umgang mit Sparschweinen kann ich nun annehmen, mindestens 100,- EUR erhalten zu haben. Dies ließe sich wieder mit einer scharfen Menge modellieren:

$$\chi_{\text{viel Geld}}(x) = \begin{cases} 1 & \textbf{wenn } x \geq 100, \\ 0 & \textbf{sonst.} \end{cases}$$

Doch auch hier komme ich in ähnliche Probleme wie im Beispiel mit dem unscharfen Begriff „große Person“: knacke ich das Sparschwein und stelle fest, daß sich nur 99,99 EUR darin befinden, ist dies nach obiger Definition bereits nicht mehr „viel Geld“, was nicht dem intuitiven Verständnis entspricht. Wie oben ist es aber möglich, mit Hilfe einer Fuzzy-Menge einen gleitenden Übergang zu definieren. Im Vergleich zum ersten Beispiel ist mein Wissensstand jedoch ein völlig anderer; in diesem Fall *weiß ich*, daß das Sparschwein „viel Geld“ enthält und benutze die Fuzzy-Menge, um die *Möglichkeit* eines bestimmten Betrages im Sparschwein mittels eines bestimmten Grades zu modellieren. Der Betrag von 100,- EUR wird so zu einem Grad von 1 möglich, 10,- EUR sind unmöglich, und dazwischen entsteht ein gleitender Übergang möglicher Sparschweinwerte. Im vorhergehenden Beispiel dagegen war die exakte Körpergröße bekannt, und die Fuzzy-Menge repräsentierte den Zugehörigkeitsgrad zu dem vagen Konzept „große Person“.

In diesem Beispiel entspricht die Verwendung der Fuzzy-Menge einer sogenannten *Possibilitätsverteilung*. Diese unterscheiden sich von Fuzzy-Mengen nur in ihrer Interpretation: eine Possibilitätsverteilung μ stellt eine *elastische Beschränkung* der Grundmenge dar. Die Fuzzy-Menge μ_C steht mithin für ein unscharfes Konzept (Prädikat) C_μ und ein konkreter Wert $\mu(\omega)$ gibt den Grad der Kompatibilität von C_μ mit ω an. Es gilt die folgende Interpretation [KGK95]:

$\mu(\omega) = 0$	dann ist ω unmöglich in C_μ,
$\mu(\omega) = 1$	heißt, ω ist mit C_μ ohne Einschränkung kompatibel; und
$\mu_x(\omega) \in (0, 1)$	gibt den Kompatibilitätsgrad an, mit dem ω in C_μ gilt.

Für obiges Beispiel würde die Fuzzy-Menge das Prädikat „viel Geld“ modellieren, μ zeigt so mit einem Wert $\mu(10) = 0$ an, daß der Betrag von 10,- EUR nicht für ein viel

Geld enthaltendes Sparschwein in Frage kommt. Man beachte, daß der Fall $\mu(\omega) = 1$ *nicht* bedeutet, daß C_μ für ω gilt — es gibt lediglich nichts, was dagegen spricht. Diese Interpretationsweise schließt also Werte aus; man bezeichnet solchermaßen repräsentierte Informationen auch als *negative* Informationen [Men00], da Werte ω mit $\mu(\omega) = 0$ sicher ausgeschlossen, für Werte mit $\mu(\omega) = 1$ dagegen keinerlei Zusagen gemacht werden.

7.2 Epistemische Interpretation von Fuzzy-Mengen

Tatsächlich besteht zwischen den beiden Interpretationsformen für Fuzzy-Mengen ein enger semantischer Zusammenhang. Jede Fuzzy-Menge, die die Normalisiertheitseigenschaft (siehe Definition 6.1.2 auf Seite 50) erfüllt, kann auch als Possibilitätsverteilung interpretiert werden. In diesem Fall wird die Fuzzy-Menge als unscharfe Beschreibung eines (realen oder abstrakten) existierenden Sachverhaltes angesehen. An obigem Beispiel verdeutlicht, können wir eine Person „Tux“ mit einer Körpergröße von 1,75 m zu einem Grad von ungefähr 0,75 als kompatibel mit dem unscharfen Begriff „große Person“ ansehen. Drehen wir nun aber die Sichtweise um und postulieren, daß Tux eine große Person *ist*, gibt uns die Fuzzy-Menge den Grad an, mit dem die *Möglichkeit* besteht, daß Tux eine Größe von gerade 1,75 m hat.

Man spricht in diesem Fall von der *epistemischen Interpretation*[1] einer Fuzzy-Menge. Dabei gibt die Fuzzy-Menge eine *unscharfe Restriktion* einer Variablen auf einer Grundmenge an.

Diese Interpretationsform ist auch bedeutsam für die hier betrachtete Anwendung von Fuzzy-Mengen innerhalb von Informationssystemen: denn dort werden eine Reihe von Techniken, wie Objekte, Klassen und Beziehungen, zur Beschreibung realer oder abstrakter Konzepte eingesetzt. Diese Beschreibungsmöglichkeiten sollen hier erweitert werden, um auch unscharfe Zustände der modellierten Konzepte ausdrücken zu können.[2]

In der Literatur wird allgemein die Forderung erhoben, daß eine Possibilitätsverteilung die Normalisiertheitseigenschaft erfüllen muß. Dies beruht auf der Überlegung [KGK95],

> *daß es für jede einen existierenden Objektzustand $x_0 \in X$ unscharf beschreibende Possibilitätsverteilung mindestens ein Element x der Referenzmenge geben muß, für das $x = x_0$ ohne jede Einschränkung für möglich gehalten wird. Andernfalls wäre die gewählte Beschreibung in sich nicht konsistent.*

[1] **Epistemologie:** <f.; -, -en; unz.> *Erkenntnislehre, Lehre vom Wissen* [zu griechisch *episteme* „Wissenschaft“ + *...logie*] (Wahrig)

[2] Beide Betrachtungsformen lassen sich durchaus in einer Anwendung gleichzeitig verwenden, wie wir in Teil IV sehen werden. Wichtig ist aber, daß in jedem Moment klar ist, welche der beiden Interpretationsformen in einem bestimmten Kontext vorliegt.

Diese Voraussetzung bezeichnet man im Kontext der Fuzzy-Theorie auch als *Closed World Assumption*, da der zwar unbekannte, aber existierende Objektzustand mit Sicherheit in der Menge der prinzipiell möglichen Objektzustände enthalten ist.

Diese Forderung ist durchaus nachvollziehbar, wenn man an einer theoretischen Betrachtung der Eigenschaften von Possibilitätsverteilungen interessiert ist. In unserem Fall interessiert jedoch der praktischen Einsatz innerhalb eines Informationssystems, in dem die imperfekten Daten ständigen Änderungen und Verarbeitungsschritten unterworfen sind. Die Fähigkeit, mit dabei auftretenden Inkonsistenzen umgehen zu können, soll, wie in Kapitel 1 beschrieben, gerade die Stärke und den Vorteil von Fuzzy-Informationssystemen im Vergleich zu klassischen Realisierungen ausmachen. Im engen Kontext der Entwurfsanwendungen hat es sich bereits als sinnvoll herausgestellt, auf die Normalisiertheitsforderung zu verzichten [Wit95, Bos96].

Und schließlich ist diese Forderung auch nicht kompatibel mit der Komplexität der realen Welt; so sehr man es sich auch wünschen mag, daß es immer eine eindeutige Antwort geben solle, so wenig ist dies in der Praxis tatsächlich möglich.

Im weiteren Verlauf der Arbeit gehen wir immer von der epistemischen Interpretationsweise einer Fuzzy-Menge aus, wenn nichts anderes vermerkt ist. Wir verzichten jedoch auf die Normalisiertheitseigenschaft, daher sprechen wir im folgenden nicht von Possibilitätsverteilungen, sondern nur von Fuzzy-Mengen.

Dies beendet Teil II dieser Arbeit. Im nächsten Teil stellen wir unser formales Modell für die Entwicklung von Fuzzy-Informationssystemen vor (siehe Abbildung 7.3 auf der nächsten Seite).

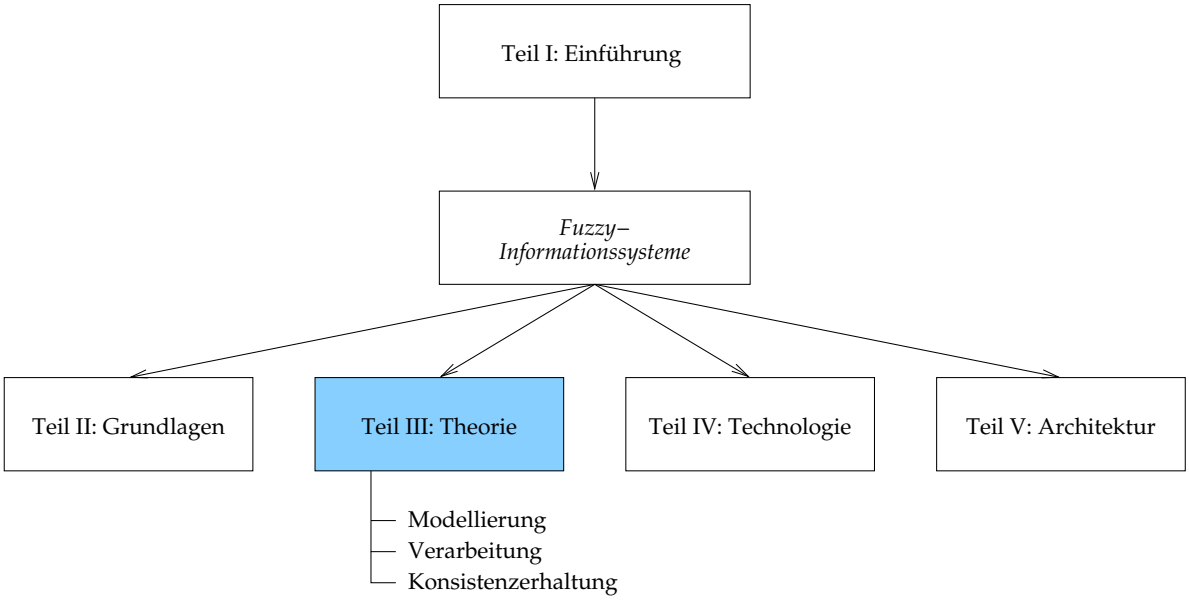


Abbildung 7.3: Gliederung Teil III

Teil III

Theorie von Fuzzy-Informationssystemen

Kapitel 8

Modellierung imperfekten Wissens

Ah! Dear Watson, now we enter the mystic room of wizardry,
where even the most brilliant of all logic minds might fail.
Sir Arthur Conan Doyle

Im letzten Kapitel haben wir gezeigt, wie sich mit Hilfe von Fuzzy-Mengen imperfekte Begriffe repräsentieren lassen. Damit existiert bereits die Basis für die Möglichkeit, unscharfes und unsicheres Wissen zu speichern. Eine einzelne Fuzzy-Menge erlaubt jedoch nur die Beschreibung jeweils einer einzelnen imperfekten Information.

Für den praktischen Einsatz ist dies aber noch nicht ausreichend. Hier kann eine vollständige Beschreibung eines Attributes oder eines Objektes schnell mehr als einen imperfekten Begriff erfordern. Beispiele sind die Spezifizierung einer Farbe *(„hell und freundlich“)*, eines Automotors *(„(sparsam und umweltfreundlich) oder elektrisch“)* oder eines Baumaterials *(„standardkonform und kostengünstig und schnell verfügbar“)*.

Solche *komplexen* imperfekten Informationen lassen sich durch logische Verknüpfung der einzelnen Ausdrücke mit den Operationen *und*, *oder* und *nicht* modellieren. Erste Ansätze für die Repräsentation solcher natürlichsprachiger Ausdrücke durch Fuzzy-Mengen hat Zadeh bereits 1978 mit seiner Semantik-Theorie PRUF entwickelt [Zad78].

Für den praktischen Einsatz solcher komplexer Fuzzy-Ausdrücke hat Boss ein Modell basierend auf einer konjunktiven Normalform von Fuzzy-Ausdrücken vorgeschlagen und im Bereich der Entwurfsanwendungen erfolgreich eingesetzt [Bos96]. Für diese Arbeit haben wir dieses Modell grundsätzlich überarbeitet und verallgemeinert, so daß es sich für eine größere Bandbreite von Anwendungen eignet. Zusätzlich stellen wir eine Reihe neuer Konzepte vor, wie die formale Behandlung semantischer Abhängigkeiten.

Das im folgenden entwickelte theoretische Modell wird bereits in Hinblick auf seinen praktischen Einsatz formuliert. So verzichten wir schon im formalen Modell auf Eigenschaften, die sich nicht oder nur sehr ineffizient realisieren lassen. Überdies stellt

sich an dieser Stelle bereits die Frage, wie ein solches Modell in ein OBJEKTORIENTIERTES SYSTEM (siehe Seite 32) eingebettet werden kann. Boss [Bos96] beispielsweise geht dazu von dem Datenmodell eines objektorientierten Datenbanksystems aus, um es schrittweise um Fuzzy-Konzepte zu erweitern. Dies ist auch der Ansatz der meisten in Kapitel 2 vorgestellten Arbeiten. Der von uns verfolgte Ansatz ist dagegen, die theoretischen Aspekte der Modellierung separat von ihrer Einbettung in eine objektorientierte Umgebung zu formulieren. Diese Vorgehensweise begründet sich mit den Argumenten, die in den Anforderungen MODERNE ARCHITEKTUR (Seite 27) und OBJEKTORIENTIERTES SYSTEM (Seite 32) vorgetragen wurden:

- Das Datenbanksystem ist nur Teil einer einzelnen Stufe eines Informationssystems (vergleiche Abbildung 1.1 auf Seite 7). Die Erweiterung des Datenmodelles einer objektorientierten Datenbank allein ist also nicht ausreichend, um komplette Fuzzy-Informationssysteme realisieren zu können.

- Zentraler Bestandteil eines Fuzzy-Informationssystems ist jedoch die objektorientierte Programmiersprache und ihre Ablaufumgebung, auf deren Basis eine Anwendung implementiert wird. Es muß daher eine Möglichkeit aufgezeigt werden, wie solche Umgebungen um Fuzzy-Technologien erweitert werden können (vergleiche Abschnitt 3.2.1: *„This dictates that capabilities for managing uncertainty and imprecision should be offered as strict extensions of existing standards.“*).

- Vielen der heute in der Praxis eingesetzten Technologien (Programmiersprachen, Ablaufumgebungen, Datenbanksysteme) liegt kein formales Datenmodell zugrunde, und selbst wenn, muß dieses weder bekannt noch durch den Anwender erweiterbar sein. Stattdessen ein beliebiges anderes Datenmodell heranzuziehen und stellvertretend zu erweitern, ist zwar theoretisch interessant, hilft jedoch praktisch nicht weiter.

- Wir betrachten die Imperfektion, die durch das Modell festgehalten werden soll, als orthogonale Systemeigenschaft — vergleichbar mit anderen Systemeigenschaften wie *Persistenz*, *Robustheit*, oder *Sicherheit*. Es ist weder praktikabel noch sinnvoll, für jede neue Eigenschaft grundsätzliche Änderungen am zugrundeliegenden Datenmodell durchzuführen.

- Moderne Informationssystem-Architekturen sehen nicht mehr vor, das Datenmodell für ein objektorientiertes Datenbanksystem als Grundlage zu erstellen, um darauf aufbauend eine Anwendung mit Hilfe einer Datenbankschnittstelle zu realisieren, wie es dem klassischen datenbankzentrischen Ansatz mit seiner zweistufigen Client/Server-Architektur entspricht. Vielmehr erfolgt die objektorientierte Modellierung einer Anwendung heute entkoppelt von einem konkreten Datenbanksystem. Wenn eine persistente Speicherung benötigt wird,

übernimmt eine geeignete Middleware die Abbildung von Objekten der verwendeten Programmiersprache auf ein Datenbanksystem, das dabei auch mit relationalen Techniken arbeiten kann.

Aus diesen Gründen verfolgen wir den Ansatz, das Repräsentationsmodell getrennt von seiner Einbindung in eine objektorientierte Programmiersprache zu formulieren. Im nächsten Teil dieser Arbeit zeigen wir dann, wie die Einbettung des Modells in eine objektorientierte Umgebung als orthogonale Erweiterung durchgeführt werden kann.

Neben der Möglichkeit, komplexe imperfekte Ausdrücke darzustellen, ermöglicht das hier entwickelte Modell eine Entkopplung der *Struktur* (Syntax) imperfekter Informationen von deren *Interpretation* (Semantik) als Fuzzy-Menge. Diesen wichtigen Unterschied werden wir in Abschnitt 8.4 diskutieren.

Anschließend definieren wir zunächst das Modell für die *Repräsentation* imperfekter Informationen. Die *Verarbeitung* dieser Daten ist Gegenstand von Kapitel 9. Einem wichtigen Teilaspekt der Verarbeitung, der *Konsistenzerhaltung*, widmen wir anschließend ein eigenes Kapitel.

8.1 Das Fuzzy-Repräsentationssystem

Wir beginnen mit der Formulierung des Repräsentationsmodells. Das Ziel ist hier zunächst ein formales Modell zu schaffen, mit dem sich die beschriebenen Arten imperfekter Informationen ausdrücken lassen.

Bereits in Kapitel 7 wurde gezeigt, wie sich vage Begriffe wie *groß* oder *umweltfreundlich* mit Hilfe von Fuzzy-Mengen repräsentieren lassen. Der klassische Ansatz zur Formulierung eines Repräsentationsmodells wäre nun, eine feste Grundmenge solcher Begriffe, ein *Vokabular* vorzugeben, dessen Elementen Fuzzy-Mengen zugeordnet werden. Aus diesen Grundbegriffen ließen sich dann durch Anwendung verschiedener Verknüpfungen neue Aussagen bilden.

Einen solchen Ansatz wählen wir jedoch für die hier betrachtete Klasse von Fuzzy-Informationssystemen nicht, da während der Laufzeit eines Systems kontinuierlich neue Fuzzy-Mengen erzeugt werden können — etwa aufgrund Daten, die von Meßsensoren geliefert werden, durch Berechnungen von Heuristiken entstehen oder von Benutzern eingegeben werden. Jede solche, beliebige Fuzzy-Menge muß aber einem Benutzer mitgeteilt werden können, und umgekehrt muß auch jede von einen Anwender angegebene Fuzzy-Menge in das System aufgenommen werden können. Mit einem vorgegebenen, statischen Vokabular läßt sich dies aber nicht gewährleisten.

Ein möglicher Modellierungsansatz hierzu wäre, ein dynamisches Vokabular zu erlauben, das sich mit der Ausführung von Operationen ständig anpaßt und erweitert.

Da wir an dieser Stelle aber vornehmlich an den Eigenschaften verschiedener Operatoren interessiert sind und diese Untersuchungen nicht mit ständigen Vokabularänderungen überladen wollen, nehmen wir für unser Modell eine wesentliche Vereinfachung vor: wir unterstellen, daß zu jeder Fuzzy-Menge, die beispielsweise aus externen Quellen stammt oder infolge von Berechnungen entsteht, auch eine syntaktische Beschreibung existiert. Wir abstrahieren dabei gleichsam von den notwendigen Änderungen eines abzählbar endlich oder unendlich großen Vokabulars und gehen von einer überabzählbar unendlich großen Menge syntaktischer Begriffe aus. Dies erlaubt uns einen einfacheren Aufbau des formalen Modells, ohne dabei praxisfern zu werden, denn zu jeder innerhalb eines Systems neu entstandenen Fuzzy-Menge läßt sich immer auch eine syntaktische Beschreibung erzeugen, sei es durch Angabe der Quelle, einer freien textuellen Syntax oder eines zugehörigen imperfekten Begriffes.

Wir nennen eine solche syntaktische Beschreibung einer Fuzzy-Menge ein *Fuzzy-Atom*, es bildet die kleinste Einheit in unserem Repräsentationsmodell. Formal ist ein Fuzzy-Atom folgendermaßen definiert:

Definition 8.1.1 (Fuzzy-Repräsentationssystem, Fuzzy-Atom) *Gegeben sei eine Menge Ω. Ein Fuzzy-Repräsentationssystem auf Ω ist ein Tripel $(\mathfrak{A}(\Omega), \mathcal{A}., \mu.)$. Dabei ist $\mathfrak{A}(\Omega)$ eine Menge von Symbolen über Ω und $\mu., \mathcal{A}.$ sind Abbildungen*

$$\begin{aligned} \mu. &: \mathfrak{A}(\Omega) \to F(\Omega), \mathcal{A} \mapsto \mu_{\mathcal{A}} \\ \mathcal{A}. &: F(\Omega) \to \mathfrak{A}(\Omega), \mu \mapsto \mathcal{A}_{\mu} \end{aligned}$$

für die $\mu. \circ \mathcal{A}. = \mathrm{id}_{F(\Omega)}$ gilt. Die Elemente von $\mathfrak{A}(\Omega)$ heißen Fuzzy-Atome. Die Fuzzy-Menge $\mu_{\mathcal{A}}$ heißt Interpretation von $\mathcal{A}$ und das Atom $\mathcal{A}_{\mu}$ ist der (ausgezeichnete) Name von μ.

Jedes Fuzzy-Atom repräsentiert einen imperfekten Begriff, dessen Syntax durch $\mathcal{A}$ ausgedrückt wird. Die *Semantik* eines solchen Atoms wird durch die Abbildung $\mu.$ in Form einer Fuzzy-Menge geliefert, die hier possibilistisch zu interpretieren ist (vergleiche Abschnitt 7.1 und dort Seite 63).

Im Vergleich zu konventionellen Repräsentationsmodellen fällt hier insbesondere die Forderung nach der Surjektivität der Abbildung $\mu.$ auf. Diese Forderung begründet sich zum einen mit der oben angeführten Notwendigkeit, daß jede beliebige innerhalb eines Systems entstandene Fuzzy-Menge einem Benutzer mitteilbar sein muß. Und zum anderen benötigen wir die Eigenschaft, jede Fuzzy-Menge syntaktisch benennen zu können, auch aus beweistechnischen Gründen.

Dies führt nun zu der ungewohnten Eigenschaft dieses Modells, daß die Domäne der Semantik zur isomorphen Teilmenge der Domäne der Syntax wird. Wie wir unten noch sehen werden, fallen die beiden Ebenen aber dennoch nicht zusammen, da es mehr wohlgeformte syntaktische Ausdrücke gibt als mögliche Denotationen.[1]

[1] Wenn man auf die Forderung der Mitteilbarkeit beliebiger Fuzzy-Mengen verzichtet, so lassen

Fuzzy-Atome alleine reichen jedoch nicht aus, um die eingangs beschriebenen komplexen Ausdrücke zu repräsentieren. Der klassische Ansatz wäre hier, die logische Verknüpfung der einzelnen imperfekten Begriffe auf die Ebene der Fuzzy-Mengen fortzusetzen und so mit Hilfe der in Abschnitt 6.1 auf Seite 49 definierten Operationen eine neue Fuzzy-Menge zu berechnen. Eine *und*-Verknüpfung zweier unsicherer Begriffe etwa ließe sich durch die Berechnung des Schnittes (vergleiche Definition 6.1.4 auf Seite 50) der beteiligten Fuzzy-Mengen erreichen. Da unser Modell für jede so entstandene Fuzzy-Menge auch ein syntaktisches Fuzzy-Atom bereithält, wäre es zunächst denkbar, diese Vorgehensweise beibehalten zu wollen.

Die Erfahrung aus der Praxis hat jedoch gezeigt, daß die Information, *wie* ein komplexer Ausdruck zustande gekommen ist, mindestens ebenso wichtig ist wie das Ergebnis, seine resultierende Fuzzy-Menge. Wir benötigen also eine Möglichkeit, die Verknüpfung einzelner Fuzzy-Atome syntaktisch festhalten zu können, ohne sie dabei auf einen einzelnen Wert reduzieren zu müssen. Darüber hinaus wird man innerhalb eines Fuzzy-Informationssystems Berechnungen aufbauend auf einer Menge zu einem Zeitpunkt verfügbarer Atome durchführen wollen — daß es zu jedem möglichen Ergebnis, jeder Fuzzy-Menge, grundsätzlich auch ein Fuzzy-Atom gibt, hilft nicht bei der Durchführung einer Berechnung.

Der hier verfolgte Ansatz ist daher, *syntaktische Ausdrücke* über Fuzzy-Atomen zu erlauben, die wiederum semantische Interpretationen in Form von Fuzzy-Mengen besitzen. Die zur Bildung solcher Ausdrücke erlaubten Verknüpfungsoperatoren sind dabei die *Negation*, die *Konjunktion* und die *Disjunktion*.

Im folgenden zeigen wir, wie in unserem Modell die Konstruktion wohlgeformter syntaktischer Ausdrücke und deren semantische Interpretation in Form von Fuzzy-Mengen erfolgt. Damit ist auch klar, wieso wie oben erwähnt die Ebenen der Syntax und Semantik nicht zusammenfallen, denn aus einfachen Fuzzy-Atomen lassen sich auf diese Weise beliebig komplexe syntaktische Ausdrücke bilden. Wie sich diese zusätzlichen syntaktischen Informationen ausnutzen lassen, um semantisch höherwertige Verarbeitungsoperatoren zu definieren, zeigen wir dann in den beiden nachfolgenden Kapiteln dieses Teils.

Wir beginnen zunächst mit der *Negation* eines Fuzzy-Atoms, die formal folgendermaßen definiert ist:

Definition 8.1.2 (Negation eines Fuzzy-Atoms) *Ein negiertes Fuzzy-Atom ist ein Ausdruck der Form $\overline{\mathcal{A}}$ oder $\neg\mathcal{A}$ mit $\mathcal{A} \in \mathfrak{A}(\Omega)$. Die Interpretation $\mu_{\overline{\mathcal{A}}}$ eines negierten Fuzzy-Atoms ist definiert als:*

$$\mu_{\overline{\mathcal{A}}} := 1 - \mu_{\mathcal{A}}$$

sich die Domänen der Syntax und Semantik auch strikter trennen, das heißt, die semantische Interpretation muß nicht mehr eine isomorphe Einbettung von $F(\Omega)$ in die Menge der Atome darstellen. Aus beweistechnischen Gründen müssen in diesem Fall aber gewisse Zusatzforderungen an μ. gestellt werden, die im wesentlichen besagen, daß zu jeder syntaktisch repräsentierbaren Fuzzy-Menge auch eine Form ihrer Inversion syntaktisch repräsentierbar sein muß.

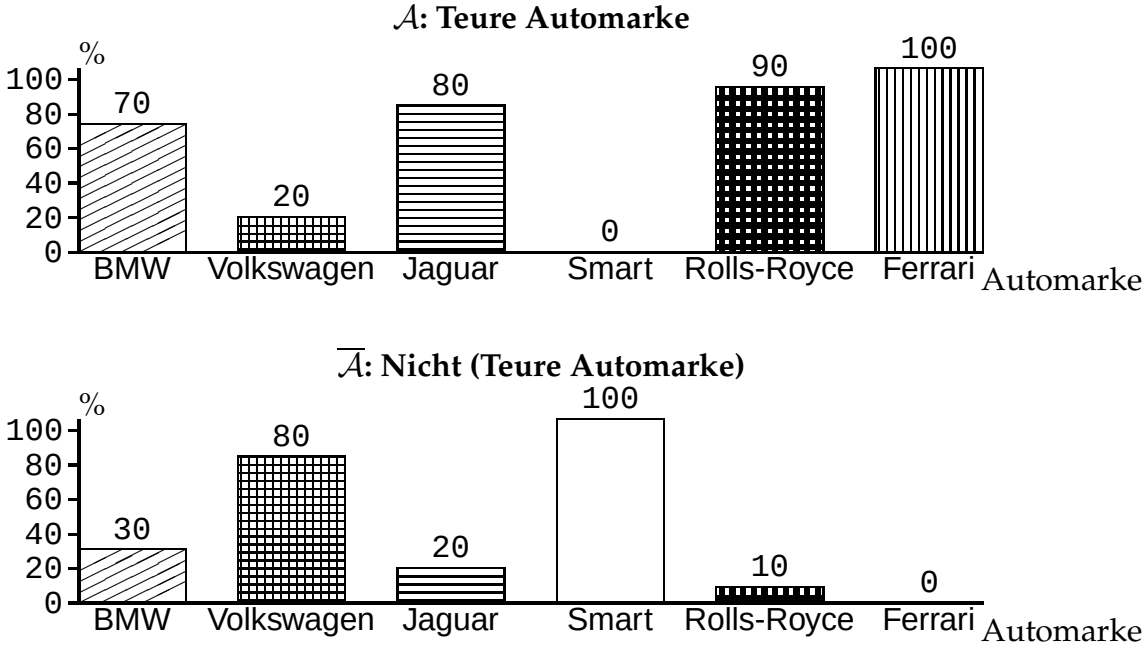


Abbildung 8.1: Fuzzy-Atom „Teure Automarke“ zusammen mit seiner Interpretation (oben) sowie seine Negation (unten)

Die Negation eines Fuzzy-Atoms, das durch eine Fuzzy-Menge in horizontaler Repräsentation beschrieben wird, ist nur bei einer endlichen Grundmenge Ω möglich. Dies stellt aber keine praktische Einschränkung dar, da in dem hier betrachteten Einsatzgebiet, nämlich der Anwendung innerhalb eines Informationssystems, grundsätzlich alle Wertebereiche durch eine endliche Anzahl von Bits repräsentiert werden.[2]

Beispiel (Fuzzy-Atom, Negation) Ein Beispiel für ein Fuzzy-Atom und seine Negation ist in Abbildung 8.1 zu sehen. Dort ist der vage Begriff *teure Automarke* zusammen mit seiner Negation und der jeweiligen Fuzzy-Interpretation gezeigt.

Wir können nun *Fuzzy-Literale* definieren:

Definition 8.1.3 (Fuzzy-Literal) *Ein Fuzzy-Literal $\mathcal{L}$ ist entweder ein Fuzzy-Atom $\mathcal{A}$ oder die Negation eines Fuzzy-Atoms $\overline{\mathcal{A}}$. Jedes Fuzzy-Literal besitzt damit eine Interpretation als Fuzzy-Menge, die mit $\mu_{\mathcal{L}}$ bezeichnet wird. Die Menge aller Fuzzy-Literale bezeichnen wir mit $\mathfrak{L}$, die Menge aller Fuzzy-Literale auf einer Grundmenge Ω mit $\mathfrak{L}(\Omega)$.*

Durch *oder*-Verknüpfung einzelner Fuzzy-Literale erhalten wir eine *Fuzzy-Klausel*:

[2] Dabei setzen wir natürlich eine effiziente Implementierung voraus, die im Zweifelsfall nicht die Negation aller Elemente der Grundmenge bestimmt, sondern die Fuzzy-Menge als negiert kennzeichnet und $\overline{\mu_{\mathcal{A}}(\omega)}$ durch Komplementbildung der Funktionswerte $\mu_{\mathcal{A}}(\omega)$ berechnet.

Definition 8.1.4 (Fuzzy-Klausel) *Eine Fuzzy-Klausel $\mathcal{K}$ ist eine endliche Menge von Fuzzy-Literalen $\mathcal{L}_i \in \mathfrak{L}(\Omega)$, $1 \leq i \leq n$:*

$$\mathcal{K} \stackrel{\text{def}}{=} \mathcal{L}_1 \vee \mathcal{L}_2 \vee \ldots \vee \mathcal{L}_n \stackrel{\text{def}}{=} \{\mathcal{L}_1, \ldots, \mathcal{L}_n\}$$

Die Interpretation einer Fuzzy-Klausel $\mathcal{K}$ als Fuzzy-Menge $\mu_\mathcal{K}$ ist gegeben durch:

$$\mu_\mathcal{K} \stackrel{\text{def}}{=} \begin{cases} \mu_\perp & \textbf{wenn } \mathcal{K} = \emptyset \\ \mu_{\mathcal{L}_1} \cup \mu_{\mathcal{L}_2} \cup \ldots \cup \mu_{\mathcal{L}_n} & \textbf{sonst} \end{cases}$$

mit $\mu_\perp(\omega) := 0 \, \forall \omega \in \Omega$. Die Menge aller Fuzzy-Klauseln bezeichnen wir mit $\mathfrak{K}$, die Menge aller Fuzzy-Klauseln auf einer Grundmenge Ω mit $\mathfrak{K}(\Omega)$.

Fuzzy-Klauseln sind also Mengen von Literalen, die aber gleichzeitig eine Interpretation als Fuzzy-Menge besitzen. Somit lassen sich die Mengenoperationen Schnitt und Vereinigung direkt auf Fuzzy-Klauseln übertragen.

Für die Fuzzy-Interpretation einer Fuzzy-Klausel werden die Fuzzy-Interpretationen der einzelnen Fuzzy-Literale vereinigt. Durch die Definition der Vereinigung über das Maximum der beteiligten Fuzzy-Mengen (vergleiche Definition 6.1.5 auf Seite 50) sind in der Ergebnismenge alle Werte mit mindestens dem gleichen Grad möglich, wie in der Fuzzy-Interpretation eines der beteiligten Fuzzy-Literale. Dies entspricht auch genau der beabsichtigten *oder*-Semantik auf Syntaxebene nach possibilistischer Interpretation auf Fuzzy-Mengenebene (vergleiche wie oben Seite 63): es soll kein Wert durch die Fuzzy-Interpretation der Fuzzy-Klausel ausgeschlossen werden, der nach mindestens einem Fuzzy-Literal möglich ist.

Beispiel (Fuzzy-Klausel) Ein Beispiel für eine Fuzzy-Klausel mit ihrer Fuzzy-Interpretation zeigt Abbildung 8.2 auf der nächsten Seite. In diesem Szenario betrachten wir das sogenannte Reengineering von Altsystemen, bei dem Informationen über bestehende, typischerweise undokumentierte Systeme durch Analyse des Programmquellcodes und sonstiger verfügbarer Artefakte gesammelt werden [Köl00]. Dabei entsteht eine Vielzahl von imperfekten Informationen [JW00], die explizit repräsentiert werden sollten, was mit unserem Modell möglich wird. Im Beispiel sind drei Fuzzy-Literale definiert, die aus verschiedenen Quellen stammende Erkenntnisse über den Typ einer Variablen *i* in einem solchen Reengineering-System repräsentieren. Die aus diesen Literalen gebildete Fuzzy-Klausel zeigt den kombinierten Kenntnisstand über die Variable *i* innerhalb des Systems an.

Eine mögliche Erweiterung von Fuzzy-Klauseln ist die Einführung eines *Sicherheitsgrades*, wie sie von Boss in [Bos96] für Mengen von vagen Anforderungen vorgeschlagen wird. Motivation für die Einführung eines solchen Grades ist dort, im Kontext der Entwurfsanwendungen die gestellten Anforderungen aufzuweichen, abhängig davon, welche *Sicherheit* oder *Priorität* ein Benutzer einer Anforderung zuordnet.

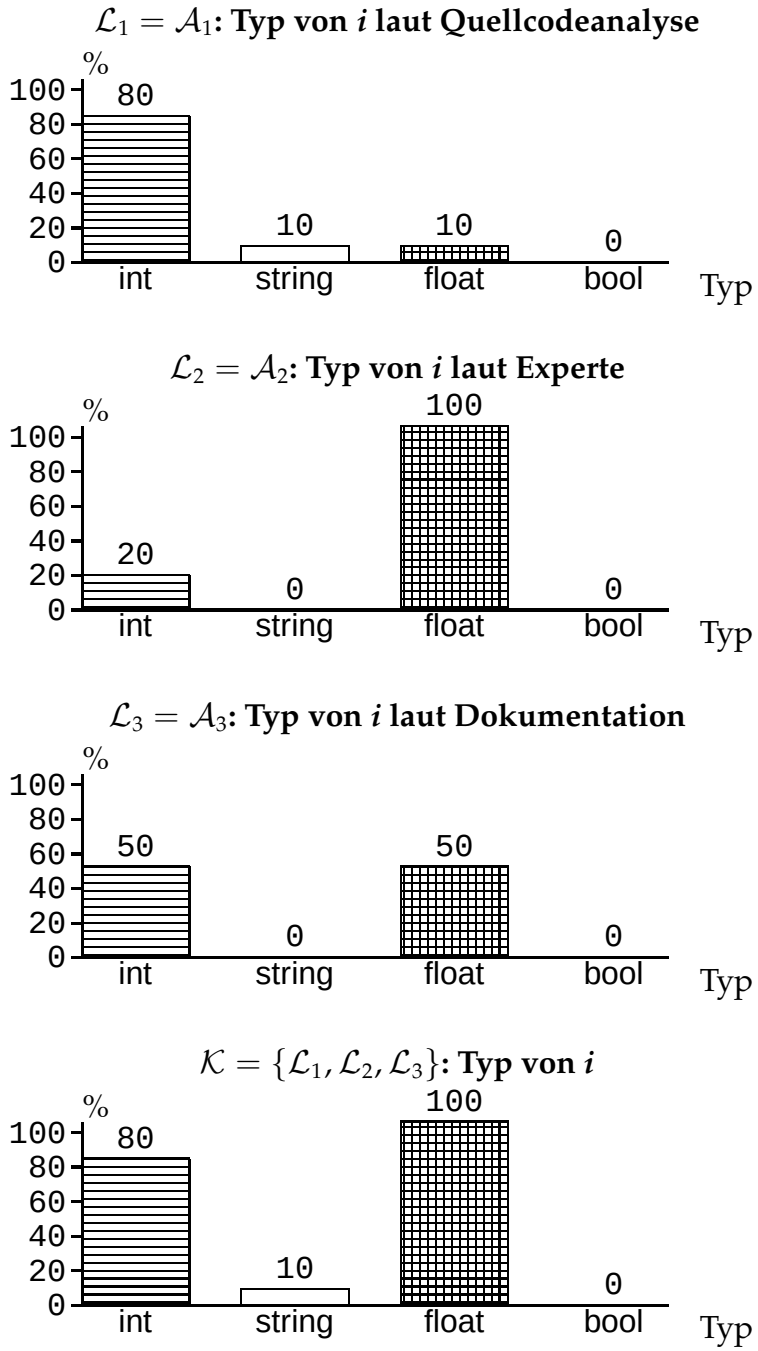


Abbildung 8.2: Fuzzy-Literale $\mathcal{L}_1$–$\mathcal{L}_3$ und daraus resultierende Fuzzy-Klausel $\mathcal{K}$

Wir verzichten in unserem Modell auf eine solche Erweiterung, und zwar aus zwei Gründen: Erstens läßt sich ein äquivalentes Ergebnis formal eleganter erreichen, wie wir unten zeigen werden. Und zweitens ist der praktische Nutzen bei der Anwendungsentwicklung zweifelhaft; die Erfahrungen mit der in [Bos96] realisierten Entwurfsanwendung haben gezeigt, daß Anwender mit zu vielen unsicheren Freiheitsgraden überfordert sind.

Möchte man dennoch die Möglichkeit haben, die Einschränkung einer Fuzzy-Klausel aufzuweichen, läßt sich dies durch die Definition eines *Sicherheitsliterals* $\mathcal{L}_\alpha$ erreichen, wobei der Grad α die Sicherheit von $\alpha = 1{,}0 =$ *völlig sicher* (keine Änderung der Fuzzy-Interpretation der Klausel) bis $\alpha = 0{,}0 =$ *völlig unsicher* (alle Werte der Grundmenge werden möglich) angibt:

Definition 8.1.5 (Sicherheitsliteral) *Ein Sicherheitsliteral $\mathcal{L}_\alpha \in \mathfrak{L}(\Omega)$ mit $\alpha \in [0,1]$ besteht aus dem Fuzzy-Atom $\mathcal{A}_\alpha \in \mathfrak{A}(\Omega)$. Dieses Atom repräsentiert die Sicherheit der Information, seine Fuzzy-Interpretation $\mu_{\mathcal{A}_\alpha}$ ist definiert als:*

$$\mu_{\mathcal{A}_\alpha}(\omega) \stackrel{\text{def}}{=} 1 - \alpha \quad \forall \omega \in \Omega.$$

Ein solchermaßen definiertes Sicherheitsliteral kann dann wie jedes andere Fuzzy-Literal zu einer Fuzzy-Klausel hinzugefügt werden:

Definition 8.1.6 (Fuzzy-Klausel mit Sicherheitsliteral) *Eine Fuzzy-Klausel mit Sicherheitsliteral $\mathcal{K}_\alpha$ ist definiert als die Vereinigung der Fuzzy-Klausel $\mathcal{K} \in \mathfrak{K}(\Omega)$ mit dem Sicherheitsliteral $\mathcal{L}_\alpha \in \mathfrak{L}(\Omega)$:*

$$\mathcal{K}_\alpha \stackrel{\text{def}}{=} \mathcal{K} \cup \{\mathcal{L}_\alpha\}.$$

Ein Beispiel für die Fuzzy-Menge einer Fuzzy-Klausel mit Sicherheitsliteral ist in Abbildung 8.3 auf der nächsten Seite gezeigt.

Auf diese Weise bleibt es einem Anwendungsentwickler überlassen, ob er den zusätzlichen Sicherheitsgrad einsetzen will. Im folgenden sprechen wir jedoch nur von Fuzzy-Klauseln; der Einsatz eines Sicherheitsliterals ist optional und anwendungsabhängig, hat jedoch aus den oben genannten Gründen keinen Einfluß auf die im folgenden betrachteten theoretischen Eigenschaften unseres Modells.

Fuzzy-Klauseln schließlich lassen sich durch *und*-Verknüpfung zu *Fuzzy-Formeln* verbinden. Mit Hilfe einer solchen Fuzzy-Formel kann dann eine komplexe imperfekte Information, wie sie eingangs beschrieben wurde, ausgedrückt werden.

Definition 8.1.7 (Fuzzy-Formel) *Eine Fuzzy-Formel $\mathcal{F}$ ist eine endliche Menge von Fuzzy-Klauseln $\mathcal{K}_i \in \mathfrak{K}(\Omega), 1 \leq i \leq m$:*

$$\mathcal{F} \stackrel{\text{def}}{=} \mathcal{K}_1 \wedge \ldots \wedge \mathcal{K}_m \stackrel{\text{def}}{=} \{\mathcal{K}_1, \ldots, \mathcal{K}_m\}$$

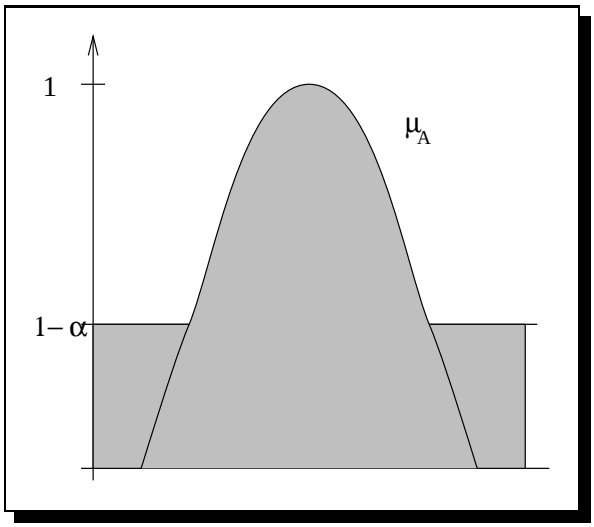


Abbildung 8.3: Fuzzy Klausel mit Sicherheitsliteral $\mathcal{L}_\alpha$

Die Interpretation einer Fuzzy-Formel $\mathcal{F}$ als Fuzzy-Menge $\mu_\mathcal{F}$ ergibt sich durch:

$$\mu_\mathcal{F} \stackrel{\text{def}}{=} \begin{cases} \mu_\top & \textbf{wenn } \mathcal{F} = \emptyset \\ \mu_{\mathcal{K}_1} \cap \mu_{\mathcal{K}_2} \cap \ldots \cap \mu_{\mathcal{K}_m} & \textbf{sonst} \end{cases}$$

mit $\mu_\top(\omega) := 1 \, \forall \omega \in \Omega$. Die Menge aller Fuzzy-Formeln bezeichnen wir mit $\mathfrak{F}$. Die Menge aller Fuzzy-Formeln auf einer Grundmenge Ω sei mit $\mathfrak{F}(\Omega)$ benannt.

Eine Fuzzy-Formel bezeichnen wir auch als *Fuzzy Konjunktive Normalform* (FKNF). Die Schnitt- und Vereinigungsmenge zweier Fuzzy-Formeln läßt sich auch hier direkt auf der Darstellung als Klauselmenge berechnen.

Entsprechend der beabsichtigten *und*-Semantik werden für die Fuzzy-Interpretation die beteiligten Fuzzy-Mengen der Fuzzy-Klauseln geschnitten (vergleiche Definition 6.1.4 auf Seite 50). Durch die possibilistische Interpretation der Fuzzy-Menge (siehe oben) werden in einer Fuzzy-Formel also alle Werte ausgeschlossen, die schon in einer der beteiligten Fuzzy-Klauseln ausgeschlossen sind. Auch dies läßt sich leicht anschaulich nachvollziehen: Repräsentieren die Fuzzy-Klauseln etwa einzelne Anforderungen, die alle gleichzeitig gelten sollen, darf ein mögliches Ergebnis keinen Wert enthalten, der nicht mit jeder der Anforderungen zu mindestens einem bestimmten Grad kompatibel ist.

Beispiel (Fuzzy-Formel) Als Beispiel für eine Fuzzy-Formel betrachten wir die Modellierung einer Anforderung zur Auswahl eines Autotyps, wie wir sie später in unserem Fuzzy-Entscheidungshilfesystem verwenden werden. Dabei soll der Motor eines auszuwählenden Fahrzeuges die folgenden vagen Eigenschaften erfüllen:

$$\text{nicht teuer} \wedge (\text{sparsam} \vee \text{biologisch})$$

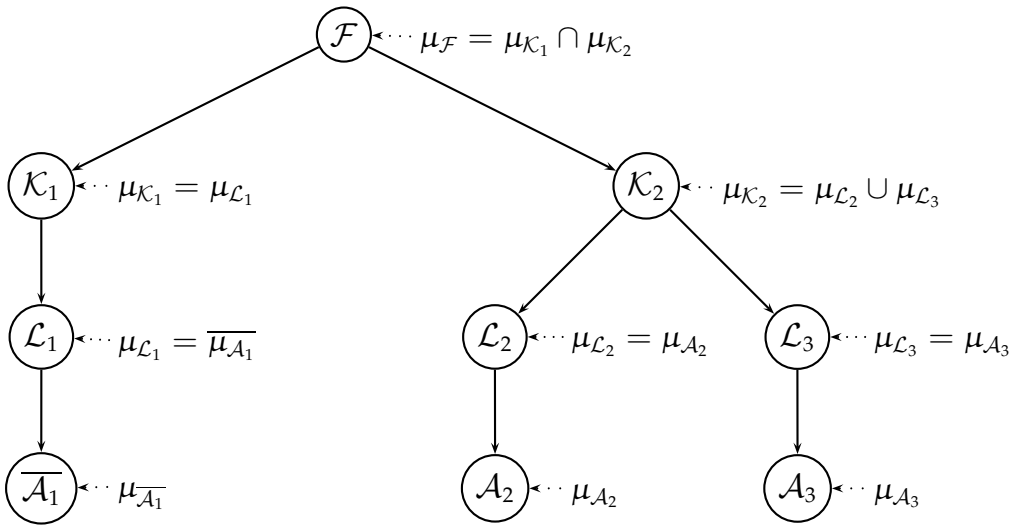


Abbildung 8.4: Beispiel für eine Fuzzy-Formel

Als Grundlage stehen bereits die Fuzzy-Atome $\mathcal{A}_1$–$\mathcal{A}_3$ zur Modellierung der Konzepte $\mathcal{A}_1$ *„teuer“* (hoher Preis), $\mathcal{A}_2$ *„sparsam“* (im Verbrauch) und $\mathcal{A}_3$ *„biologisch“* (arbeitet mit Biotreibstoff) zur Verfügung. Abbildung 8.4 zeigt, wie sich eine komplexe Fuzzy-Formel aus den einzelnen Bausteinen zusammensetzt, dabei gilt für die Formel $\mathcal{F} = \{\mathcal{K}_1, \mathcal{K}_2\} = \mathcal{K}_1 \wedge \mathcal{K}_2$, für die Klauseln $\mathcal{K}_1 = \{\mathcal{L}_1\}$ und $\mathcal{K}_2 = \{\mathcal{L}_2, \mathcal{L}_3\} = \mathcal{L}_2 \vee \mathcal{L}_3$ sowie für die Literale $\mathcal{L}_1 = \neg\mathcal{A}_1, \mathcal{L}_2 = \mathcal{A}_2$ und $\mathcal{L}_3 = \mathcal{A}_3$. Jede Komponente, wie Fuzzy-Literal, Fuzzy-Klausel und Fuzzy-Formel besitzt eine Interpretation als Fuzzy-Menge, die neben den Knoten dargestellt ist.

Man beachte, daß auch für Fuzzy-Formeln die Kontradiktionsgesetze nicht gelten (vergleiche Seite 51): So ist die Fuzzy-Interpretation von $\mathcal{L} \vee \neg\mathcal{L}$ im allgemeinen Fall nicht $\mu_\top$ und $\mathcal{K}_1 \wedge \mathcal{K}_2$ mit $\mathcal{K}_1 = \mathcal{L}, \mathcal{K}_2 = \neg\mathcal{L}$ nicht immer $\mu_\bot$. Dies ist eine typische Eigenschaft von mehrwertigen Logiken und spiegelt durchaus die Realität wieder, da man problemlos in der Lage ist, bis zu einem gewissen Grad widersprüchliche Eigenschaften zu akzeptieren. So wird etwa jemand, der sich durch dichten Nebel bewegt und dabei naß wird, gleichzeitig glauben können, daß es *regnet* und daß es *nicht regnet*. Bei einer syntaktischen Transformation — etwa um eine Formel in die konjunktive Normalform zu bringen — dürfen solche Ausdrücke also nicht entfernt werden. Die Distributivgesetze behalten jedoch ihre Gültigkeit, da die Vereinigungs- und Schnittoperatoren für Fuzzy-Mengen entsprechend gewählt wurden (vergleiche Seite 51).

8.2 Maßzahlen

Wir definieren nun noch eine Reihe von Maßzahlen, die in den nachfolgenden Kapiteln benötigt werden.

Der *Konsistenzgrad* einer Fuzzy-Klausel gibt an, wie konsistent die in ihr enthaltenen Informationen sind. Dies ist über die Interpretation einer Klausel als Fuzzy-Menge definiert:

Definition 8.2.1 (Konsistenzgrad einer Fuzzy-Klausel) *Der Konsistenzgrad C einer Fuzzy-Klausel $\mathcal{K} \in \mathfrak{K}(\Omega)$ ist definiert als:*

$$C(\mathcal{K}) \stackrel{\text{def}}{=} \sup_{\omega \in \Omega} \{\mu_{\mathcal{K}}(\omega)\}.$$

Ist in der Interpretation einer Klausel als Fuzzy-Menge mindestens ein Wert ohne Einschränkung möglich, ergibt sich ein Konsistenzgrad von 1.

Der *Inkonsistenzgrad* ist das Komplement zum Konsistenzgrad:

Definition 8.2.2 (Inkonsistenzgrad einer Fuzzy-Klausel) *Der Inkonsistenzgrad* Inc *einer Fuzzy-Klausel $\mathcal{K} \in \mathfrak{K}(\Omega)$ ist definiert als:*

$$\text{Inc}(\mathcal{K}) \stackrel{\text{def}}{=} 1 - C(\mathcal{K}).$$

Analog definieren wir den Konsistenzgrad für eine Fuzzy-Formel:

Definition 8.2.3 (Konsistenzgrad einer Fuzzy-Formel) *Der Konsistenzgrad C einer Fuzzy-Formel $\mathcal{F} \in \mathfrak{F}(\Omega)$ ist definiert als:*

$$C(\mathcal{F}) \stackrel{\text{def}}{=} \sup_{\omega \in \Omega} \{\mu_{\mathcal{F}}(\omega)\}.$$

Veranschaulicht gibt der Konsistenzgrad einer Fuzzy-Formel an, wie kompatibel die in der Formel enthaltenen Fuzzy-Klauseln zueinander sind. Ist in der Interpretation einer Formel als Fuzzy-Menge mindestens ein Wert mit dem Grad 1 möglich, ist die Formel konsistent und der Konsistenzgrad 1. Widersprechen sich die Klauseln, sinkt der Konsistenzgrad der Fuzzy-Formel bis auf 0 im Falle völliger Inkonsistenz.

Komplementär ist auch hier der Inkonsistenzgrad definiert:

Definition 8.2.4 (Inkonsistenzgrad einer Fuzzy-Formel) *Der Inkonsistenzgrad* Inc *einer Fuzzy-Formel $\mathcal{F} \in \mathfrak{F}(\Omega)$ ist definiert als:*

$$\text{Inc}(\mathcal{F}) \stackrel{\text{def}}{=} 1 - C(\mathcal{F}).$$

Diese Definitionen führen zu einer Reihe von Spezialfällen, die wir gesondert betrachten, da sie später bei der Definition von Operatoren wichtig werden. Die *absurde Fuzzy-Klausel* hat in ihrer Interpretation als Fuzzy-Menge keinen Wert, der einen Zugehörigkeitsgrad größer als 0 aufweist:

Definition 8.2.5 (Absurde Fuzzy-Klausel) *Eine absurde Fuzzy-Klausel $\mathcal{K} \in \mathfrak{K}(\Omega)$ ist eine Klausel, deren Fuzzy-Interpretation identisch verschwindet, also*

$$\mu_{\mathcal{K}}(\omega) = 0 \quad \forall \omega \in \Omega$$

erfüllt. Wir schreiben $\bot(\mathcal{K})$ für das Prädikat „$\mathcal{K}$ ist absurd". Als Abkürzung verwenden wir auch die Notation $\mathcal{K}_\bot$ für eine beliebige Klausel, die $\bot(\mathcal{K})$ erfüllt.

Semantisch entspricht dies dem Nullwert DNE (does not exist), da kein Wert aus der Basis der Fuzzy-Klausel in Frage kommt.

In der Interpretation einer *sinnleeren Fuzzy-Klausel* sind dagegen alle Werte ohne Einschränkung möglich:

Definition 8.2.6 (Sinnleere Fuzzy-Klausel) *Eine sinnleere Fuzzy-Klausel $\mathcal{K} \in \mathfrak{K}(\Omega)$ ist eine Klausel, deren Fuzzy-Interpretation konstant 1 ist, also*

$$\mu_{\mathcal{K}}(\omega) = 1 \quad \forall \omega \in \Omega$$

erfüllt. Wir schreiben $\top(\mathcal{K})$ für das Prädikat „$\mathcal{K}$ ist sinnleer". Als Abkürzung verwenden wir auch die Notation $\mathcal{K}_\top$ für eine beliebige Klausel, die $\top(\mathcal{K})$ erfüllt.

Dies ist vergleichbar mit den aus relationalen Datenbanken bekannten Nullwerten mit der Interpretation UNK (unknown): es existiert ein Wert, der jedoch unbekannt ist und somit offengehalten wird.

Analog definieren wir eine *absurde Fuzzy-Formel*:

Definition 8.2.7 (Absurde Fuzzy-Formel) *Eine absurde Fuzzy-Formel $\mathcal{F} \in \mathfrak{F}(\Omega)$ ist eine Formel, deren Fuzzy-Interpretation identisch verschwindet, also*

$$\mu_{\mathcal{F}}(\omega) = 0 \quad \forall \omega \in \Omega$$

erfüllt. Wir schreiben $\bot(\mathcal{F})$ für das Prädikat „$\mathcal{F}$ ist absurd". Analog zu Fuzzy-Klauseln verwenden wir auch hier die Abkürzung $\mathcal{F}_\bot$ für eine beliebige Formel, die $\bot(\mathcal{F})$ erfüllt.

Sowie eine *sinnleere Fuzzy-Formel*:

Definition 8.2.8 (Sinnleere Fuzzy-Formel) *Eine sinnleere Fuzzy-Formel $\mathcal{F}_\top \in \mathfrak{F}(\Omega)$ ist eine Formel, deren Fuzzy-Interpretation konstant 1 ist, also*

$$\mu_{\mathcal{F}}(\omega) = 1 \quad \forall \omega \in \Omega$$

erfüllt. Wir schreiben $\top(\mathcal{F})$ für das Prädikat „$\mathcal{F}$ ist sinnleer". Als Abkürzung verwenden wir auch hier die Notation $\mathcal{F}_\top$ für eine beliebige Formel, die $\top(\mathcal{F})$ erfüllt.

8.3 Relationen

In diesem Abschnitt definieren wir verschiedene Beziehungen auf unserem Wissensmodell, die später verwendet werden.

Wir betrachten zuerst die Definition der Gleichheit und Äquivalenz für Fuzzy-Klauseln und -Formeln. Für die Gleichheit behalten wir die übliche Definition für Mengen bei, zwei Fuzzy-Klauseln sind also genau dann gleich, wenn sie die gleiche Menge von Fuzzy-Literalen besitzen. Entsprechend sind zwei Fuzzy-Formeln gleich, wenn sie in ihrer Menge die gleichen Fuzzy-Klauseln aufweisen.

Für die Definition der *Äquivalenz* dagegen betrachten wir die Semantik der beteiligten Fuzzy-Formeln, also ihre Interpretation als Fuzzy-Menge. Haben zwei Fuzzy-Formeln $\mathcal{F}_1, \mathcal{F}_2$ die gleiche Interpretation, gilt also $\mu_{\mathcal{F}_1} = \mu_{\mathcal{F}_2}$, nennen wir sie *semantisch äquivalent*.

Später wird oft der Ausdruck verwendet, daß eine Fuzzy-Formel *in Konflikt* mit einer anderen Fuzzy-Formel (oder Fuzzy-Klausel) steht. Konflikt bedeutet hier, daß zwei Informationen inkompatibel sind, sich also — innerhalb eines Grades γ — widersprechen. Solche Informationen können widersprüchliche Anforderungen sein, die sich nicht gleichzeitig befriedigen lassen, oder inkonsistente Aussagen darstellen, die beispielsweise eine Person beschreiben. Wir präzisieren den Begriff des Konfliktes mit der folgenden Definition:

Definition 8.3.1 (Konflikt von Fuzzy-Formeln) *Eine Fuzzy-Formel $\mathcal{F}_1 \in \mathfrak{F}(\Omega)$ steht mit einer Fuzzy-Formel $\mathcal{F}_2 \in \mathfrak{F}(\Omega)$ in (γ-)Konflikt, wenn gilt: $C(\mathcal{F}_1 \cup \mathcal{F}_2) < \gamma$.*

Besteht die Fuzzy-Formel $\mathcal{F}_2$ aus nur einer Klausel $\mathcal{K}_1$, sagen wir die Fuzzy-Formel $\mathcal{F}_1$ steht mit dieser Klausel in γ-Konflikt, wenn gilt: $C(\mathcal{F}_1 \cup \{\mathcal{K}_1\}) < \gamma$.

Zur Bestimmung eines Konfliktes wird also betrachtet, ob die Fuzzy-Interpretation bei der Kombination der beiden Einzelinformationen einen Grad γ unterschreitet. Man erinnere sich hier an die possibilistische Interpretation der Fuzzy-Mengen (vergleiche Abschnitt 7.1 und dort Seite 63): Existiert in der Fuzzy-Interpretation der vereinigten Fuzzy-Formeln kein Wert ω mehr mit $\mu(\omega) \geq \gamma$, beschreibt sie auch kein Konzept, für das (zum Grade γ) kompatible Werte verfügbar sind.

Wenn festgestellt werden soll, ob eine Fuzzy-Klausel $\mathcal{K}$ in einer Fuzzy-Formel $\mathcal{F}$ enthalten ist, wird die übliche Mengenschreibweise beibehalten, also $\mathcal{K} \in \mathcal{F}$. Ebenso läßt sich ausdrücken, ob eine Fuzzy-Formel $\mathcal{F}$ Untermenge einer anderen Fuzzy-Formel $\mathcal{G}$ ist: $\mathcal{F} \subseteq \mathcal{G}$.

8.4 Entkopplung von Struktur und Interpretation

Ein wesentlicher Unterschied zwischen dem hier definierten Repräsentationsmodell und dem in existierenden Fuzzy-Datenmodellen liegt in der strikten Trennung der

Struktur einer imperfekten Information von ihrer *Interpretation* als Fuzzy-Menge.

Diese Trennung ist wichtig, da in der Struktur, also der Syntax, deutlich mehr Informationen enthalten sind als in der Interpretation, die hier die Form einer Fuzzy-Menge annimmt. Um sich den Unterschied anschaulich zu machen, kann man das Fermatsche Theorem[3] betrachten: Während die Interpretation des Theorems schlicht „wahr“ lautet, enthielt die Syntax offenbar genug Information, um unzählige Mathematiker mehrere Jahrhunderte lang in Atem zu halten.

Diese Unterscheidung ist auch für Fuzzy-Informationssysteme wichtig, da verschiedene, syntaktisch völlig unterschiedliche Fuzzy-Formeln die gleiche Interpretation als Fuzzy-Menge besitzen können (vergleiche Abschnitt 8.1 auf Seite 71). Entscheidend bei einer Weiterverarbeitung ist aber die Struktur, die einer Interpretation zugrundeliegt, da bereits kleine Änderungen auf Syntaxebene, wie das Entfernen einer Fuzzy-Klausel, zu grundlegenden Änderungen auf der Ebene der Fuzzy-Interpretation führen können.

Besonders wichtig wird dies beim Einsatz des Modells innerhalb eines objektorientierten Datenmodells und dessen Verwendung zur Realisierung von Fuzzy-Informationssystemen. Da andere Ansätze nur einzelne Fuzzy-Mengen zur Verfügung haben, um den Zustand von Attributen oder Objekten zu beschreiben, müssen Änderungen über den Kenntnisstand durch direkte Änderung der Fuzzy-Mengen fortgeschrieben werden. Dabei ist aber nicht mehr ersichtlich, wie eine Änderung auf Strukturebene eine bestimmte Fuzzy-Menge geändert hat. Wenn Änderungen aufgrund widersprüchlicher Informationen zu einer Inkonsistenz führen, gehen praktisch alle gesammelten Teilinformationen verloren, da nicht mehr ersichtlich ist, welche Änderung eine inkonsistente Information eingebracht hat.

In dem hier vorgeschlagenen Modell sind imperfekte Informationen daher in Atome, Literale, Klauseln und Formeln strukturiert, die eine Sprache wohlgeformter syntaktischer Ausdrücke definieren. Erst die *Interpretation* dieser syntaktischen Konstrukte liefert eine Fuzzy-Menge, die mit den bekannten Methoden verarbeitet werden kann. Dies ist vergleichbar mit der Bildung gültiger Ausdrücke in Programmiersprachen, etwa auf reellen Zahlen.

Jede *Änderung* über den Kenntnisstand — das Hinzufügen neuer Informationen, das Einbringen neuer Bedingungen, die Aufnahme neuer Ergebnisse einer Heuristik oder eines Meßergebnisses — wird so nicht an der resultierenden Fuzzy-Menge durchgeführt, sondern erfolgt durch Modifikation der beschreibenden Fuzzy Konjunktiven Normalform. Muß später eine Information entfernt werden, weil beispielsweise eine Heuristik inkorrekt angewendet wurde, ein defekter Meßsensor falsche Daten geliefert hat, eine Bedingung inkonsistent war oder sich unscharfe Anforderungen geändert haben, kann die entsprechende Komponente einfach aus der syntaktischen Beschreibung entfernt und die Interpretation als Fuzzy-Menge neu berechnet werden.

[3] Es gibt keine ganzzahligen Lösungen für die Gleichung $x^n + y^n = z^n$, für $n > 2$.

Wie wir später sehen werden, ist dies auch eine wichtige Voraussetzung, um die Anforderungen VERARBEITUNG UNSCHARFER DATEN (siehe Abschnitt 3.2.6 auf Seite 33) und KONSISTENZERHALTUNG (siehe Abschnitt 3.2.8 auf Seite 35) erfüllen zu können. Existierende Ansätze, die auf den Fuzzy-Mengen direkt operieren, bieten diese Flexibilität nicht, denn dort kann nur eine Fuzzy-Menge als ganzes modifiziert werden, nicht aber die einer konkreten Fuzzy-Verteilung zugrundeliegende Struktur.

8.5 Modellierung von Abhängigkeiten

Mit den im letzten Abschnitt definierten Fuzzy-Formeln ist es jetzt möglich, imperfekte Zustände beispielsweise von Attributen oder Objekten eines Informationssystems auszudrücken, wie wir im nächsten Teil zeigen werden.

Für die Entwicklung einsatzfähiger Fuzzy-Informationssysteme reicht dies jedoch noch nicht aus. Zwischen den unscharfen Beschreibungen existieren semantische Zusammenhänge, die wir mit dem Modell bis jetzt noch nicht greifen können. Boss [Bos96] führte aus diesem Grund für die Domäne der Entwurfsanwendungen *Kategorien* ein, die eine Unterscheidung von unscharfen Entwurfsanforderungen verschiedener Priorität ermöglichen. Formal sind diese über einen Vektor $W^1, \ldots, W^n$ von Wissensbasen (Boss' unsichere Formeln) modelliert; W^1 entspricht dabei der Bedingungsmenge mit der höchsten Priorität (beispielsweise strategische Entscheidungen oder gesetzliche Vorgaben) und alle nachfolgenden bis zur Kategorie n niedriger priorisierten Anforderungen (beispielsweise denen von Auftraggeber, Projektleiter, Entwerfer). Wichtig sind diese Kategorien bei Änderungen, da das Einführen einer neue Bedingung auf einer bestimmten Ebene Änderungen an niedriger priorisierten Anforderungen zur Folge haben kann. Umgekehrt darf aber auf einer Ebene keine Bedingung verletzt werden, die auf höherer Ebene gefordert wird.

Solche Wechselwirkungen zwischen unscharfen Beschreibungen finden sich nicht nur bei Entwurfsanwendungen, sondern sind ein typisches Merkmal von Fuzzy-Informationssystemen. Verallgemeinert können wir hier von *Abhängigkeiten* zwischen den imperfekten Informationen sprechen. Dieses Phänomen kennt man bereits von Wissensverwaltungssystemen, die mit einer Menge (scharfer) Aussagen umgehen müssen, wie zum Beispiel Gärdenfors beobachtet [Gär88]:

> *However, belief sets cannot be used to express that some beliefs may be* ***reasons*** *for other beliefs. [. . .] And intuitively, when we compare degrees of similarity between different epistemic states, we want the structure of reasons or justifications to count as well.*

Da eines unserer Ziele darin besteht, die Realisierung von Fuzzy-Informationssystemen aufbauend auf einem vorgegebenen Modell zu ermöglichen, müssen wir eine

Lösung für dieses Phänomen anbieten. Ansonsten würde dies wieder zu dem unerwünschten Effekt führen, daß solche Abhängigkeiten innerhalb von Anwendungen modelliert werden müßten und damit nicht als separates Datenmodell zur Verfügung stünden.

Beispiel (Abhängigkeit) Als Beispiel für eine Abhängigkeit zwischen zwei Fuzzy-Formeln betrachten wir wieder einen Fall aus dem Reengineering von Altsystemen. Für dieses Beispiel nehmen wir an, daß Informationen über den *Datentyp* einer untersuchten Variablen i in einer Fuzzy-Formel $\mathcal{F}_1$ und deren Verwendungszweck in einer Fuzzy-Formel $\mathcal{F}_2$ gesammelt werden. Die Domäne von $\mathcal{F}_1$ ist die Menge der möglichen Typen (wie *float, bool, int, enum*) und die von $\mathcal{F}_2$ die Menge der Verwendungsarten (wie *Steuerungsvariable, Berechnungsvariable, Indexvariable*). Diese beiden Informationen sind jedoch nicht unabhängig voneinander. Verdichten sich die Informationen über den Typ im weiteren Verlauf einer Programmanalyse, hat dies auch Auswirkungen auf den mutmaßlichen Verwendungszweck, da beide direkt voneinander abhängen. Beispielsweise könnte sich herausstellen, daß i mit ziemlicher Sicherheit eine *float*-Variable ist, damit würde sich auch die Sicherheit für ihre Verwendung als Berechnungsvariable erhöhen. Entsprechend sinken in dem Fall die Sicherheiten für andere Verwendungsarten, wie für die Indexvariable. Wie solche Berechnungen automatisch durchgeführt werden können, zeigen wir in Kapitel 10.

Vorher benötigen wir jedoch noch die Definition zweier weiterer Konzepte: den im nächsten Abschnitt definierten *Abhängigkeitsgraphen* sowie die in Kapitel 9 eingeführten *Transformationsoperatoren.*

8.5.1 Abhängigkeiten und Abhängigkeitsgraphen

Wenn wir diese semantischen Wechselwirkungen formalisieren, kommen wir zu dem Begriff der *Abhängigkeiten* zwischen den Bestandteilen eines Informationssystems.

In unserem Modell verwenden wir Fuzzy-Formeln, um imperfekte Aussagen über diese Bestandteile machen zu können, wie wir später noch genauer zeigen werden: Im obigen Beispiel etwa stehen solche Fuzzy-Aussagen für Programmvariablen in einem Reengineering-System.

Da wir an dieser Stelle noch nicht festlegen wollen, was diese Bestandteile sein werden, aber die Abhängigkeiten formal festhalten wollen, definieren wir einen *Abhängigkeitsgraphen*, dessen Knoten später diese Bestandteile repräsentieren werden. Da die Abhängigkeiten eben zwischen diesen Bestandteilen bestehen und die Fuzzy-Formeln nur Aussagen über diese Bestandteile repräsentieren, können wir die Abhängigkeiten nicht zwischen einzelnen Fuzzy-Formeln definieren: sie würden eine Änderung nicht überdauern. Daher sind in dem Abhängigkeitsgraph die Fuzzy-Formeln über eine Zuordnungsfunktion ξ den Knoten zugeordnet. Die Kanten wiederum repräsentieren die einzelnen Abhängigkeiten.

Definition 8.5.1 (Abhängigkeitsgraph) *Ein (Fuzzy-)Abhängigkeitsgraph ist ein gerichteter, zyklenfreier Graph $\mathfrak{G} = (V, E, \xi)$ mit der Menge aller Knoten V, der Menge der gerichteten Kanten E sowie einer Funktion ξ, die jedem Knoten $v \in V$ eine Fuzzy-Formel $\mathcal{F} \in \mathfrak{F}$ zuordnet:*

$$\xi : V \rightarrow \mathfrak{F}$$

Die Menge aller Fuzzy-Abhängigkeitsgraphen bezeichen wir mit $\mathbb{G}$. Wenn deutlich gemacht werden soll, daß die Knoten- und Kantenmengen V und E zu dem Graphen $\mathfrak{G}$ gehören, schreiben wir $V(\mathfrak{G})$ und $E(\mathfrak{G})$.

Um eine Abhängigkeit zweier Konzepte festzuhalten, die durch die Fuzzy-Formeln $\mathcal{F}_1$ und $\mathcal{F}_2$ beschrieben sind, werden diese beiden Formeln mit zwei Knoten v_1, v_2 eines Fuzzy-Abhängigkeitsgraphen assoziiert und eine gerichtete Kante (v_1, v_2) eingefügt. Anschaulich bedeutet diese Abhängigkeit, daß eine Änderung der Fuzzy-Formel $\xi(v_1)$ eine Änderung der Fuzzy-Formel $\xi(v_2)$ bewirken kann. Man beachte, daß dabei nicht gefordert wird, daß die Fuzzy-Formeln auf der gleichen Grundmenge Ω definiert sind. Bedingt durch die Graph-Eigenschaften sind Abhängigkeiten transitiv und zyklenfrei.

Damit haben wir Abhängigkeiten zunächst *syntaktisch* festgehalten. Für die *Semantik* einer solchen Abhängigkeit werden wir in den nächsten beiden Kapiteln bei der Betrachtung der Verarbeitung Operatoren für Abhängigkeitsgraphen einführen und zeigen, wie die Anwendung eines Operators auf eine Fuzzy-Formel entsprechende Änderungen auf den abhängigen Fuzzy-Formeln auslöst.

Beispiel (Abhängigkeitsgraph) Als Beispiel für einen Abhängigkeitsgraphen zeigen wir, wie sich die von Boss definierten Wissensfamilien $\overrightarrow{\mathcal{W}} = \langle \mathcal{W}^1, \mathcal{W}^2, \ldots, \mathcal{W}^n \rangle$ modellieren lassen, wobei wir hierfür $\mathcal{W}^1, \ldots, \mathcal{W}^n \in \mathfrak{F}(\Omega)$ annehmen: man erhält den Spezialfall eines zu einer Liste degenerierten Abhängigkeitsgraphen $\mathfrak{G}_{\overrightarrow{\mathcal{W}}}(V, E, \xi)$ mit:

$$\begin{aligned} V &= \{v_1, \ldots, v_n\}, \\ E &= \{(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)\}, \\ \xi &= \{[v_1; \mathcal{W}^1], [v_2; \mathcal{W}^2], \ldots, [v_n; \mathcal{W}^n]\}. \end{aligned}$$

Man beachte, daß entsprechend dem Modell von [Bos96] hierbei alle Fuzzy-Formeln auf der gleichen Grundmenge Ω definiert sind. Im Allgemeinen Fall ist dies bei Abhängigkeitsgraphen, wie zuvor gesagt, nicht mehr erforderlich.

Die Menge aller direkt von einer Fuzzy-Formel abhängigen Fuzzy-Formeln bezeichnen wir als *Abhängigkeitsmenge*:

Definition 8.5.2 (Abhängigkeitsmenge) *Gegeben sei ein Fuzzy-Abhängigkeitsgraph $\mathfrak{G} = (V, E, \xi)$. Die Menge aller Knoten $\Gamma_{\mathfrak{G}}(v)$, mit $\Gamma_{\mathfrak{G}}(v) \subseteq V$, die von einem Knoten*

$v \in V$ abhängig ist:

$$\Gamma_{\mathfrak{G}}(v) \stackrel{\text{def}}{=} \{v' \in V | (v, v') \in E\}$$

bezeichnen wir als Abhängigkeitsmenge von v.

Um alle Fuzzy-Formeln zu ermitteln, die in einem Graph von einer Fuzzy-Formel erreicht werden können, bilden wir die *Transitive Abhängigkeitsmenge*:

Definition 8.5.3 (Transitive Abhängigkeitsmenge) *Gegeben sei ein Abhängigkeitsgraph $\mathfrak{G} = (V, E, \xi)$. Die Menge aller Fuzzy-Formeln $\Gamma^*_{\mathfrak{G}}(v)$, mit $\Gamma^*_{\mathfrak{G}}(v) \subseteq \mathcal{V}$, die über einen Pfad p von dem Knoten $v \in V$ erreichbar sind:*

$$\Gamma^*_{\mathfrak{G}}(v) \stackrel{\text{def}}{=} \{v' \in V | v \stackrel{p}{\rightsquigarrow} v'\}$$

bezeichnen wir als transitive Abhängigkeitsmenge von v.

Wir benötigen nun noch eine Reihe von Definitionen für die Bearbeitung von Fuzzy-Abhängigkeitsgraphen. Bei der Notation lehnen wir uns dabei an [Bol79] und [Cou90] an.

Wenn in einem Abhängigkeitsgraphen ein einzelner Knoten ausgezeichnet werden soll, bilden wir einen *Graphen mit Quelle:*

Definition 8.5.4 (Abhängigkeitsgraph mit Quelle) *Gegeben sei ein Abhängigkeitsgraph $\mathfrak{G} = (V, E, \xi)$ sowie ein beliebiger, ausgezeichneter Knoten $s \in V$, der als Quelle bezeichnet wird. Das Tupel $\langle \mathfrak{G}, s \rangle$ bezeichnen wir als Abhängigkeitsgraph mit Quelle. Die Menge aller Abhängigkeitsgraphen mit Quelle bezeichnen wir entsprechend mit $\langle \mathbb{G}, \mathbb{V} \rangle$.*

Die Menge aller von einem Knoten ausgehenden Kanten bezeichnen wir als *Kantenmenge eines Knotens*:

Definition 8.5.5 (Kantenmenge eines Knotens) *Gegeben sei ein Fuzzy-Abhängigkeitsgraph $\mathfrak{G} = (V, E, \xi)$. Die Menge aller von einem Knoten $v \in V$ ausgehenden Kanten $\overrightarrow{\Phi}_{\mathfrak{G}}(v)$, mit $\overrightarrow{\Phi}_{\mathfrak{G}}(v) \subseteq E$, ist definiert als:*

$$\overrightarrow{\Phi}_{\mathfrak{G}}(v) \stackrel{\text{def}}{=} \{(v, v') \in E | v' \in V\}$$

Umgekehrt definieren wir die Menge aller in einen Knoten $v \in V$ hineinführenden Kanten $\overleftarrow{\Phi}_{\mathfrak{G}}(v)$, mit $\overleftarrow{\Phi}_{\mathfrak{G}}(v) \subseteq E$, durch:

$$\overleftarrow{\Phi}_{\mathfrak{G}}(v) \stackrel{\text{def}}{=} \{(v', v) \in E | v' \in V\}.$$

Diese Definitionen erlauben es nun, Abhängigkeiten explizit zu modellieren und klar von der Programmlogik zu trennen. Sie lassen sich so in ein Datenmodell einbetten und separat von einer einzelnen Anwendung wiederverwenden, wie wir im nächsten Teil dieser Arbeit zeigen werden.

Kapitel 9

Verarbeitung imperfekter Daten

The temperature of Heaven can be rather accurately computed from available data. Our authority is Isaiah 30:26, "Moreover, the light of the Moon shall be as the light of the Sun and the light of the Sun shall be sevenfold, as the light of seven days." Thus Heaven receives from the Moon as much radiation as we do from the Sun, and in addition seven times seven (49) times as much as the Earth does from the Sun, or fifty times in all. The light we receive from the Moon is one ten-thousandth of the light we receive from the Sun, so we can ignore that. With these data we can compute the temperature of Heaven. The radiation falling on Heaven will heat it to the point where the heat lost by radiation is just equal to the heat received by radiation, i.e., Heaven loses fifty times as much heat as the Earth by radiation. Using the Stefan-Boltzmann law for radiation, $(H/E)^4 = 50$, *where E is the absolute temperature of the earth (300 °K), gives H as 798 °K (525 °C). The exact temperature of Hell cannot be computed, but it must be less than 444.6 °C, the temperature at which brimstone or sulphur changes from a liquid to a gas. Revelations 21:8 says "But the fearful, and unbelieving . . . shall have their part in the lake which burneth with fire and brimstone." A lake of molten brimstone means that its temperature must be at or below the boiling point, or 444.6 °C (Above this point it would be a vapor, not a lake.) We have, then, that Heaven, at 525 °C is hotter than Hell at 445 °C.*

"Applied Optics", vol. 11, A14, 1972

Bis jetzt haben wir den statischen Aspekt des Umgangs mit imperfekten Daten betrachtet: die Modellierung und Strukturierung der Informationen mit Hilfe von Fuzzy-Literalen, -Klauseln und -Formeln sowie Fuzzy-Abhängigkeitsgraphen.

Wie wir jedoch bereits bei der Anforderung VERARBEITUNG UNSCHARFER DATEN (Abschnitt 3.2.6 auf Seite 33) festgestellt haben, ist es eine typische Eigenschaft von Informationssystemen, daß sich Daten ändern. Die dynamischen Aspekte imperfekter Informationen müssen also von unserem Modell unterstützt werden.

In diesen Kapitel werden wir diese Anforderung präzisieren: was sind überhaupt sinnvolle Operationen, die auf solchen imperfekten Daten ausgeführt werden können?

9.1 Operationen auf imperfekten Daten

In Kapitel 2 haben wir bereits festgestellt, daß sich andere Ansätze typischerweise nicht mit der Verarbeitung unscharfer Daten beschäftigen. Als Grund haben wir unter anderem ausgemacht, daß die meisten existierenden Arbeiten einen datenbankzentrischen Ansatz verfolgen. Dabei stehen für Fuzzy-Daten die gleichen Operationen zur Verfügung wie für scharfe Daten. Diese klassischen Datenbankoperationen sind:

- das **Einfügen** neuer Daten *(Insert)*,
- das **Löschen** vorhandener Daten *(Delete)* und
- das **Ändern** vorhandener Daten *(Update)*.

Diese Operationen sind aber nicht ausreichend, um imperfekte Daten innerhalb eines Fuzzy-Informationssystems zu verarbeiten, und zwar aus den folgenden Gründen:

1. **Verarbeitung beschränkt sich nicht auf Persistenz.** Das Einfügen, Löschen und Ändern sind typische *Datenbank-Operationen*, die nur den Aspekt der persistenten Speicherung betreffen. Diese jedoch ist optional, wie wir in der Anforderung OBJEKTORIENTIERTES SYSTEM (siehe Abschnitt 3.2.5 auf Seite 32) festgestellt haben. Unverzichtbar ist dagegen die Verarbeitung innerhalb der Methoden eines Objektes, wie in der Anforderung VERARBEITUNG UNSCHARFER DATEN (Abschnitt 3.2.6 auf Seite 33) ausgeführt wird. Hierbei gelten aber andere Anforderungen als bei der persistenten Speicherung in einer Datenbank, denn diese Verarbeitung beschränkt sich typischerweise gerade nicht auf das einfache Setzen, Ändern und Auslesen eines Attributwertes.

2. **Trennung von Konzept und Zustand erforderlich.** Es muß unterschieden werden zwischen *Konzepten*, die mit imperfekten Informationen behaftet sein können (wie Attribute oder Objekte in einem Informationssystem) und dem imperfekten *Zustand* dieser Konzepte (ausgedrückt durch Abhängigkeitsgraphen und den darin enthaltenen Fuzzy-Formeln). Wie wir im nächsten Teil sehen werden, erreichen wir damit eine Entkopplung, ähnlich wie wir bereits die Struktur und die Interpretation imperfekter Daten in unserem Modell entkoppelt haben.

3. **Neue Informationen implizieren keine Weltänderung.** Die oben beschriebenen Operationen wendet man an, wenn die Welt sich ändert — zum Beispiel, wenn ein neuer Mitarbeiter hinzukommt *(insert)*, ein Kontostand sich ändert *(update)* oder ein Artikel nicht mehr hergestellt wird *(delete)*. Das Informationssystem muß also seine Miniwelt mit der geänderten externen Welt wieder in Einklang bringen. Nach der klassischen Einteilung von [KM91] entspricht dies einem *Update*. Demgegenüber steht der Fall, daß die Welt sich nicht ändert, aber neue Informationen über sie bekannt werden. In diesem Fall muß eine sogenannte *Revision* des vorhandenen Wissens durchgeführt werden.

Um sich den Unterschied zwischen den Änderungen bei einem *Update* und einer *Revision* zu veranschaulichen, kann man eine Anwendung zur Verwaltung von Zeugenaussagen betrachten, die beispielsweise eine Person beschreiben. Modelliert man Personen als Objekt, wird man ein Attribut „Körpergröße“ einführen, das mit einem scharfen Wert belegt werden kann (wie 1,80 m). Im Fall des Zeugenaussagenverwaltungssystems ist diese jedoch erst einmal unbekannt, es liegt lediglich eine Reihe von Aussagen vor, die den Wert einschränken. Die Aussage eines ersten Zeugen könnte lauten, daß die Person *groß* ist. Kommt nun die Aussage eines zweiten Zeugen hinzu — zum Beispiel, daß die Person mindestens 1,75 m groß ist — soll diese Aussage die erste keineswegs ersetzen!

Eine Änderungsoperation ist hier also nicht gefragt. Die Welt hat sich nicht geändert, die beschriebene Person ist immer noch die gleiche, es wurden lediglich neue Informationen über sie bekannt. Mit den oben beschriebenen Operationen wäre es jetzt nur möglich, die neue Information zusätzlich abzuspeichern. Was passiert aber, wenn eine dritte Aussage hinzukommt, die lauten könnte, daß die Person „sehr klein“ war? In diesem Fall haben wir einen zumindest teilweisen *Widerspruch*, eine *Inkonsistenz* (vergleiche Definition 8.3.1 auf Seite 82). Wir würden erwarten, daß uns das System wenigstens auf diese Tatsache aufmerksam macht. Dies ließe sich erreichen, wenn unser Aussagenverwaltungssystem die Fuzzy-Mengen der beteiligten Aussagen betrachtet und eine kombinierte Fuzzy-Menge berechnet. Damit wären wir jedoch in genau der Situation, die wir eingangs kritisiert haben: wir müßten für diesen Anwendungsfall eine eigene Logik realisieren.

Genau von diesem Ansatz wollten wir jedoch abrücken. Es stellt sich die Frage, ob dieser Fall formalisiert und allgemeine Operationen zum Umgang mit der *Konsistenz* von Fuzzy-Informationen angeboten werden können.

9.1.1 Konsistenzerhaltung

Aus der Fuzzy-Theorie sind natürlich Möglichkeiten zur Bearbeitung von Fuzzy-Mengen bekannt. Aber reichen diese für unser Modell und unser Einsatzgebiet schon aus? Weisbrod zum Beispiel untersucht in seiner Dissertation [Wei96] das possibilistische und das evidenzgestützte Schließen. Vereinfacht ausgedrückt, schränkt das

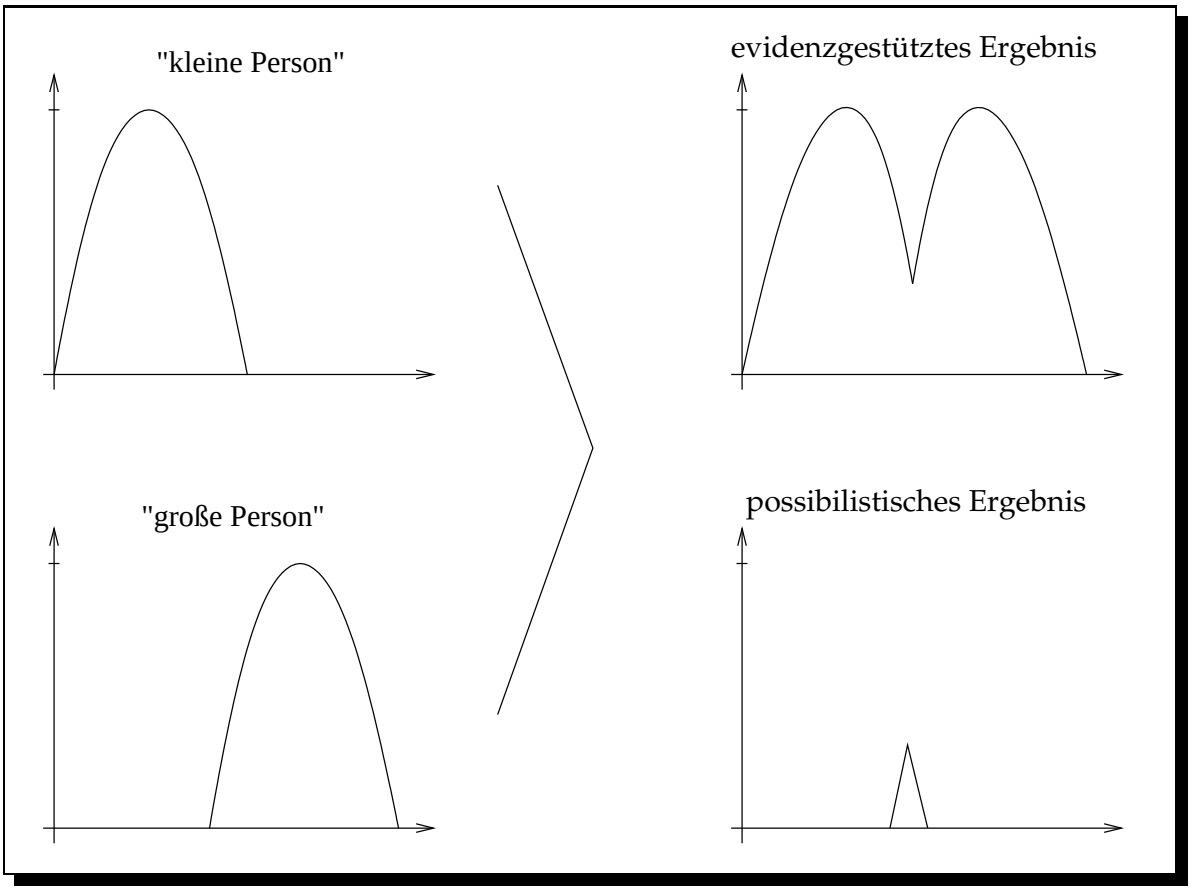


Abbildung 9.1: Inkonsistente Informationen und deren Kombination durch possibilistisches und evidenzgestütztes Schließen

possibilistische Schließen durch fortwährende Schnittmengenbildung die Verteilung immer weiter ein, während das evidenzgestützte Schließen durch Vereinigung der Fuzzy-Mengen den möglichen Wertebereich immer weiter ausdehnt.

Beide Ansätze führen zu Problemen bei inkonsistenten Informationen: im ersten Fall verschwindet die Verteilung, so daß schließlich kein Wert mehr in Frage kommt; im zweiten Fall werden immer mehr Werte möglich, bis schließlich alle Werte der Grundmenge enthalten sind. Wir haben dies auch als den absurden beziehungsweise den sinnleeren Zustand bezeichnet (vergleiche Abschnitt 8.2 auf Seite 80). Ein Beispiel für zwei Zeugenaussagen, die die Größe einer Person sehr unterschiedlich beschreiben, zeigt Abbildung 9.1. Nach der Kombination beider Aussagen erhält man eine Verteilung, die schon fast alle Körpergrößen als gut möglich repräsentiert, beziehungsweise kaum noch Werte übrig läßt. Weisbrod weist richtigerweise darauf hin, daß für eine sinnvolle Verarbeitung mit solchen Verfahren die Eigenschaften der Informationsquellen bekannt sein müssen. Er unterteilt sie daher in „mitteilsame Experten“,

die mit Informationen nicht „hinter dem Berg“ halten, auch wenn sie möglicherweise inkonsistent sind, und in „vorsichtige Experten“, die nur Informationen preisgeben, derer sie sich absolut sicher sind. Die erste Expertenart kann dann mit dem possibilistischen Schließen verarbeitet werden, während für letztere auf evidenzbasierte Verfahren zurückgegriffen wird.

Wenn man dieses Phänomen aus Datenbanksicht betrachtet, stellt man fest, daß es sich hierbei um eine *Metainformation* handelt, welche die Art der eigentlichen Information beschreibt. In der Praxis wird man jedoch typischerweise genau diese Metainformation nicht zur Verfügung haben, wie wir schon bei der Diskussion der Anforderung KONSISTENZERHALTUNG (siehe Abschnitt 3.2.8 auf Seite 35) diskutiert haben: Zum einen können unsere Informationen aus einer Vielzahl von Quellen kommen — wer kann zum Beispiel bei einer einzelnen Aussage pro Zeuge entscheiden, ob dieser vorsichtig oder mitteilsam ist? Wer soll diese Entscheidung treffen, wenn innerhalb eines Systems Hunderttausende, Millionen von Einzelinformationen eintreffen, wie in der Verkehrssteuerung oder in einem Internet-basierten Informationssystem? Ist es überhaupt sinnvoll, diese Betrachtungsweise aufrechtzuerhalten, wenn die Informationen nicht von Experten stammen, sondern von Heuristiken oder Meßsensoren?

Und selbst wenn diese Metainformation verfügbar ist, löst dieser Ansatz nicht das Grundproblem: Denn treten trotz aller Vorsicht Inkonsistenzen auf, führt dies unweigerlich zu den degenerierten Verteilungen. In obigem Beispiel kann bereits eine dritte Information ausreichen, den Zustand völliger Inkonsistenz zu erreichen.

Dies ist nicht tragbar, denn schließlich war der robuste Umgang mit Unsicherheiten und Inkonsistenzen gerade der Grund, warum klassische Informationssysteme nicht ausreichend sind und fuzzy-basierte Ansätze überhaupt erst eingesetzt werden sollen. Die aus der Literatur bekannten Verfahren bieten jedoch gerade für dieses Problem keine Lösung an, was ursächlich für die fehlende Akzeptanz von Fuzzy-Ansätzen im Bereich der Informationssysteme sein dürfte.

Die bekannten Verfahren zur Verarbeitung von Fuzzy-Daten sind offensichtlich nicht ausreichend. Gesucht sind Operatoren, die die Erhaltung der Konsistenz bei der Verarbeitung imperfekter Daten garantieren können. Durch die hier vorgenommene Trennung von Syntax und Semantik erhalten wir völlig neue Möglichkeiten, die deutlich über die einfache Kombination von Fuzzy-Mengen hinausgehen. Beispielsweise können wir Inkonsistenzen vermeiden, indem wir aus einer Information in Form einer Fuzzy-Formel eine Klausel herausnehmen, bevor sie mit einer zweiten Formel kombiniert wird. Dadurch *ändern* sich entsprechend die beteiligten Fuzzy-Mengen, was Verarbeitungsverfahren zuläßt, die weder mit der possibilistischen, noch mit der evidenzgestützten Variante Ähnlichkeit haben.

Um dies zu ermöglichen, brauchen wir eine Methode, um die Syntax einer Fuzzy-Formel geeignet zu modifizieren. Dieses Problem ist aber bereits aus dem Bereich Wissensrevision in der Künstlichen Intelligenz bekannt. Die Grundidee hierbei ist, sich von der Vorstellung der monoton ändernden Wissensmenge zu verabschieden:

Der Erwerb neues Wissens kann dazu führen, daß alte Informationen überflüssig werden und entfernt werden müssen, damit ein konsistenter Zustand wiederhergestellt werden kann.

9.1.2 Konzepte der Wissensrevision

Im Bereich der Künstlichen Intelligenz kennt man schon länger das Problem, daß (scharfe) Aussagen in einem Wissensverwaltungssystem *(truth maintenance system)* inkonsistent zueinander werden können [GRS00]:

> *Und das werden Wissensbasen im Laufe ihrer Verwendung immer: sei es, dass sich die Welt verändert, sei es, dass sich das Wissen über die Welt verändert!*

Man brauchte also Verfahren, die es ermöglichten, Informationen zu einer solchen Wissensbasis hinzuzufügen und dabei gleichzeitig einen konsistenten Zustand zu bewahren. Ein klassisches Beispiel ist die Aussagenmenge $\{a_1, a_2\}$ mit

a_1 = *„alle Vögel können fliegen“*,
a_2 = *„Pinguine sind Vögel“*

und der Schlußfolgerung a_3:

a_3 = *„Pinguine können fliegen“*.

Soweit ist unsere Wissensbasis noch konsistent. Kommt jedoch eine vierte Aussage a_4 hinzu:

a_4 = *„Pinguine können nicht fliegen“*

haben wir ein Problem. An diesem Beispiel zeigt sich schon, daß es keine eindeutige Lösung gibt, um einen konsistenten Zustand wiederherzustellen. Eine einfache Möglichkeit wäre, in so einem Fall alle alten Aussagen zu entfernen und nur die neue aufzunehmen, was natürlich sehr unbefriedigend ist. Aus diesem Grund wurde das *Prinzip der minimalen Änderung* formuliert, das besagt, daß nur solche Änderungen an der Wissensmenge durchgeführt werden dürfen, die absolut notwendig sind (Harman, zitiert in [Gär88]):

> *When changing beliefs in response to new evidence, you should continue to believe as many of the old beliefs as possible.*

Der Widerspruch, einerseits keine eindeutige Änderungsoperation angeben zu können, andererseits aber nicht völlig auf eine Formalisierung verzichten zu wollen, wurde erstmalig in den Arbeiten von Alchourrón, Makinson und Gärdenfors aufgelöst [AGM85]. Bei diesem Ansatz werden neben der sogenannten *Expansion* die konsistenzerhaltenden Operationen *Revision* und *Kontraktion* definiert und eine Reihe von Postulaten aufgestellt, die diese Operationen erfüllen müssen. Diese *AGM-Postulate*, oft auch nur als *Gärdenfors-Postulate* bezeichnet, formalisieren wünschenswerte Eigenschaften dieser Operationen, etwa daß im Falle einer Inkonsistenz nicht einfach sämtliche alten Informationen gelöscht werden dürfen.

Im einzelnen haben diese Operationen die folgende Bedeutung:

Expansion Die Expansion fügt eine neue Aussage zu einer Wissensbasis hinzu. Sie ist als einzige der beschriebenen eine monotone Operation, da die vorhandenen Informationen nicht geändert werden; dadurch kann es vorkommen, daß eine Wissensbasis, die durch Expansion um eine inkompatible Information erweitert wurde (wie im Beispiel der nicht-fliegenden Pinguine) inkonsistent wird.

Revision Die Revision fügt eine Information zu einer Wissensbasis hinzu, stellt dabei aber gleichzeitig sicher, daß wieder ein konsistenter Zustand erreicht wird. Um dies zu erreichen, werden gegebenenfalls vorhandene Informationen entfernt, die mit der neuen Information in Widerspruch stehen. Die Revision ist daher eine *nichtmonotone* Operation. Wie oben angedeutet, ist es aber nicht eindeutig, welche Informationen entfernt werden müssen: Schlimmstenfalls verliert man alle bisher bekannten Aussagen.

Kontraktion Die Kontraktion ist das Gegenstück zur Revision. Es wird eine bisher als gültig angenommene Aussage aus der Wissensbasis entfernt. Auch die Kontraktion ist eine nichtmonotone Operation, da es notwendig sein kann, mehr als nur die zu entfernende Information aus der Wissensbasis zu nehmen. Es reicht beispielsweise nicht, nur die Aussage a_3 zu entfernen, nach der Pinguine fliegen können — ein System könnte diese sofort wieder aus den ersten beiden Fakten herleiten.

Wissensrevision wurde von Alchourrón, Gärdenfors und Makinson ursprünglich für deduktiv abgeschlossene Satzmengen formuliert [AGM85,Gär88,Gär92]. Dieses auch als *Theorierevision* bezeichnete Verfahren ist jedoch für den praktischen Einsatz denkbar ungeeignet, da die Berechnung des Abschlusses sehr aufwendig ist. In der Praxis verwendet man daher die sogenannte *Basisrevision*, bei der lediglich eine endliche Faktenbasis modifiziert wird [Neb90, Neb91, Neb92].[1]

[1] Nebel [Neb89] hat gezeigt, daß die Basisrevision als Spezialfall einer Theorierevision durch partielle Schnittrevision aufgefaßt werden kann, wobei die Selektionsfunktion nur Elemente aus der Basis auswählt.

Die im folgenden anstehende Aufgabe wird daher sein, diese Operationen in sinnvoller Weise auf das hier verwendete Modell zu übertragen und effiziente Implementierungsmöglichkeiten aufzuzeigen.

9.1.3 Weitere Operationen

Bis jetzt haben wir den Aspekt der konsistenzerhaltenden Verarbeitung betont, da dieser in der Literatur bisher stark vernachlässigt wurde. Zusätzlich benötigen wir aber auch eine Reihe „einfacher“ Operationen, auf die wir im folgenden kurz eingehen.

Neben den einfachen Mengenoperationen Vereinigung und Schnitt, die wir bereits im letzten Kapitel definiert haben, benötigen wir die Negation für Fuzzy-Formeln und -Klauseln. Aus formalen Gründen führen wir zusätzlich den Begriff der Addition und Subtraktion von Fuzzy-Klauseln auf Fuzzy-Formeln ein.

Für die zuvor besprochenen konsistenzerhaltenden Operationen benötigen wir außerdem einen Transformationsbegriff, der eine Umsetzung von Fuzzy-Mengen sowie Fuzzy-Klauseln und -Formeln, die auf verschiedenen Grundmengen definiert sind, ermöglicht.

Für die Umwandlung der Fuzzy-Informationen in scharfe Informationen schließlich werden Defuzzifizierungsoperatoren benötigt.

In der Tabelle 9.1 auf der nächsten Seite fassen wir die besprochenen Operationen zusammen. Den konsistenzerhaltenden Operationen widmen wir dabei ein eigenes Kapitel. Im folgenden betrachten wir zunächst die übrigen Operationen, die auf Fuzzy-Literalen, -Klauseln und Fuzzy-Formeln ausgeführt werden können.

9.2 Operationen auf Fuzzy-Atomen und -Literalen

Wir beginnen mit der Definition von Operationen auf Fuzzy-Atomen und Fuzzy-Literalen.

Das Beispiel aus dem Reengineering (vergleiche Abschnitt 8.5, Seite 85) hat gezeigt, daß Abhängigkeiten zwischen Fuzzy-Formeln auftreten können, die auf unterschiedlichen Grundmengen Ω definiert sind. Für die Operationen, die wir im nächsten Kapitel betrachten, benötigen wir daher eine Möglichkeit, Fuzzy-Interpretationen auf verschiedenen Domänen ineinander umzurechnen. Dies geschieht mittels sogenannter *Transformationsfunktionen:*

Definition 9.2.1 (Transformationsfunktion) *Eine Transformationsfunktion Θ bildet eine Fuzzy-Menge $\mu \in F(\Omega)$ auf eine Fuzzy-Menge $\mu' \in F(\Omega')$ ab:*

$$\Theta_{\Omega,\Omega'} : F(\Omega) \rightarrow F(\Omega').$$

Operator	Name	Argumente	Definition
$\cup$	Vereinigung	$\mathfrak{K}(\Omega) \times \mathfrak{K}(\Omega) \to \mathfrak{K}(\Omega)$ $\mathfrak{F}(\Omega) \times \mathfrak{F}(\Omega) \to \mathfrak{F}(\Omega)$	Nr. 8.1.4, Seite 75 Nr. 8.1.7, Seite 77
$\cap$	Schnitt	$\mathfrak{K}(\Omega) \times \mathfrak{K}(\Omega) \to \mathfrak{K}(\Omega)$ $\mathfrak{F}(\Omega) \times \mathfrak{F}(\Omega) \to \mathfrak{F}(\Omega)$	Nr. 8.1.4, Seite 75 Nr. 8.1.7, Seite 77
$\neg$	Negation	$\mathfrak{F}(\Omega) \to \mathfrak{F}(\Omega)$	Nr. 9.3.2, Seite 99
$+$	Addition	$\mathfrak{F}(\Omega) \times \mathfrak{K}(\Omega) \to \mathfrak{F}(\Omega)$ $\mathfrak{F}(\Omega) \times \mathfrak{F}(\Omega) \to \mathfrak{F}(\Omega)$	Nr. 9.3.3, Seite 100
$-$	Subtraktion	$\mathfrak{F}(\Omega) \times \mathfrak{K}(\Omega) \to \mathfrak{F}(\Omega)$ $\mathfrak{F}(\Omega) \times \mathfrak{F}(\Omega) \to \mathfrak{F}(\Omega)$	Nr. 9.3.4, Seite 100
Θ	Transformation	$F(\Omega) \to F(\Omega')$ $\mathfrak{F}(\Omega) \to \mathfrak{F}(\Omega')$	Nr. 9.2.1, Seite 96 Nr. 9.3.1, Seite 99
δ	Domänenabbildung	$\bigcup_\Omega \mathfrak{F}(\Omega) \to \bigcup_\Omega \{\Omega\}$	Nr. 9.3.5, Seite 100
∇	Defuzzifizierung	$\bigcup_\Omega \mathfrak{F} \to \bigcup_\Omega 2^\Omega$	Nr. 9.3.6, Seite 100
$+_\gamma$	γ-Expansion	$\mathfrak{F}(\Omega) \times \mathfrak{F}(\Omega) \to \mathfrak{F}(\Omega)$	Nr. 10.1.1, Seite 102
$\oplus_\gamma$	γ-Revision	$\mathfrak{F}(\Omega) \times \mathfrak{F}(\Omega) \to \mathfrak{F}(\Omega)$	Nr. 10.2.1, Seite 108
$\oplus_\gamma^R$	epistemische γ-Revision	$\mathfrak{F}(\Omega) \times \mathfrak{F}(\Omega) \to \mathfrak{F}(\Omega)$	Nr. 10.2.13, Seite 122
$\ominus_\gamma$	γ-Kontraktion	$\mathfrak{F}(\Omega) \times \mathfrak{F}(\Omega) \to \mathfrak{F}(\Omega)$	Nr. 10.3.1, Seite 124
$+_\gamma^{\mathfrak{G}}$	Standard Graph-γ-Expansion	$\langle \mathbb{G}, \mathbb{V} \rangle \times \mathfrak{F} \to \mathbb{G}$	Nr. 10.4.1, Seite 130
$\tilde{+}_\gamma^{\mathfrak{G}}$	Strenge Graph-γ-Expansion	$\langle \mathbb{G}, \mathbb{V} \rangle \times \mathfrak{F} \to \mathbb{G}$	Nr. 10.4.3, Seite 134
$\oplus_\gamma^{\mathfrak{G}}$	Graph-γ-Revision	$\langle \mathbb{G}, \mathbb{V} \rangle \times \mathfrak{F} \to \mathbb{G}$	Nr. 10.5.1, Seite 137
$\ominus_\gamma^{\mathfrak{G}}$	Graph-γ-Kontraktion	$\langle \mathbb{G}, \mathbb{V} \rangle \times \mathfrak{F} \to \mathbb{G}$	Nr. 10.6.1, Seite 143

Tabelle 9.1: Operationen auf Fuzzy-Atomen, -Klauseln und -Formeln

Die Transformation eines Fuzzy-Atoms ergibt so ein neues Fuzzy-Atom, dessen Name man in einer praktischen Anwendung durch *„Transformation von ⟨Begriff⟩ auf ⟨Domäne⟩“* ausdrücken wird und dessen Fuzzy-Interpretation sich auf die oben beschriebene Weise berechnet.

Beispiel (Transformationsfunktion) Wir greifen den schon gezeigten Einsatzfall des Reengineering wieder auf. Die erste Domäne repräsentiert hier Informationen über den Typ einer Variablen: $\Omega_1 = \text{typ} = \{int, string, float, bool\}$, die zweite die möglichen Verwendungsarten $\Omega_2 = \text{art} = \{steuerung, ausgabe, index, berechnung, eingabe\}$. Wir können nun eine Transformationsfunktion $\Theta_{\text{typ,art}}$ definieren, die eine Fuzzy-Menge von der Domäne Ω_1 (typ) in eine Fuzzy-Menge auf der Domäne Ω_2 (art) umrechnet:

$$\Theta_{\text{typ,art}}(\mu_1)(\omega_2) = \begin{cases} \mu_1(\text{int}) & \textbf{wenn } \omega_2 = \text{index} \\ \mu_1(\text{float}) & \textbf{wenn } \omega_2 = \text{berechnung} \\ \mu_1(\text{bool}) & \textbf{wenn } \omega_2 = \text{steuerung} \\ \mu_1(\text{string}) & \textbf{sonst.} \end{cases}$$

Ein Beispiel für die Umrechnung einer konkreten Fuzzy-Menge $\mu_1 \in F(\text{typ})$ in eine Fuzzy-Menge $\mu_2 \in F(\text{art})$ zeigt Abbildung 9.2.

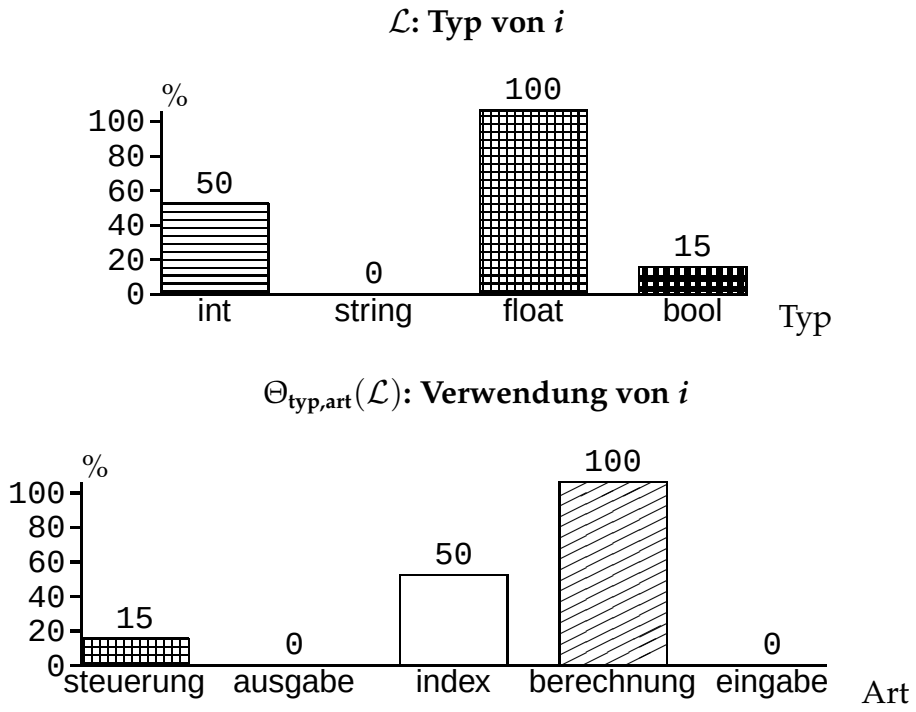


Abbildung 9.2: Transformation einer Fuzzy-Menge

9.3 Operationen auf Fuzzy-Klauseln und Fuzzy-Formeln

Bis jetzt haben wir lediglich die Transformation von Fuzzy-Mengen definiert. Für die in den nächsten Kapiteln beschriebenen Operatoren benötigen wir aber die Transformation von Fuzzy-Formeln, die wir in diesem Abschnitt einführen:

Definition 9.3.1 (Formeltransformation) *Die Fortsetzung $\Theta^{\mathcal{F}}_{\Omega,\Omega'}$ einer Transformationsfunktion $\Theta_{\Omega,\Omega'}$ auf Fuzzy-Formeln über Ω ist die folgendermaßen definierte Abbildung:*

$$\Theta^{\mathcal{F}}_{\Omega,\Omega'} : \mathfrak{F}(\Omega) \to \mathfrak{F}(\Omega')$$

Dabei gilt für $\mathcal{F}' \in \mathfrak{F}(\Omega')$:

$$\mathcal{F}' \stackrel{\text{def}}{=} \{\mathcal{K}'\} \quad mit \quad \mathcal{K}' \stackrel{\text{def}}{=} \{\mathcal{L}'\}, \quad \mathcal{L}' \stackrel{\text{def}}{=} \mathcal{A}'$$

und schließlich

$$\mu_{\mathcal{A}'} = \Theta_{\Omega,\Omega'}(\mu_{\mathcal{F}}).$$

Bei einer solchen Transformation entsteht also eine neue Formel, die unabhängig von der ursprünglichen syntaktischen Form nur eine einzige Klausel enthält, da die Transformation über die Fuzzy-Interpretation durchgeführt wird. Dies geschieht, weil sich die transformierte Fuzzy-Formel so erheblich effizienter berechnen läßt. Für die hier betrachteten Anwendungen ergeben sich daraus keine Nachteile; grundsätzlich ist es aber auch möglich, alle syntaktischen Bestandteile einzeln zu transformieren, um daraus eine neue Fuzzy-Formel zu rekonstruieren. Dieser Weg sollte jedoch nur verfolgt werden, wenn eine Anwendung die strukturellen Informationen auch für solchermaßen transformierte Formeln benötigt.

Bei der *Negation* von Fuzzy-Formeln verfolgen wir ein ähnliches Prinzip:

Definition 9.3.2 (Negation von Fuzzy-Formeln) *Für die Negation $\overline{\mathcal{F}}$ einer Fuzzy-Formel $\mathcal{F} \in \mathfrak{F}(\Omega)$ definieren wir zunächst ein Fuzzy-Atom $\mathcal{A}_{\overline{\mathcal{F}}}$ mit $\mu_{\mathcal{A}_{\overline{\mathcal{F}}}} \stackrel{\text{def}}{=} \mu_{\mathcal{F}}$. Das Fuzzy-Literal $\mathcal{L}_{\overline{\mathcal{F}}} \in \mathfrak{L}(\Omega)$ ist dann definiert als die Negation dieses Atoms:*

$$\mathcal{L}_{\overline{\mathcal{F}}} \stackrel{\text{def}}{=} \neg\mathcal{A}_{\overline{\mathcal{F}}}$$

Mit Hilfe dieses Literals können wir nun eine Fuzzy-Klausel $\mathcal{K}_{\overline{\mathcal{F}}} \in \mathfrak{K}(\Omega)$ definieren:

$$\mathcal{K}_{\overline{\mathcal{F}}} \stackrel{\text{def}}{=} \{\mathcal{L}_{\overline{\mathcal{F}}}\}$$

Die Negation einer Fuzzy-Formel ergibt sich damit als die Menge aus der Klausel $\mathcal{K}_{\overline{\mathcal{F}}}$:

$$\overline{\mathcal{F}} \stackrel{\text{def}}{=} \{\mathcal{K}_{\overline{\mathcal{F}}}\}.$$

Wie bei der Transformation wird also die Negation auf der Basis der Fuzzy-Interpretation einer Formel berechnet, um daraus die negierte Fuzzy-Formel zu konstruieren. Hier gelten die gleichen Argumente wie oben; allerdings ist der Effizienzgewinn hier besonders hoch, da die Negation der syntaktischen Form mit anschließender Wiederherstellung der konjunktiven Normalform schlimmstenfalls eine exponentielle Laufzeit benötigt. Die Fuzzy-Interpretation ist jedoch bei beiden Verfahren identisch. Entsprechend sollte man eine syntaktische Negation nur dann vornehmen, wenn die strukturelle Form für negierte Fuzzy-Formeln tatsächlich benötigt wird.

Aus praktischen Gründen definieren wir nun noch die *Addition* und die *Subtraktion* von Klauseln und Formeln:

Definition 9.3.3 (Addition von Klauseln und Formeln) *Die Addition einer Fuzzy-Klausel $\mathcal{K} \in \mathfrak{K}(\Omega)$ zu einer Fuzzy-Formel $\mathcal{F} \in \mathfrak{F}(\Omega)$ ist definiert als:*

$$\mathcal{F} + \mathcal{K} \stackrel{\text{def}}{=} \mathcal{F} \cup \{\mathcal{K}\}.$$

Entsprechend ist die Addition zweier Fuzzy-Formeln $\mathcal{F}_1, \mathcal{F}_2 \in \mathfrak{F}(\Omega)$ definiert als:

$$\mathcal{F}_1 + \mathcal{F}_2 \stackrel{\text{def}}{=} \mathcal{F}_1 \cup \mathcal{F}_2.$$

Definition 9.3.4 (Subtraktion von Klauseln und Formeln) *Die Subtraktion einer Fuzzy-Klausel $\mathcal{K} \in \mathfrak{K}(\Omega)$ von einer Fuzzy-Formel $\mathcal{F} \in \mathfrak{F}(\Omega)$ ist definiert als:*

$$\mathcal{F} - \mathcal{K} \stackrel{\text{def}}{=} \mathcal{F} \backslash \{\mathcal{K}\}.$$

Die Subtraktion zweier Fuzzy-Formeln $\mathcal{F}_1, \mathcal{F}_2 \in \mathfrak{F}(\Omega)$ ist entsprechend definiert als:

$$\mathcal{F}_1 - \mathcal{F}_2 \stackrel{\text{def}}{=} \mathcal{F}_1 \backslash \mathcal{F}_2.$$

Ebenfalls aus formalen Gründen benötigen wir eine Möglichkeit, den Definitionsbereich der Fuzzy-Interpretation einer Fuzzy-Formel zu ermitteln:

Definition 9.3.5 (Domänenabbildung) *Die Domänenabbildung δ ordnet einer Fuzzy-Formel $\mathcal{F} \in \mathfrak{F}(\Omega)$ die Grundmenge Ω ihrer Fuzzy-Interpretation zu:*

$$\delta : \mathcal{F} \mapsto \Omega.$$

Mit Hilfe dieser Definition können wir ermitteln, ob zwei Fuzzy-Formeln in ihrer Interpretation als Fuzzy-Menge auf der gleichen Grundmenge Ω definiert sind.

Die Defuzzifizierung bildet eine Fuzzy-Formel auf eine Untermenge des Wertebereiches ihrer Interpretation als Fuzzy-Menge ab:

Definition 9.3.6 (Defuzzifizierung) *Ein Defuzzifizierungsoperator ∇ bildet eine Fuzzy-Formel $\mathcal{F} \in \mathfrak{F}(\Omega)$ auf Elemente der Potenzmenge von Ω ab:*

$$\nabla : \mathfrak{F}(\Omega) \rightarrow 2^{\Omega}.$$

Man beachte, daß wir damit lediglich die Syntax für die Defuzzifizierung festgelegt haben — zur Berechnung einer konkreten Ergebnismenge wird ein Defuzzifizierungsverfahren wie die in Abschnitt 6.3 auf Seite 57 vorgestellte Maximum-Methode benötigt.

Kapitel 10

Konsistenzerhaltung: Expansion, Revision und Kontraktion

My speciality is being right when other people are wrong.
George Bernard Shaw

Wir widmen uns nun genauer dem Aspekt der KONSISTENZERHALTUNG bei der VERARBEITUNG UNSCHARFER DATEN. Zur Erfüllung dieser Anforderungen übertragen wir die im letzten Kapitel vorgestellten Operationen Expansion, Revision und Kontraktion auf unser Fuzzy-Modell. Alle diese Operationen unterscheiden sich von den bekannten Verarbeitungsverfahren darin, daß sie die Erhaltung der Konsistenz garantieren. Um dies zu ermöglichen, sind sie jedoch — mit Ausnahme der Expansion — nicht-monoton.

Wir betrachten zunächst die Definition dieser Operationen für einzelne Fuzzy-Formeln. Während wir für die Expansion auf Vorarbeiten zurückgreifen können [Wit95, Bos96], zeigen wir in dieser Arbeit erstmalig, welche Eigenschaften alle in diesem Modell möglichen Revisionsoperatoren erfüllen müssen [Wit02]. Wir betrachten darüber hinaus den Fall der Kontraktion auf Fuzzy-Formeln. Anschließend erfolgt die Verallgemeinerung dieser Operationen auf die in Kapitel 8.5 eingeführten Fuzzy-Abhängigkeitsgraphen.

10.1 Expansion von Fuzzy-Formeln

Die Expansion ist eine monotone Erweiterung einer Fuzzy-Formel $\mathcal{F}$ um eine Fuzzy-Formel $\mathcal{G}$. Monoton bedeutet hier, daß keine der in der Formel $\mathcal{F}$ enthaltenen Fuzzy-Klauseln bei einer Expansionsoperation entfernt werden.

Die aus der Literatur für den aussagenlogischen Fall bekannte Expansion fügt eine neue Information immer hinzu, auch wenn danach ein inkonsistenter Zustand entsteht. Bei dieser Formulierung der Expansion erreicht eine Formel den absurden (oder sinnleeren) Zustand, in dem kein Wert mehr möglich ist (beziehungsweise alle Werte möglich werden). Für die von uns vorgesehenen Anwendungen ist dies jedoch nicht sinnvoll, da während des Verarbeitungsprozesses durchaus zeitweise Inkonsistenzen auftreten können, die dann im späteren Verlauf bereinigt werden. Solche Inkonsistenzen sind eine wünschenswerte Eigenschaft von Fuzzy-Informationssystemen, denn der kontrollierte Umgang mit ihnen soll ja gerade einen der Vorteile gegenüber klassischen Realisierungen ausmachen. Das heißt aber nicht, daß ein solches System beliebig inkonsistent werden darf. Vielmehr muß es möglich sein, Inkonsistenzen graduell und genau kontrolliert zuzulassen.

Der hier vorgeschlagene Ansatz ist daher, Operationen auf Fuzzy-Formeln und Abhängigkeitsgraphen mit einem Konsistenzgrad γ auszustatten, den das Ergebnis einer Operation mindestens erreichen muß. Entsprechend definieren wir im folgenden eine *γ-Expansionsoperation*, bei der γ den minimalen Konsistenzgrad der Ergebnisformel *nach einer erfolgreichen* Expansionsoperation angibt. Wenn der geforderte Konsistenzgrad nicht erreicht werden kann, ist die Operation *erfolglos*. In diesem Fall bleibt die Formel unverändert, der Konsistenzgrad also gleich. Da der Konsistenzgrad bei einer γ-Expansion nie zunehmen kann (denn sie ist ja monoton), bedeutet dies auch, daß es keinen Sinn macht, bei einer γ-Expansion einen höheren Konsistenzgrad zu fordern, als die beteiligten Formeln vor der Expansion aufweisen.

Die geforderten Eigenschaften einer γ-Expansionsoperation werden formal in einer Reihe von Postulaten festgehalten. Anschließend wird untersucht, welche Operationen diese Eigenschaften erfüllen, und ein Algorithmus für die γ-Expansion angegeben.

Zuerst legen wir aber die Syntax für eine γ-Expansion fest:

Definition 10.1.1 (γ-Expansion) *Eine γ-Expansion $+_\gamma$ ist eine Abbildung, die zwei Fuzzy-Formeln $\mathcal{F}, \mathcal{G} \in \mathfrak{F}(\Omega)$ und einem Wert $\gamma \in [0,1]$ eine weitere Fuzzy-Formel aus $\mathfrak{F}(\Omega)$ zuordnet:*

$$+_\gamma : \mathfrak{F}(\Omega) \times \mathfrak{F}(\Omega) \to \mathfrak{F}(\Omega)$$

und die den unten aufgeführten Postulaten $E_\gamma 1$–$E_\gamma 6$ (Seite 103f) genügt.

Enthält die Formel $\mathcal{G}$ nur eine Klausel $\mathcal{K}$, erlauben wir anstelle von $\mathcal{F} +_\gamma \{\mathcal{K}\}$ die abkürzende Schreibweise $\mathcal{F} +_\gamma \mathcal{K}$. Aus stilistischen Gründen reden wir auch einfach von der Expansion von Fuzzy-Formeln, wenn klar ist, daß nicht die aussagenlogische Variante, sondern die γ-Expansion gemeint ist.

10.1.1 Die Postulate der γ-Expansion

Es verbleibt die Aufgabe, die Semantik der γ-Expansion präzise festzulegen. Dafür gehen wir von den bekannten AGM-Postulaten aus, die sich als Grundlage für die Wissensrevision etabliert haben, und übersetzen diese auf unser Fuzzy-Modell. Die Originalpostulate haben wir zum Vergleich in Anhang A.2 auf Seite 308 aufgeführt. Änderungen an den Postulaten werden dabei nur soweit nötig vorgenommen; wenn eine Änderung erforderlich ist, richten wir uns bei der Neufassung stets an der Intention aus, die hinter den Originalpostulaten stand. Anschließend wird untersucht, welche Eigenschaften sich aus den Postulaten für die jeweiligen Operatoren ergeben. Im Fall der Expansion mag der formale Aufwand für das einfache Ergebnis etwas hoch erscheinen, wir zeigen an dieser Stelle aber trotzdem die Übersetzung im Detail, um zum einen eine einheitliche Vorgehensweise beizubehalten und zum anderen den Leser in die Idee der Wissensrevision und unsere Vorgehensweise bei der Übersetzung der Postulate einzuführen. Überdies werden wir bei der Betrachtung der γ-Revision auf die Postulate und die daraus folgenden Eigenschaften der γ-Expansion zurückgreifen.

Eine wesentliche Änderung zu den Original-Postulaten ist, daß wir keinen Inferenzmechanismus vorsehen. Wie bereits in Abschnitt 9.1.2 auf Seite 94 diskutiert, wäre der damit verbundene Aufwand zu hoch für den geplanten Einsatzzweck: denn Ziel unseres Modells ist vor allem, ein geeignetes Abbild von Imperfektion für eine Miniwelt zu schaffen, also imperfekte Zustände zu repräsentieren. Wichtig dabei ist, wie oben ausgeführt, einen bestimmen Grad an innerer Konsistenz bei der Ausführung von Operatoren garantieren zu können. Eine der Grundideen bei der Übersetzung der Postulate war daher, diese Konsistenz auf der Basis der Fuzzy-Interpretation einer Formel mit Hilfe des $C(\cdot)$-Operators (vergleiche Definition 8.2.3 auf Seite 80) zu bestimmen.

Um den Vergleich mit den Original-Postulaten zu erleichtern, wurde die ursprüngliche Numerierung beibehalten. Postulat $E_\gamma 2$, das aus Gründen der Übersichtlichkeit in zwei Teilpostulate aufgesplittet wurde, ist daher mit Buchstaben indiziert.

Für alle diese Postulate gilt: $\mathcal{F}, \mathcal{G}$ bezeichnen Fuzzy-Formeln aus $\mathfrak{F}(\Omega)$.

$E_\gamma 1$ (Stabilität) Das Ergebnis einer γ-Expansionsoperation ist wieder eine Fuzzy-Formel:

$$\mathcal{F} +_\gamma \mathcal{G} \in \mathfrak{F}(\Omega)$$

$E_\gamma 2a$ (Erfolg) Wenn die Informationen $\mathcal{F}$ und $\mathcal{G}$ zusammen mindestens den Konsistenzgrad γ erreichen, soll die γ-Expansion erfolgreich sein:

$$\textbf{Wenn } C(\mathcal{F} \cup \mathcal{G}) \geq \gamma, \textbf{dann } \mathcal{G} \subseteq \mathcal{F} +_\gamma \mathcal{G}$$

$E_\gamma 2b$ (Mißerfolg) Unterschreitet der Konsistenzgrad von $\mathcal{F}$ vereinigt mit $\mathcal{G}$ den Wert γ, soll die Ausgangsformel unverändert bleiben:

$$\textbf{Wenn } C(\mathcal{F} \cup \mathcal{G}) < \gamma, \textbf{dann } \mathcal{F} +_\gamma \mathcal{G} = \mathcal{F}$$

E_γ3 (Zunahme) Die Informationsmenge nimmt zu — sie expandiert:

$$\mathcal{F} +_\gamma \mathcal{G} \supseteq \mathcal{F}$$

E_γ4 (Invarianz) Ist die Information $\mathcal{G}$ schon bekannt, gilt Invarianz:

$$\mathcal{G} \subseteq \mathcal{F} \Rightarrow \mathcal{F} +_\gamma \mathcal{G} = \mathcal{F}$$

E_γ5 (Monotonie) Sei $\mathcal{F}' \subseteq \mathcal{F}$. Wenn die Expansion $\mathcal{F} +_\gamma \mathcal{G}$ erfolgreich oder die Expansion $\mathcal{F}' +_\gamma \mathcal{G}$ nicht erfolgreich ist, gilt die Monotonie:

$$\textbf{Wenn } (\mathcal{G} \subseteq \mathcal{F}+_\gamma\mathcal{G}) \vee (\mathcal{G} \not\subseteq \mathcal{F}'+_\gamma\mathcal{G}), \textbf{ dann } \mathcal{F}'+_\gamma\mathcal{G} \subseteq \mathcal{F}+_\gamma\mathcal{G}$$

E_γ6 (Vorhersehbarkeit) Das Ergebnis der γ-Expansion soll nur Fuzzy-Klauseln enthalten, die aus $\mathcal{F}$ oder $\mathcal{G}$ stammen:

$$\mathcal{F} +_\gamma \mathcal{G} \subseteq \mathcal{F} \cup \mathcal{G}$$

Die meisten dieser Postulate lassen sich dabei leicht nachvollziehen. Postulat E_γ1 garantiert, daß die Expansion einer Fuzzy-Formel wieder eine Fuzzy-Formel ergibt.

Die Postulate E_γ2a und E_γ2b regeln den Fall der erfolgreichen und erfolglosen Expansion. Im Erfolgsfall sollen die Informationen der Fuzzy-Formel $\mathcal{G}$ im Ergebnis enthalten sein; im Fall des Mißerfolges bleibt die Ausgangsformel unverändert. Diese Formulierung unterscheidet sich von der Originalform: dort wird eine Expansion immer durchgeführt, im Falle einer Inkonsistenz erreicht man bei dieser Formulierung den absurden Zustand, bei dem kein Wert mehr in Frage kommt. Wir weisen dagegen die neue Information zurück, um die Einhaltung der Konsistenz garantieren zu können. Wird ein Verhalten der γ-Expansionsoperation wie bei den Original-Postulaten gewünscht, läßt sich dies einfach erreichen, indem der Konsistenzgrad γ auf 0 gesetzt wird.

Das dritte Postulat hält das Prinzip der minimalen Änderung für die γ-Expansion fest: Informationen sollen nicht grundlos entfernt werden.

Falls die neue Information $\mathcal{G}$ bereits bekannt war, soll sich durch eine γ-Expansionsoperation nichts ändern. Dieser Fall der „degenerierten“ Expansion wird durch Postulat E_γ4 ausgedrückt.

Das Monotonie-Postulat stellt sicher, daß bei der Expansion einer Teilmenge von $\mathcal{F}$ auch die Ergebnisse in einer Untermengenbeziehung zueinander stehen. Im Gegensatz zu den AGM-Postulaten garantieren wir allerdings nur die Monotonie im Falle der erfolgreichen Expansion von $\mathcal{F}$ mit $\mathcal{G}$, beziehungsweise der nicht-erfolgreichen Expansion von $\mathcal{F}'$ mit $\mathcal{G}$.

Die bisher aufgestellten Postulate verbieten nicht, daß die Formel $\mathcal{F}+_\gamma\mathcal{G}$ zusätzlich zu den Fuzzy-Klauseln aus $\mathcal{G}$ neue Klauseln enthält, die vorher nicht in $\mathcal{F}$ enthalten waren. Dieses, gelegentlich auch als *Besserwisser-Syndrom* bezeichnete Phänomen wollen wir jedoch ausschließen, was mit Hilfe von Postulat E_γ6 geschieht.

10.1.2 Die γ-Expansionsoperation

Der nächste Schritt ist das Aufstellen einer konstruktiven Operation, die alle oben aufgestellten Postulate erfüllt.

Wir betrachten hierzu zunächst den Fall, daß die Expansion erfolgreich ist. Dann folgt aus den Postulaten $E_\gamma 3$ und $E_\gamma 6$:

$$\mathcal{F} \subseteq \mathcal{F} +_\gamma \mathcal{G} \subseteq \mathcal{F} \cup \mathcal{G}$$

Aus dem Postulat $E_\gamma 2a$ folgt weiterhin:

$$\mathcal{G} \subseteq \mathcal{F} +_\gamma \mathcal{G}$$

Wenn wir dies kombinieren, erhalten wir:

$$\mathcal{F} \cup \mathcal{G} \subseteq \mathcal{F} +_\gamma \mathcal{G} \subseteq \mathcal{F} \cup \mathcal{G}$$

Woraus unmittelbar folgt, daß

$$\mathcal{F} +_\gamma \mathcal{G} = \mathcal{F} \cup \mathcal{G} \tag{10.1}$$

Das heißt, die Vereinigung ist die einzig mögliche Operation für eine erfolgreiche γ-Expansion.

Im Falle einer nicht-erfolgreichen Expansion folgt aus Postulat $E_\gamma 2b$:

$$\mathcal{F} +_\gamma \mathcal{G} = \mathcal{F} \tag{10.2}$$

Damit können wir die γ-Expansion folgendermaßen definieren:

Satz 10.1.2 (Expansionssatz) *Seien $\mathcal{F}, \mathcal{G} \in \mathfrak{F}(\Omega)$. Die einzige Operation, die alle γ-Expansionspostulate erfüllt, ist:*

$$\mathcal{F} +_\gamma \mathcal{G} \stackrel{\text{def}}{=} \begin{cases} \mathcal{F} \cup \mathcal{G} & \textbf{wenn } C(\mathcal{F} \cup \mathcal{G}) \geq \gamma, \\ \mathcal{F} & \textbf{sonst} \end{cases}$$

Beweis. *Die obige Definition folgt aus (10.1) und (10.2). Es muß noch gezeigt werden, daß mit dieser γ-Expansionsoperation alle Postulate erfüllt sind. Die Postulate $E_\gamma 2a$, $E_\gamma 2b$, $E_\gamma 3$, $E_\gamma 6$ sind bereits durch die Definition erfüllt. Postulat $E_\gamma 1$ ist trivialerweise erfüllt, da wir in jedem Fall wieder eine Fuzzy-Formel als Ergebnis erhalten. Durch die Mengeneigenschaften der Fuzzy Konjunktiven Normalformen ist auch Postulat $E_\gamma 4$ (Invarianz) erfüllt. Postulat $E_\gamma 5$ läßt sich ebenfalls leicht zeigen: Im Fall der erfolgreichen Expansion $\mathcal{F} +_\gamma \mathcal{G}$ gilt:*

$$\begin{aligned} \mathcal{G} \subseteq \mathcal{F} +_\gamma \mathcal{G} \quad \wedge \quad \mathcal{F}' \subseteq \mathcal{F} \quad &\Rightarrow \quad \mathcal{F}' +_\gamma \mathcal{G} = \mathcal{F}' \cup \mathcal{G} \\ &\subseteq \quad \mathcal{F} +_\gamma \mathcal{G} = \mathcal{F} \cup \mathcal{G} \end{aligned}$$

Falls die Expansion erfolglos ist, gilt:

$$\begin{aligned} \mathcal{G} \not\subseteq \mathcal{F}' +_\gamma \mathcal{G} \quad \wedge \quad \mathcal{F}' \subseteq \mathcal{F} \quad &\Rightarrow \quad \mathcal{F}' +_\gamma \mathcal{G} = \mathcal{F} \\ &\subseteq \quad \mathcal{F} +_\gamma \mathcal{G} = \mathcal{F} \end{aligned}$$

Womit die Behauptung bewiesen ist. ■

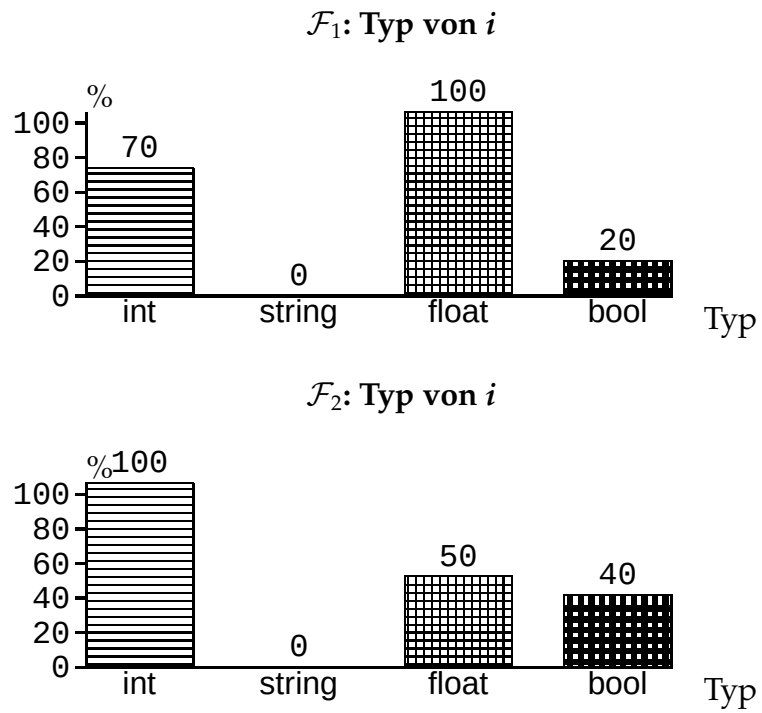


Abbildung 10.1: Beispielformeln für eine γ-Expansion

Somit ist gezeigt, daß es genau eine γ-Expansionsoperation gibt, die die aufgestellten Postulate erfüllt, nämlich die Mengenvereinigung.

Beispiel (γ-Expansion) Wir zeigen die Expansion einer Fuzzy-Formel an dem Beispiel des Reengineering. Hierzu werden wieder die Informationen betrachtet, die über den Typ einer Variablen i gesammelt und mit Hilfe einer Fuzzy-Formel $\mathcal{F}_1$ repräsentiert werden (siehe Abbildung 10.1). Es soll nun eine neue Information, repräsentiert durch die Formel $\mathcal{F}_2$, durch γ-Expansion hinzugefügt werden. Wichtig hierbei ist der geforderte Konsistenzgrad γ, den das Ergebnis nach der Expansion erreichen soll: verlangen wir einen Konsistenzgrad von $\gamma = 0{,}8$, ist die Expansion erfolglos (da $C(\mathcal{F}_1 \cup \mathcal{F}_2) = 0{,}7 < 0{,}8$) und die Ausgangformel bleibt unverändert, also $\mathcal{F}_1 +_{0,8} \mathcal{F}_2 = \mathcal{F}_1$.

Bei einem geforderten Konsistenzgrad von $\gamma = 0{,}6$ dagegen ist die Expansion erfolgreich; das Ergebnis ist in Abbildung 10.2 auf der nächsten Seite zu sehen.

10.1.3 Der γ-Expansionsalgorithmus

In diesem Abschnitt definieren wir den Algorithmus zur Berechnung der γ-Expansion.

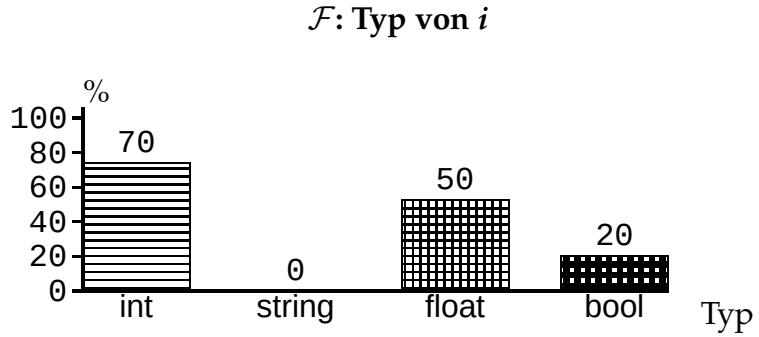


Abbildung 10.2: Ergebnis der γ-Expansion $\mathcal{F}_1 +_{0.6} \mathcal{F}_2$

Wie bei allen nachfolgenden Algorithmen wird eine objektorientierte Umgebung vorausgesetzt; die Beschreibung erfolgt in Form von Pseudocode, der sich an die Programmiersprache *Java* anlehnt.

Der Expansionsalgorithmus wird dabei als Methode einer Formel-Klasse beschrieben. Die Parameter der Methode sind die Fuzzy-Formel $\mathcal{G}$, um die expandiert werden soll, sowie der geforderte Konsistenzgrad γ. Als Ergebnis wird die Formel, gespeichert in der Objektvariable $\mathcal{F}$, gemäß der γ-Expansionsoperation verändert: $\mathcal{F}' = \mathcal{F} +_\gamma \mathcal{G}$ (Algorithmus 10.1.1).

Algorithmus 10.1.1 γ-Expansion

```
1  STATUS FORMEL.γ-EXPANSION( formel G, konsistenzgrad γ )
2  begin
3     F' := this.F.vereinigung(G);
4
6     if F'.konsistenzgrad() ≥ γ
7        then
8              this.F := F';
9              return(ERFOLG);
10       else
11             return(MISSERFOLG);
12    fi
13 end
```

Zur Durchführung der Expansion wird testweise die Mengenvereinigung der aktuellen Formel (referenziert durch *this*.$\mathcal{F}$) mit der neuen Formel berechnet. Erreicht die Vereinigung den geforderten Konsistenzgrad γ, ist dies das Ergebnis einer erfolgreichen Expansion. Andernfalls bleibt die Formel unverändert.

10.2 Revision von Fuzzy-Formeln

Bisher haben wir inkonsistente Informationen zurückgewiesen, um die Integrität unserer Wissensmenge nicht kompromittieren zu müssen. Im Allgemeinen Fall ist dies jedoch nicht möglich, da man sich in den meisten Einsatzgebieten neuen Informationen nicht einfach verschließen kann, nur weil sie nicht in das aufgebaute Weltbild passen. Hier wird es also notwendig, bisher als gültig angenommene Informationen aufzugeben, um wieder einen konsistenten Zustand zu erreichen. Aus diesem Grund ist die Revision eine nicht-monotone Operation. Gleichwohl dürfen nicht beliebig viele vorhandene Informationen entfernt werden, sondern nur gerade soviele, um eine Inkonsistenz zu verhindern; dies gibt schon das in Abschnitt 9.1.2 vorgestellte Prinzip der minimalen Änderung vor.

Nun wird eine solche Revision meist kein eindeutiges Ergebnis haben, da es oft mehrere Möglichkeiten geben wird, einen Widerspruch aufzulösen. Die grundsätzliche Schwierigkeit besteht darin, die Informationsmenge aufzuspüren, die gerade noch konsistent zueinander ist und mit der neuen Information zusammenpaßt.

Die Herausforderung ist dabei, dies konstruktiv und in effizienter Weise zu ermöglichen, denn eine rein theoretische Betrachtung dieses Problems liefert zwar Lösungsmöglichkeiten, die sich aber nicht direkt in eine praktische Lösung umsetzen lassen. Der hier verfolgte Lösungsansatz ist, wie bei der Expansion wünschenswerte Eigenschaften der Revision mit Hilfe von Postulaten festzuhalten. Danach untersuchen wir, welche Operatoren diese Postulate erfüllen.

Analog zur Expansion legen wir zuerst die Syntax für die γ-Revision fest:

Definition 10.2.1 (γ-Revision) *Eine γ-Revision $\oplus_\gamma$ ist eine Abbildung, die zwei Fuzzy-Formeln $\mathcal{F}, \mathcal{G} \in \mathfrak{F}(\Omega)$ und einem Wert $\gamma \in [0,1]$ eine weitere Fuzzy-Formel aus $\mathfrak{F}(\Omega)$ zuordnet:*

$$\oplus_\gamma : \mathfrak{F}(\Omega) \times \mathfrak{F}(\Omega) \to \mathfrak{F}(\Omega)$$

und die mindestens den Postulaten $R_\gamma 1$–$R_\gamma 6$ (Seite 109) genügt.

Enthält die Formel $\mathcal{G}$ nur eine Klausel $\mathcal{K}$, erlauben wir statt $\mathcal{F} \oplus_\gamma \{\mathcal{K}\}$ auch die abkürzende Schreibweise $\mathcal{F} \oplus_\gamma \mathcal{K}$. Auch hier schreiben wir kurz Revision, wenn aus dem Zusammenhang klar ist, daß die γ-Revision gemeint ist.

Im Gegensatz zur γ-Expansion werden bei einer γ-Revision die Klauseln der neuen Formel immer hinzugefügt, es sei denn, ihr Konsistenzgrad ist bereits kleiner als das geforderte γ: In diesem Fall könnte die Ergebnisformel, unabhängig davon wie sie revidiert wird, nie die neuen Information beinhalten und gleichzeitig den geforderten Konsistenzgrad erreichen. Die Ausgangsformel soll dann, wie bei der γ-Expansion, unverändert bleiben.

10.2.1 Postulate der γ-Revision

In diesem Abschnitt werden die Postulate für die Revision von Fuzzy-Formeln mit einem vorgegebenen Konsistenzgrad γ definiert. Hierzu nehmen wir ebenfalls die AGM-Postulate (siehe Anhang A.3 auf Seite 308) als Vorlage. Während es jedoch bei der γ-Expansion möglich war, die meisten Postulate analog zum aussagenlogischen Fall zu definieren, müssen bei der γ-Revision einige Änderungen und Erweiterungen durchgeführt werden, die wir im Anschluß an die Postulate motivieren. Grundsätzlich orientieren wir uns bei solchen Änderungen wiederum an der Intention, die hinter den Original-Postulaten steht.

Wie bei der γ-Expansion wird auch bei der γ-Revision keine Aussage über den Konsistenzgrad einer Fuzzy-Formel *vor* einer γ-Revisionsoperation gemacht. Wenn der geforderte Konsistenzgrad nicht erreicht werden kann, bleibt die Formel unverändert.

Um den Vergleich mit den Original-Postulaten leichter zu machen, wurde wieder die ursprüngliche Numerierung beibehalten. Postulat $R_\gamma 2$, das aus Gründen der Übersichtlichkeit in zwei Teilpostulate aufgesplittet wurde, ist daher mit Buchstaben indiziert.

Für alle nachfolgenden Postulate gilt: $\mathcal{F}, \mathcal{G}, \mathcal{H}$ sind Fuzzy-Formeln aus $\mathfrak{F}(\Omega)$.

$R_\gamma 1$ (Stabilität) Nach einer Revisionsoperation soll wieder eine Fuzzy-Formel vorhanden sein:

$$\mathcal{F} \oplus_\gamma \mathcal{G} \in \mathfrak{F}(\Omega)$$

$R_\gamma 2a$ (Priorität) Wenn die Fuzzy-Formel $\mathcal{G}$ mindestens den Konsistenzgrad γ erreicht, soll sie nach der Revisionsoperation in der Ergebnisformel enthalten sein:

$$\textbf{Wenn } C(\mathcal{G}) \geq \gamma, \textbf{ dann } \mathcal{G} \subseteq \mathcal{F} \oplus_\gamma \mathcal{G}$$

$R_\gamma 2b$ (Mißerfolg) Wenn der Konsistenzgrad der Fuzzy-Formel $\mathcal{G}$ kleiner als γ ist, bleibt die Ausgangsformel unverändert:

$$\textbf{Wenn } C(\mathcal{G}) < \gamma, \textbf{ dann } \mathcal{F} \oplus_\gamma \mathcal{G} = \mathcal{F}$$

$R_\gamma 3$ (Vorhersehbarkeit) Das Ergebnis der Revision soll nur Fuzzy-Klauseln enthalten, die aus $\mathcal{F}$ oder $\mathcal{G}$ stammen:

$$\mathcal{F} \oplus_\gamma \mathcal{G} \subseteq \mathcal{F} \cup \mathcal{G}$$

$R_\gamma 4$ (Kompatibilität) Falls die Fuzzy-Klauseln der Fuzzy-Formel $\mathcal{G}$ mit den Klauseln aus $\mathcal{F}$ verträglich sind, ist die Expansion in der Revision enthalten:

$$\textbf{Wenn } \mathcal{G} \subseteq \mathcal{F} +_\gamma \mathcal{G} \textbf{ und } C(\mathcal{F} +_\gamma \mathcal{G}) \geq \gamma, \textbf{ dann } \mathcal{F} \oplus_\gamma \mathcal{G} \supseteq \mathcal{F} +_\gamma \mathcal{G}$$

$R_\gamma 5$ (Konsistenz) Der Konsistenzgrad der Ergebnisformel muß im Falle der erfolgreichen γ-Revision mindestens γ erreichen:

$$\textbf{Wenn } C(\mathcal{G}) \geq \gamma, \textbf{ dann } C(\mathcal{F} \oplus_\gamma \mathcal{G}) \geq \gamma$$

$R_\gamma 6$ (Identität) Bei einer Revision mit verschiedenen, aber semantisch äquivalenten Fuzzy-Formeln ist auch das Ergebnis semantisch äquivalent:

$$\textbf{Wenn } \mu_\mathcal{G} = \mu_\mathcal{H}, \textbf{ dann } \mu_{\mathcal{F} \oplus_\gamma \mathcal{G}} = \mu_{\mathcal{F} \oplus_\gamma \mathcal{H}}$$

Die Postulate $R_\gamma 1$–$R_\gamma 6$ sind die sogenannten *Basis-Postulate*, die jeder Revisionsoperator erfüllen muß. Darüber hinaus gibt es zwei Zusatz-Postulate, die das Verhalten bei iterativen Änderungen kontrollieren:

$R_\gamma 7$ Wenn zwei Formeln $\mathcal{G}$ und $\mathcal{H}$ erfolgreich nacheinander durch γ-Revision und γ-Expansion zu einer Formel $\mathcal{F}$ hinzugefügt werden können, ist das Ergebnis eine Obermenge der Revision von $\mathcal{F}$ mit $\mathcal{G} \cup \mathcal{H}$:

$$\textbf{Wenn } \mathcal{G} \cup \mathcal{H} \subseteq (\mathcal{F} \oplus_\gamma \mathcal{G}) +_\gamma \mathcal{H}, \textbf{ dann } \mathcal{F} \oplus_\gamma (\mathcal{G} \cup \mathcal{H}) \subseteq (\mathcal{F} \oplus_\gamma \mathcal{G}) +_\gamma \mathcal{H}$$

$R_\gamma 8$ Erweiterung von Postulat $R_\gamma 4$:

$$\textbf{Wenn } \mathcal{G} \cup \mathcal{H} \subseteq (\mathcal{F} \oplus_\gamma \mathcal{G}) +_\gamma \mathcal{H}, \textbf{ dann } (\mathcal{F} \oplus_\gamma \mathcal{G}) +_\gamma \mathcal{H} \subseteq \mathcal{F} \oplus_\gamma (\mathcal{G} \cup \mathcal{H})$$

Das erste Postulat hält wie bei der γ-Expansion fest, daß das Ergebnis der γ-Revision einer Fuzzy-Formel wieder eine Fuzzy-Formel sein soll.

Die Postulate $R_\gamma 2a$ und $R_\gamma 2b$ regeln den Fall der erfolgreichen beziehungsweise erfolglosen Revision. Wie oben erläutert, kann eine Revision nur dann erfolgreich sein, wenn die neuen Informationen mindestens den Konsistenzgrad γ erreichen. Diese beiden Postulate weichen dabei von dem entsprechenden AGM-Postulat ab, da wir nicht in jedem Fall dem neuen Wissen Vorrang geben wollen; insbesondere, wenn dies zu einem zu niedrigen Konsistenzgrad führen würde.

Das dritte Postulat gibt uns, ähnlich wie bei der γ-Expansion das Postulat $E_\gamma 6$, eine obere Grenze für das Ergebnis der γ-Revision.

Eine Revision wird üblicherweise dann durchgeführt, wenn es einen Konflikt zwischen den Informationen der neuen und denen der alten Fuzzy-Formel gibt. Es ist jedoch wünschenswert, die Revision auch für den Fall zu definieren, daß es keinen solchen Konflikt gibt, die Revision also zu einer Expansion wird. Dies wird durch Postulat $R_\gamma 4$ ermöglicht. Bei diesem Postulat ist die weitere Voraussetzung wichtig, daß der Konsistenzgrad der expandierten Formel mindestens γ erreichen muß, da es möglich ist, daß $\mathcal{G}$ schon vor dieser Revision in $\mathcal{F}$ enthalten war, der Konsistenzgrad von $\mathcal{F}$ alleine aber kleiner als γ ist.

Postulat $R_\gamma 5$ stellt sicher, daß das Ergebnis einer erfolgreichen Revision mindestens den Konsistenzgrad γ erreicht.

Bei der Durchführung der Revision kommt es vor allem auf die Interpretation der beteiligten Formeln als Fuzzy-Mengen an. Insbesondere soll die Revision mit syntaktisch unterschiedlichen, aber semantisch äquivalenten Fuzzy-Formeln zu einem äquivalenten Ergebnis führen, was durch das Postulat $R_\gamma 6$ gesichert wird.

Die beiden Zusatz-Postulate $R_\gamma 7$ und $R_\gamma 8$ schließlich kontrollieren das Verhalten der γ-Revision bei iterativen Änderungen. Sie fordern nämlich, daß $\mathcal{F} \oplus_\gamma \mathcal{G} = (\mathcal{F} \oplus_\gamma \mathcal{G}_1) \cup \mathcal{G}_2$ gilt für jede Zerlegung von $\mathcal{G}$ in zwei Teilformeln $\mathcal{G}_1$ und $\mathcal{G}_2$ mit $\mathcal{G} = \mathcal{G}_1 \cup \mathcal{G}_2$, für die $C(\mathcal{G}_1) \geq \gamma$ und $C((\mathcal{F} \oplus_\gamma \mathcal{G}_1) \cup \mathcal{G}_2) \geq \gamma$ ist.

10.2.1.1 Eigenschaften der γ-Revision

Um sich zu veranschaulichen, wie die Postulate die Menge der möglichen Operatoren einschränken, kann man die triviale Revisionsoperation betrachten: diese löscht bei jeder Revision sämtliches altes Wissen und nimmt nur die neuen Klauseln in die Formel auf:

$$\mathcal{F} \oplus_\gamma \mathcal{G} \stackrel{\text{def}}{=} \mathcal{G}.$$

Man kann jedoch schnell sehen, daß mit dieser Operation das Postulat $R_\gamma 4$ im Allgemeinen nicht erfüllt ist. Mithin stellt sie also keine wünschenswerte γ-Revisionsoperation dar. Man beachte jedoch, daß eine leichte Abwandlung dieser Operation alle Postulate erfüllt: Prüfe zuerst, ob sich die neue Fuzzy-Formel $\mathcal{G}$ durch einfache γ-Expansion hinzufügen läßt, also $\mathcal{G} \subseteq \mathcal{F} +_\gamma \mathcal{G}$ gilt. Wenn ja, ist dies das Ergebnis der γ-Revision, wenn nein, setze $\mathcal{F} \oplus_\gamma \mathcal{G} := \mathcal{G}$, lösche also alle existierenden Klauseln und füge nur die neuen Klauseln in die Ergebnisformel ein. Da diese Operation offensichtlich keine maximale Ergebnismenge garantieren kann, werden wir für praktisch verwendete Operatoren zusätzlich noch eine Maximalitätsforderung erheben. Zuvor betrachten wir im nächsten Abschnitt jedoch, welche Revisionsoperatoren grundsätzlich in Frage kommen.

Bei der Definition der γ-Revision wird absichtlich keine Annahme über den Konsistenzgrad einer Formel *vor* der Operation gemacht. Dies ist sinnvoll, da es dadurch möglich ist, den Konsistenzgrad für eine γ-Revision höher als den der Ausgangsformel anzusetzen. Im Erfolgsfall wird die Revisionsoperation den Konsistenzgrad auf den neuen Wert erhöhen. So ist es zum Beispiel in einer Entwurfsanwendung sinnvoll, den Konsistenzgrad zu Beginn, wenn noch viele Unklarheiten herrschen, auf einen niedrigen Wert zu setzen und im weiteren Verlauf schrittweise zu erhöhen. Würde man bei der Definition der γ-Revision einen bestimmten Konsistenzgrad voraussetzen, wäre eine solche Vorgehensweise nicht möglich.

Aus den aufgestellten Revisions-Postulaten lassen sich, zusammen mit den Postulaten der γ-Expansion, noch einige nützliche Eigenschaften der γ-Revision ableiten.

Die erste betrifft den Fall, daß kein Konflikt zwischen den beteiligten Formeln existiert. Die Revision entspricht dann einer Expansion:

Lemma 10.2.2 (Revision durch Expansion) *Seien $\mathcal{F}, \mathcal{G} \in \mathfrak{F}(\Omega)$. Wenn die Expansion $\mathcal{F} +_\gamma \mathcal{G}$ erfolgreich ist, entspricht die γ-Revision der γ-Expansion:*

$$\textbf{Wenn } \mathcal{G} \subseteq \mathcal{F} +_\gamma \mathcal{G} \textbf{ und } C(\mathcal{F} +_\gamma \mathcal{G}) \geq \gamma, \textbf{ dann } \mathcal{F} \oplus_\gamma \mathcal{G} = \mathcal{F} +_\gamma \mathcal{G} = \mathcal{F} \cup \mathcal{G}.$$

Beweis. *Es sei vorausgesetzt, daß $\mathcal{G} \subseteq \mathcal{F} +_\gamma \mathcal{G}$ und $C(\mathcal{F} +_\gamma \mathcal{G}) \geq \gamma$. Dann folgt aus dem Expansionssatz 10.1.2 (Seite 105), daß*

$$\mathcal{F} +_\gamma \mathcal{G} = \mathcal{F} \cup \mathcal{G}.$$

Daraus folgt mit Postulat $R_\gamma 3$ und $R_\gamma 4$:

$$\mathcal{F} \oplus_\gamma \mathcal{G} \subseteq \mathcal{F} \cup \mathcal{G} \subseteq \mathcal{F} \oplus_\gamma \mathcal{G},$$

woraus unmittelbar die Behauptung folgt. ■

Der aufmerksame Leser wird sich gefragt haben, warum für die Revision nicht wie bei der Expansion die Invarianz für den Fall gefordert wird, daß die neue Information schon bekannt ist. Tatsächlich läßt sich die Invarianzeigenschaft aus den existierenden Postulaten ableiten, wie das nachfolgende Lemma zeigt:

Lemma 10.2.3 (Invarianz) *Seien $\mathcal{F}, \mathcal{G} \in \mathfrak{F}(\Omega)$. Falls die neue Information schon bekannt ist ($\mathcal{G} \subseteq \mathcal{F}$) und die Fuzzy-Formel $\mathcal{F}$ den Konsistenzanforderungen genügt ($C(\mathcal{F}) \geq \gamma$), bleibt $\mathcal{F}$ unverändert:*

$$\begin{aligned} & (\mathcal{G} \subseteq \mathcal{F}) \wedge (C(\mathcal{F}) \geq \gamma) \\ \Rightarrow \quad & \mathcal{F} \oplus_\gamma \mathcal{G} = \mathcal{F}. \end{aligned}$$

Beweis. *Es sei $\mathcal{G} \subseteq \mathcal{F}$ und $C(\mathcal{F}) \geq \gamma$. Dann folgt mit dem Postulat $E_\gamma 3$ der Expansion:*

$$\mathcal{G} \subseteq \mathcal{F} \Rightarrow \mathcal{G} \subseteq \mathcal{F} +_\gamma \mathcal{G}.$$

Daraus folgt mit Hilfe des Lemmas 10.2.2:

$$\mathcal{F} \oplus_\gamma \mathcal{G} = \mathcal{F} +_\gamma \mathcal{G} = \mathcal{F} \cup \mathcal{G} = \mathcal{F},$$

womit das Lemma bewiesen ist. ■

Dies entspricht auch dem Prinzip der minimalen Änderung: Falls die neue Information schon bekannt ist und den Konsistenzanforderungen genügt, ist die minimal benötigte Änderung überhaupt keine Änderung.

Was passiert aber, wenn die neue Information zwar bekannt ist, aber die vorhandene Fuzzy-Formel den geforderten Konsistenzgrad nicht erreicht? In diesem Fall wird

die γ-Revisionsoperation den geforderten Konsistenzgrad herstellen, indem sie in Konflikt stehende Klauseln entfernt.

Wenn eine γ-Revisionsoperation beide Zusatz-Postulate erfüllt, sind die beiden Mengen $(\mathcal{F}\oplus_\gamma\mathcal{G})+_\gamma\mathcal{H}$ und $\mathcal{F}\oplus_\gamma(\mathcal{G}\cup\mathcal{H})$ tatsächlich gleich, wie das folgende Lemma zeigt.

Lemma 10.2.4 *Seien $\mathcal{F},\mathcal{G},\mathcal{H}\in\mathfrak{F}(\Omega)$. Aus $R_\gamma 7$ und $R_\gamma 8$ folgt:*

$$\textbf{Wenn } \mathcal{G}\subseteq\mathcal{F}\oplus_\gamma\mathcal{G} \textbf{ und } \mathcal{H}\subseteq(\mathcal{F}\oplus_\gamma\mathcal{G})+_\gamma\mathcal{H}, \textbf{ dann } (\mathcal{F}\oplus_\gamma\mathcal{G})+_\gamma\mathcal{H}=\mathcal{F}\oplus_\gamma(\mathcal{G}\cup\mathcal{H})$$

Beweis. *Wir beginnen mit der Fuzzy-Formel $\mathcal{F}$. Voraussetzung sei, daß sowohl die γ-Revision um $\mathcal{G}$ als auch die γ-Expansion um $\mathcal{H}$ erfolgreich ist. Dann folgt aus $R_\gamma 7$ und $R_\gamma 8$:*

$$\mathcal{F}\oplus_\gamma(\mathcal{G}\cup\mathcal{H})\subseteq(\mathcal{F}\oplus_\gamma\mathcal{G})+_\gamma\mathcal{H}\subseteq\mathcal{F}\oplus_\gamma(\mathcal{G}\cup\mathcal{H}),$$

woraus unmittelbar die Behauptung folgt. ■

Für den im nächsten Abschnitt gezeigten Revisionssatz benötigen wir außerdem noch zwei weitere Hilfssätze.

Lemma 10.2.5 *Seien $\mathcal{F}\in\mathfrak{F}(\Omega)$ und $\mu\in F(\Omega)-\{\mu_\perp\}$ beliebig. Dann existiert eine Formel $\mathcal{G}\in\mathfrak{F}(\Omega)$ mit $\mathcal{F}\cap\mathcal{G}=\emptyset$ und $\mu_\mathcal{G}=\mu$.*

Beweis. *Es ist leicht zu sehen, daß es unendlich viele Klauseln $\mathcal{K}\in\mathfrak{K}(\Omega)$ mit $\mu_\mathcal{K}=\mu$ gibt. Nur endlich viele dieser Klauseln können in $\mathcal{F}$ enthalten sein. Wähle also eine Klausel $\mathcal{K}\in\mathfrak{K}(\Omega)-\mathcal{F}$ mit $\mu_\mathcal{K}=\mu$ und setze $\mathcal{G}:=\{\mathcal{K}\}$.* ■

Lemma 10.2.6 *Seien $\mathcal{F},\mathcal{G}_1,\mathcal{G}_2\in\mathfrak{F}(\Omega)$ und $\gamma\in[0,1]$ beliebig. Falls $\mu_{\mathcal{G}_1}=\mu_{\mathcal{G}_2}$ und $C(\mathcal{G}_1)=C(\mathcal{G}_2)\geq\gamma$, dann*

$$(\mathcal{F}\oplus_\gamma\mathcal{G}_1)\cup\mathcal{G}_2=(\mathcal{F}\oplus_\gamma\mathcal{G}_2)\cup\mathcal{G}_1$$

Insbesondere gilt, falls zusätzlich $\mathcal{F},\mathcal{G}_1,\mathcal{G}_2$ paarweise disjunkt sind,

$$(\mathcal{F}\oplus_\gamma\mathcal{G}_1)-\mathcal{G}_1=(\mathcal{F}\oplus_\gamma\mathcal{G}_2)-\mathcal{G}_2$$

Beweis. *Da $\mu_{\mathcal{G}_1}=\mu_{\mathcal{G}_2}$ gilt $C((\mathcal{F}\oplus_\gamma\mathcal{G}_1)\cup\mathcal{G}_2)=C((\mathcal{F}\oplus_\gamma\mathcal{G}_1)\cup\mathcal{G}_1)=C(\mathcal{F}\oplus_\gamma\mathcal{G}_1)\geq\gamma$ und analog $C((\mathcal{F}\oplus_\gamma\mathcal{G}_2)\cup\mathcal{G}_1)\geq\gamma$. Aus $R_\gamma 7$ und $R_\gamma 8$ folgt dann*

$$(\mathcal{F}\oplus_\gamma\mathcal{G}_1)\cup\mathcal{G}_2=\mathcal{F}\oplus_\gamma(\mathcal{G}_1\cup\mathcal{G}_2)=(\mathcal{F}\oplus_\gamma\mathcal{G}_2)\cup\mathcal{G}_1$$

und damit die Behauptung. ■

10.2.2 Die γ-Revisionsoperation

Nachdem festgelegt ist, welche Eigenschaften die γ-Revision erfüllen soll, muß ein geeigneter Operator gefunden werden. Dieser ist jedoch im Gegensatz zur γ-Expansion nicht eindeutig festgelegt. Bei der Bestimmung des für die Praxis verwendeten Operators gehen wir daher im folgenden schrittweise vor: Zunächst veranschaulichen wir die Eigenschaften möglicher Revisionsoperatoren, bevor sie im *Revisionssatz* formalisiert werden. Mit diesem Satz sind die Eigenschaften gültiger Operatoren festgelegt, er gibt aber noch keine effiziente, in der Praxis einsetzbare Operation an. Eine solche wird anschließend basierend auf der Idee einer *Relevanzordnung* von Fuzzy-Klauseln entwickelt und algorithmisiert.

Um sich den Ergebnisraum möglicher Operatoren zu veranschaulichen, kann man eine nicht-konstruktive γ-Revisionsoperation betrachten: Gegeben seien zwei Fuzzy-Formeln $\mathcal{F}, \mathcal{G} \in \mathfrak{F}(\Omega)$. Wir bestimmen nun die Menge aller möglichen Ergebnisse $\mathfrak{R}_{\mathcal{F}\oplus_\gamma\mathcal{G}}$, die die Formel $\mathcal{G}$ enthalten und deren Konsistenzgrad mindestens den Wert γ erreicht. Ist der Konsistenzgrad der Formel $\mathcal{G}$ kleiner als γ, existiert keine solche Formel, die Ausgangsformel $\mathcal{F}$ bleibt dann per Definition unverändert:

Definition 10.2.7 (Ergebnisse der γ-Revision) *Seien $\mathcal{F}, \mathcal{G} \in \mathfrak{F}(\Omega)$. Die Menge aller möglichen Ergebnisse $\mathfrak{R}_{\mathcal{F}\oplus_\gamma\mathcal{G}}$ der γ-Revision $\mathcal{F} \oplus_\gamma \mathcal{G}$ ist definiert durch:*

$$\mathfrak{R}_{\mathcal{F}\oplus_\gamma\mathcal{G}} \stackrel{\text{def}}{=} \begin{cases} \{\mathcal{F} +_\gamma \mathcal{G}\} & \textbf{wenn } C(\mathcal{F}\cup\mathcal{G}) \geq \gamma \\ \{\mathcal{F}\} & \textbf{wenn } C(\mathcal{G}) < \gamma \\ \{\mathcal{F}' | \mathcal{G} \subseteq \mathcal{F}' \subseteq \mathcal{F}\cup\mathcal{G} \wedge C(\mathcal{F}') \geq \gamma\} & \textbf{sonst}. \end{cases}$$

Man kann leicht sehen, daß alle Elemente dieser Menge die Basis-Postulate $R_\gamma 1$–$R_\gamma 6$ erfüllen.

Als nicht-konstruktive Operation gibt uns diese Definition aber leider keinen Anhaltspunkt dafür, wie eine konkrete Ergebnismenge berechnet werden könnte. Der typischerweise in der Theorie der Wissensrevision verfolgte Ansatz ist, aus dieser Menge mit Hilfe einer Selektionsfunktion ein geeignetes Resultat auszuwählen. Eine solche Vorgehensweise hätte in der Praxis jedoch katastrophale Laufzeiteigenschaften.

Es stellt sich nun die Frage, ob es nicht möglich ist, weitere Aussagen über mögliche Revisionsoperatoren zu machen. Wir wissen bereits, daß das Ergebnis eines Operators, der alle Postulate erfüllt, ein Element aus der Potenzmenge von $\mathcal{F}\cup\mathcal{G}$ sein muß. Etwas anders ausgedrückt, muß ein Ergebnis im Falle einer erfolgreichen Revision $\mathcal{F} \oplus_\gamma \mathcal{G}$ die neue Formel $\mathcal{G}$ und ein zu ihr γ-kompatibles Element aus der Potenzmenge von $\mathcal{F}$ enthalten. Wenn wir dazu eine Ordnung auf den Elementen der Potenzmenge herstellen, läßt sich ein gültiger Operator konstruieren, indem man genau das erste Element aus der Ordnung nimmt, das mit $\mathcal{G}$ kompatibel ist und es mit diesem vereinigt. Interessanterweise gilt auch der umgekehrte Schluß: zu jedem gültigen Revisionsoperator gibt es eine äquivalente Abbildung, die auf einer Ordnung der Potenzmenge beruht. Beides zeigen wir im folgenden Revisionssatz:

Satz 10.2.8 (Revisionssatz) *Seien $\mathcal{F} \in \mathfrak{F}(\Omega)$ und $\gamma \in [0,1]$ beliebig, aber fest gewählt. Dann gilt: eine Abbildung $\mathcal{F}^{\oplus_\gamma} : \mathfrak{F}(\Omega) \to \mathfrak{F}(\Omega)$, $\mathcal{G} \mapsto \mathcal{F} \oplus_\gamma \mathcal{G}$ genügt genau dann den Postulaten $R_\gamma 1$–$R_\gamma 8$, wenn es eine Aufzählung $2^{\mathcal{F}} = \{\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_{2^{|\mathcal{F}|}-1}\}$ der Potenzmenge von $\mathcal{F}$ gibt mit*

$$\mathcal{F} \oplus_\gamma \mathcal{G} = \begin{cases} \mathcal{F} & \textbf{wenn } C(\mathcal{G}) < \gamma, \\ \mathcal{F} \cup \mathcal{G} & \textbf{wenn } C(\mathcal{F} \cup \mathcal{G}) \geq \gamma, \\ \mathcal{F}_i \cup \mathcal{G} & \textbf{sonst, wobei } i = \min\{j | C(\mathcal{G} \cup \mathcal{F}_j) \geq \gamma\}. \end{cases}$$

Beweis. *Wir zeigen zuerst die Richtung „⇐“: Gegeben sei die oben definierte Aufzählung. Wir überprüfen die Erfüllung der einzelnen Postulate:*

$R_\gamma 1$ *($\mathcal{F} \oplus_\gamma \mathcal{G} \in \mathfrak{F}(\Omega)$) Erfüllt. In allen drei Fällen ist das Ergebnis eine Fuzzy-Formel aus $\mathfrak{F}(\Omega)$.*

$R_\gamma 2a$ *(**Wenn** $C(\mathcal{G}) \geq \gamma$, **dann** $\mathcal{G} \subseteq \mathcal{F} \oplus_\gamma \mathcal{G}$) Erfüllt. Wenn $C(\mathcal{G}) \geq \gamma$, ist $\mathcal{F} \oplus_\gamma \mathcal{G}$ entweder $\mathcal{F} \cup \mathcal{G}$ oder $\mathcal{F}_i \cup \mathcal{G}$; in beiden Fällen ist $\mathcal{G}$ im Ergebnis enthalten.*

$R_\gamma 2b$ *(**Wenn** $C(\mathcal{G}) < \gamma$, **dann** $\mathcal{F} \oplus_\gamma \mathcal{G} = \mathcal{F}$) Erfüllt. In diesem Fall gilt $\mathcal{F} \oplus_\gamma \mathcal{G} = \mathcal{F}$.*

$R_\gamma 3$ *($\mathcal{F} \oplus_\gamma \mathcal{G} \subseteq \mathcal{F} \cup \mathcal{G}$) Erfüllt. In allen drei Fällen ist das Ergebnis eine Untermenge von $\mathcal{F} \cup \mathcal{G}$.*

$R_\gamma 4$ *(**Wenn** $\mathcal{G} \subseteq \mathcal{F} +_\gamma \mathcal{G}$ **und** $C(\mathcal{F} +_\gamma \mathcal{G}) \geq \gamma$, **dann** $\mathcal{F} \oplus_\gamma \mathcal{G} \supseteq \mathcal{F} +_\gamma \mathcal{G}$) Erfüllt. In diesem Fall gilt $C(\mathcal{F} \cup \mathcal{G}) \geq \gamma$ und somit $\mathcal{F} \oplus_\gamma \mathcal{G} = \mathcal{F} \cup \mathcal{G} = \mathcal{F} +_\gamma \mathcal{G}$.*

$R_\gamma 5$ *(**Wenn** $C(\mathcal{G}) \geq \gamma$, **dann** $C(\mathcal{F} \oplus_\gamma \mathcal{G}) \geq \gamma$) Erfüllt. Wenn $C(\mathcal{G}) \geq \gamma$, ist $\mathcal{F} \oplus_\gamma \mathcal{G}$ entweder $\mathcal{F} \cup \mathcal{G}$ und somit $C(\mathcal{F} \oplus_\gamma \mathcal{G}) \geq \gamma$, oder $\mathcal{G} \cup \mathcal{F}_i$ mit einem $\mathcal{F}_i$, für das $C(\mathcal{G} \cup \mathcal{F}_i) \geq \gamma$ gilt. Die Menge $\{j | C(\mathcal{G} \cup \mathcal{F}_j)\}$ ist überdies nicht leer, da sie mindestens dasjenige j mit $\mathcal{F}_j = \emptyset$ enthält; somit ist $\mathcal{F}_i$ wohldefiniert.*

$R_\gamma 6$ *(**Wenn** $\mu_\mathcal{G} = \mu_\mathcal{H}$, **dann** $\mu_{\mathcal{F} \oplus_\gamma \mathcal{G}} = \mu_{\mathcal{F} \oplus_\gamma \mathcal{H}}$) Erfüllt. Wenn $\mu_\mathcal{G} = \mu_\mathcal{H}$, gilt insbesondere auch $C(\mathcal{G}) = C(\mathcal{H})$. Da nur der Konsistenzgrad darüber entscheidet, wie die γ-Revision durchgeführt wird, ist auch das Ergebnis semantisch äquivalent.*

$R_\gamma 7$ und $R_\gamma 8$ *Anstelle die beiden Postulate einzeln zu beweisen, zeigen wir die strengere Gleichung $\mathcal{G} \cup \mathcal{H} \subseteq (\mathcal{F} \oplus_\gamma \mathcal{G}) +_\gamma \mathcal{H} \Rightarrow \mathcal{F} \oplus_\gamma (\mathcal{G} \cup \mathcal{H}) = (\mathcal{F} \oplus_\gamma \mathcal{G}) +_\gamma \mathcal{H}$, aus deren Gültigkeit die Erfüllung der beiden Postulate direkt folgt:*

1. Fall: *$C(\mathcal{G} \cup \mathcal{H}) < \gamma$. Dann gilt:*

$$\begin{aligned} & C((\mathcal{F} \oplus_\gamma \mathcal{G}) +_\gamma \mathcal{H}) \leq C(\mathcal{G} \cup \mathcal{H}) < \gamma \\ \overset{\text{Satz 10.1.2}}{\Rightarrow} \quad & (\mathcal{F} \oplus_\gamma \mathcal{G}) +_\gamma \mathcal{H} = \mathcal{F} \oplus_\gamma \mathcal{G} \quad \wedge \quad C(\mathcal{F} \oplus_\gamma \mathcal{G}) < \gamma \\ \overset{R_\gamma 5}{\Rightarrow} \quad & C(\mathcal{G}) < \gamma \\ \overset{R_\gamma 2b}{\Rightarrow} \quad & (\mathcal{F} \oplus_\gamma \mathcal{G}) +_\gamma \mathcal{H} = \mathcal{F} \oplus_\gamma \mathcal{G} = \mathcal{F} \end{aligned}$$

Andererseits gilt $\mathcal{F} \oplus_\gamma (\mathcal{G} \cup \mathcal{H}) = \mathcal{F}$, also ist die Bedingung erfüllt.

2. Fall: *$C(\mathcal{G} \cup \mathcal{H}) \geq \gamma \wedge C(\mathcal{F} \cup \mathcal{G}) \geq \gamma$. Dann gilt auch $C(\mathcal{G}) \geq \gamma$ und weiter*

$$\begin{aligned} & \mathcal{G} \cup \mathcal{H} \subseteq (\mathcal{F} \oplus_\gamma \mathcal{G}) +_\gamma \mathcal{H} \\ \Rightarrow\ & \mathcal{G} \cup \mathcal{H} \subseteq (\mathcal{F} \cup \mathcal{G}) +_\gamma \mathcal{H} \\ \Rightarrow\ & C(\mathcal{F} \cup \mathcal{G} \cup \mathcal{H}) \geq \gamma \end{aligned}$$

wobei letzteres gelten muß, da die Annahme $C(\mathcal{F} \cup \mathcal{G} \cup \mathcal{H}) < \gamma$ zu einem Widerspruch führt:

$$\begin{aligned} & \mathcal{G} \cup \mathcal{H} \subseteq (\mathcal{F} \cup \mathcal{G}) +_\gamma \mathcal{H} = \mathcal{F} \cup \mathcal{G} \\ \Rightarrow\ & \mathcal{F} \cup \mathcal{G} \cup \mathcal{H} \subseteq \mathcal{F} \cup \mathcal{G} \\ \Rightarrow\ & C(\mathcal{F} \cup \mathcal{G} \cup \mathcal{H}) \geq C(\mathcal{F} \cup \mathcal{G}) \geq \gamma \end{aligned}$$

Damit ist also $(\mathcal{F} \oplus_\gamma \mathcal{G}) +_\gamma \mathcal{H} = \mathcal{F} \cup \mathcal{G} \cup \mathcal{H}$.
Andererseits gilt wegen $C(\mathcal{F} \cup \mathcal{G} \cup \mathcal{H}) \geq \gamma$ auch $\mathcal{F} \oplus_\gamma (\mathcal{G} \cup \mathcal{H}) = \mathcal{F} \cup \mathcal{G} \cup \mathcal{H}$, also ist die Bedingung hier ebenfalls erfüllt.

3. Fall: *$C(\mathcal{G} \cup \mathcal{H}) \geq \gamma \wedge C(\mathcal{F} \cup \mathcal{G}) < \gamma$. Dann gilt:*

$$\begin{aligned} & \mathcal{F} \oplus_\gamma \mathcal{G} = \mathcal{F}_i \cup \mathcal{G} && \text{mit } i = \min\{j | C(\mathcal{G} \cup \mathcal{F}_j) \geq \gamma\} \\ \wedge\ & \mathcal{F} \oplus_\gamma (\mathcal{G} \cup \mathcal{H}) = \mathcal{F}_k \cup (\mathcal{G} \cup \mathcal{H}) && \text{mit } k = \min\{j | C(\mathcal{G} \cup \mathcal{H} \cup \mathcal{F}_j) \geq \gamma\} \end{aligned}$$

Ferner gilt nach Voraussetzung $\mathcal{H} \subseteq (\mathcal{F} \oplus_\gamma \mathcal{G}) +_\gamma \mathcal{H}$ sowie nach $E_\gamma 3$

$$\begin{aligned} & (\mathcal{F} \oplus_\gamma \mathcal{G}) && \subseteq (\mathcal{F} \oplus_\gamma \mathcal{G}) +_\gamma \mathcal{H} \\ \Rightarrow\ & (\mathcal{F} \oplus_\gamma \mathcal{G}) \cup \mathcal{H} && \subseteq (\mathcal{F} \oplus_\gamma \mathcal{G}) +_\gamma \mathcal{H} \\ \overset{E_\gamma 6}{\Rightarrow}\ & (\mathcal{F} \oplus_\gamma \mathcal{G}) +_\gamma \mathcal{H} && = (\mathcal{F} \oplus_\gamma \mathcal{G}) \cup \mathcal{H} = \mathcal{G} \cup \mathcal{F}_i \cup \mathcal{H} \\ \Rightarrow\ & C(\mathcal{G} \cup \mathcal{F}_i \cup \mathcal{H}) && = C((\mathcal{F} \oplus_\gamma \mathcal{G}) +_\gamma \mathcal{H}) \\ & && \overset{\text{Satz 10.1.2}}{\geq} \min(C(\mathcal{F} \oplus_\gamma \mathcal{G}), \gamma) \\ & && \overset{R_\gamma 5}{\geq} \min(\gamma, \gamma) = \gamma \end{aligned}$$

Also ist $k \leq i$. Andererseits gilt

$$\begin{aligned} & C(\mathcal{G} \cup \mathcal{H} \cup \mathcal{F}_k) \geq \gamma \\ \Rightarrow\ & C(\mathcal{G} \cup \mathcal{F}_k) \geq \gamma \\ \Rightarrow\ & i \leq k \end{aligned}$$

also ist $i = k$ und

$$(\mathcal{F} \oplus_\gamma \mathcal{G}) +_\gamma \mathcal{H} = \mathcal{G} \cup \mathcal{F}_i \cup \mathcal{H} = \mathcal{G} \cup \mathcal{H} \cup \mathcal{F}_k = \mathcal{F} \oplus_\gamma (\mathcal{G} \cup \mathcal{H})$$

Also sind auch diese beiden Postulate erfüllt.

Als nächstes wird die Richtung „⇒“ betrachtet.

Für den Fall $C(\mathcal{G}) < \gamma$ folgt aus Postulat R_γ2b, daß $\mathcal{F} \oplus_\gamma \mathcal{G} = \mathcal{F}$.

Im Fall $C(\mathcal{F} \cup \mathcal{G}) \geq \gamma$ folgt aus Lemma 10.2.2 (Seite 112), daß $\mathcal{F} \oplus_\gamma \mathcal{G} = \mathcal{F} \cup \mathcal{G}$.

Der dritte Fall ist $C(\mathcal{G}) \geq \gamma \wedge C(\mathcal{F} \cup \mathcal{G}) < \gamma$. Ein Operator $\oplus_\gamma$ gemäß R_γ1–R_γ8 sei fest gewählt. Definiere eine Aufzählung $\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_{2^{|\mathcal{F}|}-1}$ der Teilformeln von $\mathcal{F}$ induktiv wie folgt:

$$\begin{aligned} \mathcal{F}_0 &:= \mathcal{F} \\ \mathcal{F}_{i+1} &:= \begin{cases} (\mathcal{F} \oplus_\gamma \mathcal{G}_i) - \mathcal{G}_i & \textbf{falls } C(\mathcal{G}_i) \geq \gamma \\ \textit{beliebig aus } 2^{\mathcal{F}} - \{\mathcal{F}_0, \ldots, \mathcal{F}_i\} & \textbf{sonst} \end{cases} \end{aligned}$$

wobei $\mathcal{G}_i \in \mathfrak{F}(\Omega)$ eine beliebige Formel mit $\mathcal{F} \cap \mathcal{G}_i = \emptyset$ und

$$\mu_{\mathcal{G}_i}(\omega) = \begin{cases} 1 & \textbf{falls } \mu_{\mathcal{F}_j}(\omega) < \gamma \textit{ für } 0 \leq j \leq i, \\ 0 & \textbf{sonst} \end{cases}$$

ist. Die Existenz solcher $\mathcal{G}_i$ und die Unabhängigkeit der $\mathcal{F}_i$ von der speziellen Wahl der $\mathcal{G}_i$ folgt aus den Lemma 10.2.5 und 10.2.6 (siehe Seite 113).

Sei nun $\mathcal{G} \in \mathfrak{F}(\Omega)$ mit $C(\mathcal{G}) \geq \gamma$, $C(\mathcal{F} \cup \mathcal{G}) < \gamma$ beliebig gewählt. Dann ist die Menge $\{j | C(\mathcal{G} \cup \mathcal{F}_j) \geq \gamma\}$ nicht leer (sie enthält mindestens dasjenige j mit $\mathcal{F}_j = \emptyset$), und sie enthält nicht $\mathcal{F}_0 = \mathcal{F}$. Folglich ist $i := \min\{j | C(\mathcal{G} \cup \mathcal{F}_j) \geq \gamma\}$ wohldefiniert, und es gilt $i > 0$. Zu zeigen ist: $\mathcal{F} \oplus_\gamma \mathcal{G} = \mathcal{G} \cup \mathcal{F}_i$.

Sei nun $\varepsilon > 0$ beliebig. Aus R_γ5 folgt $C(\mathcal{F} \oplus_\gamma \mathcal{G}) \geq \gamma$, folglich existiert ein $\omega \in \Omega$ mit $\mu_{\mathcal{F} \oplus_\gamma \mathcal{G}}(\omega) \geq \gamma - \varepsilon$. Aus R_γ2a folgt dann $\mu_{\mathcal{G}}(\omega) \geq \gamma - \varepsilon$. Ferner ergibt sich aus der Wahl von i, daß für $0 \leq j < i$ gilt: $C(\mathcal{G} \cup \mathcal{F}_j) < \gamma$, insbesondere $\mu_{\mathcal{F}_j}(\omega) < \gamma$. Damit ist $\mu_{\mathcal{G}_{i-1}}(\omega) = 1$ und folglich $C((\mathcal{F} \oplus_\gamma \mathcal{G}) \cup \mathcal{G}_{i-1}) \geq \min(\mu_{\mathcal{F} \oplus_\gamma \mathcal{G}}(\omega), \mu_{\mathcal{G}_{i-1}}(\omega)) \geq \gamma - \varepsilon$. Da $\varepsilon > 0$ beliebig war, gilt auch $C((\mathcal{F} \oplus_\gamma \mathcal{G}) \cup \mathcal{G}_{i-1}) \geq \gamma$. Aus R_γ7 und R_γ8 folgt dann

$$(\mathcal{F} \oplus_\gamma \mathcal{G}) \cup \mathcal{G}_{i-1} = \mathcal{F} \oplus_\gamma (\mathcal{G} \cup \mathcal{G}_{i-1})$$

Weiter existiert wegen $C(\mathcal{G} \cup \mathcal{F}_i) \geq \gamma$ ein $\omega' \in \Omega$ mit $\mu_{\mathcal{G}}(\omega') \geq \gamma - \varepsilon$ und $\mu_{\mathcal{F}_i}(\omega') \geq \gamma - \varepsilon$. Wie oben folgt daraus $\mu_{\mathcal{G}_{i-1}}(\omega') = 1$ und $C(\mathcal{F}_i \cup \mathcal{G}_{i-1} \cup \mathcal{G}) \geq \gamma$. Nach Definition von $\mathcal{F}_i$ gilt $\mathcal{F}_i \cup \mathcal{G}_{i-1} = ((\mathcal{F} \oplus_\gamma \mathcal{G}_{i-1}) - \mathcal{G}_{i-1}) \cup \mathcal{G}_{i-1} = (\mathcal{F} \oplus_\gamma \mathcal{G}_{i-1}) \cup \mathcal{G}_{i-1} = \mathcal{F} \oplus_\gamma \mathcal{G}_{i-1}$, wobei die letzte Gleichheit aus $C(\mathcal{G}_{i-1}) \geq \mu_{\mathcal{G}_{i-1}}(\omega') = 1$ und R_γ2a folgt. Insgesamt ist also $C((\mathcal{F} \oplus_\gamma \mathcal{G}_{i-1}) \cup \mathcal{G}) = C(\mathcal{F}_i \cup \mathcal{G}_{i-1} \cup \mathcal{G}) \geq \gamma$. Aus R_γ7 und R_γ8 folgt dann

$$(\mathcal{F} \oplus_\gamma \mathcal{G}_{i-1}) \cup \mathcal{G} = \mathcal{F} \oplus_\gamma (\mathcal{G} \cup \mathcal{G}_{i-1})$$

und in Kombination mit der entsprechenden Gleichung oben

$$(\mathcal{F} \oplus_\gamma \mathcal{G}_{i-1}) \cup \mathcal{G} = (\mathcal{F} \oplus_\gamma \mathcal{G}) \cup \mathcal{G}_{i-1}$$

Da ohne Beschränkung der Allgemeinheit $\mathcal{G}_{i-1}$ disjunkt zu $\mathcal{F} \cup \mathcal{G}$ gewählt werden kann, läßt sich dies zu

$$((\mathcal{F} \oplus_\gamma \mathcal{G}_{i-1}) - \mathcal{G}_{i-1}) \cup \mathcal{G} = \mathcal{F} \oplus_\gamma \mathcal{G}$$

umformen und durch Einsetzen der Definition von $\mathcal{F}_i$ ergibt sich daraus die Behauptung. ■

Man sieht also, daß jeder Revisionsoperator, der den Axiomen $R_\gamma 1$–$R_\gamma 8$ genügt, zu einem Operator äquivalent ist, der die Teilformeln von $\mathcal{F}$ gemäß einer festgelegten Reihenfolge durchsucht, die erste Teilformel nimmt, die mit $\mathcal{G}$ mindestens zum Niveau γ konsistent ist und diese mit $\mathcal{G}$ vereinigt.

Da offensichtlich nur eine Teilmenge der in Frage kommenden Operatoren die Maximalitätsforderung erfüllt, führen wir noch folgendes Zusatz-Postulat ein:

$R_\gamma 9$ (Informationserhaltung) Ein Revisionsoperator muß eine maximale Ergebnismenge bestimmen:

$$\textbf{Wenn } C(\mathcal{G}) \geq \gamma, \textbf{ gilt für alle } \mathcal{H}, \mathcal{F} \oplus_\gamma \mathcal{G} \subset \mathcal{H} \subseteq \mathcal{F} \cup \mathcal{G} : C(\mathcal{H}) < \gamma$$

Es darf also im Erfolgsfall keine Klausel aus $\mathcal{F} \cup \mathcal{G}$ geben, die sich zu der Ergebnismenge hinzufügen läßt, ohne daß der geforderte Konsistenzgrad unterschritten wird.

Es stellt sich nun die Frage, wie eine für den praktischen Einsatz geeignete Operation gefunden werden kann. Eine pragmatische Lösung dafür sind Revisionsoperatoren basierend auf epistemischen Relevanzordnungen, die wir im nächsten Abschnitt einführen.

10.2.2.1 Ordnungen auf Fuzzy-Formeln

Nach dem Revisionssatz erhält man einen gültigen Operator durch die Angabe einer Ordnung auf den Elementen der Potenzmenge einer Fuzzy-Formel. In der Praxis ist es jedoch leichter, eine Ordnung auf den Elementen einer Formel anzugeben, also auf den einzelnen Fuzzy-Klauseln. Auch wenn aufgrund einer Revision beispielsweise einige Klauseln aufgegeben werden müssen, bleibt oft eine Wahl zwischen verschiedenen Klauselmengen, die man im Ergebnis behalten kann. Abhängig von einer Anwendung wird man vielleicht den zeitlich jüngeren Klauseln Vorrang geben wollen, oder denjenigen Klauseln mit einem höheren Konsistenzgrad.

Um diese *Relevanz* formal festzuhalten, führt man Ordnungen auf den Klauseln einer Formel ein. Die zwei wichtigsten Ordnungen sind dabei die *epistemische Relevanz* von Nebel [Neb90] sowie die *epistemische Verschanzung* von Gärdenfors [Gär88]. Im folgenden betrachten wir nur Revisionsoperationen basierend auf epistemischen Re-

levanzordnungen; nähere Informationen zu den Eigenschaften von epistemischen Verschanzungsordnungen erhält man in [Wit95].[1]

Nebel hat die epistemische Relevanz für den aussagenlogischen Fall eingeführt, um bei einer Revision oder Kontraktion entscheiden zu können, welche der beteiligten Aussagen *epistemisch relevanter* sind. Tritt eine Inkonsistenz auf, wird diejenige Aussage zuerst entfernt, die die kleinste epistemische Relevanz aufweist.

Eine epistemische Relevanzordnung ist zunächst nichts weiter als eine totale Ordnung auf einer Referenzmenge:[2]

Definition 10.2.9 (Epistemische Relevanzordnung) *Eine epistemische Relevanzordnung $\prec_R$ einer Menge M ist eine totale Ordnung mit Maximum; das heißt, es gibt eine irreflexive, asymmetrische, transitive und konnexe Relation auf M, so daß gilt:*

$$\begin{array}{lll}
\forall x \in M: & x \not\prec_R x & \textit{(Irreflexivität)} \\
\forall x, y \in M: & x \prec_R y \Rightarrow y \not\prec_R x & \textit{(Asymmetrie)} \\
\forall x, y, z \in M: & x \prec_R y \wedge y \prec_R z \Rightarrow x \prec_R z & \textit{(Transitivität)} \\
\forall x, y \in M: & x \prec_R y \vee y \prec_R x \vee x = y & \textit{(Konnexität)} \\
\exists x_{max} \in M : \forall y \in M: & y \prec_R x_{max} \vee y = x_{max} & \textit{(maximales Element)}
\end{array}$$

Eine solchermaßen definierte Ordnung läßt sich auch auf den Klauseln einer Fuzzy-Formel anwenden. Dies bildet die Grundlage für eine γ-Revision basierend auf einer epistemischen Relevanzordnung. Dazu bilden wir eine Aufzählung der Teilformeln einer Fuzzy-Formel nach der epistemischen Relevanz der einzelnen Klauseln:

Definition 10.2.10 (Aufzählung einer Formel nach epistemischer Relevanz) *Gegeben sei eine Fuzzy-Formel $\mathcal{F} \in \mathfrak{F}(\Omega)$ mit den Klauseln $\{\mathcal{K}_0, \ldots, \mathcal{K}_{n-1}\}$ in aufsteigender epistemischer Relevanz, das heißt, es gilt $\mathcal{K}_i \prec_R \mathcal{K}_j$ für alle $i < j$. Wir definieren eine Aufzählung der Elemente der Potenzmenge von $\mathcal{F}$ wie folgt: Sei $c := 2^{|\mathcal{F}|} - 1$ und $\mathcal{F}_i = \{\mathcal{K}_{i_1}, \mathcal{K}_{i_2}, \ldots, \mathcal{K}_{i_m}\}$ eine Teilmenge von $\mathcal{F}$. Diese Teilmenge erhält dann den Index*

$$c - \sum_{j=1}^{m} 2^{i_j},$$

also denjenigen Index i, für den der Wert $c - i$ bei Darstellung im Binärsystem genau an den Stellen $i_1, i_2, \ldots, i_m$ eine 1 aufweist.

Damit erhalten wir eine Aufzählung durch

$$2^{\mathcal{F}} = \{\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_c\}$$

[1] Grundsätzlich lassen sich mit epistemischen Verschanzungsordnungen nicht so effiziente Operationen realisieren wie mit epistemischen Relevanzordnungen, da sie sich nicht allein aufgrund der syntaktischen Eigenschaften von Fuzzy-Formeln bestimmen lassen.

[2] Nebel verlangt tatsächlich nur eine total beschränkte Quasiordnung. Für unseren Einsatzzweck ist eine totale Ordnung aber einfacher zu verwenden, daher haben wir die Definition einer epistemischen Relevanzordnung hier angepaßt.

wobei $\mathcal{F}_0 = \mathcal{F}$ und $\mathcal{F}_c = \emptyset$ gilt. Wie man leicht einsieht, hat diese Aufzählung überdies die Eigenschaft, daß aus $\mathcal{F}_i \subseteq \mathcal{F}_j$ auch $i \geq j$ folgt, was im folgenden noch nützlich sein wird.

Man kann sich diese Aufzählung einer Fuzzy-Formel $\mathcal{F}$ auch als Binärbaum veranschaulichen, wobei die Wurzel diejenige Klausel bildet, die die größte epistemische Relevanz besitzt, die inneren Knoten von den weiteren Klauseln in absteigender epistemischer Relevanz gebildet werden und die Blätter schließlich die Teilmengen der Formel bilden, die durch Vereinigung der Knoten auf einem Pfad von der Wurzel entstehen. Man erhält so einen Baum mit $2^{|\mathcal{F}|}$ Blättern; die Ordnung der Unterformeln von $\mathcal{F}$ erhält man, indem man diese Blätter durchgeht, beginnend mit demjenigen Blatt, für das $\mathcal{F}_0 = \mathcal{F}$ gilt.

10.2.2.2 Revision basierend auf epistemischen Relevanzordnungen

Die obige Definition gibt uns jetzt die Möglichkeit, allein basierend auf einer epistemischen Relevanzordnung der Klauseln einer Formel einen gültigen Revisionsoperator festzulegen. Dies hat überdies den Vorteil, daß wir nicht mehr $2^{|\mathcal{F}|}$ Elemente in eine Ordnung bringen müssen, sondern nur noch $|\mathcal{F}|$ Elemente:

Definition 10.2.11 (γ-Revision mit epistemischer Relevanzordnung) *Gegeben seien die Fuzzy-Formeln $\mathcal{F}, \mathcal{G} \in \mathfrak{F}(\Omega)$, eine epistemische Relevanzordnung $\prec_R$ auf den Klauseln von $\mathcal{F}$, sowie eine Aufzählung $\{\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_{2^{|\mathcal{F}|}-1}\}$ von $2^{\mathcal{F}}$ nach Definition 10.2.10 auf der vorherigen Seite. Der zu dieser epistemischen Relevanzordnung gehörende γ-Revisionsoperator $\oplus_\gamma^R$ ist dann folgendermaßen definiert:*

$$\mathcal{F} \oplus_\gamma^R \mathcal{G} = \begin{cases} \mathcal{F} & \textbf{wenn } C(\mathcal{G}) < \gamma, \\ \mathcal{F} \cup \mathcal{G} & \textbf{wenn } C(\mathcal{F} \cup \mathcal{G}) \geq \gamma, \\ \mathcal{F}_i \cup \mathcal{G} & \textbf{sonst, wobei } i = \min\{j | C(\mathcal{F}_j \cup \mathcal{G}) \geq \gamma\}. \end{cases}$$

Nach dem Revisionssatz (Satz 10.2.8 auf Seite 115) erfüllt dieser Operator die Postulate $R_\gamma 1$–$R_\gamma 8$. Er erfüllt ebenfalls Postulat $R_\gamma 9$, was in Satz 10.2.14 auf Seite 122 gezeigt wird.

Damit sind wir auf dem Weg zu einem praktisch einsetzbaren Operator einen Schritt weiter. Es verbleibt jedoch die Aufgabe, eine geeignete Ordnung auf den Fuzzy-Klauseln einer Formel festzulegen. Prinzipiell könnte man die Ordnung von einem Anwender bestimmen lassen, was aber entsprechend aufwendig ist und nur in Ausnahmefällen Sinn machen dürfte. Idealerweise sollte sich eine Ordnung anhand der Eigenschaften einer Klausel selbst bestimmen lassen. Einige Möglichkeiten dazu, die auch miteinander kombiniert werden können, sind:

Ordnung nach Zeit: Die Klauseln werden nach dem Zeitpunkt ihres Hinzufügens zu einer Fuzzy-Formel sortiert. Je nach Einsatzgebiet kann man dann entweder den zeitlich jüngeren oder älteren Klauseln Priorität geben.

Ordnung nach Anzahl der Literale: Je mehr Literale eine Klausel besitzt, desto höher ist ihre Relevanz. Dies kann in Anwendungsfällen sinnvoll sein, bei denen die Literale Einzelinformationen repräsentieren, die beispielsweise von Meßsensoren geliefert oder in Heuristiken berechnet werden.

Ordnung nach Konsistenzgrad: Je größer der Konsistenzgrad einer Klausel, desto höher ist ihre epistemische Relevanz.

Ordnung nach Sicherheitsliteral: Wenn ein Klausel mit Sicherheitsliteral verwendet wird (vergleiche Definition 8.1.6 auf Seite 77), kann man den Sicherheitsgrad α als Ordnungskriterium verwenden.

Dies sind Eigenschaften, die sich entweder rein syntaktisch oder anhand der Fuzzy-Interpretation bestimmen lassen. Unter Kenntnis der entsprechenden Anwendungsdomäne wird es meist möglich sein, eine geeignete Ordnung festzulegen, die die Eigenheiten des Einsatzgebietes berücksichtigt. Zusätzlich lassen sich Domänenkenntnisse ausnutzen, um anwendungsspezifische Ordnungen zu definieren, die auch die Semantik einer Klausel berücksichtigen. Man beachte, daß diese Kriterien für sich im Allgemeinen noch keine totale Ordnung, sondern nur eine Halbordnung festlegen. Erst durch eine Kombination oder Einbeziehung zusätzlicher (auch anwendungsspezifischer) Merkmale erhält man die von der epistemischen Relevanzordnung geforderte totale Ordnung.

Da wir an dieser Stelle keine konkrete Anwendungsdomäne voraussetzen, gehen wir im folgenden von einer Ordnung nach dem Konsistenzgrad einer Klausel aus. Da mehrere Klauseln den gleichen Konsistenzgrad haben können, sei zusätzlich definiert, daß bei gleichem Konsistenzgrad die zeitlich jüngere Klausel die höhere epistemische Relevanz hat. Diese epistemische Relevanzordnung bezeichen wir mit $O_{\prec_R}$:

Definition 10.2.12 (Epistemische Relevanzordnung $O_{\prec_R}$) *Eine Fuzzy-Klausel $\mathcal{K}_1 \in \mathfrak{K}(\Omega)$ ist epistemisch relevanter als eine Fuzzy-Klausel $\mathcal{K}_2 \in \mathfrak{K}(\Omega)$, geschrieben $\mathcal{K}_2 \prec_R \mathcal{K}_1$, wenn $C(\mathcal{K}_1) > C(\mathcal{K}_2)$ gilt. Sind beide Konsistenzgrade gleich ($C(\mathcal{K}_1) = C(\mathcal{K}_2)$), gilt: diejenige Klausel, die zuletzt eingefügt wurde, hat die höhere epistemische Relevanz.*

Man beachte, daß dabei implizit vorausgesetzt wird, daß niemals zwei gleich-konsistente Klauseln zum gleichen Zeitpunkt eingefügt werden können. Ein Datenbanksystem kann dies beispielsweise durch Serialisierung garantieren.

10.2.2.3 Effiziente Revisionsoperatoren basierend auf epistemischer Relevanz

Mit Hilfe der obigen Ordnung und der Revision basierend auf epistemischer Relevanz (Definition 10.2.11 auf der vorherigen Seite) haben wir zwar einen konstruktiven Operator. Dieser ist aber immer noch nicht für einen praktischen Einsatz geeignet,

da die Suche nach dem kompatiblen Element mit dem kleinsten Index in der Potenzmenge einer Formel unakzeptabel langsam wäre. Gesucht ist also ein Operator, der ein äquivalentes Ergebnis liefert, aber effizienter zu berechnen ist.

Wenn man sich die Aufzählung nach epistemischer Relevanz wie oben beschrieben als Binärbaum veranschaulicht, kommt man zu einem äquivalenten Operator, wenn man eine Binärsuche in diesem Baum durchführt. Dabei entscheidet man an jedem Knoten, ob eine Klausel hinzugefügt werden kann, beginnend mit der relevantesten Klausel an der Wurzel des Baumes. Das Blatt bildet dann diejenige Teilmenge der ursprünglichen Formel, die mit den Klauseln der neuen Formel maximal kompatibel ist. Den entsprechenden Operator erhält man, wenn man von dem Index i dieses Blattes ausgeht und die Binärdarstellung des Wertes $2^{|\mathcal{F}|} - 1 - i$ betrachtet: Man beginnt mit der Prüfung des höchstwertigen Bits, welches der Klausel mit der größten epistemischen Relevanz entspricht. Ist diese Klausel kompatibel, wird sie in die Ergebnismenge aufgenommen. Diesen Prozeß setzt man nacheinander mit allen Klauseln in den niedrigerwertigen Bitpositionen fort, was zu der folgenden konstruktiven, effizient realisierbaren Revisionsoperation führt:

Definition 10.2.13 (Effizienter γ-Revisionsoperator mit epistemischer Relevanz) *Die γ-Revision einer Fuzzy-Formel $\mathcal{F} \in \mathfrak{F}(\Omega)$ um eine Fuzzy-Formel $\mathcal{G} \in \mathfrak{F}(\Omega)$ wird berechnet, indem die Fuzzy-Formel $\mathcal{G}$ nacheinander um alle Fuzzy-Klauseln aus $\mathcal{F}$ in der Reihenfolge ihrer epistemischen Relevanz γ-expandiert wird. Hierbei sei $\mathcal{F} = \{\mathcal{K}_1, \ldots, \mathcal{K}_n\}$ mit $\mathcal{K}_1 \succ_R \ldots \succ_R \mathcal{K}_n$:*

$$\mathcal{F} \oplus_\gamma^R \mathcal{G} \stackrel{\text{def}}{=} \begin{cases} ((\mathcal{G} +_\gamma \mathcal{K}_1) +_\gamma \ldots) +_\gamma \mathcal{K}_n & \textbf{wenn } C(\mathcal{G}) \geq \gamma, \\ \mathcal{F} & \textbf{sonst} \end{cases}$$

Es ist leicht zu sehen, daß dieser Operator äquivalent zu dem in Definition 10.2.11 auf Seite 120 gezeigten Revisionsoperator ist. Es verbleibt jetzt noch zu zeigen, daß auch Postulat $R_\gamma 9$ erfüllt ist:

Satz 10.2.14 (Maximale Ergebnismenge) *Der Revisionsoperator nach Definition 10.2.13 liefert eine maximale Ergebnismenge, das heißt, es existiert keine Klausel $\mathcal{K}_i \in \mathcal{F} \cup \mathcal{G}$, die zu $\mathcal{F} \oplus_\gamma^R \mathcal{G}$ hinzugefügt werden kann, so daß $C(\mathcal{F} \oplus_\gamma^R \mathcal{G} \cup \{\mathcal{K}_i\}) \geq \gamma$ gilt.*

Beweis. *Beweis durch Widerspruch. Sei $\mathcal{F}' = \mathcal{F} \oplus_\gamma^R \mathcal{G} = ((\mathcal{G} +_\gamma \mathcal{K}_1) +_\gamma \ldots) +_\gamma \mathcal{K}_n$, mit $\mathcal{K}_1, \ldots, \mathcal{K}_n \in \mathcal{F}$ und $\mathcal{K}_1 \succ_R \ldots \succ_R \mathcal{K}_n$. Wir nehmen an, es existiere eine Klausel $\mathcal{K}_i$, die sich zu der Ergebnismenge hinzufügen läßt, ohne daß die Konsistenzbedingung verletzt wird:*

$$\exists \mathcal{K}_i \in (\mathcal{F} \cup \mathcal{G}), \mathcal{K}_i \notin \mathcal{F}' : C(\mathcal{F}' \cup \{\mathcal{K}_i\}) \geq \gamma$$

Dann gilt:

$$\begin{aligned} & C((((\mathcal{G} +_\gamma \mathcal{K}_1) +_\gamma \ldots) +_\gamma \mathcal{K}_{i-1}) \cup \{\mathcal{K}_i\}) \geq \gamma, \quad \textit{da } C(\mathcal{F}' \cup \{\mathcal{K}_i\}) \geq \gamma \\ \Rightarrow \quad & \mathcal{K}_i \in ((((((\mathcal{G} +_\gamma \mathcal{K}_1) +_\gamma \ldots) +_\gamma \mathcal{K}_{i-1}) +_\gamma \mathcal{K}_i) +_\gamma \mathcal{K}_{i+1}) +_\gamma \ldots) +_\gamma \mathcal{K}_n \\ \Rightarrow \quad & \mathcal{K}_i \in \mathcal{F}' \end{aligned}$$

Dies ist ein Widerspruch zur Annahme $\mathcal{K}_i \notin \mathcal{F}'$, also ist die Ergebnismenge maximal. ■

Man kann sich leicht veranschaulichen, daß jede Aufzählung $2^{\mathcal{F}}$, die mit der Untermengenbeziehung kompatibel ist, das Maximalitätspostulat erfüllt. Es muß also aus $\mathcal{F}_i \supseteq \mathcal{F}_j$ folgen, daß $i \leq j$ gilt, was für die epistemische Relevanzordnung offensichtlich erfüllt ist.

Damit sind wir jetzt am Ziel: wir haben einen Revisionsoperator, der zum einen den aufgestellten Postulaten genügt und zum anderen effizient realisierbar ist.

10.2.3 Der γ-Revisionsalgorithmus

Wir zeigen nun, wie die unter 10.2.13 definierte γ-Revisionsoperation als Algorithmus formuliert werden kann.

Zur γ-Revision einer Fuzzy-Formel $\mathcal{F}$ um eine Fuzzy-Formel $\mathcal{G}$ wird zunächst überprüft, ob $\mathcal{G}$ den geforderten Konsistenzgrad γ erreicht. Ist dies der Fall, beginnt die Berechnung der Ergebnisformel mit der Aufnahme aller Klauseln aus $\mathcal{G}$. Dann werden nacheinander alle Klauseln aus $\mathcal{F}$ gemäß ihrer epistemischen Relevanz durch γ-Expansion hinzugefügt, wie in Algorithmus 10.2.1 auf der nächsten Seite gezeigt.

In dem Algorithmus wird dabei ein sogenannter *Iterator* verwendet, um die Klauselmenge einer Formel sortiert nach der übergebenen Ordnung o durchlaufen zu können.

Die Methode gibt schließlich einen Statuswert zurück, der angibt, ob die Operation erfolgreich oder erfolglos war. In beiden Fällen enthält das Formel-Objekt den durch den γ-Revisionsoperator definierten Wert.

10.3 Kontraktion von Fuzzy-Formeln

Die nächste Operation, die wir betrachten, ist die *Kontraktion*. Dabei geht es um das Problem, eine bestimmte Information aus einer Fuzzy-Formel zu entfernen (vergleiche Abschnitt 9.1.2 auf Seite 94).

Eine Information wieder „loszuwerden“ kann tatsächlich eine genauso schwierige Aufgabe sein, wie sie ursprünglich einzubringen. Es gilt zwei Fälle der Kontraktion auseinanderzuhalten:

Syntaktische Kontraktion Der einfachste Fall ist, eine Klausel oder eine Klauselmenge aus einer Formel zu entfernen. Diese Variante wird verwendet, wenn innerhalb einer Anwendung Informationen zurückgezogen werden müssen. Beispiele sind der Fall eines defekten Meßsensors, der falsche Daten geliefert hat, einer Heuristik, die sich im nachhinein als nicht anwendbar herausstellt oder einer Datenquelle, die inkorrekte Daten geliefert hat. Dank unseres Repräsentationsmodells ist es einfach möglich, solche Informationen zu identifizieren und

Algorithmus 10.2.1 γ-Revision basierend auf epistemischer Relevanz

```
1  STATUS FORMEL.γ-REVISION( formel G, konsistenzgrad γ, ordnung o )
2  begin
3     // Überprüfe, ob die neue Formel den geforderten Konsistenzgrad erreicht
4     if G.konsistenzgrad() < γ
5        then return(MISSERFOLG);
6     fi
7
9     // Die Ergebnisformel enthält alle Klauseln aus G
10    F' := G;
11
13    // Nun fügen wir die Klauseln der aktuellen Formel durch γ-Expansion hinzu
14    // Dazu benutzen wir einen Iterator, der die Klauseln dieser Formel
15    // sortiert nach der Ordnung o anbietet
16
18    iterator := sortierterKlauselIterator(this.F,o);
19    while K := iterator.next() do
20          F'.γ-expansion({K}, γ);
21    od
22
24    this.F := F';
25    return(ERFOLG);
26 end
```

herauszunehmen — vorausgesetzt natürlich, die Implementierung erlaubt eine eindeutige Identifizierung (siehe auch Abschnitt 8.4 auf Seite 82).

Semantische Kontraktion Den zweiten Fall bezeichnen wir in dieser Arbeit als *semantische* Kontraktion, da es hierbei auf die Interpretation einer Information als Fuzzy-Menge ankommt. Deshalb ist es nicht möglich, eine Klauselmenge syntaktisch auszuwählen, um sie eindeutig zu entfernen. Dieser Fall tritt immer dann auf, wenn Wissen gezielt ausgeschlossen werden sollen — Beispiele sind die sprichwörtlichen Ausnahmen von der Regel *(„täglich außer Montags“)*, die in Heuristiken modelliert sein können, als Ausschlüsse formulierte Anforderungen oder entdeckte Gegenbeispiele, die in eine Fuzzy-Formel aufgenommen werden sollen. Wie nachfolgend gezeigt wird, ist diese Art der Kontraktion mit der zuvor betrachteten Revision verwandt.

Der erste Fall entspricht der schon definierten Subtraktion von Fuzzy-Formeln (siehe Definition 9.3.4 auf Seite 100). Der zweite Fall entspricht einer γ-Kontraktion, deren Syntax wir folgendermaßen festlegen:

Definition 10.3.1 (γ-Kontraktion) *Eine γ-Kontraktion $\ominus_\gamma$ ist eine Abbildung, die zwei Fuzzy-Formeln $\mathcal{F}, \mathcal{G} \in \mathfrak{F}(\Omega)$ und einem Wert $\gamma \in [0,1]$ eine weitere Fuzzy-Formel aus $\mathfrak{F}(\Omega)$ zuordnet:*

$$\ominus_\gamma : \mathfrak{F}(\Omega) \times \mathfrak{F}(\Omega) \to \mathfrak{F}(\Omega)$$

und die den Postulaten $K_\gamma 1$–$K_\gamma 6$ auf Seite 126 genügt.

Enthält die Formel $\mathcal{G}$ nur eine Klausel $\mathcal{K}$, erlauben wir anstelle von $\mathcal{F} \ominus_\gamma \{\mathcal{K}\}$ auch die abkürzende Schreibweise $\mathcal{F} \ominus_\gamma \mathcal{K}$.

Nun ist aus der aussagenlogischen Wissenverarbeitung bekannt, daß es einen engen Zusammenhang zwischen der Revision und der Kontraktion gibt, da sie sich gegenseitig über die Harper/Levi-Identitäten ausdrücken lassen (vergleiche Anhang A.6). Übertragen auf unser Modell ließe sich die Kontraktion damit folgendermaßen definieren:

$$\mathcal{F} \ominus_\gamma \mathcal{G} = (\mathcal{F} \oplus_\gamma \overline{\mathcal{G}}) \cap \mathcal{F}$$

Es stellt sich nun die Frage, ob dies eine sinnvolle Definition für die γ-Kontraktion von Fuzzy-Formeln darstellt. Betrachten wir dazu ein kleines Beispiel: wir gehen wieder von dem Reengineering-Szenario aus und möchten aus einer Fuzzy-Formel $\mathcal{F} = \{\mathcal{K}_1, \mathcal{K}_2\}$ (siehe Abbildung 10.3 auf der nächsten Seite) die Information $\mathcal{G}$ entfernen, wonach der Typ *int* nicht mehr in Frage kommt (siehe Abbildung 10.4 auf Seite 127).

Wie man leicht nachrechnen kann, ist das Ergebnis der Kontraktionsoperation

$$\mathcal{F} \ominus_\gamma \mathcal{G} = (\mathcal{F} \oplus^R_{0.5} \overline{\mathcal{G}}) \cap \mathcal{F} = \mathcal{F},$$

das heißt, $\mathcal{F}$ bleibt unverändert; der Typ *int* kommt also weiterhin mit einer Sicherheit von 0,80 in Frage. Was ist passiert?

Tatsächlich erzeugt diese Definition eine sehr kontraintuitiv arbeitende Operation, denn nach einer Kontraktion ist keinesfalls gesichert, daß die zu entfernende Eigenschaft nicht immer noch in der Interpretation der Fuzzy-Formel enthalten ist. An obigem Beispiel kann man sehen, daß diese Operation versucht, die Information *„die Variable ist int **und nur int**“* zu entfernen — eine solche Klausel ist aber nicht Bestandteil unserer Fuzzy-Formel, kann also auch nicht entfernt werden. Insofern ist diese Operation korrekt, aber ist dies wirklich ein wünschenswertes und intuitives Verhalten?

Wir schlagen für den hier betrachteten Einsatzzweck eine andere Kontraktionsoperation vor, die nach unserer Ansicht einfacher von einem Entwickler von Fuzzy-Informationssystemen eingesetzt werden kann: nach einer Kontraktion soll die Negation der zu entfernenden Information im Ergebnis enthalten sein. Die oben beschriebene Kontraktionsoperation läßt sich überdies mit Hilfe dieser Operation berechnen, indem das Ergebnis zusätzlich mit der Ausgangsformel geschnitten wird.

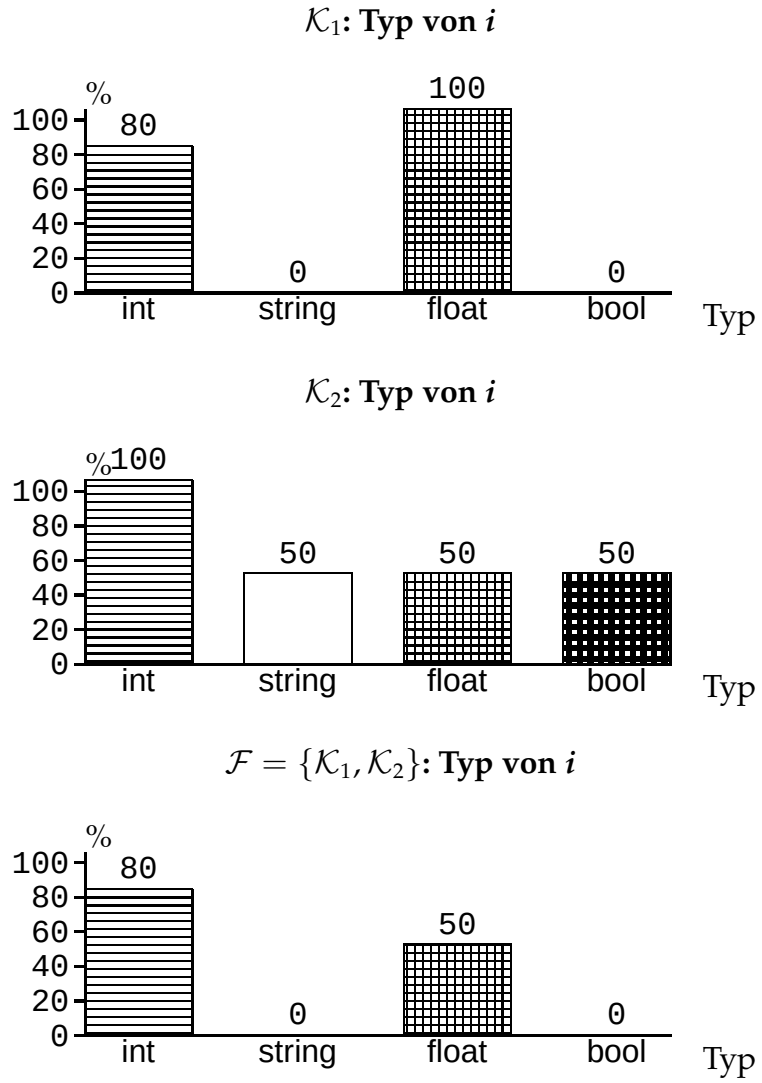


Abbildung 10.3: Beispielformel mit zwei Klauseln für eine γ-Kontraktion

10.3.1 Postulate der γ-Kontraktion

Wir adaptieren wieder die bekannten AGM-Postulate für unsere γ-Kontraktion (siehe Anhang A.4 auf Seite 309).

Auch hier legt der Grad γ fest, daß das Ergebnis mindestens den Konsistenzgrad γ erreichen muß. Für alle Postulate gilt: $\mathcal{F}, \mathcal{G}, \mathcal{H}$ bezeichnen Fuzzy-Formeln aus $\mathfrak{F}(\Omega)$.

K_γ1 (Stabilität) Nach einer Kontraktionsoperation soll wieder eine Fuzzy-Formel vorhanden sein:

$$\mathcal{F} \ominus_\gamma \mathcal{G} \in \mathfrak{F}(\Omega)$$

K_γ2 (Vorhersehbarkeit) Die Kontraktion von $\mathcal{F}$ um $\mathcal{G}$ soll nur Klauseln enthalten,

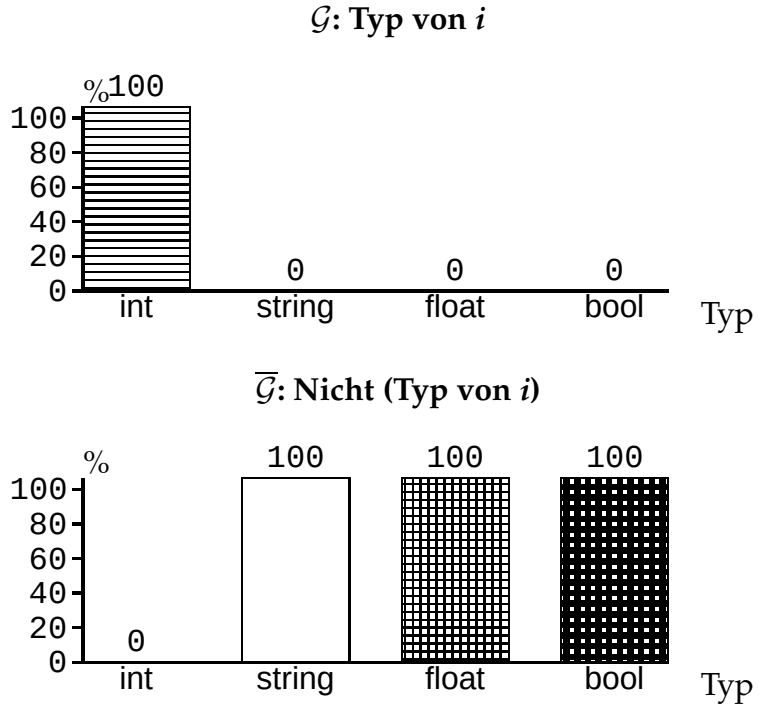


Abbildung 10.4: Zu entfernende Information $\mathcal{G}$ und deren Negation

die aus $\mathcal{F}$ oder nicht $\mathcal{G}$ stammen:

$$\mathcal{F} \ominus_\gamma \mathcal{G} \subseteq \mathcal{F} \cup \overline{\mathcal{G}}$$

K_γ3 (Kompatibilität) Wenn $\overline{\mathcal{G}}$ keiner Klausel in $\mathcal{F}$ widerspricht, sollen die Klauseln aus $\mathcal{F}$ und $\overline{\mathcal{G}}$ in der Kontraktion enthalten sein:

$$\textbf{Wenn } C(\mathcal{F} \cup \overline{\mathcal{G}}) \geq \gamma, \textbf{ dann } \mathcal{F} \ominus_\gamma \mathcal{G} \supseteq \mathcal{F} \cup \overline{\mathcal{G}}$$

K_γ4a (Erfolg) Die Kontraktion soll erfolgreich sein:

$$\textbf{Wenn } C(\overline{\mathcal{G}}) \geq \gamma, \textbf{ dann } \overline{\mathcal{G}} \subseteq \mathcal{F} \ominus_\gamma \mathcal{G}$$

K_γ4b (Mißerfolg) Wenn der Konsistenzgrad von $\overline{\mathcal{G}}$ kleiner als γ ist, bleibt die Ausgangsformel unverändert:

$$\textbf{Wenn } C(\overline{\mathcal{G}}) < \gamma, \textbf{ dann } \mathcal{F} \ominus_\gamma \mathcal{G} = \mathcal{F}$$

K_γ4c (Konsistenz) Das Ergebnis einer erfolgreichen Kontraktion erreicht mindestens den Konsistenzgrad γ:

$$\textbf{Wenn } C(\overline{\mathcal{G}}) \geq \gamma, \textbf{ dann } C(\mathcal{F} \ominus_\gamma \mathcal{G}) \geq \gamma$$

$K_\gamma 5$ (Informationserhaltung) Die Kontraktion soll eine maximale Ergebnismenge bestimmen:

$$\textbf{Wenn } C(\overline{\mathcal{G}}) \geq \gamma, \textbf{ gilt für alle } \mathcal{H}, \mathcal{F} \ominus_\gamma \mathcal{G} \subset \mathcal{H} \subseteq \mathcal{F} \cup \overline{\mathcal{G}} : C(\mathcal{H}) < \gamma$$

$K_\gamma 6$ (Identität) Bei einer Kontraktion mit verschiedenen, aber semantisch äquivalenten Fuzzy-Formeln ist auch das Ergebnis semantisch äquivalent:

$$\textbf{Wenn } \mu_{\mathcal{G}} = \mu_{\mathcal{H}}, \textbf{ dann } \mu_{\mathcal{F} \ominus_\gamma \mathcal{G}} = \mu_{\mathcal{F} \ominus_\gamma \mathcal{H}}$$

Das erste Postulat sichert wie bei der Expansion und Revision, daß das Ergebnis einer Kontraktionsoperation wieder eine Fuzzy-Formel darstellt.

Postulat $K_\gamma 2$ definiert die Kontraktionseigenschaft: die Ergebnisformel enthält keine neuen Informationen mit Ausnahme der Information $\overline{\mathcal{G}}$, die wir aus den oben angeführten Gründen in die Ergebnisformel aufnehmen.

Gibt es keinen Widerspruch zwischen $\mathcal{F}$ und $\overline{\mathcal{G}}$, muß auch nichts aus $\mathcal{F}$ entfernt werden; diesen Fall regelt Postulat $K_\gamma 3$.

Wenn der Konsistenzgrad von $\overline{\mathcal{G}}$ mindestens den Grad γ erreicht, soll die Kontraktion erfolgreich sein (Postulat $K_\gamma 4a$); das Ergebnis muß in diesem Fall mindestens den Konsistenzgrad γ erreichen (Postulat $K_\gamma 4c$). Andernfalls haben wir wie bei der Expansion oder Revision eine erfolglose Operation, bei der die Ausgangsformel unverändert bleibt (Postulat $K_\gamma 4b$).

Postulat $K_\gamma 5$ fordert, nicht mehr Informationen zu entfernen als notwendig; bei der Revision entspricht dies dem Postulat $R_\gamma 8$.

Wie bei der Expansion und Revision soll die syntaktische Form der beteiligten Fuzzy-Formeln keine Auswirkungen auf das Ergebnis einer Kontraktion haben, was durch Postulat $K_\gamma 6$ festgehalten wird.

10.3.2 Die γ-Kontraktionsoperation

Auch hier gilt es jetzt, eine effiziente Operation zu finden, die die oben aufgestellten Postulate erfüllt.

Man kann anhand der Postulate leicht sehen, daß für jeden Operator $\ominus_\gamma$ gelten muß, daß er (ähnlich wie bei der Revision) eine Abbildung $\mathcal{F}^{\ominus_\gamma} : \mathfrak{F}(\Omega) \rightarrow \mathfrak{F}(\Omega)$, $\mathcal{G} \mapsto \mathcal{F} \ominus_\gamma \mathcal{G}$ durchführt, wobei die Ergebnisformel im Fall $C(\overline{\mathcal{G}}) < \gamma$ gleich $\mathcal{F}$, ansonsten eine maximale mit $\overline{\mathcal{G}}$ kompatible Teilmenge von $\mathcal{F}$ sein muß.

Eine Möglichkeit, die diese Forderung offensichtlich erfüllt, ist die γ-Kontraktion auf die γ-Revision zurückzuführen, wie es oben gezeigt wurde. Dies führt zu folgender Definition:

Definition 10.3.2 (γ-Kontraktionsoperator) *Seien $\mathcal{F}, \mathcal{G} \in \mathfrak{F}(\Omega)$. Der γ-Kontraktionsoperator $\ominus_\gamma$ ist definiert durch:*

$$\mathcal{F} \ominus_\gamma \mathcal{G} \stackrel{\text{def}}{=} \mathcal{F} \oplus_\gamma^R \overline{\mathcal{G}}$$

Wir müssen nun überprüfen, ob die so definierte Operation die Kontraktionspostulate erfüllt.

Satz 10.3.3 (Kontraktionssatz) *Die in 10.3.2 definierte Operation erfüllt die γ-Kontraktionspostulate.*

Beweis. *Es wird die Erfüllung der einzelnen Postulate überprüft:*

$K_\gamma 1$ $(\mathcal{F} \ominus_\gamma \mathcal{G} \in \mathfrak{F}(\Omega))$ *Erfüllt. Dies folgt direkt aus der Erfüllung des γ-Revisionspostulates $R_\gamma 1$.*

$K_\gamma 2$ $(\mathcal{F} \ominus_\gamma \mathcal{G} \subseteq \mathcal{F} \cup \overline{\mathcal{G}})$ *Erfüllt. Dies folgt direkt aus dem Postulat $R_\gamma 3$:*

$$\mathcal{F} \ominus_\gamma \mathcal{G} = \mathcal{F} \oplus_\gamma \overline{\mathcal{G}} \stackrel{R_\gamma 3}{\subseteq} \mathcal{F} \cup \overline{\mathcal{G}}$$

$K_\gamma 3$ (**Wenn** $C(\mathcal{F} \cup \overline{\mathcal{G}}) \geq \gamma$, **dann** $\mathcal{F} \ominus_\gamma \mathcal{G} \supseteq \mathcal{F} \cup \overline{\mathcal{G}}$) *Erfüllt.*

$$\begin{aligned} C(\mathcal{F} \cup \overline{\mathcal{G}}) \geq \gamma &\Rightarrow \mathcal{F} \oplus_\gamma \overline{\mathcal{G}} = \mathcal{F} \cup \overline{\mathcal{G}} \quad \textit{(Lemma 10.2.2)} \\ &\Rightarrow \mathcal{F} \ominus_\gamma \mathcal{G} = \mathcal{F} \cup \overline{\mathcal{G}} \supseteq \mathcal{F} \cup \overline{\mathcal{G}} \end{aligned}$$

$K_\gamma 4a$ (**Wenn** $C(\overline{\mathcal{G}}) \geq \gamma$, **dann** $\overline{\mathcal{G}} \subseteq \mathcal{F} \ominus_\gamma \mathcal{G}$) *Erfüllt.*

$$\begin{aligned} C(\overline{\mathcal{G}}) \geq \gamma &\Rightarrow \overline{\mathcal{G}} \stackrel{R_\gamma 2a}{\subseteq} \mathcal{F} \oplus_\gamma \overline{\mathcal{G}} \\ &\Rightarrow \overline{\mathcal{G}} \subseteq (\mathcal{F} \ominus_\gamma \mathcal{G}) \end{aligned}$$

$K_\gamma 4b$ (Wenn $C(\overline{\mathcal{G}}) < \gamma$, **dann** $\mathcal{F} \ominus_\gamma \mathcal{G} = \mathcal{F}$**)** *Erfüllt. Dies folgt direkt aus dem γ-Revisionspostulat $R_\gamma 2b$.*

$K_\gamma 4c$ (Wenn $C(\overline{\mathcal{G}}) \geq \gamma$, **dann** $C(\mathcal{F} \ominus_\gamma \mathcal{G}) \geq \gamma$**)** *Erfüllt. Dies folgt direkt aus dem γ-Revisionspostulat $R_\gamma 5$.*

$K_\gamma 5$ (Für alle $\mathcal{H}, (\mathcal{F} \ominus_\gamma \mathcal{G}) \subset \mathcal{H} \subseteq \mathcal{F} \cup \overline{\mathcal{G}}$ **gilt:** $C(\mathcal{H}) < \gamma$**)** *Erfüllt. Dies folgt direkt aus dem Satz 10.2.14 über maximale Ergebnismengen bei der γ-Revision.*

$K_\gamma 6$ (Wenn $\mu_\mathcal{G} = \mu_\mathcal{H}$, **dann** $\mu_{\mathcal{F} \ominus_\gamma \mathcal{G}} = \mu_{\mathcal{F} \ominus_\gamma \mathcal{H}}$**)** *Erfüllt. Dies folgt direkt aus dem γ-Revisionspostulat $R_\gamma 6$.* ■

Algorithmus 10.3.1 γ-Kontraktion

```
1  STATUS FORMEL.γ-KONTRAKTION( formel G, konsistenzgrad γ, ordnung o )
2  begin
3     // Berechne die Negation von G
4     G.negiere_formel();
5
7     // Berechne die γ-Revision dieser Formel um ¬G
8     status := this.γ-revision(G, γ, o);
9
11    // Die Kontraktion ist erfolgreich, wenn die Revision erfolgreich war
12    return(status);
13 end
```

10.3.3 Der γ-Kontraktionsalgorithmus

Auch für die γ-Kontraktion zeigen wir einen möglichen Algorithmus. Dieser läßt sich sehr einfach darstellen, da die Kontraktion auf die Revision zurückgeführt wird (Algorithmus 10.3.1).

10.4 Expansion von Fuzzy-Abhängigkeitsgraphen

Nachdem wir die Operationen Expansion, Revision und Kontraktion für einzelne Fuzzy-Formeln definiert haben, steht jetzt die Erweiterung auf Fuzzy-Abhängigkeitsgraphen an.

Die Grundidee bei allen Graph-Operationen ist dabei, daß Änderungen, die an einem bestimmten Knoten des Graphen eingebracht werden, über die Kanten zu den abhängigen Knoten fortgeschrieben werden. Damit lassen sich Berechnungen formalisieren, wie sie im Beispiel in Abschnitt 8.5 auf Seite 85 angerissen wurden.

Bei der Formulierung dieser Graph-Operationen behalten wir die grundsätzliche Vorgehensweise bei, die notwendigen Eigenschaften dieser Operatoren formal mit Hilfe von Postulaten festzuhalten, aufgrund derer dann geeignete Möglichkeiten zu ihrer Definition untersucht werden.

Wir legen zuerst die Syntax für eine Graph-Expansion fest:

Definition 10.4.1 (Graph-γ-Expansion) *Eine Graph-γ-Expansion $+_\gamma^{\mathfrak{G}}$ ist eine Abbildung, die einem Fuzzy-Abhängigkeitsgraphen mit Quelle $\langle \mathfrak{G}, s\rangle \in \langle \mathbb{G}, \mathbb{V}\rangle$, einem Wert $\gamma \in [0,1]$ und einer Fuzzy-Formel $\mathcal{F}$ mit $\mathcal{F}, \xi(s) \in \mathfrak{F}(\Omega)$ einen weiteren Abhängigkeitsgraphen zuordnet:*

$$+_\gamma^{\mathfrak{G}} : \langle \mathbb{G}, \mathbb{V}\rangle \times \mathfrak{F} \to \mathbb{G}$$

und die den Postulaten $GE_\gamma 1$–$GE_\gamma 6$ (Seite 131) genügt.

Die Expansion eines Abhängigkeitsgraphen wird an einem ausgezeichneten Knoten gestartet, der sogenannten Quelle (vergleiche Definition 8.5.4 auf Seite 87). Von dieser Quelle ausgehend wird die neue Information durch den Graphen propagiert, wie wir unten noch zeigen werden. Zunächst muß jedoch die Semantik der Graph-Expansion festgelegt werden.

10.4.1 Postulate der Graph-Expansion

Auch für die Graph-Expansion definieren wir Postulate, die eine Operation notwendigerweise erfüllen muß. Hierzu gibt es aber, im Gegensatz zur γ-Expansion einzelner Fuzzy-Formeln, keine analogen Arbeiten aus dem Gebiet der aussagenlogischen Wissensverarbeitung. Bei der Formulierung der Postulate nehmen wir jedoch Rückgriff auf die bereits definierten konsistenzerhaltenden Operationen für Fuzzy-Formeln.

Bei der Behandlung von Abhängigkeitsgraphen gibt es mehrere Möglichkeiten, sinnvolle Expansions-Operationen festzulegen. Unsere Vorgehensweise ist daher, grundlegende Eigenschaften mit Hilfe von Basis-Postulaten zu formulieren, die für alle Varianten der Graph-Expansion Geltung haben sollen, und die Eigenschaften verschiedener Ausprägungen durch eine Menge von Zusatzpostulaten festzuhalten.

Für alle nachfolgenden Postulate gilt: $\mathfrak{G} = (V, E, \xi) \in \mathbb{G}$ ist ein Fuzzy-Abhängigkeitsgraph, $\mathcal{F} \in \mathfrak{F}(\Omega)$ bezeichnet eine Fuzzy-Formel, der Knoten $s \in V$ eine Quelle in $\mathfrak{G}$, für die $\xi(s) \in \mathfrak{F}(\Omega)$ gelten muß, und $\mathfrak{G}' = \langle \mathfrak{G}, s \rangle +_\gamma^{\mathfrak{G}} \mathcal{F} = (V', E', \xi')$:

GE$_\gamma$1 (Stabilität) Das Ergebnis einer Graph-Expansionsoperation ist wieder ein Fuzzy-Abhängigkeitsgraph:

$$\langle \mathfrak{G}, s \rangle +_\gamma^{\mathfrak{G}} \mathcal{F} \in \mathbb{G}$$

GE$_\gamma$2 (Graphinvarianz) Die Knotenmenge sowie die Struktur des Graphen bleiben erhalten:

$$V = V' \textbf{ und } E = E'$$

GE$_\gamma$3 (Erfolg) Wenn der Konsistenzgrad von $\mathcal{F}$ vereinigt mit $\xi(s)$ mindestens den geforderten Wert γ erreicht, wird die Fuzzy-Formel, die dem Knotens s zugeordnet ist, um $\mathcal{F}$ γ-expandiert:

$$\textbf{Wenn } C(\xi(s) \cup \mathcal{F}) \geq \gamma, \textbf{ dann } \xi'(s) = \xi(s) +_\gamma \mathcal{F}$$

GE$_\gamma$4 (Mißerfolg) Wenn der Konsistenzgrad von $\mathcal{F}$ vereinigt mit $\xi(s)$ kleiner als γ ist, bleibt der Abhängigkeitsgraph unverändert:

$$\textbf{Wenn } C(\xi(s) \cup \mathcal{F}) < \gamma, \textbf{ dann } \langle \mathfrak{G}, s \rangle +_\gamma^{\mathfrak{G}} \mathcal{F} = \mathfrak{G}$$

GE$_\gamma$5 (Abhängigkeit) Es werden außer $\xi(s)$ nur Fuzzy-Formeln von Knoten geändert, die in der transitiven Abhängigkeitsmenge von s enthalten sind:

$$\forall v \in V' : \xi(v) \neq \xi'(v) \Rightarrow v \in \Gamma^*_{\mathfrak{G}}(s) \cup \{s\}$$

GE$_\gamma$6 (Konsistenz) Für alle Knoten, deren zugeordnete Fuzzy-Formel sich geändert hat, gilt: der Konsistenzgrad erreicht mindestens den Wert γ:

$$\forall v \in V' : \xi(v) \neq \xi'(v) \Rightarrow C(\xi'(v)) \geq \gamma$$

Dies sind unsere Basis-Postulate für eine Graph-Expansion. Das erste Postulat sichert, ähnlich wie im atomaren Fall, daß das Ergebnis einer Graph-Expansion wieder ein Abhängigkeitsgraph ist.

Das zweite Postulat legt fest, daß die Struktur des Graphen selbst bei einer Expansion nicht verändert werden darf. Es werden also weder Knoten hinzugefügt noch entfernt oder Abhängigkeiten verändert. Lediglich die mit den Knoten assoziierten Fuzzy-Formeln dürfen modifiziert werden.

Den Erfolgsfall definiert das dritte Postulat: Wenn die neue Information konsistent mit der Quelle im Graphen ist, wird sie über eine γ-Expansion hinzugefügt. Dies stellt eine untere Schranke für den Erfolgsfall dar, da wir hier nichts über Änderungen an anderen Fuzzy-Formeln aussagen; insbesondere also nichts über abhängige Formeln.

Die Expansion schlägt fehl, wenn die neue Information nicht konsistent mit der Quelle ist. In diesem Fall bleibt der Abhängigkeitsgraph unverändert (GE$_\gamma$4).

Das Postulat GE$_\gamma$5 legt fest, daß sich nur solche Fuzzy-Formeln ändern dürfen, bei denen eine transitive Abhängigkeit zu der Fuzzy-Formel der Quelle besteht.

Falls Änderungen an Fuzzy-Formeln außer der Quellformel stattfinden, müssen diese mindestens den geforderten Konsistenzgrad γ erreichen, was mit Postulat GE$_\gamma$6 festgehalten wird.

Behandlung der Abhängigkeiten

Offen blieb bis jetzt noch die Frage, wie mit Informationen umgegangen werden soll, bei denen eine Abhängigkeit von der Quelle besteht. Im Gegensatz zur atomaren Expansion gibt es hier aber, wie oben angedeutet, keine eindeutige Möglichkeit, eine solche Expansion durchzuführen.

Der erste Fall, den wir betrachten, bildet die Standardvariante der Graph-Expansion. Nach der Expansion der Quelle werden alle abhängigen Fuzzy-Formeln γ-revidiert, um deren Konsistenz mit der neu eingebrachten Information garantieren zu können. Dazu wird die neue Information nacheinander auf die jeweiligen Domänen der abhängigen Fuzzy-Formeln transformiert und auf der Fuzzy-Formel jedes Knotens eine γ-Revision angestoßen.

Die durch den Graph transformierte Formel $\mathcal{F}_v^\theta$ definieren wir wie folgt:

Definition 10.4.2 (Knotentransformation) *Gegeben sei eine Fuzzy-Formel $\mathcal{F} \in \mathfrak{F}$, ein Abhängigkeitsgraph mit Quelle $\langle \mathfrak{G}, s \rangle \in \langle \mathbb{G}, \mathbb{V} \rangle$ sowie die Menge aller Transformationsfunktionen für alle $u \in \Gamma^*_{\mathfrak{G}}(s) \cup \{s\}$ und alle $(u, v) \in \overrightarrow{\Phi}_{\mathfrak{G}}(u)$: $\Theta^{\mathcal{F}}_{\delta(\xi(u)),\delta(\xi(v))}$. Dann ist die Knotentransformation $\mathcal{F}_v^\theta$ von $\mathcal{F}$ für die Knoten $v \in \Gamma^*_{\mathfrak{G}}(s) \cup \{s\}$ definiert als:*

$$\mathcal{F}_v^\theta \stackrel{\text{def}}{=} \begin{cases} \mathcal{F} & \textbf{wenn } v = s \\ \bigcup\limits_{(u,v) \in \overleftarrow{\Phi}_{\mathfrak{G}}(v)} \Theta^{\mathcal{F}}_{\delta(\xi(u)),\delta(\xi(v))}(\mathcal{F}_u^\theta) & \textbf{sonst.} \end{cases}$$

Wenn ein Knoten mehr als einen Vorgänger hat, vereinigen wir die Transformationen der Vorgängerknoten, um die Knotentransformation dieses Knotens zu berechnen. Dies geschieht, da es bei der Transformation über verschiedene Domänen zu kleinen Abweichungen kommen kann, was durch die Bildung der Vereinigungsmenge kompensiert wird.

Beispiel (Knotentransformation) Ein Beispiel für eine Fuzzy-Formel, die durch einen Abhängigkeitsgraphen propagiert wird, zeigt Abbildung 10.5 auf der nächsten Seite. Dort soll eine Graph-Expansion an dem Quellknoten $s = v_1$ um die Fuzzy-Formel $\mathcal{F}$ durchgeführt werden. Der Beispielgraph zeigt für jeden Knoten die transformierte Fuzzy-Formel.

Standard Graph-Expansion Für diese erste Variante der Graph-Expansion können wir die Behandlung der Abhängigkeiten jetzt formal mit Hilfe des folgenden Zusatz-Postulates definieren; hierbei sei wieder $\langle \mathfrak{G}, s \rangle \in \langle \mathbb{G}, \mathbb{V} \rangle$ mit $\mathcal{F}, \xi(s) \in \mathfrak{F}(\Omega)$ und $\mathfrak{G}' = \langle \mathfrak{G}, s \rangle +^{\mathfrak{G}}_{\gamma} \mathcal{F} = (V', E', \xi')$:

GE$_\gamma$7 (Propagation) Alle von der Quelle abhängigen Fuzzy-Formeln werden γ-revidiert:

$$\forall v \in \Gamma^*_{\mathfrak{G}}(s) : \xi'(v) = \xi(v) \oplus^R_\gamma \mathcal{F}_v^\theta$$

Wir propagieren also nur die neue Information durch den Abhängigkeitsgraphen. Eine Alternative wäre, das Gesamtergebnis der Graph-Expansion an jedem Knoten durch den Graphen zu propagieren; dies würde jedoch für die meisten Anwendungsfälle eine unverhältnismäßig starke Restriktion jedes Knotens zur Folge haben. Überdies läßt sich eine solche Operation mit Hilfe der bisher definierten Mittel ausdrücken, indem das Ergebnis eines Knotens durch eine zusätzliche Graph-Expansion propagiert wird.

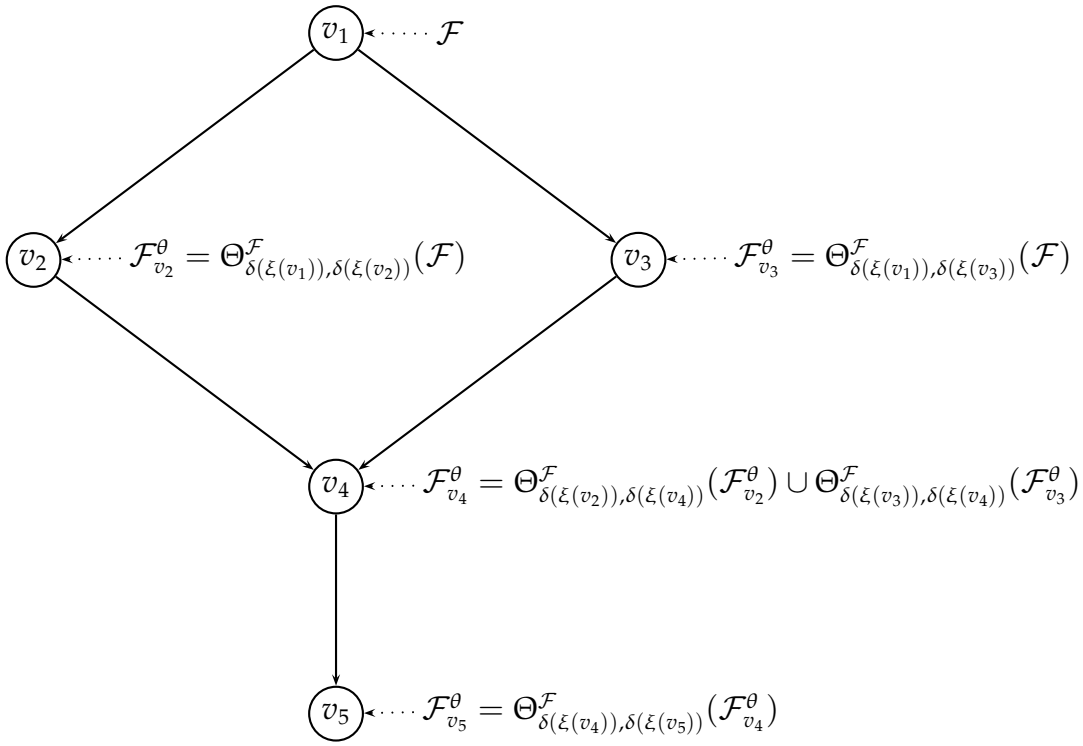


Abbildung 10.5: Propagation einer Fuzzy-Formel durch Knotentransformation

Strenge Graph-Expansion Die zweite Variante stellt eine streng monotone Graph-Expansion dar: hierbei werden alle von der Quelle abhängigen Knoten γ-expandiert; ist dies nicht möglich, schlägt die Graph-Expansion fehl und der Ausgangsgraph bleibt unverändert:

Definition 10.4.3 (Strenge Graph-γ-Expansion) *Eine strenge Graph-γ-Expansion $\tilde{+}^{\mathfrak{G}}_{\gamma}$ ist eine Abbildung, die einem Fuzzy-Abhängigkeitsgraphen mit Quelle $\langle \mathfrak{G}, s \rangle \in \langle \mathbb{G}, \mathbb{V} \rangle$, einem Wert $\gamma \in [0,1]$ und einer Fuzzy-Formel $\mathcal{F}$ mit $\mathcal{F}, \xi(s) \in \mathfrak{F}(\Omega)$ einen weiteren Abhängigkeitsgraphen zuordnet:*

$$\tilde{+}^{\mathfrak{G}}_{\gamma} : \langle \mathbb{G}, \mathbb{V} \rangle \times \mathfrak{F} \to \mathbb{G}$$

und die den Postulaten GE$_\gamma$1, GE$_\gamma$2, GE$_\gamma$5 und GE$_\gamma$6 (Seite 131) sowie GE$_\gamma$3', GE$_\gamma$4' und GE$_\gamma$7' (siehe unten) genügt.

Für die strenge Graph-Expansion gilt eine verschärfte Form der Postulate GE$_\gamma$3 und GE$_\gamma$4 sowie eine geänderte Fassung des Zusatz-Postulates GE$_\gamma$7; auch hier ist wieder $\langle \mathfrak{G}, s \rangle \in \langle \mathbb{G}, \mathbb{V} \rangle$ mit $\mathcal{F}, \xi(s) \in \mathfrak{F}(\Omega)$ und $\mathfrak{G}' = \langle \mathfrak{G}, s \rangle +^{\mathfrak{G}}_{\gamma} \mathcal{F} = (V', E', \xi')$:

GE$_\gamma$3' (Erfolg) Wenn der Konsistenzgrad von $\mathcal{F}^{\theta}_{v}$ vereinigt mit $\xi(v)$ für alle Knoten v in $\Gamma^{*}_{\mathfrak{G}}(s) \cup \{s\}$ mindestens den geforderten Wert γ erreicht, wird die Fuzzy-

Formel des Knotens s um $\mathcal{F}$ γ-expandiert:

$$\textbf{Wenn } \forall v \in \Gamma^*_{\mathfrak{G}}(s) \cup \{s\} : C(\xi(v) \cup \mathcal{F}^\theta_v) \geq \gamma, \textbf{dann } \xi'(s) = \xi(s) +_\gamma \mathcal{F}$$

GE$_\gamma$4′ (Mißerfolg) Wenn der Konsistenzgrad einer Formel, die einem Knoten aus $\Gamma^*_{\mathfrak{G}}(s) \cup \{s\}$ zugeordnet ist, vereinigt mit $\mathcal{F}^\theta_v$ kleiner als γ ist, bleibt der Abhängigkeitsgraph unverändert:

$$\textbf{Wenn } \exists v \in \Gamma^*_{\mathfrak{G}}(s) \cup \{s\} : C(\xi(s) \cup \mathcal{F}^\theta_v) < \gamma, \textbf{dann } \langle \mathfrak{G}, s\rangle \tilde{+}^{\mathfrak{G}}_\gamma \mathcal{F} = \mathfrak{G}$$

GE$_\gamma$7′ (Propagation) Alle von der Quelle abhängigen Fuzzy-Formeln werden γ-expandiert:

$$\forall v \in \Gamma^*_{\mathfrak{G}}(s) : \xi'(v) = \xi(v) +_\gamma \mathcal{F}^\theta_v$$

Die Postulate GE$_\gamma$3 und GE$_\gamma$4 werden also dahingehend erweitert, daß eine Expansion der durch den Graph propagierten, transformierten Information an jedem Knoten erfolgreich sein muß. Andernfalls schlägt die strenge Graph-Expansion fehl. Postulat GE$_\gamma$7′ legt fest, daß die abhängigen Knoten nicht revidiert, sondern expandiert werden.

Man beachte, daß es für eine Anwendung durchaus Sinn machen kann, einen anderen als einen der vordefinierten Graph-Expansionsoperatoren zu verwenden. Beispielsweise könnte auf allen abhängigen Fuzzy-Formeln eine Expansion durchgeführt werden, aber nur soweit, wie dies möglich ist. Im Gegensatz zu der strengen Graph-Expansion würde also bei dem ersten Knoten abgebrochen, bei dem die Expansion nicht mehr durchgeführt werden kann. Solche Alternativoperationen lassen sich nach dem gleichen Prinzip in dem hier vorgestellten Rahmen definieren, um sie dann bei der Anwendungsentwicklung zu implementieren und einzusetzen.

10.4.2 Operationen der Graph-Expansion

Die einzig mögliche Operation zur Standard-Graph-Expansion läßt sich direkt aus den Postulaten ablesen:

Definition 10.4.4 (Standard-Graph-γ-Expansionsoperation) *Gegeben sei ein Abhängigkeitsgraph mit Quelle $\langle \mathfrak{G}, s\rangle \in \langle \mathbb{G}, \mathbb{V}\rangle$ sowie $\mathcal{F}, \xi(s) \in \mathfrak{F}(\Omega)$. Die Standard-Graph-$\gamma$-Expansionsoperation ist dann folgendermaßen definiert:*

$$\langle \mathfrak{G}, s\rangle +^{\mathfrak{G}}_\gamma \mathcal{F} \stackrel{\text{def}}{=} \begin{cases} \mathfrak{G} & \textbf{wenn } C(\xi(s) \cup \mathcal{F}) < \gamma \\ (V, E, \xi') & \textbf{sonst} \end{cases}$$

mit

$$\xi' : \; V \to \mathfrak{F},$$
$$\xi'(v) \stackrel{\text{def}}{=} \begin{cases} \xi(v) +_\gamma \mathcal{F} & \textbf{wenn } v = s, \\ \xi(v) \oplus^R_\gamma \mathcal{F}^\theta_v & \textbf{wenn } v \in \Gamma^*_{\mathfrak{G}}(s) \\ \xi(v) & \textbf{sonst.} \end{cases}$$

Algorithmus 10.4.1 Standard-Graph-Expansion

```
1  STATUS ABHÄNGIGKEITSGRAPH.γ-EXPANSION( quelle s, formel F,
3                                         konsistenzgrad γ, ordnung o)
4  begin
5     // Ist die neue Information kompatibel mit der Fuzzy-Formel der Quelle?
6     status := this.knoten(s).formel().γ-expansion(F, γ);
7     if status = MISSERFOLG
8        then return(MISSERFOLG);
9     fi
10
12    // Erfolg, nun werden alle von der Quelle abhängigen Formeln γ-revidiert
13    // Die auf den Knoten v transformierte Information ist mit F_v^θ bezeichnet
14    foreach v ∈ this.knoten(s).transitiveAbhängigkeitsmenge() do
15       this.knoten(v).formel().γ-revision(F_v^θ, γ, o);
16    od
17
19    return(ERFOLG);
21 end
```

Analog können wir die Operation für den Fall der strengen Graph-Expansion definieren:

Definition 10.4.5 (Strenge Graph-γ-Expansionsoperation) *Gegeben sei ein Abhängigkeitsgraph $\mathfrak{G} = (V, E, \xi)$ mit Quelle $\langle \mathfrak{G}, s \rangle \in \langle \mathbb{G}, \mathbb{V} \rangle$ sowie eine Fuzzy-Formel $\mathcal{F}$ mit $\mathcal{F}, \xi(s) \in \mathfrak{F}(\Omega)$. Die strenge Graph-$\gamma$-Expansionsoperation ist folgendermaßen definiert:*

$$\langle \mathfrak{G}, s \rangle \tilde{+}_\gamma^{\mathfrak{G}} \mathcal{F} \stackrel{\text{def}}{=} \begin{cases} \mathfrak{G} & \textbf{wenn } \exists v \in \Gamma_{\mathfrak{G}}^*(s) \cup \{s\} : C(\xi(v) \cup \mathcal{F}_v^\theta) < \gamma \\ (V, E, \xi') & \textbf{sonst} \end{cases}$$

mit

$$\xi' : \; V \to \mathfrak{F},$$
$$\xi'(v) \stackrel{\text{def}}{=} \begin{cases} \xi(v) +_\gamma \mathcal{F}_v^\theta & \textbf{wenn } v \in \Gamma_{\mathfrak{G}}^*(s) \cup \{s\} \\ \xi(v) & \textbf{sonst.} \end{cases}$$

Auch hier läßt sich leicht sehen, daß diese Operation die einzig mögliche Definition ist, die alle Postulate erfüllt.

10.4.3 Algorithmen für die Graph-Expansion

Wir zeigen die Berechnung der Graph-Expansion ebenfalls in algorithmischer Form. Die Operation ist als Methode eines Abhängigkeitsgraphen dargestellt; die Berechnung der durch den Graph propagierten Information $\mathcal{F}_v^\theta$ (siehe Definition 10.4.2 auf

Algorithmus 10.4.2 Strenge Graph-Expansion

```
1  STATUS ABHÄNGIGKEITSGRAPH.STRENGE-γ-EXPANSION( quelle s, formel F,
3                                                  konsistenzgrad γ )
4  begin
5    // Da wir noch nicht wissen, ob alle Expansionen erfolgreich sein werden,
6    // führen wir alle Expansionsoperationen auf einer Kopie von G aus
7    G' := this.clone();
8
10   foreach v ∈ (G'.knoten(s).transitiveAbhängigkeitsmenge() ∪ {s}) do
11     status := G'.knoten(v).formel().γ-expansion(F_v^θ, γ);
12     if status = MISSERFOLG
13       then return(MISSERFOLG);
14     fi
15   od
16
18   // Alle Expansionsoperationen waren erfolgreich,
19   // damit ist auch die Graph-Expansion erfolgreich
20   this := G';
21   return(ERFOLG);
22 end
```

Seite 133) wird dabei nicht explizit gezeigt. Die Standardform der Graph-Expansion zeigt Algorithmus 10.4.1 auf der vorherigen Seite. Dabei gibt die Methode *formel()* die einem Knoten zugeordnete Fuzzy-Formel zurück.

Bei der strengen Graph-Expansion werden dagegen alle abhängigen Knoten γ-expandiert, sie ist nur erfolgreich, wenn diese Expansion an allen Knoten erfolgreich durchgeführt werden kann (Algorithmus 10.4.2). Man beachte, daß hierbei keine epistemische Relevanzordnung auf den Klauseln gegeben sein muß, da ausschließlich Expansionsoperationen durchgeführt werden.

10.5 Revision von Fuzzy-Abhängigkeitsgraphen

Nach der Expansion erweitern wir nun die Revision auf Fuzzy-Abhängigkeitsgraphen. Wir legen auch hier zuerst die Syntax für eine Graph-Revision fest:

Definition 10.5.1 (Graph-γ-Revision) *Eine Graph-γ-Revision $\oplus_\gamma^{\mathfrak{G}}$ ist eine Abbildung, die einem Fuzzy-Abhängigkeitsgraphen mit Quelle $\langle \mathfrak{G}, s\rangle \in \langle \mathbb{G}, \mathbb{V}\rangle$, einem Wert $\gamma \in [0,1]$ und einer Fuzzy-Formel $\mathcal{F}$ mit $\mathcal{F}, \xi(s) \in \mathfrak{F}(\Omega)$ einen weiteren Abhängigkeitsgraphen zuordnet:*

$$\oplus_\gamma^{\mathfrak{G}} : \langle \mathbb{G}, \mathbb{V}\rangle \times \mathfrak{F} \to \mathbb{G}$$

und die den Postulaten $GR_\gamma 1$–$GR_\gamma 7$ (siehe Seite 138) genügt.

10.5.1 Postulate der Graph-Revision

Wir definieren nun die Postulate, die ein Operator zur Graph-Revision erfüllen muß.

Für alle nachfolgenden Postulate gilt: $\mathfrak{G} = (V, E, \xi) \in \mathbb{G}$ ist ein Fuzzy-Abhängigkeitsgraph, $\mathcal{F}$ bezeichnet eine Fuzzy-Formel aus $\mathfrak{F}(\Omega)$, der Knoten $s \in V$ eine Quelle in $\mathfrak{G}$ mit $\xi(s) \in \mathfrak{F}(\Omega)$ und $\mathfrak{G}' = \langle \mathfrak{G}, s \rangle \oplus_\gamma^{\mathfrak{G}} \mathcal{F} = (V', E', \xi')$:

GR$_\gamma$1 (Stabilität) Das Ergebnis einer Graph-Revisionsoperation ist wieder ein Fuzzy-Abhängigkeitsgraph:

$$\langle \mathfrak{G}, s \rangle \oplus_\gamma^{\mathfrak{G}} \mathcal{F} \in \mathbb{G}$$

GR$_\gamma$2 (Graphinvarianz) Die Knotenmenge sowie die Struktur des Graphen bleiben erhalten:

$$V = V' \textbf{ und } E = E'$$

GR$_\gamma$3 (Erfolg) Wenn die Formel $\mathcal{F}$ mindestens den Konsistenzgrad γ erreicht, wird der Knoten s um $\mathcal{F}$ γ-revidiert:

$$\textbf{Wenn } C(\mathcal{F}) \geq \gamma, \textbf{dann } \xi'(s) = \xi(s) \oplus_\gamma^R \mathcal{F}$$

GR$_\gamma$4 (Mißerfolg) Wenn der Konsistenzgrad von $\mathcal{F}$ kleiner als γ ist, bleibt der Abhängigkeitsgraph unverändert:

$$\textbf{Wenn } C(\mathcal{F}) < \gamma, \textbf{dann } \langle \mathfrak{G}, s \rangle \oplus_\gamma^{\mathfrak{G}} \mathcal{F} = \mathfrak{G}$$

GR$_\gamma$5 (Abhängigkeit) Es werden außer der Fuzzy-Formel $\xi(s)$ nur Fuzzy-Formeln von Knoten geändert, die in der transitiven Abhängigkeitsmenge von s enthalten sind:

$$\forall v \in V' : \xi(v) \neq \xi'(v) \Rightarrow v \in \Gamma_{\mathfrak{G}}^*(s) \cup \{s\}$$

GR$_\gamma$6 (Konsistenz) Für alle Knoten, deren zugeordnete Fuzzy-Formel sich geändert hat, gilt: der Konsistenzgrad erreicht mindestens den Wert γ:

$$\forall v \in V' : \xi(v) \neq \xi'(v) \Rightarrow C(\xi'(v)) \geq \gamma$$

Postulat GR$_\gamma$1 ist die bekannte Stabilitätsforderung. Das Postulat GR$_\gamma$2 definiert die Eigenschaft der Graphinvarianz für die Graph-Revision. Eine Graph-Revision ist erfolgreich, wenn die neue Information mindestens den geforderten Konsistenzgrad γ erreicht (Postulat GR$_\gamma$3). Andernfalls ist die Graph-Revision erfolglos und der Ausgangsgraph bleibt unverändert (Postulat GR$_\gamma$4). Ändern dürfen sich bei einer Graph-Revision nur diejenigen Fuzzy-Formeln, die Knoten in der transitiven Abhängigkeitsmenge der Quelle zugeordnet sind, was Postulat GR$_\gamma$5 formalisiert. Jede geänderte Fuzzy-Formel muß dabei mindestens den geforderten Konsistenzgrad γ erreichen, was durch Postulat GR$_\gamma$6 ausgedrückt wird.

Dies sind wieder die Basis-Postulate, die für alle Graph-Revisionsoperationen Geltung haben sollen. Für die Behandlung der Abhängigkeiten bei der Graph-Revision legen wir den Standardfall mit folgendem Zusatzpostulat fest:

GR$_\gamma$7 (Propagation) Alle von der Quelle abhängigen Fuzzy-Formeln werden γ-revidiert:

$$\forall v \in \Gamma^*_{\mathfrak{G}}(s) : \xi'(v) = \xi(v) \oplus^R_\gamma \mathcal{F}^\theta_v$$

Man beachte, daß die Revision an einem Knoten v in $\Gamma^*_{\mathfrak{G}}(s)$ scheitern kann, wenn der Konsistenzgrad von $\mathcal{F}^\theta_v$ kleiner als γ ist. Damit ist jedoch kein Scheitern der Graph-Revision verbunden.

10.5.2 Operation zur Graph-Revision

Auch bei der Graph-Revision ist die einzig mögliche Operation eindeutig durch die Postulate bestimmt:

Definition 10.5.2 (Graph-γ-Revisionsoperation) *Sei $\langle \mathfrak{G}, s\rangle \in \langle \mathbb{G}, \mathbb{V}\rangle$ mit $\xi(s), \mathcal{F} \in \mathfrak{F}(\Omega)$ und $\mathfrak{G} = (V, E, \xi)$. Die Graph-$\gamma$-Revisionsoperation sei folgendermaßen definiert:*

$$\langle \mathfrak{G}, s\rangle \oplus^{\mathfrak{G}}_\gamma \mathcal{F} \stackrel{\text{def}}{=} \begin{cases} \mathfrak{G} & \textbf{wenn } C(\mathcal{F}) < \gamma \\ (V, E, \xi') & \textbf{sonst} \end{cases}$$

mit

$$\begin{aligned} \xi' :&\ V \to \mathfrak{F}, \\ &\xi'(v) \stackrel{\text{def}}{=} \begin{cases} \xi(v) \oplus^R_\gamma \mathcal{F}^\theta_v & \textbf{wenn } v \in \Gamma^*_{\mathfrak{G}}(s) \cup \{s\} \\ \xi(v) & \textbf{sonst.} \end{cases} \end{aligned}$$

Die Erfüllung der Postulate läßt sich, wie bei der Graph-Expansion, leicht anhand der Definition nachvollziehen.

Beispiel (Graph-Revision) Um zu zeigen, wie die entwickelten Formalismen der Abhängigkeiten, γ-Revision und Transformationsfunktionen bei einer Graph-Revision zusammenspielen, führen wir das Beispiel aus der Programmanalyse fort.

Analysiert wird wieder der Typ und Verwendungszweck einer Programmvariablen i. Beide Informationen sind unsicher und verändern sich im Laufe der Zeit durch weitere Analysen. Dabei besteht eine Abhängigkeit vom Typ zum Verwendungszweck, wie bereits in einem vorhergehenden Beispiel dargelegt wurde (siehe Abschnitt 8.5 auf Seite 85).

Hier interessiert nun vor allem die automatische Verarbeitung einer solchen Abhängigkeit. Für dieses Beispiel modellieren wir einen einfachen Fuzzy-Abhängigkeitsgraphen $\mathfrak{G} = (V, E, \xi) \in \mathbb{G}$ mittels

$$\begin{aligned} V &= \{v_1, v_2\}, \\ E &= \{(v_1, v_2)\}, \\ \xi &= \{[v_1, \mathcal{F}_1], [v_2, \mathcal{F}_2]\}. \end{aligned}$$

$\mathcal{F}_1 = \{\mathcal{K}_1\}$**: Typ von *i***

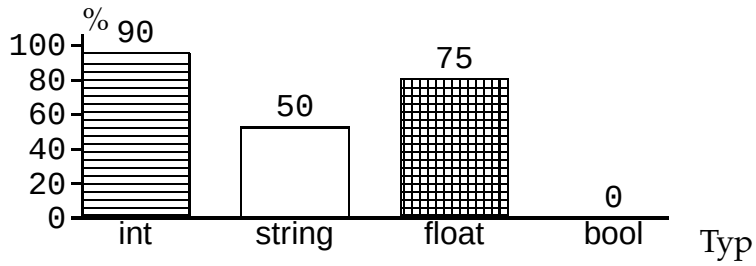
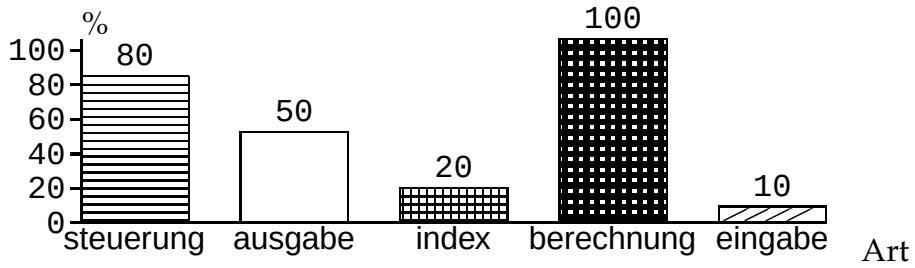
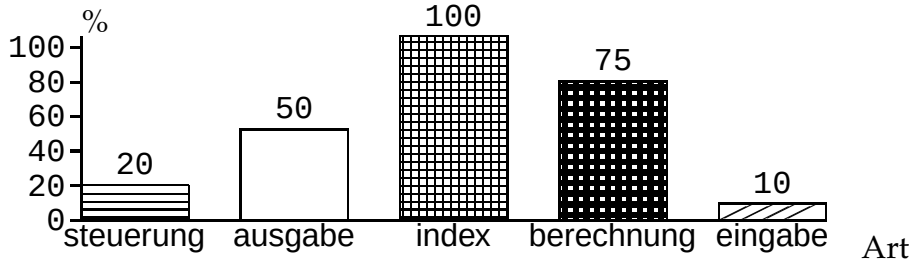


Abbildung 10.6: Fuzzy-Formel für den Typ von i

$\mathcal{K}_2$**: Verwendung von *i***



$\mathcal{K}_3$**: Verwendung von *i***



$\mathcal{F}_2 = \{\mathcal{K}_2, \mathcal{K}_3\}$**: Verwendung von *i***

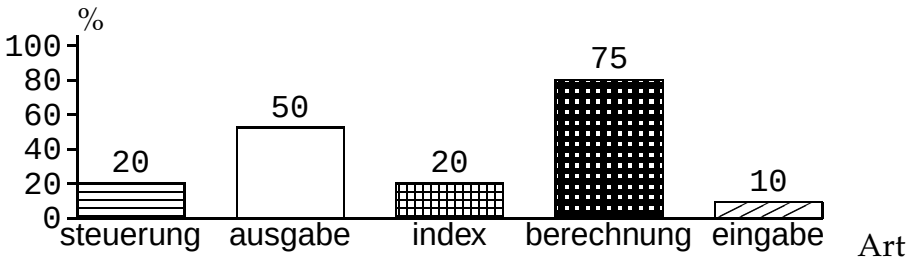
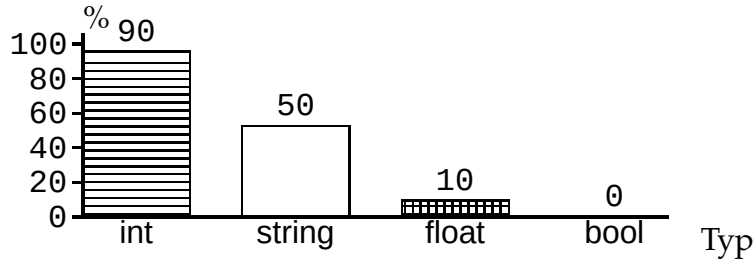


Abbildung 10.7: Fuzzy-Formel für die Verwendung von i

$\mathcal{F}_3 = \{\mathcal{K}_4\}$: **Typ von *i***



$(\mathcal{F}_3)^{\theta}_{v_2} = \{\mathcal{K}_5\}$: **Verwendung von *i***

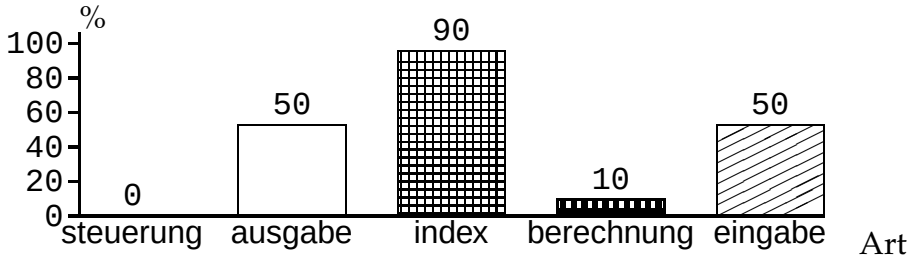


Abbildung 10.8: Neue Information $\mathcal{F}_3$ über den Typ von i und deren Transformation $(\mathcal{F}_3)^{\theta}_{v_2}$ auf Knoten v_2

Die vorhandenen Informationen über den Typ $\mathcal{F}_1 = \{\mathcal{K}_1\}$ und den Verwendungszweck $\mathcal{F}_2 = \{\mathcal{K}_2, \mathcal{K}_3\}$ sind in den Abbildungen 10.6 und 10.7 (Seite 140) gezeigt.

Erhalten wir nun im weiteren Verlauf der Programmanalyse neue Informationen über den Typ der Variablen i, sorgt das Verfahren der Graph-Revision dafür, daß die Fuzzy-Formel, die den Verwendungszweck beschreibt, entsprechend modifiziert wird. Die neue Information $\mathcal{F}_3 = \{\mathcal{K}_4\}$ ist in Abbildung 10.8 gezeigt: hier haben wir also erfahren, daß die Variable mit großer Sicherheit eine *int*-Variable ist. Wir fügen diese Information durch Graph-Revision hinzu, wobei wir einen Konsistenzgrad von 0,5 für das Ergebnis fordern:

$$\mathfrak{G}' = \langle \mathfrak{G}, v_1 \rangle \oplus^{\mathfrak{G}}_{0,5} \mathcal{F}_3.$$

Die beteiligten Fuzzy-Mengen ändern sich durch die Graph-Revision folgendermaßen:

$$\xi'(v_1) = \mathcal{F}_4 \stackrel{\text{def}}{=} \xi(v_1) \oplus_{0,5} \mathcal{F}_3,$$
$$\xi'(v_2) = \mathcal{F}_5 \stackrel{\text{def}}{=} \xi(v_2) \oplus_{0,5} (\mathcal{F}_3)^{\theta}_{v_2}.$$

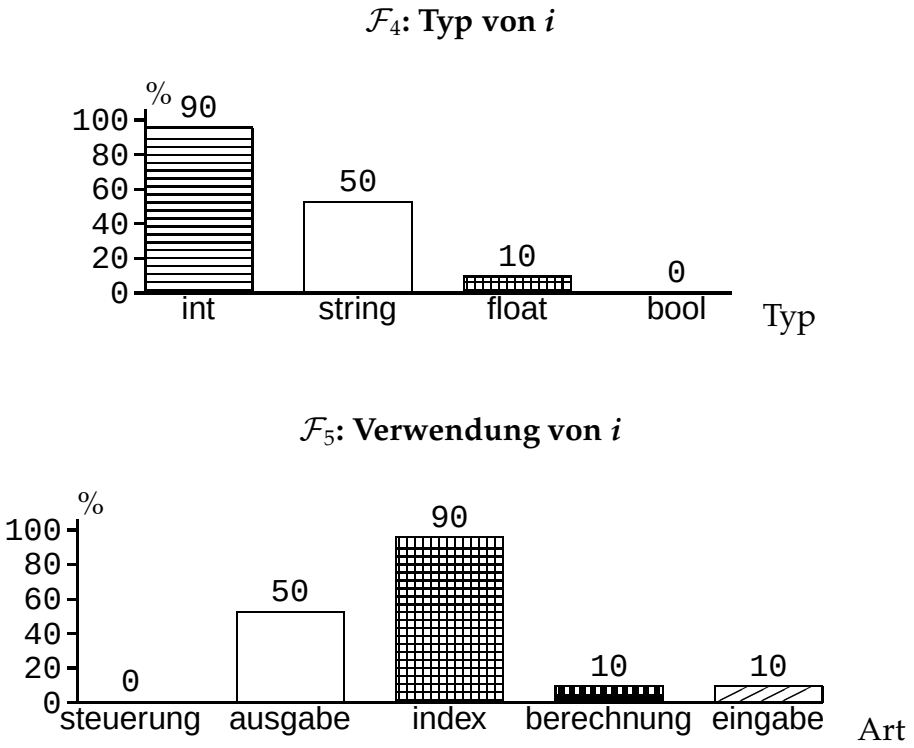


Abbildung 10.9: Revidierte Fuzzy-Formeln für den Typ ($\mathcal{F}_4$) und die Verwendung von i ($\mathcal{F}_5$)

Die auf den Knoten v_2 transformierte Fuzzy-Formel $(\mathcal{F}_3)^{\theta}_{v_2}$ ist ebenfalls in Abbildung 10.8 auf der vorherigen Seite gezeigt; die Umrechnung

$$(\mathcal{F}_3)^{\theta}_{v_2} = \Theta^{\mathcal{F}}_{\text{typ,art}}(\mathcal{F}_3)$$

erfolgte dabei mit der gleichen Transformationsfunktion wie in Beispiel 9.2 auf Seite 98 gezeigt.

Durch γ-Revision erhalten wir als Ergebnis die neuen Fuzzy-Formeln

$$\mathcal{F}_4 = \{\mathcal{K}_1, \mathcal{K}_4\},$$
$$\mathcal{F}_5 = \{\mathcal{K}_3, \mathcal{K}_5\}$$

wie sich leicht nachrechnen läßt (Abbildung 10.9).

Die Fuzzy-Klausel $\mathcal{K}_2$ wurde also durch die γ-Revision entfernt, als Folge hat sich der Sicherheitsgrad für die Verwendung als *index*-Variable entsprechend erhöht.

10.5.3 Algorithmus für die Graph-Revision

Der Algorithmus für die Graph-Revision ist dem der Standard-Graph-Expansion sehr ähnlich. Der wesentliche Unterschied liegt darin, daß lediglich die neue Infor-

mation den geforderten Konsistenzgrad erreichen muß: in diesem Fall werden die Quelle und alle von ihr abhängigen Knoten γ-revidiert (Algorithmus 10.5.1).

Algorithmus 10.5.1 Graph-Revision

```
1  STATUS ABHÄNGIGKEITSGRAPH.γ-REVISION( quelle s, formel F,
3                                        konsistenzgrad γ, ordnung o)
4  begin
5    // Hat die neue Information mindestens den Konsistenzgrad γ?
6    if F.konsistenzgrad() < γ
7      then return(MISSERFOLG);
8    fi
9
11   // Formel F erreicht den geforderten Konsistenzgrad, wir müssen
12   // nun die Quelle und alle von ihr abhängigen Knoten γ-revidieren
13   foreach v ∈ (this.knoten(s).transitiveAbhängigkeitsmenge() ∪ {s}) do
14     this.knoten(v).formel().γ-revision(F_v^θ, γ, o);
15   od
16
18   return(ERFOLG);
19 end
```

10.6 Kontraktion von Fuzzy-Abhängigkeitsgraphen

Als letzten Fall schließlich betrachten wir die Kontraktion von Fuzzy-Abhängigkeitsgraphen. Die Syntax der Graph-Kontraktion ist definiert durch:

Definition 10.6.1 (Graph-γ-Kontraktion) *Eine Graph-γ-Kontraktion $\ominus_\gamma^{\mathfrak{G}}$ ist eine Abbildung, die einem Fuzzy-Abhängigkeitsgraphen mit Quelle $\langle \mathfrak{G}, s\rangle \in \langle \mathbb{G}, \mathbb{V}\rangle$, einem Wert $\gamma \in [0,1]$ und einer Fuzzy-Formel $\mathcal{F}$ mit $\mathcal{F}, \xi(s) \in \mathfrak{F}(\Omega)$ einen weiteren Abhängigkeitsgraphen zuordnet:*

$$\ominus_\gamma^{\mathfrak{G}} : \langle \mathbb{G}, \mathbb{V}\rangle \times \mathfrak{F} \to \mathbb{G}$$

und die den Postulaten $GK_\gamma 1$–$GK_\gamma 7$ (Abschnitt 10.6.1) genügt.

10.6.1 Postulate der Graph-Kontraktion

Auch hier formulieren wir die notwendigen Eigenschaften der Graph-Kontraktion mit Hilfe von Postulaten.

Für alle nachfolgenden Postulate gilt: $\mathfrak{G} = (V, E, \xi) \in \mathbb{G}$ ist ein Fuzzy-Abhängigkeitsgraph, $\mathcal{F}$ bezeichnet eine Fuzzy-Formel aus $\mathfrak{F}(\Omega)$, der Knoten $s \in V$ eine Quelle in $\mathfrak{G}$ mit $\xi(s) \in \mathfrak{F}(\Omega)$ und $\mathfrak{G}' = \langle \mathfrak{G}, s\rangle \ominus_\gamma^{\mathfrak{G}} \mathcal{F} = (V', E', \xi')$:

GK$_\gamma$1 (Stabilität) Das Ergebnis einer Graph-Kontraktionsoperation ist wieder ein Fuzzy-Abhängigkeitsgraph:

$$\langle \mathfrak{G}, s \rangle \ominus_\gamma^{\mathfrak{G}} \mathcal{F} \in \mathbb{G}$$

GK$_\gamma$2 (Graphinvarianz) Die Knotenmenge sowie die Struktur des Graphen bleiben erhalten:

$$V = V' \textbf{ und } E = E'$$

GK$_\gamma$3 (Erfolg) Wenn die Formel $\overline{\mathcal{F}}$ mindestens den Konsistenzgrad γ erreicht, wird eine γ-Kontraktion des Knotens s um $\mathcal{F}$ durchgeführt:

$$\textbf{Wenn } C(\overline{\mathcal{F}}) \geq \gamma, \textbf{dann } \xi'(s) = \xi(s) \ominus_\gamma \mathcal{F}$$

GK$_\gamma$4 (Mißerfolg) Wenn der Konsistenzgrad von $\overline{\mathcal{F}}$ kleiner als γ ist, bleibt der Abhängigkeitsgraph unverändert:

$$\textbf{Wenn } C(\overline{\mathcal{F}}) < \gamma, \textbf{dann } \langle \mathfrak{G}, s \rangle \ominus_\gamma \mathcal{F} = \mathfrak{G}$$

GK$_\gamma$5 (Abhängigkeit) Es werden außer s nur Fuzzy-Formeln von Knoten geändert, die in der transitiven Abhängigkeitsmenge von s enthalten sind:

$$\forall v \in V' : \xi(v) \neq \xi'(v) = v \in \Gamma_{\mathfrak{G}}^*(s) \cup \{s\}$$

GK$_\gamma$6 (Konsistenz) Für alle Knoten, deren zugeordnete Fuzzy-Formel sich geändert hat, gilt: der Konsistenzgrad erreicht mindestens den Wert γ:

$$\forall v \in V' : \xi(v) \neq \xi'(v) \Rightarrow C(\xi'(v)) \geq \gamma$$

Dies sind wieder die Basis-Postulate, die für alle Graph-Kontraktionsoperationen Geltung haben sollen. Den Standardfall legen wir mit folgendem Zusatzpostulat fest:

GK$_\gamma$7 (Propagation) Alle von der Quelle abhängigen Fuzzy-Formeln werden γ-kontrahiert:

$$\forall v \in \Gamma_{\mathfrak{G}}^*(s) : \xi'(v) = \xi(v) \ominus_\gamma \mathcal{F}_v^\theta$$

10.6.2 Operation zur Graph-Kontraktion

Die entsprechende Operation zur Graph-Kontraktion läßt sich wieder direkt aus den Postulaten ableiten:

Definition 10.6.2 (Graph-γ-Kontraktionsoperation) *Gegeben sei ein Abhängigkeitsgraph mit Quelle $\langle \mathfrak{G}, s\rangle \in \langle \mathbb{G}, \mathbb{V}\rangle$ mit $\xi(s), \mathcal{F} \in \mathfrak{F}(\Omega)$ und $\mathfrak{G} = (V, E, \xi)$. Die Graph-$\gamma$-Kontraktionsoperation ist dann folgendermaßen definiert:*

$$\langle \mathfrak{G}, s\rangle \ominus_{\gamma}^{\mathfrak{G}} \mathcal{F} \stackrel{\text{def}}{=} \begin{cases} \mathfrak{G} & \textbf{wenn } C(\overline{\mathcal{F}}) < \gamma \\ (V, E, \xi') & \textbf{sonst} \end{cases}$$

mit

$$\begin{aligned} \xi' : \; & V \to \mathfrak{F}, \\ & \xi'(v) \stackrel{\text{def}}{=} \begin{cases} \xi(v) \ominus_{\gamma} \mathcal{F}_v^{\theta} & \textbf{wenn } v \in \Gamma_{\mathfrak{G}}^{*}(s) \cup \{s\} \\ \xi(v) & \textbf{sonst.} \end{cases} \end{aligned}$$

Auch hier läßt sich die Erfüllung der Postulate leicht anhand der Definition nachvollziehen.

10.6.3 Algorithmus für die Graph-Kontraktion

Die Graph-Kontraktion kann analog zur Graph-Revision algorithmisiert werden; wir müssen lediglich überprüfen, ob die Negation der neuen Information den geforderten Konsistenzgrad erreicht und dann eine γ-Kontraktion auf der Quelle sowie aller von ihr abhängigen Knoten ausführen.

Der fertige Algorithmus 10.6.1 ist auf der nächsten Seite zu sehen.

Dies beendet Teil III dieser Arbeit. Im nächsten Teil legen wir die technischen Grundlagen für die Entwicklung von Fuzzy-Informationssystemen (siehe Abbildung 10.10 auf der nächsten Seite).

Algorithmus 10.6.1 Graph-Kontraktion

```
1  STATUS ABHÄNGIGKEITSGRAPH.γ-KONTRAKTION( quelle s, formel F,
3                                            konsistenzgrad γ, ordnung o)
4  begin
5     // Berechne die Negation von F
6     F.negiere_formel();
7     // Erreicht die Negation von F mindestens den Konsistenzgrad γ?
8     if F.konsistenzgrad() < γ
9        then return(MISSERFOLG);
10    fi
11
13    // Die Negation von F erreicht den geforderten Konsistenzgrad, wir müssen
14    // nun die Quelle und alle von ihr abhängigen Knoten γ-kontrahieren
15    foreach v ∈ (this.knoten(s).transitiveAbhängigkeitsmenge() ∪ {s}) do
16       this.knoten(v).formel().γ-kontraktion(F_v^θ, γ, o);
17    od
18
20    return(ERFOLG);
21 end
```

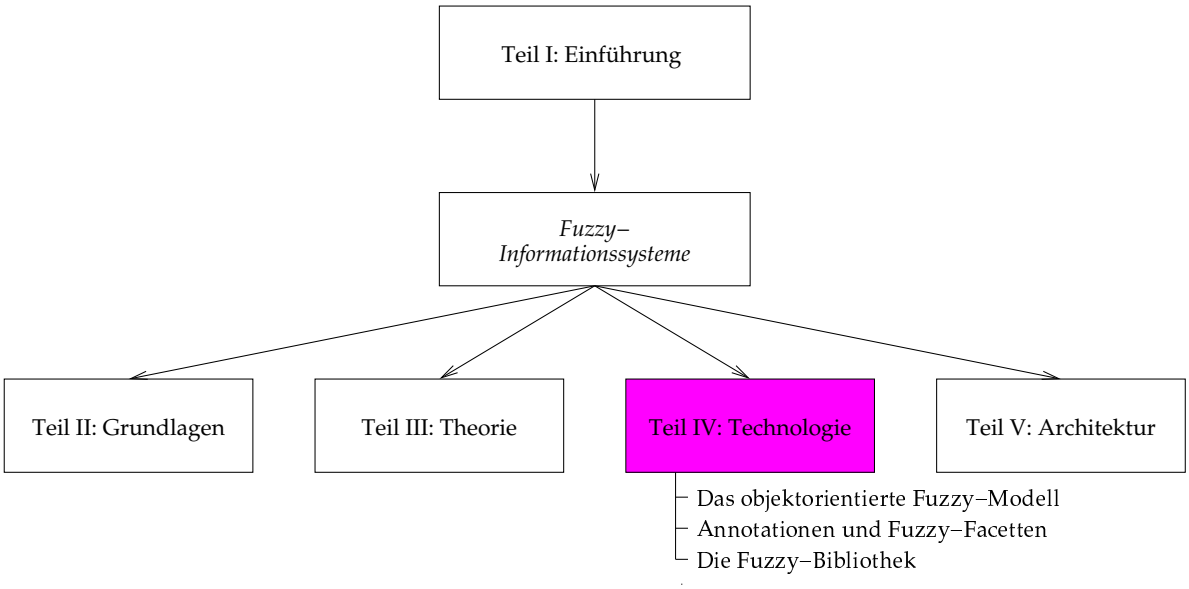


Abbildung 10.10: Gliederung Teil IV

Teil IV

Technologie von Fuzzy-Informationssystemen

Kapitel 11

Das objektorientierte Fuzzy-Modell

Fuzzy theory is wrong, wrong, and pernicious.
What we need is more logical thinking, not less.
The danger of fuzzy logic is that it will encourage the sort
of imprecise thinking that has brought us so much trouble.
Fuzzy logic is the cocaine of science.

Prof. William Kahan, University of California at Berkeley

Im letzten Teil dieser Arbeit haben wir unser theoretisches Fuzzy-Modell beschrieben. Der nächste Schritt zur Entwicklung von Fuzzy-Informationssystemen ist die Einbettung dieses Modells in objektorientierte Umgebungen, wie von der Anforderung OBJEKTORIENTIERTES SYSTEM (Seite 32) gefordert.

Dazu betrachten wir zunächst verschiedene Möglichkeiten einer solchen Einbettung und leiten anschließend unsere eigenen Erweiterungen ab. Um das objektorientierte Fuzzy-Modell gefahrlos einsetzen zu können, erfolgt danach seine Formalisierung.

11.1 Objektorientierung und Fuzzy-Theorie

Bevor wir unser theoretisches Fuzzy-Modell in ein objektorientiertes Datenmodell einbetten können, muß zunächst präziser definiert werden, was mit einem „objektorientierten Fuzzy-Datenmodell“ gemeint ist. Denn wie bereits bei der Untersuchung der vorhandenen Ansätze in Kapitel 2 aufgezeigt wurde, verfolgen verschiedene Autoren völlig unterschiedliche Vorgehensweisen, um objektorientierte Technologie und Fuzzy-Theorie miteinander zu verbinden.

Um dieses Gebiet zu systematisieren, kategorisieren wir zunächst die verschiedenen existierenden Ansätze, indem wir von den konkret geleisteten Arbeiten abstrahieren, und so die oft nur implizit genannte Vorgehensweise offenlegen. Damit kommen wir

zu sechs verschiedenen Ausprägungen objektorientierter Fuzzy-Modelle, unter denen auch Mischformen möglich sind. Mit Hilfe dieser Kategorisierung wird es möglich, den in dieser Arbeit verfolgten Ansatz präzise zu positionieren.

Unsere Kategorisierung umfaßt im einzelnen:

Formale Datenmodelle Hier steht die Erweiterung eines formalen Datenmodells um Fuzzy-Aspekte im Vordergrund. Als großes Hindernis für die Forschung hat sich erwiesen, daß es im Gegensatz zum relationalen Datenmodell kein einheitliches formales objektorientiertes Modell gibt. Als Folge weisen die verschiedenen Arbeiten eine sehr große Bandbreite unterschiedlicher Schwerpunkte und Eigenschaften auf.

Objektorientiertes Datenbankmanagementsystem Die Herangehensweise ist hier die Erweiterung eines objektorientierten Datenbanksystems mit dem Ziel, die Speicherung und den Zugriff auf Fuzzy-Objekte zu ermöglichen. Dabei werden unterschiedliche Vorgehensweisen verfolgt, die sich in den folgenden Unterausprägungen widerspiegeln:

- Die Erweiterung des Datenbankmanagementsystems selbst, um mit „Fuzzy-Mengen“ als neuen Datentyp umgehen zu können;
- Die Betrachtung von Fuzzy-Anfragen auf scharfen oder fuzzy Daten;
- Die Definition eines Datenbankschemas, welches Fuzzy-Mengen als eigenständige Klasse beinhaltet, um dieses dann in einem konkreten Anwendungsfall einsetzen zu können; sowie
- Die Erweiterung des dem Datenbankmanagementsystem zugrundeliegenden Objektmodells um Fuzzy-Konzepte.

Konzeptionelle Datenmodelle Bei den konzeptionellen Ansätzen wird eine Modellierungssprache um Fuzzy-Aspekte erweitert. Typischerweise beschränken sich solche Arbeiten darauf, eine graphische Notation zu entwerfen oder zu erweitern. Motiviert wird dies meist mittels einfacher Anwendungsbeispiele („Joe spricht gut Englisch, aber nur schlecht Französisch“).

Objektorientierte Programmiersprachen Hier steht die Definition einer neuen oder die Erweiterung einer bestehenden objektorientierten Programmiersprache im Vordergrund. Bei der Erweiterung kann man wiederum zwei Möglichkeiten unterscheiden:

- Die Erweiterung einer Sprache um neue Ausdrucksmittel zum Umgang mit Fuzzy-Mengen, was durch eine Änderung der die Sprache definierenden Grammatik erreicht werden kann.
- Das Aufsetzen von Fuzzy-Konzepten mit Hilfe von Erweiterungsmöglichkeiten, die durch die Sprache angeboten werden.

Im ersten Fall wird also eine neue Programmiersprache mit eigener Grammatik definiert, die eine mehr oder weniger große Ähnlichkeit mit einer existierenden Sprache hat, wogegen im zweiten Fall eine existierende Sprache unangetastet bleibt und Fuzzy-Konzepte durch vorgesehene Erweiterungsmöglichkeiten eingebracht werden.

Objektorientierte Konzepte Manche Autoren vermeiden es von vorneherein, auf konkrete Technologien Bezug zu nehmen, und sprechen stattdessen allgemein von der Erweiterung von Konzepten wie dem der Objekte, Attribute, Klassen oder der objektorientierten Modellierung um Fuzzy-Aspekte. Oft werden diese konzeptionellen Ideen mit einfachen Beispielen oder ad-hoc Notationen illustriert.

Objektorientierte Anwendungen Wenn konkrete Anwendungsfälle betrachtet werden, kommen dabei auch objektorientierte Technologien zum Einsatz. Dies wird dann allgemein mit Begriffen wie „Eine objektorientierte Anwendung zur ...“ beschrieben, wobei die entwickelten Konzepte typischerwise so anwendungsspezifisch sind, daß sie sich nicht auf andere Anwendungsbereiche übertragen lassen.

Nach unserer Auffassung muß ein Ansatz, der Objektorientierung und Fuzzy-Theorie verbinden will, seine Position präzise darlegen. Tatsächlich versäumen es aber die meisten Autoren von objektorientierten Fuzzy-Datenmodellen, ihre Sichtweise zu problematisieren oder sie auch nur explizit zu machen. Die angesprochene Arbeit von Boss [Bos96] beispielsweise beginnt mit der Erweiterung des formalen Datenmodells des objektorientierten Datenbanksystems O_2, verwendet für die praktische Umsetzung dagegen das auf einem anderen Datenmodell basierende Datenbanksystem OBST und setzt für die Implementierungsbeispiele schließlich die Programmiersprache C++ ein, die auf einem wiederum anderen Datenmodell beruht — und das, ohne auf diese Brüche explizit hinzuweisen.

Insbesondere wenn mehrere Ebenen vermischt betrachtet, implizite Voraussetzungen über Einsatzzweck oder Entwurfseinschränkungen gemacht werden oder es versäumt wird, die betrachtete Ebene explizit zu nennen, ist eine klare Vermittlung der Möglichkeiten und Ziele eines solchen Ansatzes verfehlt worden.

Für die Entwicklung unseres objektorientierten Fuzzy-Datenmodells gehen wir daher folgendermaßen vor: zunächst rekapitulieren wir kurz die wichtigsten Eigenschaften objektorientierter Modelle zusammen mit deren bekannten Fuzzy-Erweiterungen. Diese diskutieren wir in Hinblick auf die in Kapitel 3 gestellten Anforderungen. Aus den gezeigten Möglichkeiten und den aufgestellten Anforderungen bilden wir die Synthese in Form unseres eigenen objektorientierten Fuzzy-Datenmodells. Dieses beschreiben wir sowohl anschaulich als auch mit Hilfe eines formalen Datenmodells. Die nachfolgenden Kapitel widmen sich dann der technischen Umsetzung der entwickelten Konzepte.

11.1.1 Objektorientierte Konzepte und bekannte Erweiterungen

Um diese Fuzzyfizierung systematisch angehen zu können, beschreiben wir im folgenden zunächst die wichtigsten objektorientierten Konzepte (angelehnt an [Bud97]) und deren mögliche Fuzzy-Varianten, wie sie sich in der Literatur finden.

Klassen und Objekte

Die Grundidee der Objektorientierung ist, alles als ein *Objekt* aufzufassen. Objekte besitzen einen *Zustand*, der technisch über eine Menge von Variablen, den *Objektvariablen* oder *Attributen* realisiert wird. Der Objektzustand heißt *gekapselt*, da er von anderen Objekten nicht eingesehen oder geändert werden darf. Berechnungen werden in den *Methoden* eines Objektes durchgeführt; diese können auf Instanzvariablen zugreifen, um sie zu lesen oder zu ändern. Der Aufruf einer Methode wird auch als die Übermittlung einer *Nachricht* an ein Objekt bezeichnet.

Jedes Objekt gehört zu einer *Klasse*, es ist *Instanz* dieser Klasse. Die Klasse definiert den Zustandsraum und das Verhalten, also die Attribute und Methoden ihrer Instanzen, während das einzelne Objekt eine konkrete Ausprägung des Zustandsraumes, also eine zeitlich veränderliche Belegung der Attribute mit Werten repräsentiert. Jedes Objekt kann unabhängig von seinem Zustand über einen eindeutigen und zeitlich unveränderlichen *Bezeichner* (object identifier, oid) referenziert werden.

Fuzzyfizierung Der Begriff „Fuzzy-Objekt“ wird in vielen verschiedenen Bedeutungen verwendet. Im einfachsten Fall handelt es sich dabei um ein Objekt, das Fuzzy-Informationen in irgendeiner Form bereithält — beispielsweise über ein Attribut vom Typ „Fuzzy-Menge“. Einige Ansätze verwenden die Idee einer Fuzzy-Instanzbeziehung, bei der ein Objekt eine graduelle Zugehörigkeit zu einer Klasse aufweist.

Unter einer „Fuzzy-Klasse“ wird entweder die schon erwähnte Fuzzy-Instanzbeziehung verstanden, oder ein konzeptionelles Modell, das mit Unsicherheit behaftet ist. Im zweiten Fall bleiben die vorgestellten Ideen meist sehr vage, wie die Idee der „Hypothetischen Modellierung“ im UFO-Modell (vergleiche Abschnitt 2.2.3 auf Seite 15), da nicht auf konkrete Anwendungen oder Implementierungsmöglichkeiten eingegangen wird.

„Fuzzy-Methoden“ als solche werden in der Literatur nicht erwähnt, da der Fokus meist auf Datenmodellen objektorientierter Datenbanksysteme liegt. Dabei wird implizit davon ausgegangen, daß die Methoden mit den Erweiterungen umgehen können, also eine Verarbeitung der imperfekten Informationen möglich ist. Eine konkrete Unterstützung der VERARBEITUNG UNSCHARFER DATEN (vergleiche Anforderung 3.2.6 auf Seite 33) wird aber nicht angeboten.

Attribute

Attribute dienen zur Speicherung des Zustandes eines Objektes. Sie haben einen bestimmten *Typ* als Wertebereich und können so entweder Referenzen auf Objekte einer bestimmten Klasse oder Elemente eines primitiven Datentyps aufnehmen. Primitive Datentypen lassen sich weiterhin in *einfachwertige* und *mehrwertige* Typen unterscheiden.

Fuzzyfizierung Attribute als der Datenspeicher von Objekten bieten die erste Angriffsstelle für die Einbringung von Fuzzy-Techniken. Sie dienen typischerweise dazu, Zustände abstrakter oder realer Konzepte aufzunehmen — erweitert man sie um Fuzzy-Aspekte, lassen sich auch imperfekte Zustände dieser Konzepte ausdrücken. Beispielsweise kann ein Objekt dazu dienen, eine Person zu repräsentieren. Fuzzy-Attribute dieses Objektes können dann imperfekte Zustände etwa des Alters oder der Körpergröße aufnehmen.

Die Grundidee fast aller objektorientierten Fuzzy-Modelle ist daher, entweder einen neuen primitiven Typ oder eine neue Klasse „Fuzzy-Menge“ einzuführen. Abhängig vom Fokus einer Arbeit werden anschließend meist verschiedene Auswirkungen diskutiert, die beispielsweise die Klassenhierarchie, Vorbelegungen oder den Wertebereich von Attributen betreffen.

Vererbung

Zwischen Klassen besteht eine *Vererbungshierarchie*, dabei *erbt* eine Klasse Attribute und Methoden von einer *(Einfachvererbung)* oder mehreren *(Mehrfachvererbung)* Oberklassen.

Zusätzlich ermöglicht es das Konzept der *Substituierbarkeit*, ein Objekt überall dort zu verwenden, wo ein Objekt seiner Oberklasse erwartet wird.

Fuzzyfizierung In der Literatur werden Fuzzy-Vererbungsbeziehungen typischerweise als graduelle Übergänge der *„ist-ein“* („is-a“)-Beziehung verstanden. So wird es möglich, daß eine Klasse nur zu einem bestimmten Grad ihrer Oberklasse angehört. Beispielsweise könnte es eine Klasse „Trike“ geben, die mit ihren Oberklassen „Auto“ und „Motorrad“ nur zu einem bestimmten Grad kompatibel ist. Typischerweise beschäftigen sich solche Ansätze dann mit den Auswirkungen auf die Attribute bei unscharfer Vererbung.

In solchen Modellen gehören Objekte auch nicht mehr eindeutig zu einer Klasse, sondern nur noch zu einem bestimmten Grad. Manche Ansätze schlagen dabei vor, diesen Zugehörigkeitsgrad über eine Vorbelegung zu berechnen, dabei wird die graduelle Vererbung („ein Trike *ist-ein* Auto mit Grad 0,6 und *ist-ein* Motorrad mit Grad

0,8“) direkt auf die Objektinstanzen dieser Klassen übertragen; dies überführt eine fuzzy „ist-ein“-Vererbungsbeziehung direkt auf eine fuzzy „instanz-von“ („instance-of“) Klassenzugehörigkeit.

11.1.2 Diskussion der Fuzzyfizierungen

An Vorschlägen zu objektorientierten Fuzzy-Modellen existiert offensichtlich kein Mangel. Bevor wir die existierenden Ansätze kritisch untersuchen und ein eigenes Modell aufstellen können, müssen wir daher die Anforderungen präzisieren. Dies ist ein Punkt, der in der Literatur typischerweise vernachlässigt wird, denn wie wir schon bei der Kritik am UFO-Modell (siehe Abschnitt 2.2.3 auf Seite 15) anklingen ließen, kann eine sinnfreie Erweiterung nur um der Fuzzifizierung selbst willen nicht Ziel eines solchen Modells sein.

Unser Hauptkritikpunkt richtet sich an dieser Stelle nicht gegen konkret vorgenommene Fuzzy-Erweiterungen, sondern vor allem gegen die Vorgehensweise, die fast durchweg auf klare Anforderungsanalysen und Zielsetzungen verzichtet. Dies wird zwar stellenweise in der Literatur erkannt [CCV97]:

> *These proposals, for the most part, focus on introducing fuzzy set theory for one particular concept and/or one kind of uncertainty and are not consistent or systematic in carrying out the extensions throughout the selected object-oriented model.*

blieb aber bisher ohne Konsequenzen. Als Folge bleibt der Nutzen solcher Modelle meist unklar und praktische Einsatzmöglichkeiten diffus. Gefragt ist also eine systematische Vorgehensweise bei der Entwicklung eines Fuzzy-Datenmodells.

Unsere in Kapitel 3 aufgestellten Anforderungen geben einen klaren Rahmen für die an dieser Stelle durchzuführenden Arbeiten vor. Ein objektorientiertes Fuzzy-Modell muß:

Anwendungsrelevant sein, das heißt die vorgeschlagenen Konzepte müssen einen klaren Nutzwert für die Entwicklung von Fuzzy-Informationssystemen haben (vergleiche Anforderungen OBJEKTORIENTIERTES SYSTEM auf Seite 32 und PARTIELLE UNSCHÄRFE auf Seite 30); sowie

Implementierbar sein, also als orthogonale Erweiterung existierender Standards und Technologien durchgeführt werden können (vergleiche Anforderungen MODERNE ARCHITEKTUR auf Seite 27 und INTEROPERABILITÄT auf Seite 34).

Es muß nun eine Auswahl hinsichtlich Art und Umfang der Fuzzifizierungen getroffen werden, die den aufgestellten Anforderungen genügt. Wir betrachten die verschiedenen Konzepte im einzelnen.

Fuzzy-Attribute Wie oben ausgeführt, wird für Attribute typischerweise ein neuer Datentyp „Fuzzy-Menge“ eingeführt; Objekte mit Attributen dieses Typs sind dann in der Lage, einen imperfekten Zustand zu repräsentieren.

Wir schließen uns diesen Ansätzen nicht an. Attribute sollen auch in einem objektorientierten Fuzzy-Datenmodell nur die vom Typsystem angebotenen, scharfen Datentypen annehmen. Dies begründet sich vor allem mit der Anforderung INTEROPERABILITÄT (Abschnitt 3.2.7 auf Seite 34): für Systemteile, die nicht mit imperfekten Informationen umgehen können, muß es weiterhin eine scharfe Sicht auf ein Gesamtsystem geben.

Zwar bieten auch wir einen neuen Fuzzy-Datentyp an: unsere Fuzzy Konjunktive Normalform (siehe Definition 8.1.7 auf Seite 77), da einfache Fuzzy-Mengen, wie in Kapitel 8 diskutiert, nicht ausdrucksstark genug sind und keine Entkopplung von Repräsentation und Interpretation ermöglichen (vergleiche Abschnitt 8.4 auf Seite 82). Um nun diesen Typ einsetzen, aber trotzdem die Anforderung nach der INTEROPERABILITÄT erfüllen zu können, verwenden wir das Konzept der *Annotation mit Facetten*, das verschiedene Sichtweisen auf den Wert eines Attributes ermöglicht, ähnlich wie in Frame-basierten Datenmodellen. Jedem Attribut kann dabei eine Annotation zugeordnet werden. In einer solchen Annotation können dann eine oder mehrere Facetten angelegt werden, die abhängig von ihrem Typ verschiedene Zusatzinformationen aufnehmen. Beispielsweise könnte in einem Personen-Objekt das Attribut „Größe“ annotiert werden, um in einer Fuzzy-Facette mit Hilfe einer Fuzzy Konjunktiven Normalform imperfekte Informationen über die Körpergröße zu speichern. Wir werden das Konzept der Annotation mit Facetten in Abschnitt 11.2.1 genauer diskutieren.

Zwischen den Attributen eines oder mehrerer Objekte kann es Abhängigkeiten geben (vergleiche Abschnitt 8.5 auf Seite 84), die bei einer Zustandsänderung berücksichtigt werden müssen. Wir haben bereits in Zusammenhand mit den theoretischen Modell argumentiert, daß diese explizit repräsentiert werden müssen, damit sie nicht in einem anwendungsspezifischen Format abgelegt werden. Trotzdem wurden in der Literatur hierzu keine ähnlichen Arbeiten gefunden. Wir werden in Abschnitt 11.2 zeigen, wie das in dieser Arbeit entwickelte Konzept der Abhängigkeitsgraphen mit Objekt-Attributen verknüpft werden kann.

Die Verwaltung von imperfekten Informationen mit Hilfe von Facetten zu Einzelattributen ist überdies nicht ausreichend für alle Modellierungsanforderungen, da mit ihnen keine Annotation von Beziehungen möglich ist. Wir erlauben daher auch das Anlegen von Facetten zu Gruppen von Attributen, sogenannten Attributmengen. Diesen Aspekt werden wir in Abschnitt 11.2.3 ausführen.

Objekte Wie oben erläutert, muß der Begriff „Fuzzy-Objekt“ weiter präzisiert werden. Sofern hierbei von einem Objekt mit imperfektem Zustand die Rede ist, enthält

unser Modell durch die Aufnahme von Attributen mit Fuzzy-Facetten bereits Fuzzy-Objekte.

Auf den Aspekt der graduellen Zugehörigkeit zu einer Klasse werden wir weiter unten im Zusammenhang mit Klassen und Vererbung eingehen.

Eine neue Erweiterung ist die Möglichkeit, ein Objekt als ganzes mit imperfekten Zusatzinformationen zu versehen. Damit wird es unter anderem möglich, den Reifegrad eines Objektes an sich zu verwalten, wie wir in Abschnitt 11.2.4 noch genauer ausführen werden.

Klassen und Vererbung Wir verzichten in unserem Modell bewußt auf Fuzzy-Erweiterungen, die die Klassenhierarchie oder die Schemaebene betreffen. Dafür gibt es zwei wesentliche Gründe: Erstens läßt sich eine solche Änderung nicht mit Hilfe existierender Laufzeitumgebungen abbilden. Kompatibilität ist jedoch eine unserer zentralen Anforderungen (vergleiche Abschnitt 3.2.1 auf Seite 27 und dort [MS97]: *„This dictates that capabilities for managing uncertainty and imprecision should be offered as strict extensions of existing standards."*).

Der zweite Grund ist konzeptioneller Natur. Wie wir zuvor kritisiert haben, berücksichtigen die Autoren vorhandener Ansätze in den allermeisten Fällen nie den *Einsatz* eines solches Fuzzy-Datenmodells. Die Grundidee bei Fuzzy-Datenbanken und -Informationssystemen ist aber, eine bessere Modellierung realer oder abstrakter Konzepte zu erreichen, die mit Imperfektion behaftet sind. Wie in Abschnitt 1.1 ausgeführt, geschieht dies nicht aus Selbstzweck, sondern um bessere, robustere, kostengünstigere Systeme bauen zu können.

Daraus folgt zunächst nur, daß ein Fuzzy-Modell in der Lage sein muß, solche Imperfektionen repräsentieren zu können. Ansätze, die darüber hinaus *auch das Modell selbst* unsicher machen, lassen sich damit noch nicht motivieren. Tatsächlich sind auch die Beispiele, mit denen solche Ansätze typischerweise illustriert werden, nicht struktureller, sondern inhaltlicher Natur: Die Tatsache, daß ein Objekt zu „60%" einer Oberklasse und zu „40%" einer anderen angehört (und damit Teile der Eigenschaften und Fähigkeiten beider Oberklassen erbt), ist kein außergewöhnlicher Vorgang, sondern tritt in modernen objektorientierten System ständig auf — zum Beispiel bei der Entwicklung graphischer Oberflächen, die aus vorhandenen Komponenten bausteinhaft zusammengesetzt werden. Wie heute jeder Student weiß, würde man so eine Anforderung daher nicht über eine statische Klassenvererbung, sondern dynamisch mit Hilfe von Entwurfsmustern wie dem *Dekorierer* [GHJV95] modellieren.

Schließlich bleibt bei der rein strukturellen Betrachtung der vorhandenen Ansätze die Frage offen, was bei statischer Vererbung für die Ausführung von Methoden gelten soll. Eine Methode nur zu „60%" und eine weitere zu „40%" auszuführen kann offensichtlich nicht sinnvoll sein; wird dagegen nur genau eine Methode (vielleicht die mit dem höheren Zugehörigkeitsgrad) ausgeführt, bleibt unklar, wozu genau die Fuzzy-Vererbung eingeführt wurde. Auch wir können dafür keine pratikable Lösung

anbieten, die konsequente Entwurfsentscheidung muß daher lauten, keine Fuzzy-Vererbungsbeziehungen zuzulassen.

Methoden Methoden im objektorientierten Modell dienen der Verarbeitung von Objekt-Attributen und kapseln in vielen Systemen zusätzlich den Zugriff auf diese Attribute. Es ist daher unverständlich, daß dieser wesentliche Aspekt des objektorientierten Modells bei der Betrachtung von Fuzzy-Erweiterungen ignoriert wird, wie wir schon bei der Anforderung VERARBEITUNG UNSCHARFER DATEN (vergleiche Abschnitt 3.2.6 auf Seite 33) ausgeführt haben.

Obwohl wir keine Fuzzy-Erweiterung auf der Ebene der Klassenzugehörigkeit und der Vererbung einführen, bleibt dennoch die Frage bestehen, wie die Verarbeitung imperfekter Objektzustände unterstützt werden soll. Unser Ansatz gibt hierzu Hilfestellungen von zwei Seiten:

Konsistenzerhaltende Operationen Die in einem Informationssystem verwalteten Daten stellen bekanntermaßen einen hohen Wert dar, den es zu schützen gilt. Ein wichtiger Aspekt dabei ist die KONSISTENZERHALTUNG, worauf wir auch in der gleichnamigen Anforderung (siehe Seite 35) hingewiesen haben. Eine wesentliche Leistung unseres Modells ist daher die Verarbeitung mit Hilfe der Fuzzy Konjunktiven Normalformen, der Fuzzy-Abhängigkeitsgraphen und den damit verbundenen konsistenzerhaltenden Operationen.

Unterstützung durch Bibliotheken und Komponenten Zur Realisierung eines lauffähigen Systems muß eine geeignete Infrastruktur angeboten werden, die den Umgang mit imperfekten Informationen unterstützt und eine effiziente Realisierung von Fuzzy-Informationssystemen ermöglicht. Dazu gehört insbesondere die Einbettung des Konzeptes der Annotationen und Facetten in existierende Programmsprachen, sowie die Realisierung der im letzten Teil dieser Arbeit vorgestellten theoretischen Konzepte. Wir werden diesen Aspekt in den nachfolgenden Kapiteln dieses Teils genauer ausführen.

Das Ziel muß es sein, sowohl die Definition als auch die Implementierung von Algorithmen auf einer semantisch höheren Ebene als der einzelner Fuzzy-Mengen zu ermöglichen. Idealerweise benutzt ein Entwickler von Fuzzy-Informationssystemen dabei abstrakte Operationen zur Modellierung und Verarbeitung imperfekter Informationen, ohne mit deren Repräsentation in Form einzelner Fuzzy-Mengen in Berührung zu kommen.

Während aber heute kein Anwendungsentwickler mehr auf die Idee käme, elementare Ganzzahloperationen wie eine Division selber auf Bitebene zu implementieren, scheint dieser Abstraktionsgedanke im Bereich der Fuzzy-Informationssysteme noch neu zu sein. Dies dürfte vor allem mit dem Problem zusammenhängen, daß die konkreten Verarbeitungsschritte immer anwendungsspezifisch sind, wie wir schon bei

der Anforderung VERARBEITUNG UNSCHARFER DATEN (siehe Abschnitt 3.2.6 auf Seite 33) angemerkt haben. Trotzdem wird es eine Reihe von Eigenschaften geben, die anwendungsübergreifend für die verwendeten Operatoren gelten sollen, ohne diese immer von Grund auf neu definieren zu müssen. Es ist daher sinnvoll, eine Zwischenschicht anzubieten, die eine Definition anwendungsspezifischer Operatoren aufbauend auf vorgegebenen Primitiven ermöglicht. Genau dies leisten unsere Graph-Operationen, die grundsätzliche Eigenschaften durch die jeweiligen Basis-Postulate festlegen, aber darüber hinaus verschiedene Ausprägungen durch Definition spezifischer Zusatz-Postulate erlauben, wie in Abschnitt 10.4 auf Seite 130 für die Graph-Expansion demonstriert wird. Allen diesen Operatoren ist gemein, daß sie die Erhaltung der Konsistenz bei der Verarbeitung garantieren, was eine Grundvoraussetzung für den Einsatz von Fuzzy-Technologie in Informationssystemen darstellt.

11.2 Definition des objektorientierten Fuzzy-Modells

In diesem Abschnitt beschreiben wir unseren Vorschlag im Detail. Dabei handelt es sich um Fuzzy-Erweiterungen des objektorientierten Datenmodells auf der Ebene von:

- Objekten,
- Attributen und
- Attributmengen.

Die einzelnen Erweiterungen werden zunächst anschaulich motiviert; anschließend definieren wir unser objektorientiertes Fuzzy-Modell formal.

Zuvor gehen wir jedoch auf ein Grundprinzip unseres Modells genauer ein: die oben schon angedeutete Trennung zwischen einer *Strukturkomponente*, wie einem Attribut, und dessen *Fuzzyfizierung*.

11.2.1 Trennung von Konzept und Fuzzyfizierung

Ein wesentlicher Aspekt unseres Ansatzes ist, über die einfache kanonische Fuzzyfizierung objektorientierter Konzepte hinaus separate Betrachtungsebenen einzuführen. Damit wird eine *Entkopplung* von Konzepten und deren Fuzzy-Zuständen erreicht, ähnlich wie wir im theoretischen Modell bereits die Struktur und die Interpretation imperfekter Informationen entkoppelt haben (vergleiche Abschnitt 8.4 auf Seite 82).

Dies begründet sich mit den Anforderungen PARTIELLE UNSCHÄRFE (Abschnitt 3.2.4 auf Seite 30) und INTEROPERABILITÄT (Abschnitt 3.2.7 auf Seite 34):

- Ein Informationssystem soll nur genau dort Fuzzy-Teile beinhalten, wo dies für das Gesamtsystem Vorteile bringt. Daraus folgt, daß es möglich sein muß, präzise festzulegen, welche Systemteile imperfekte Informationen aufnehmen. Im Extremfall kann dies bedeuten, daß nur eine einzige Objektinstanz einer Klasse imperfekte Informationen aufweist, und dort auch nur in einem einzelnen Attribut.
- Solche Fuzzy-Informationssysteme müssen auch weiterhin kompatibel mit klassischen Systemteilen bleiben. Daraus folgt, daß es eine Sicht auf ein Fuzzy-Informationssystem geben muß, in der es weiterhin als klassisches, scharfes System erscheint — mit scharfen Attributen und Objekten. Fuzzy-Informationen müssen zusätzliche Eigenschaften sein, die je nach Bedarf ein- oder ausgeblendet werden können.

Diese Entkopplung erreichen wir konzeptionell durch die Einführung von *Annotationen* und *Facetten*. Damit erhält man die Möglichkeit, ein Fuzzy-Informationssystem in mehreren Sichten zu sehen: einer klassischen, scharfen Sicht, die aus Kompatibilitätsgründen erforderlich ist, sowie einer Fuzzy-Sicht, in der die Imperfektionen erkennbar und verwendbar werden. Zusätzlich erschließen wir mit dieser Trennung die Möglichkeit, Zusatz- und Metawissen anderer Ausprägung, zum Beispiel aussagenlogische Bedingungen oder statistische Informationen, nahtlos in ein solches System einzubringen und potentiell sogar miteinander zu verknüpfen.

Annotationen und Facetten

Die Grundidee, Wissen über ein Objekt einer Diskurswelt in mehrere Sichtweisen aufzusplitten, geht auf eine Arbeit von Marvin Minsky [Min75] zurück.

Angewendet auf unser Modell erlauben wir, Strukturkomponenten des objektorientierten Modells zu *annotieren*. Eine solche Annotation bildet einen Container, in dem eine oder mehrere *Facetten* eines konkreten Typs zur Aufnahme von Zusatz- oder Metainformationen angelegt werden können. Beispielsweise könnte ein einzelnes Objekt-Attribut annotiert und mit einer Facette zur Aufnahme einer Fuzzy Konjunktiven Normalform ausgestattet werden.

In einfacherer Form wird dieser Ansatz auch in [Bos96] verwendet:[1] während dort jedoch nur Attribute um Facetten erweitert werden, betrachten wir das objektorientierte Modell aus einer Metasicht und erweitern Teile dieses Metamodells, wie Objekte, Attribute oder Attributmengen, um Annotationen, die dann die beschriebenen Facetten aufnehmen können.

[1] Die Terminologie ist auch leicht unterschiedlich. Für Boss spaltet sich ein Attribut in mehrere Facetten auf, von denen eine dann den klassischen, scharfen Wert aufnimmt. Die Gesamtheit dieser Facetten bildet somit die Annotation. Bei uns dagegen bleibt das Attribut selbst scharf, erst durch Annotation können in den einzelnen Facetten Zusatz- und Metainformationen etwa in Form einer Fuzzy Konjunktiven Normalform abgelegt werden.

Syntaktisch denotieren wir dabei die Tatsache, daß eine Strukturkomponente k annotiert ist, mit dem Zeichen „↑“. Weist eine Strukturkomponente k vom Typ t_k eine Facette f_i vom Typ t_{f_i} auf, kennzeichnen wir dies mit k: $t_k{\uparrow}f_i{:}t_{f_i}$. Der Wert dieser Facette wird mit $k{\uparrow}f_i$ bezeichnet.

Beispiel (Attribut-Facetten) Wir greifen auf das Beispiel der Zeugenaussagenverwaltung zurück. Eine gesuchte Person sei durch ein Objekt o_{person} beschrieben, das unter anderem ein Attribut *o.größe* enthält. Auf diesem Attribut definieren wir nun zwei Facetten:

$$\texttt{größe} : \texttt{Integer}{\uparrow}\texttt{Beschreibung: String},$$
$$\texttt{größe} : \texttt{Integer}{\uparrow}\texttt{Fuzzy\_größe: FKNF};$$

Die erste Facette ist vom Typ *String*, während die zweite eine Fuzzy Konjunktive Normalform speichern kann. Mit Hilfe dieser Facetten können wir nun eine Zeugenaussage „die Person ist mittelgroß“ sowohl in ihrer ursprünglichen textuellen Form, als auch als Fuzzy-Aussage speichern:

$$\texttt{größe}{\uparrow}\texttt{Beschreibung} = \texttt{"mittelgroß"};$$
$$\texttt{größe}{\uparrow}\texttt{Fuzzy\_größe}.\gamma\texttt{-expansion}(\mathcal{F}_{\text{mittelgroß}}, \gamma);$$

Ein Fuzzy-Zeugenaussagenverwaltungssystem könnte bei Modifikation der Facetten gleichzeitig den scharfen Wert des Attributes setzen, indem die Fuzzy Konjunktive Normalform defuzzifiert wird. So erreichen wir Kompatibilität mit Systemteilen, die nicht mit den Fuzzy-Informationen umgehen können: aus deren Sicht existiert weiterhin ein „normaler“ scharfer Wert, der problemlos gelesen und weiterverarbeitet werden kann.[2]

An diesem Beispiel läßt sich auch nachvollziehen, warum der Typ einer Facette unabhängig von dem Typ des annotierten Attributes ist: die (imperfekten) Zusatzinformationen müssen sich keineswegs auf den Typ des Attributes oder einen kompatiblen Untertyp beschränken.

Semantik von Annotationen

Bei der Einführung des Konzeptes der Abhängigkeitsgraphen in Abschnitt 8.5.1 auf Seite 85 wurde offen gelassen, zwischen welchen Konzepten solche Abhängigkeiten

[2] Das *Setzen* eines solchermaßen annotierten scharfen Wertes ändert natürlich nicht die Werte der einzelnen Facetten in der Annotation — solche Änderungen müßten dann von den Methoden berücksichtigt werden, die mit den Facetten operieren. In vielen Systemen ist das Setzen/Auslesen von Attributen über *set/get*-Methoden gekapselt, diese ließen sich leicht für einen solchen Zweck erweitern.

bestehen, und diese nur abstrakt als Knoten in einem Abhängigkeitsgraphen modelliert. Auf diese Weise konnten wir unser formales Modell unabhängig von einem konkreten Datenmodell beschreiben.

An dieser Stelle können wir jetzt die Verbindung zwischen dem Konzept des Abhängigkeitsgraphen und unserem objektorientierten Fuzzy-Modell herstellen: Ein Knoten in einem Fuzzy-Abhängigkeitsgraphen entspricht gerade einer Annotation. Eine solche Annotation kann auf verschiedenen Strukturkomponenten des objektorientierten Datenmodells definiert werden, zum Beispiel auf Objekten oder Attributen. Die genaue Definition der annotierbaren Konzepte ist Bestandteil der nachfolgenden Abschnitte.

Wir ändern dafür die Zuordnungsfunktion ξ des Abhängigkeitsgraphen so, daß sie jetzt einem Knoten, also einer Annotation, eine Menge von Facetten zuordnet — wir erlauben hier neben Fuzzy-Facetten auch Facetten anderen Typs, wie im obigen Beispiel gezeigt wurde. Für Facetten vom Typ „Fuzzy Konjunktive Normalform“ stehen damit alle in Tabelle 9.1 auf Seite 97 gezeigten Verarbeitungsoperationen zur Verfügung; Facetten anderen Typs werden typischerweise andere Operationen aufweisen.

Durch das Einfügen einer Kante in diesen Abhängigkeitsgraphen läßt sich eine gerichtete Abhängigkeit zwischen Annotationen ausdrücken, die dann für alle Facetten einer Annotation gilt.

Die in Annotationen gespeicherten Informationen lassen sich mit den folgenden beiden Ausprägungen charakterisieren:

Anforderungen, die an ein annotiertes Konzept gestellt werden. Die Idee ist hierbei, einen scharfen Wert durch imperfekte Bedingungen und Einschränkungen einzugrenzen. Die konsistenzerhaltenden Verarbeitungsoperationen stellen dabei sicher, daß keine widersprüchlichen Anforderungen gestellt werden können.

Metainformationen, die über einen einfachen scharfen Wert hinausgehende Informationen zu einem annotierbaren Konzept bereithalten. Auch hier stellen die Operationen sicher, daß die gespeicherten Metainformationen konsistent zueinander sind.

Annotationen schränken also den gültigen Wertebereich eines Attributes dynamisch ein, wie es auch der possibilistischen Interpretation der Fuzzy-Mengen entspricht (vergleiche Abschnitt 7.1, Seite 63). Die Anforderungen und Metainformationen werden dabei in Form von Fuzzy Konjunktiven Normalformen formuliert, wobei einzelne Aussagen in Form von Fuzzy-Atomen ausgedrückt werden, die dann aussagenlogisch miteinander verknüpft werden können.[3]

[3]Da wir außer Fuzzy-Facetten auch Facetten anderen Typs zulassen, lassen sich mit Hilfe unseres Modells auch Informationen erfassen, die mit Hilfe anderer Repräsentationsformen erzeugt wurden. Verschiedene Informationsarten können so einheitlich verwaltet werden. Hierdurch ergeben sich zusätzliche Möglichkeiten bei der Informationsintegration und insbesondere der Kombination von Informationen unterschiedlichen Typs; dies geht jedoch über das Thema dieser Arbeit hinaus.

11.2.2 Fuzzy-Attribute

Wir beginnen mit der Erweiterung von Attributen um Fuzzy-Konzepte. Wie oben motiviert, beschränken wir uns im Gegensatz zu anderen Fuzzy-Datenmodellen nicht darauf, für Attribute nur einen zusätzlichen Datentyp „Fuzzy-Menge“ einzuführen. Stattdessen führen wir die beschriebene Trennung zwischen der Strukturkomponente „Attribut“ und deren Fuzzyfizierung durch: Ein Attribut wird so zu einem annotierbaren Konzept, das Zusatzinformationen in Form einer oder mehrerer Facetten bereithalten kann.

Dabei schränken wir weder den möglichen Typ einer Facette ein, noch die Domäne einer Fuzzy-Facette. Dies geschieht aus zwei Gründen: erstens setzen wir keine streng typisierte Programmiersprache voraus. Zwar erfolgt die gezeigte Beispielimplementierung in Java, alle Konzepte sollen aber auch mit den im Internetbereich populären Skriptsprachen realisierbar sein, von denen viele kein ausgeprägtes Typsystem besitzen. Zweitens gibt es keinen festen Zusammenhang zwischen dem Typ eines Attributes und dem seiner Facetten, oder seiner Domäne und der Domäne einer Fuzzy-Facette. So kann es bei Attributen beliebigen Typs sinnvoll sein, Zusatzinformationen etwa in Form einer Zeichenkette abzulegen.

Attribut-Annotationen sind möglich für:

Einfachwertige Attribute, die einen primitiven Datentyp aufnehmen. Eine Annotation in Form einer Fuzzy Konjunktiven Normalform kann hier verwendet werden, um beispielsweise einen voraussichtlichen Wert aufzunehmen, oder eine einschränkende Bedingung festzuhalten.

Mehrwertige Attribute, wie Vektoren oder Felder. Eine Annotation kann hier mehrere Formen annehmen:

- eine einzelne Fuzzy Konjunktive Normalform, die nach einer geeigneten Defuzzifizierung zu einem mehrwertigen scharfen Ergebnis wird;
- eine einzelne Fuzzy Konjunktive Normalform, die Metainformationen wie die voraussichtliche Länge eines mehrwertigen Attributes festhält;
- eine mehrwertige Annotation, deren einzelne Fuzzy Konjunktive Normalformen mögliche Werte der entsprechenden Elemente des mehrwertigen Attributes repräsentieren, also jedes Element einzeln einschränken.

Objektwertige Attribute, die Referenzen auf andere Objekte halten. Eine Fuzzy-Annotation kann hier etwas über die Qualität einer Referenz aussagen. Wenn es die eingesetzte Programmiersprache erlaubt, kann auch die Domäne der Fuzzy-Interpretation eine Menge von Objektreferenzen gleichen Typs sein, womit eine Referenz selbst einen imperfekten Zustand annehmen kann.

Wie im vorhergehenden Abschnitt erläutert, bildet eine solche Attribut-Annotation einen Knoten in einem Fuzzy-Abhängigkeitsgraphen. Durch das Einfügen einer Kante kann eine gerichtete Abhängigkeit zwischen Attributen ausgedrückt werden, die dann bei einer Verarbeitung mit Hilfe der Operationen Graph-Expansion, -Revision und -Kontraktion automatisch berücksichtigt wird.

11.2.3 Fuzzy-Attributmengen

Als nächstes erweitern wir das Konzept der Annotation einfacher Attribute auf die Annotation von *Attributmengen*.

Eine Attributmenge ist definiert als eine Menge bestehend aus den Attributen eines oder mehrerer Objekte. Damit erhalten wir die Möglichkeit, diese Attribute nicht nur einzeln, sondern auch in einer spezifischen Kombination mit einer Annotation und somit mit imperfekten Zusatz- oder Metainformationen zu versehen.

Anschaulich läßt sich eine solche Attributmenge als die Einführung einer ad-hoc Beziehung verstehen. Im Gegensatz zu den klassischen Beziehungen, die bereits während der Modellierung festgelegt werden, haben solche ad-hoc Beziehungen folgende Eigenschaften:

- sie gelten nur für ausgewählte Instanzen, also gerade nicht für alle Objekte einer Klasse;
- sie können zur Laufzeit erzeugt und verändert werden;
- sie erlauben die Modellierung von Fuzzy-Beziehungen, die im Gegensatz zu den klassischen Beziehungen nicht scharf sein müssen; und
- sie ermöglichen die Modellierung von Beziehungen, die abhängig vom Zustand eines Objektes sind.

Folgende Beispiele mögen die Verwendung veranschaulichen: Eine Fuzzy-Anwendung für den Bereich Maschinenbau soll Artefakte verwalten, ein Objektmodell repräsentiert dafür Eigenschaften wie Material, Abmessungen und Volumen. Wenn nun eine vage Eigenschaft wie *„gut verschweißbar“* festgehalten werden soll, läßt sich diese nicht an einem einzelnen Objekt oder Attribut festmachen — ein solches Konzept macht erst bei der Kombination zweier Materialien Sinn. In unserem Modell wird dafür eine Attributmenge definiert, die die Material-Attribute der beteiligten Objekte umfaßt. Diese Attributmenge wird dann annotiert, wobei eine Facette der Annotation eine Metainformation wie die Schweißbarkeit aufnehmen kann. Beispielsweise wird die Kombination „Stahl an Stahl“ sehr gut verschweißbar sein, „Stahl an Styropor“ dagegen gar nicht.

Ein anderes Beispiel findet sich bei einer Anwendung im Bereich der Architektur, wenn eine Bedingung wie *„farblich passend“* für die Materialien und Farben eines

Raumes repräsentiert werden soll — auch diese läßt sich nicht sinnvoll an einem einzelnen Attribut oder Objekt aufhängen.

Obwohl solche Fälle häufig auftreten und mit Hilfe eines Datenmodells abstrahierbar sein sollten, findet sich in der Literatur kein vergleichbares Konzept — nur mit Hilfe von Fuzzy-Attributen alleine lassen sich solche Phänomene nur sehr unzureichend ausdrücken.

Dank unserer Vorgehensweise der Trennung von Konzept und Fuzzy-Annotation lassen sich solche Fuzzy-Attributmengen einfach modellieren: Eine Attributmenge wird zu einem weiteren annotierbaren Konzept, das mit den bekannten Fuzzy-Facetten versehen werden kann.

Ein weiteres Einsatzgebiet liegt in der Repräsentation von objektübergreifenden Bedingungen, wie sie [Bos96] als offene Forschungsfrage skizziert: dabei sollen imperfekte Metainformationen wie „Objekt X ist preiswerter als Objekt Y“ modelliert werden, was sich mit Hilfe unserer Attributmengen elegant ausdrücken läßt. Hierfür würde man eine Attributmenge über die Preis-Attribute der beiden Objekte definieren, deren Annotation die gewünschte Bedingung als Anforderung aufnimmt.

Das Konzept der Abhängigkeiten bleibt auch hier erhalten; einer Attributmenge kann eine Annotation zugeordnet werden, die dann einen Knoten in einem Fuzzy-Abhängigkeitsgraphen bildet.

11.2.4 Fuzzy-Objekte

Als letzten Fall schließlich erlauben wir die Annotation von Objekten als ganzes. Damit können innerhalb der Facetten Zusatzinformationen über das Objekt abgelegt werden, angefangen von einfachen textuellen Informationen *(„erster Entwurf!“)* über Bedingungen, die das Objekt als ganzes betreffen *(„muß persistent gespeichert werden“)*, bis hin zu imperfekten Metainformationen wie die *Aktualität*, die *Informationsreife* (vergleiche Abschnitt 2.3.2 auf Seite 18) oder die *Wichtigkeit* eines Objektes.

Fuzzy-Objekte sind somit eine weiteres annotierbares Konzept und können daher genauso wie Attribute und Attributmengen mit Abhängigkeiten ausgestattet sein.

11.2.5 Formalisierung

In diesem Abschnitt formalisieren wir die oben aufgestellten Erweiterungen.

Nun haben wir in der Einleitung zu Kapitel 8 (Seite 69) kritisiert, daß die Erweiterung eines theoretischen Modells allein nicht ausreicht, sondern mit einer praktischen Umsetzung gekoppelt sein muß. An dieser Stelle werden wir daher, wie schon beim theoretischen Fuzzy-Modell, nur solche Eigenschaften in das formale objektorientierte Fuzzy-Modell aufnehmen, die auch praktisch umgesetzt werden können. Eine solche Umsetzung muß dabei als orthogonale Systemerweiterung durch-

geführt werden können, wie es auch in der Anforderung INTEROPERABILITÄT (siehe Abschnitt 3.2.7 auf Seite 34) verlangt wird.

Das hier entwickelte formale Objektmodell ist dabei abstrahiert von einer konkreten Programmiersprache oder einem bestimmten Datenbankmanagementsystem formuliert, da dies bei der Vielzahl der im Einsatz befindlichen Systeme zu kurz greifen würde. Es soll vielmehr aufzeigen, wie bei einer gegebenen Umgebung die Einbettung des Fuzzy-Modells systematisch durchgeführt werden kann. Aus diesem Grund enthält das objektorientierte Fuzzy-Modell auch nur solche Konzepte, die sich in nahezu allen Varianten objektorientierter Systeme finden lassen, so daß sich die Erweiterung einer konkreten Programmiersprache oder Ablaufumgebung überall durchführen lassen sollte.

Bei der Aufstellung des formalen Fuzzy-Modells gehen wir schrittweise vor: Zunächst rekapitulieren wir ein formales Objektmodell, das die klassische Trennung von Datenmodell, Schema und Instanz durchführt. In einem zweiten Schritt erfolgt dessen Erweiterung um Annotationen und Facetten. Hierfür wird ein Annotations-Schema sowie eine dazu kompatible Annotations-Instanz definiert, mit denen die annotierbaren Konzepte sowie die Facetten-Typen festgelegt werden. Das Ziel ist dabei, eine Implementierung des objektorientierten Fuzzy-Modells mit existierenden Entwicklungs- und Laufzeitumgebungen zu ermöglichen, ohne dabei Eingriffe in Übersetzer oder Bibliotheken durchführen zu müssen.

Das formale Objektmodell

Für die Definition eines Datenmodells benötigt man zunächst ein Typsystem, bestehend aus primitiven Typen, Typkonstruktoren und dazu passenden Operatoren:

Definition 11.2.1 (Objektorientiertes Datenmodell) *Ein objektorientiertes Datenmodell* Π *ist ein Tripel*

$$\Pi = (P, T, O)$$

mit

(Primitive Typen) *der Menge P aller primitiven Typen. Dabei muß P mindestens alle Referenztypen* ref *k enthalten, wobei k ein Klassenname aus der Menge aller möglichen Klassennamen darstellt:* $k \in \mathbb{K}$*. Ein Typ ist formal die Menge seiner Ausprägungen. Elemente von* ref *-Typen werden auch Objektidentifikatoren (oids) genannt.*

(Typkonstruktoren) *der Menge T aller Typkonstruktoren (wie struct, set, list). Ein objektorientiertes Datenmodell muß mindestens den Typkonstruktor „struct" enthalten, der einer Menge von (Attributname, Typ)-Paaren einen neuen Typ, nämlich die Menge der Tupel mit Elementen aus den Attributtypen zuordnet. Dabei darf ein Attributname nur einmal einem Typ zugeordnet werden.*

(Operatoren) *der Menge O aller Operatoren, die auf den Typen ausgeführt werden können.*

Die konkret existierenden primitiven Typen, Typkonstruktoren und Operatoren werden dabei abhängig von dem Datenmodell der verwendeten Programmiersprache sein.

Für ein objektorientiertes Informationssystem wird dann auf der Basis eines konkreten Datenmodells ein *Schema* definiert, das die benötigten Klassendefinitionen enthält:

Definition 11.2.2 (Objektorientiertes Schema) *Ein objektorientiertes Schema Σ ist ein Tripel*

$$\Sigma = (\Pi, K, S)$$

mit

(Datenmodell) *einem objektorientierten Datenmodell Π.*

(Klassennamen) *der partiell geordneten Menge K von Klassennamen. Dabei gilt für alle $k_1, k_2 \in K$: wenn $k_1 < k_2$, dann ist k_2 Unterklasse von k_1.*

(Klassendefinition) *der Abbildung*

$$S : K \to T^*(P)$$

die jedem Klassennamen einen Typ zuordnet; dieser muß unter Verwendung des Typkonstruktors „struct“ entstanden sein. Dabei bezeichnet $T^*(P)$ den Abschluß der Menge P unter Anwendung der Typkonstruktoren T aus Π.

Für diese Abbildung fordert man üblicherweise eine Reihe von Konsistenzbedingungen, etwa um die Kompatibilität von Typen in Vererbungsbeziehungen sicherzustellen. Wir verweisen den Leser hierfür auf die Standardliteratur [AHV95, Kapitel 21.2, Definition 21.2.3].

Zur Laufzeit eines Systems entsteht auf der Basis eines gegebenen Schemas eine *Instanz* eines objektorientierten Informationssystems:

Definition 11.2.3 (Objektorientierte Instanz) *Eine objektorientierte Instanz I ist ein Tripel*

$$I = (\Sigma, O, Z)$$

mit

(Schema) *einem objektorientierten Schema Σ.*

(Extensionsabbildung) *der Extensionsabbildung O, die jedem $k \in K$ eine Teilmenge von* ref *k zuordnet.*

(Zustandsabbildung) *der Zustandsabbildung Z, die jedem Objektidentifikator $o \in O(k)$ einen Wert $Z(o) \in S(k)$ zuordnet.*

Die hier gezeigte Trennung in Schema und Instanz ist vor allem im Datenbankbereich verbreitet, wo sich eine eigene Disziplin, der Datenbankentwurf, nur mit der Erstellung eines geeigneten Schemas beschäftigt. Bei dieser Sichtweise entspricht das Schema dem Datenbankschema und die Instanz einer konkreten Ausprägung in Form einer Datenbasis.

In dieser Arbeit liegt der Fokus dagegen mehr auf dem Bereich der objektorientierten Programmiersprachen, wo diese Trennung meist nur implizit existiert, da viele Sprachen das Erzeugen neuer Klassen auch zur Laufzeit erlauben. Das vorgestellte Modell ist aber auch mit dieser Sichtweise kompatibel, wenn man von einem während der Laufzeit veränderlichen Schema Σ ausgeht.

Erweiterung um Annotationen und Facetten

Getrennt von dem eigentlichen Schema einer Anwendung definieren wir jetzt ein *Annotations-Schema*, das genau diejenigen Klassen enthält, die zur Annotation von Schemaelementen mittels Facetten verwendet werden können:

Definition 11.2.4 (Objektorientiertes Annotations-Schema) *Ein objektorientiertes Annotations-Schema Γ ist definiert als das Tripel*

$$\Gamma = (\Pi, F, S')$$

mit

(Datenmodell) *einem objektorientierten Datenmodell Π.*

(Facettennamen) *der partiell geordneten Menge F von Facetten-Klassennamen. Dabei gilt wie oben für alle $f_1, f_2 \in F$: wenn $f_1 < f_2$, dann ist f_2 Unterklasse von f_1.*

(Klassendefinition) *der Abbildung*

$$S' : F \rightarrow T^*(P)$$

die jedem Facettennamen einen Facettentyp zuordnet; auch dieser muß unter Verwendung des Typkonstruktors „struct“ entstanden sein. Hier gelten die gleichen Konsistenzbedingungen wie oben beschrieben.

Für ein Fuzzy-Informationssystem enthält ein solches Annotations-Schema mindestens Facetten mit dem Namen „FKNF“, denen ein zur Implementierung von Fuzzy Konjunktiven Normalformen geeigneter Typ zugeordnet sein muß.

Definition 11.2.5 (Objektorientierte Annotations-Instanz) *Die mit einer objektorientierten Informationssystem-Instanz I kompatible Annotations-Instanz G ist ein Quadrupel*

$$G = (I, \Gamma, \mathfrak{G}, \mathcal{Z})$$

mit

(Informationssystem-Instanz) *einer objektorientierten Informationssystem-Instanz I.*

(Annotations-Schema) *einem objektorientierten Annotations-Schema Γ.*

(Abhängigkeitsgraph) *einem Abhängigkeitsgraphen $\mathfrak{G} = (V, E, \xi)$ (siehe Definition 8.5.1 auf Seite 86). Ein Knoten $v \in V$ dieses Graphen entspricht einer Annotation, die auf einer atomaren oder einer zusammengesetzten Strukturkomponente möglich ist. Wir definieren zunächst die atomaren Komponenten σ, dies sind Objektidentifikatoren $o \in O(k)$ mit $k \in K$ oder Tupel (o, a) bestehend aus einem Objektidentifikator $o \in O(k)$ und einem Attributnamen a, der im Typ $T(k)$ definiert ist:*

$$\sigma := \bigcup_{k \in K} O(k) \cup \{(o, a) | \exists k \in K : o \in O(k) \wedge a \in Attributname(T(k))\}$$

Eine zusammengesetzte Strukturkomponente ist eine Menge von atomaren Strukturkomponenten σ. Die Menge aller Knoten V des Abhängigkeitsgraphen läßt sich so als Untermenge der Potenzmenge von σ definieren:

$$V \subseteq 2^{\sigma}$$

Die Kantenmenge E ist eine beliebige zyklenfreie Teilmenge von $V \times V$. Jede Kante repräsentiert eine gerichtete Abhängigkeit zwischen zwei Annotationen (vergleiche Abschnitt 8.5 auf Seite 84).

Die Zuordnungsfunktion ξ ordnet jedem Knoten, also jeder Annotation, eine Menge von Facetten zu. Jede Facette ist dabei formal der Objektidentifikator einer Instanz eines Facettentyps aus Γ:

$$\xi : V \rightarrow 2^{\mathbb{E}}$$

wobei $\mathbb{E}$ die Menge aller Objektidentifikatoren von Instanzen von Facettentypen bezeichnet.

(Zustandsabbildung) *der Zustandsabbildung $\mathcal{Z}$, die jeder Facette $f \in \xi(v)$ einen Wert $\mathcal{Z}(f) \in S'(f)$ zuordnet.*

Wir modellieren Facetten hier als Objekte, weil sie sich so harmonisch in das objektorientierte Modell einfügen. Überdies ermöglicht diese Modellierung, eine Facette zwei verschiedenen Annotationen zuzuordnen.

Dieses Modell trennt demnach zwischen annotierbaren Konzepten — hier Objekte, Attribute und Attributmengen — und den darauf möglichen Annotationen. Für uns sind im folgenden nur Fuzzy-Annotationen interessant, das Modell erlaubt es aber trotzdem, Annotationen beliebigen Typs zuzuweisen; für eine Anwendung könnte zum Beispiel eine textuelle Annotation oder ein aussagenlogischer Ausdruck sinnvoll sein.

Ein konkretes objektorientiertes Fuzzy-Informationssystem besteht somit aus (vergleiche Abbildung 11.1 auf der nächsten Seite):

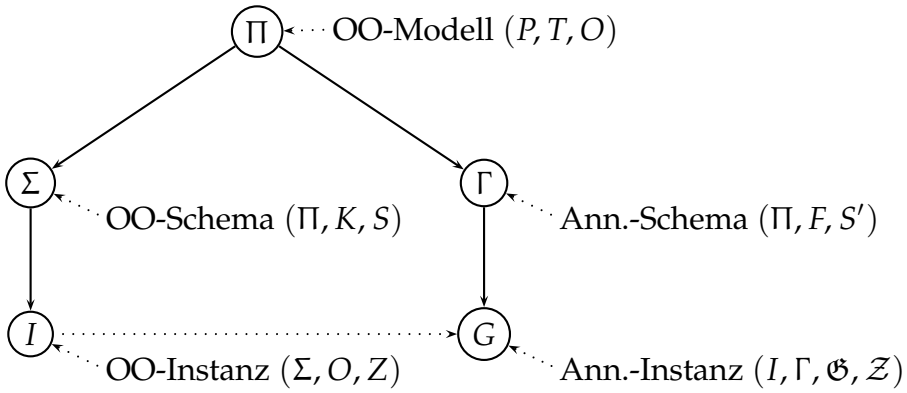


Abbildung 11.1: Erweiterung des objektorientierten Datenmodells um Annotationen

1. einem Datenmodell Π, das von der verwendeten Programmiersprache festgelegt ist und unverändert bleibt,
2. einem Schema Σ, in dem die Klassendefinitionen des Systems erfolgen,
3. einem Annotationsschema Γ, das zur Annotation verwendbare Klassen festlegt; sowie
4. einer Instanz I mit den zur Laufzeit existierenden Objekten und
5. der zugehörigen (kompatiblen) Annotations-Instanz G, die Objekte mit Metainformationen, insbesondere also Fuzzy-Objekte, aufnimmt.

In den restlichen beiden Kapiteln dieses Teils betrachten wir nun die praktische Umsetzung dieses Modells.

Kapitel 12

Annotationen und Fuzzy-Facetten

C'est magnifique, mais ce n'est pas l'Informatique
Bosquet [on seeing the IBM 4341]

In diesem Kapitel gehen wir auf die technische Umsetzung unseres objektorientierten Fuzzy-Datenmodells ein. Wir beginnen mit der Beschreibung denkbarer Entwurfsalternativen zu seiner Realisierung, die wir in Hinblick auf die gestellten Anforderungen und ihre praktische Eignung diskutieren.

Anschließend erfolgt die Beschreibung der ausgewählten Lösung in Form eines sogenannten Entwurfsmusters.

12.1 Einbettung des Fuzzy-Datenmodells

Nach der Entwicklung des formalen Fuzzy-Objektmodells ist der nächste Schritt seine Implementierung basierend auf existierenden Technologien. Dazu gehört mindestens eine objektorientierte Sprache, der zugehörige Übersetzer oder Interpreter, sowie die für ein lauffähiges System notwendigen Bibliotheken.

Grundsätzlich ergeben sich hier zwei Möglichkeiten:

Sprachänderung Eine vorhandene Programmiersprache wird von ihrer Syntax erweitert, so daß Annotationen und Facetten durch eingebaute Sprachmittel verwendet werden können.

Spracheinbettung Die entwickelten Konzepte werden über Mittel, die eine Sprache selbst zur Verfügung stellt, modelliert und stehen somit als Erweiterung zur Verfügung.

Die erste Möglichkeit wird an dieser Stelle nicht betrachtet. Sie verletzt schon unsere Anforderung nach INTEROPERABILITÄT (vergleiche Abschnitt 3.2.7 auf Seite 34), da ein praktischer Einsatz durch den mit einer solchen Änderung verbundenen immensen Aufwand unrealistisch wird. Aber selbst unabhängig von der Frage der Praktikabilität halten wir dies für keine sinnvolle Vorgehensweise, wie bereits in der Einleitung zu Kapitel 8 (Seite 69f) ausgeführt wurde: Imperfektion stellt für uns eine orthogonale Systemeigenschaft dar, gleichberechtigt neben anderen Eigenschaften wie Persistenz oder Sicherheit. Es ist weder wünschenswert noch sinnvoll, für jede neue Eigenschaft grundlegende Änderungen am verwendeten Datenmodell durchführen zu wollen.

12.1.1 Realisierungsalternativen

Aus den genannten Gründen verbleibt also die Möglichkeit der Spracheinbettung. Dabei bieten sich wiederum verschiedene Realisierungsmöglichkeiten:

Klassenhierarchie Zur Erweiterung einer existierenden Klasse um neue Fähigkeiten bietet das objektorientierte Modell den Mechanismus der Vererbung. Die naheliegende Lösung zur Erweiterung eines existierenden Klassenschemas um Annotationen und Facetten wäre daher, jede zu annotierende Klasse durch Vererbung mit einer Unterklasse zu versehen. In der Unterklasse können zusätzliche Attribute dann Referenzen auf Annotationsobjekte mit den Facetten speichern, die ihrerseits in einer eigenen Klassenhierarchie modelliert werden würden.

Lösungen dieser Art werden auch in [Bos96] diskutiert und wegen des hohen Aufwandes verworfen. Ein solches System wäre nicht nur schwer wartbar, sondern es wird durch die explizite Modellierung mit Hilfe einer statischen Klassenhierarchie auch unmöglich, dynamisch imperfekte Informationen zur Laufzeit zu erzeugen und durch Annotation an Objekte oder Attribute anzufügen.

Diese Variante ist zwar problemlos mit Hilfe gängiger Sprachen realisierbar, aber es bleibt dennoch die Frage nach einer eleganteren Lösung offen.

Aspektorientierte Programmierung Eine weitere Möglichkeit ist der Einsatz der sogenannten *Aspektorientierten Programmierung* (aspect-oriented programming, AOP) [KLM+97]. Bei diesem Ansatz werden Teile der Struktur und Semantik eines Programmes als Aspekte ausgelagert und später mit dem anwendungsspezifischen Programmcode verwoben.

Diese Variante ist durchaus interessant, wurde jedoch aus einer Reihe von Gründen nicht in Betracht gezogen:

- Für den praktischen Einsatz werden spezielle Vorübersetzer benötigt, die Aspekt- und Programmcode miteinander verweben. Diese sind jedoch nur für

wenige der gängigen objektorientierten Programmiersprachen verfügbar, wodurch viele der in der Praxis populären Sprachen von einem Einsatz unserer Fuzzy-Technologie ausgeschlossen würden.

- Es gibt keine sprachspezifischen Standards für AOP-Erweiterungen, so daß es selbst für ein und dieselbe Programmiersprache verschiedene Ansätze und Vorübersetzer gibt. Damit fehlt eine gemeinsame Basis, auf der Fuzzy-Erweiterungen und Anwendungsentwicklung aufsetzen können.
- Die Aspektorientierte Programmierung hat noch eine sehr geringe Verbreitung. Da bereits der Einsatz der Fuzzy-Technologie für sich schon eine große Herausforderung für den Praktiker darstellt, sollten die Einstiegshürden durch den Einsatz weiterer ungewohnter Konzepte nicht noch höher gelegt werden.

Es stellt sich also die Frage, ob nicht eine Synthese der obigen Konzepte möglich ist, mit der sich die Vorteile beider vereinen lassen, ohne dabei die Nachteile aufzugreifen. Glücklicherweise gibt es hierfür eine Lösung, namentlich der Einsatz eines *Entwurfsmusters*, das wir im folgenden Abschnitt vorstellen.

Das entwickelte Muster läßt sich leicht mit Hilfe gängiger objektorientierter Programmiersprachen realisieren und benötigt dafür keine zusätzlichen Vorübersetzer, im Gegensatz zu einer auf Aspektorientierter Programmierung basierenden Variante. Dabei erhält diese Lösung die Möglichkeit, separate, orthogonale Sichtweisen auf einem System zu definieren; die Fuzzy-Informationen können so mit den scharfen Daten verwoben werden, ohne daß getrennte Klassenhierarchien wie in der ersten Lösungsalternative nötig wären.

12.2 Das Annotations-Entwurfsmuster

In diesem Abschnitt beschreiben wir die im folgenden verwendete Lösung zur praktischen Umsetzung des objektorientierten Fuzzy-Datenmodells. Da sich der hier entwickelte Ansatz auch in anderen Gebieten einsetzen läßt, beschreiben wir ihn abstrahiert in Form eines Entwurfsmusters.

Die Idee der Entwurfsmuster *(Design Patterns)* geht zurück auf eine Arbeit im Bereich der Architektur [AIS+77]:

> *Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.*

Im Bereich des objektorientierten Entwurfs dienen sie vor allem dazu, wiederverwendbare Lösungen aufzuzeigen, die durch den gezielten Einsatz von Abstraktion und Entkopplung einen hohen Grad an Flexibilität aufweisen. Die Arbeit von

[GHJV95] war dabei die erste, die die Idee der Entwurfsmuster als neue Abstraktionsebene systematisch in den objektorientierten Entwurf eingebracht hat.

Zur Dokumentation von Entwurfsmustern gibt es einen präzise festgelegten Rahmen, der in [GHJV95] definiert ist. Die folgende Beschreibung folgt daher diesem Standard.

12.2.1 Zweck

Erweitere dynamisch die Struktur von Objekten.

12.2.2 Motivation

Anwendungen, die in einem objektorientierten System Metainformationen wie imperfektes Wissen, Bedingungen oder sonstige Zusatzinformationen einsetzen wollen, müssen dafür dynamisch die Struktur von Objekten erweitern können. So kann beispielsweise ein Attribut eines Objektes einer Reihe von aussagenlogischen Einschränkungen unterliegen, mit einem textuellen Kommentar versehen oder von einem imperfekten Zustand, repräsentiert durch eine Fuzzy-Beschreibung, überlagert werden. Dies muß jedoch transparent erfolgen, damit andere Systemteile, die von den Metainformationen keinen Gebrauch machen wollen, unbeeinträchtigt weiter mit den betroffenen Objekten arbeiten können.

Die Speicherung und Verwaltung dieser Metainformationen soll dabei dynamisch erfolgen. Solche Metainformationen werden mit Teilen des objektorientierten Datenmodells assoziiert, zum Beispiel mit einem einzelnen Attribut eines Objektes. Wir nennen die Teile des objektorientierten Datenmodells, für die Metainformationen verwaltbar sein sollen, *Strukturkomponenten*. Um sie zu bestimmen, betrachten wir das objektorientierte Datenmodell aus einer Metasicht: mögliche Komponenten sind so beispielsweise Objekte oder Attribute, aber auch daraus zusammengesetzte Strukturen wie eine Menge von Attributen.

Ferner soll es möglich sein, mehr als eine Art von Metainformation an einer Strukturkomponente festzumachen. Beispielsweise kann für ein Attribut sowohl eine unscharfe Information in Form einer Fuzzy-Menge, als auch eine textuelle Zusatzinformation in Form einer Zeichenkette vorliegen.

Die hier gezeigte Lösung verwendet dafür *Annotationen*, die an Strukturkomponenten aufgehängt werden. Eine solche Annotation spaltet sich in eine oder mehrere *Facetten* (facets) auf, welche dann die gewünschten Metainformationen aufnehmen (vergleiche Abschnitt 11.2.1 auf Seite 158). Eine Annotation bildet also den Container zur Verwaltung der einzelnen Facetten.

Die Entscheidung, welche Strukturkomponenten annotiert werden sollen, wird beim Einsatz des Musters gefällt.

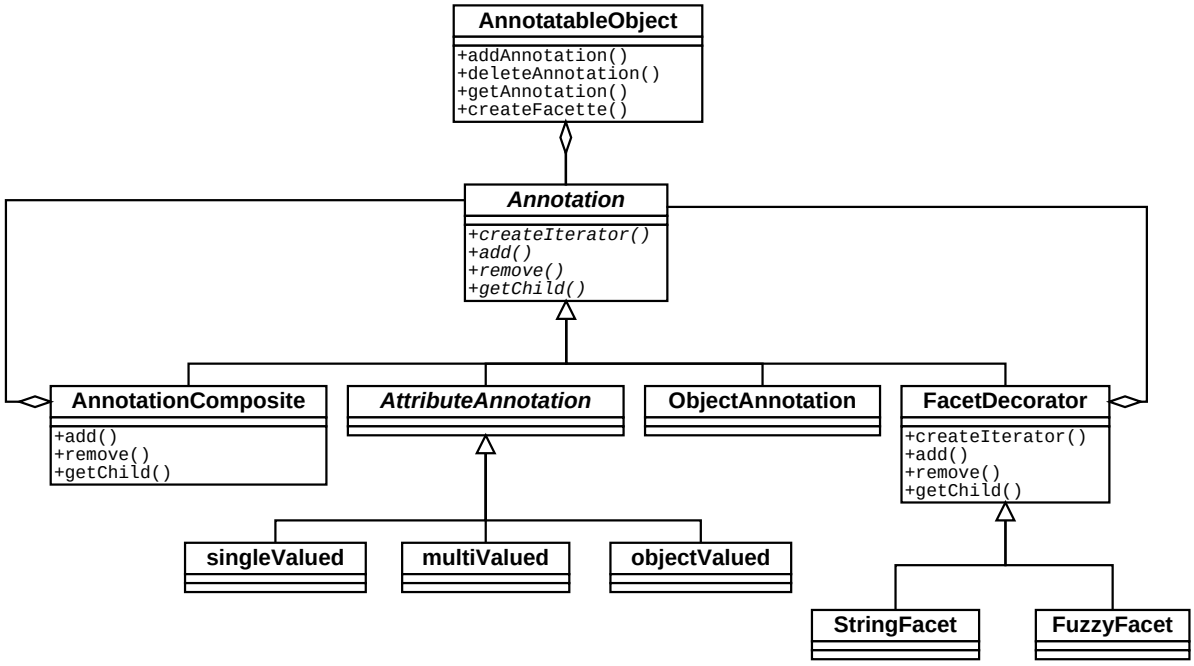


Abbildung 12.1: Einsatzbeispiel des Annotations-Entwurfsmusters zur Annotation von Objekten, Attributen und Attributmengen mit String- und Fuzzy-Facetten

Als Einsatzbeispiel betrachten wir ein Modell für Fuzzy-Informationssysteme, das Metainformationen in Form von String- und Fuzzy-Facetten verwalten kann (siehe Abbildung 12.1). In diesem Beispiel sind Zusatzinformationen für Objekte und Attribute erlaubt, wobei letztere nochmal nach ihrer Art unterteilt sind.

Ferner lassen sich aus diesen Komponenten zusammengesetzte Strukturen annotieren, was durch den Einsatz des Musters *Kompositum*[1] (Composite) innerhalb unseres Entwurfsmusters erreicht wird. Die abstrakte Klasse *Annotation* bietet eine einheitliche Schnittstelle für sowohl atomare als auch komplexe Annotationskomponenten.

Die Art der zu verwaltenden Metainformation ist ebenfalls dem Anwender des Musters überlassen. Jede Strukturkomponente wird dabei mit einer oder mehreren Facetten überlagert, von denen jede dann einen konkreten Typ von Metainformation aufnimmt. Im obigen Beispiel sind eine `StringFacet`, die einen einfachen textuellen Kommentar verwaltet und eine `FuzzyFacet`, die eine komplexe Fuzzy-Formel (vergleiche Definition 8.1.7 auf Seite 77) aufnehmen kann, definiert. Konzeptionell wird dies durch den Einsatz eines weiteren Entwurfsmusters realisiert, des *Dekorierers* (Decorator). Dieses Muster erlaubt es, ein Objekt dynamisch um zusätzliche Funktionalität zu erweitern: In diesem Fall wird ein Annotations-Objekt dynamisch in einem

[1] Dieses sowie die weiteren erwähnten Entwurfsmuster sind in [GHJV95] dokumentiert.

Dekorierer-Objekt eingefaßt, das dann die erweiterte Funktionalität abhängig vom Facettentyp anbietet.

Zusätzlich bieten wir einen *Iterator* an, der — aufbauend auf dem gleichnamigen Entwurfsmuster — eine einheitliche Schnittstelle zum Durchlaufen der Komponenten einer komplexen Annotation, und davon wiederum ihrer Facetten, erlaubt (siehe Abbildung 12.3 auf der nächsten Seite). Iteratoren ermöglichen es, eine komplexe Struktur auf verschiedene Weise zu durchlaufen. Da der Zustand eines Durchlaufs im Iterator gekapselt ist, läßt sich eine Struktur zum gleichen Zeitpunkt mehrfach durchlaufen, indem mehrere Instanzen eines Iterators angelegt werden.

12.2.3 Anwendbarkeit

Das Annotations-Entwurfsmuster ist anwendbar, wenn:

- eine Anwendung mit Metainformationen umgehen möchte, diese jedoch klar von der Objektstruktur getrennt sein sollen;
- Metainformationen ohne Änderung der Klassendefinition der zu annotierenden Objekte verwaltet werden sollen;
- Metainformationen dynamisch hinzugefügt und entfernt werden können sollen;
- die Strukturkomponenten des objektorientierten Datenmodells, an denen Metainformationen aufgehängt werden, frei definierbar sein sollen;
- verschiedene Arten von Metainformationen gleichzeitig für eine Strukturkomponente verwaltbar sein sollen;
- Objekte mit Annotationen transparent von Systemteilen verwendbar sein sollen, die nicht mit den Metainformationen umgehen können.

12.2.4 Struktur

Die Struktur des Annotations-Entwurfsmusters in Form eines UML-Diagramms ist in Abbildung 12.2 auf der nächsten Seite gezeigt. Den zusätzlich angebotenen Iterator zeigt Abbildung 12.3.

12.2.5 Teilnehmer

Die Funktion der einzelnen Klassen ist hierbei:

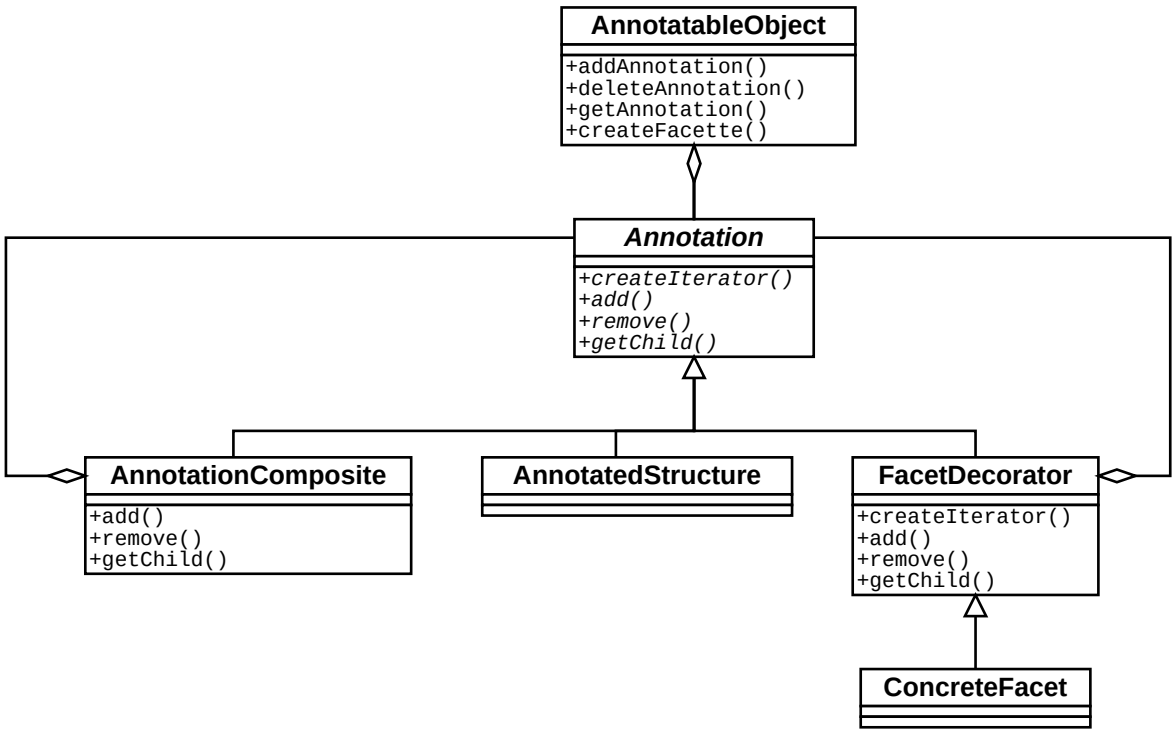


Abbildung 12.2: Struktur des Annotations-Entwurfsmusters

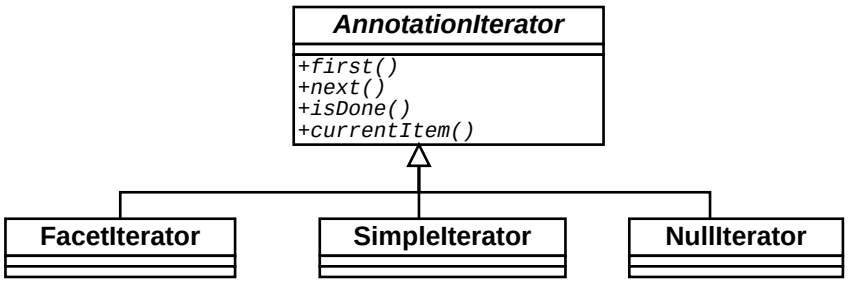


Abbildung 12.3: Iterator für Annotationen

AnnotatableObject definiert die Schnittstelle für ein annotierbares Objekt. Jedem annotierbaren Objekt können dabei beliebig viele Annotationen zugeordnet sein. Beim Einsatz des Entwurfsmusters wird man typischerweise die Oberklasse aller Objekte, die annotierbar sein sollen, von dieser Klasse erben lassen.

Annotation definiert die Schnittstelle für alle Objekte innerhalb einer Kompositumstruktur und beinhaltet die Methoden, um diese Struktur aufzubauen. Eine Annotation kann dynamisch um Facetten erweitert werden, die anwendungsspezifische Metainformationen aufnehmen.

AnnotationComposite dient zum Aufbau der Kompositumstruktur. Die Verwendung entspricht dem Entwurfsmuster Kompositum.

AnnotatedStructure ist ein Blatt innerhalb des Kompositums. Bei der Anwendung des Musters können beliebige Annotationsstrukturen (wie Attribute oder Objekte) definiert werden, die sich dann über das Kompositum zu komplexeren Strukturen zusammensetzen lassen.

FacetDecorator dient zur Erweiterung einer Annotation um Facetten. Unterklassen des `FacetDecorator` definieren die anwendungsspezifischen Typen von Metainformationen. Die Verwendung hier entspricht dem Entwurfsmuster Dekorierer.

AnnotationIterator definiert die Schnittstelle für den Zugriff auf die Facetten einer Annotation, sowie für die Navigation in zusammengesetzten Annotationsstrukturen. Die Verwendung entspricht dem Entwurfsmuster Iterator.

NullIterator implementiert eine konkrete Schnittstelle des `AnnotationIterator`. Dieser Iterator ist immer fertig mit einem Durchlauf, sein Einsatz liegt in der vereinfachten Behandlung von Grenzfällen.

SimpleIterator implementiert einen konkreten Iterator für den Durchlauf zusammengesetzter Annotationsstrukturen.

FacetIterator implementiert einen konkreten Iterator für den Durchlauf aller Facetten einer Annotation.

12.2.6 Interaktionen

Zur Speicherung von Metainformationen wird eine Annotation angelegt, die verschiedene Informationstypen in einzelnen Facetten aufnimmt. Eine Annotation wiederum bietet als Oberklasse die Schnittstelle für das dynamische Anlegen oder Entfernen von Metainformationen für Strukturkomponenten an. Die Klasse `Annotation` kombiniert dabei einen Dekorierer mit einem Kompositum: der Dekorierer erweitert eine Annotation um eine spezifische Art von Metainformation, während das Kompositum es erlaubt, neben atomaren Strukturen auch beliebige Kombinationen davon zu annotieren.

Die zusätzlich angebotenen Iteratoren erlauben es, die einzelnen Facetten einer Annotation sowie die Struktur komplexer Annotationen zu durchlaufen.

12.2.7 Konsequenzen

Das Muster ermöglicht es, zur Laufzeit Objekte eines Systems für die Aufnahme zusätzlicher Informationen zu erweitern. Es eignet sich insbesondere, um Metainformationen zu einzelnen Objekten oder Attributen zu speichern, ohne die Klassendefinition dieser Objekte selbst ändern zu müssen. Es muß lediglich eine Vererbungsbeziehung zu der Klasse `AnnotatableObject` hergestellt werden.

Das Muster unterstützt die Annotation zusammengesetzter Strukturen durch ein Kompositum und erlaubt es ferner, eine Annotation um Facetten verschiedenen Typs zu erweitern, so daß Systemteile, die unterschiedliche Arten von Metainformationen verwalten möchten, konfliktfrei koexistieren und kooperieren können.

Existierende Systemteile, die keinen Gebrauch von den Metainformationen machen können, bleiben durch den Einsatz dieses Musters unbeeinträchtigt. Gleichzeitig profitieren die Anwender des Musters von der Möglichkeit, Metainformationen direkt an ihrer Quelle aufzuhängen, sie also nicht als separate, vom eigentlichen System getrennte Hierarchie realisieren müssen.

12.2.8 Implementierung

Das Muster läßt sich mit allen gängigen objektorientierter Sprachen einsetzen. Bei der Art der möglichen Strukturkomponenten können sich aber Einschränkungen ergeben, etwa bei der Annotation einer Objektreferenz.

Das Entwurfsmuster wurde exemplarisch mit der Programmiersprache Java realisiert; nähere Hinweise zur dieser Implementierung finden sich in der vom Autor betreuten Studienarbeit [Lan97].

12.2.9 Beispielcode

Wir benutzen dieses Muster zur Einbettung von Fuzzy-Konzepten in existierende objektorientierte Systeme. In der in Abbildung 12.1 auf Seite 175 vorgestellten Fassung erlauben wir die Annotation von Attributen, Objekten sowie daraus zusammengesetzten Strukturen. Eine Annotation kann hier mit einer komplexen Fuzzy-Formel oder einer einfachen Zeichenkette versehen werden.

Für unser Beispiel zum Reengineering könnte eine mögliche Verwendung so aussehen: ein Objekt der Klasse `VarInfo` nimmt Informationen über den ermittelten Typ und Verwendungszweck einer Programmvariablen auf. Für dieses Beispiel gibt es ein skalares Attribut `typ`, sowie ein objektwertiges Attribut `zweck`. Die ermittelten Informationen über den Typ sind in einer komplexen Fuzzy-Formel gespeichert, die in einer `FuzzyComponent` abgelegt werden (die Realisierung von Fuzzy-Formeln stellen wir im nächsten Kapitel vor). Der Konstruktor einer Annotation besitzt als Parameter eine Referenz auf das zu annotierende Objekt, einen Klassennamen und optional einen Attributnamen.

Zuerst muß also die Klasse `VarInfo` als Unterklasse von `AnnotatableObject` deklariert werden:

```
class VarInfo extends AnnotatableObject {
    VarTyp     typ;
    VarZweck   zweck;
}
```

Nun ist es möglich, die geerbte Funktionalität zum Anlegen von Annotationen und Facetten zu benutzen. Das folgende Programmstück zeigt die Annotation der beiden Attribute, das Anlegen von Facetten für eine bestimmte Annotation und die Iteration durch alle Facetten einer Annotation.

```
// erzeuge Instanz der Klasse VarInfo
VarInfo varInfo = new VarInfo();
// scharfe Zustände von "typ" und "zweck"
varInfo.typ = "int";
varInfo.zweck = "Laufvariable";

// Annotiere das skalare Attribut "typ"
singleValued typAnnotation = new singleValued( varInfo, 'typ' );
varInfo.addAnnotation( typAnnotation );
// Annotiere das objektwertige Attribut "zweck"
objectValued zweckAnnotation = new objectValued( varInfo, 'zweck' );
varInfo.addAnnotation( zweckAnnotation );

// Annotiere zusätzlich das Objekt selbst
objectAnnotation varInfoAnnotation = new objectAnnotation( varInfo );
varInfo.addAnnotation( varInfoAnnotation );

// füge eine String-Facette zur Annotation des Typs hinzu
varInfo.addAnnotation( new StringFacet(
    typAnnotation, 'StringDescription', 'vermutlich int'));
// füge eine Fuzzy-Facette zur Annotations des Zwecks hinzu
varInfo.addAnnotation( new FuzzyFacet(
    zweckAnnotation, 'FuzzyDescription', fuzzyComponent ));

// iteriere durch die Facetten des Typs
AnnotationIterator typIterator = typAnnotation.createIterator();
for (typIterator.first(); !typIterator.isDone(); typIterator.next()) {
    Annotation typFacet = typIterator.currentItem();
    ...
}
```

Um auf die Annotationen eines Objektes zuzugreifen, wird die Methode `getAnnotation` benutzt. Diese verlangt als Parameter eine `Annotation` als Platzhalter für die gesuchte Annotation, hier als `dummyAnnotation` bezeichnet:

```
// finde die Annotation für das Attribut "typ"
Annotation dummyAnnotation = new singleValued( varInfo, 'typ');
gesuchteAnnotation = varInfo.getAnnotation( dummyAnnotation );
```

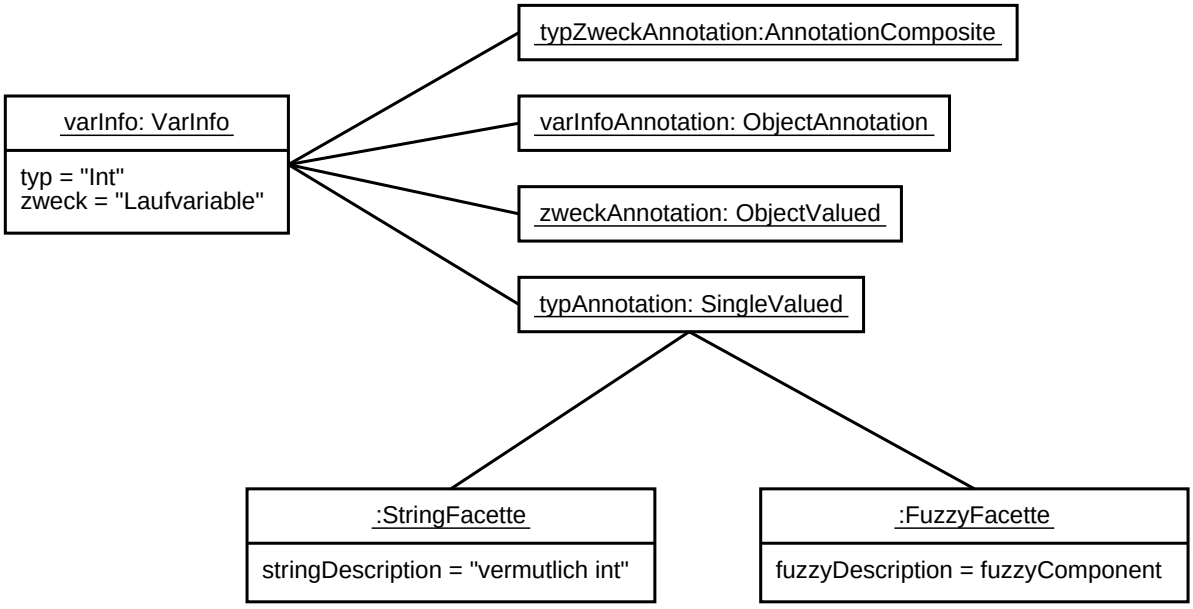


Abbildung 12.4: Objektstruktur mit Annotationen und Facetten

Möchte man mehrere Attribute gemeinsam annotieren, wird zunächst eine komplexe Annotationsstruktur aus den einzelnen Attributen aufgebaut, und diese dann genauso wie im atomaren Fall annotiert. Bei einer Annotation der Attribute `Typ` und `Zweck` gemeinsam sieht dies dann folgendermaßen aus:

```
// erzeuge eine Annotation für die Kombination
// der Attribute ”typ” und ”zweck”
AnnotationComposite typZweckAnnotation = new Annotation( varInfo );
typZweckAnnotation.add( typAnnotation );
typZweckAnnotation.add( zweckAnnotation );
varInfo.addAnnotation( typZweckAnnotation );
```

Abbildung 12.4 zeigt einen Ausschnitt aus der aufgebauten Objektstruktur.

12.2.10 Bekannte Verwendungen

Wir benutzen dieses Entwurfsmuster, um objektorientierte Systeme um Fuzzy-Techniken zu erweitern. Dabei ist ein wichtiger Aspekt die Kompatibilität mit klassischen Systemkomponenten, die mit Fuzzy-Informationen nicht umgehen können.

Eine weitere Einsatzmöglichkeit liegt im Dokumentenmanagement, wenn eine vorgegebene, komplexe Dokumentstruktur annotiert werden soll. Beispiele für solche Dokumente sind Bücher, Artikel, Webseiten oder Lehrmaterial, das zu Lernzwecken,

zum Austausch zwischen Reviewern oder für sonstige Anmerkungen annotiert werden soll. Hierbei muß wieder sowohl die Art der Annotation (textueller Kommentar, ergänzende Graphik, Internet-Adresse, Ton- oder Videodokument) als auch die annotierbare Struktur frei definierbar sein. Die Realisierung eines solchen Systems läßt sich elegant mit Hilfe unseres Annotations-Musters durchführen.

12.2.11 Verwandte Muster

Das Annotations-Entwurfsmuster gehört zur Klasse der sogenannten *Strukturmuster*. Es hat Ähnlichkeit mit dem Muster *Dekorierer* (siehe [GHJV95]), das auch innerhalb dieses Musters eingesetzt wird. Während jedoch das Ziel des Dekorierers ist, Objekte dynamisch um weitere Eigenschaften zu erweitern, geht es bei der Annotation vornehmlich darum, dynamisch die Struktur von Objekten zu erweitern.

12.3 Fazit

Mit dem in diesem Kapitel vorgestellten Annotations-Entwurfsmuster existiert nun die Möglichkeit, das Konzept der Annotation mit Facetten (siehe Abschnitt 11.2.1 auf Seite 158) in ein existierendes objektorientiertes System einzubetten. Auf diese Weise erreichen wir die oben geforderte Trennung zwischen der klassischen, scharfen Sicht eines Systems und zusätzlichen Metaebenen, die beispielsweise eine Fuzzy-Sicht beinhalten können.

Durch diese Vorgehensweise erhalten wir eine elegante Einbettungsmöglichkeit des im letzten Kapitel vorgestellten formalen objektorientierten Fuzzy-Datenmodells: Teile des objektorientierten Modells, die mit imperfekten Zuständen behaftet sein können, werden als annotierbare Strukturkomponenten aufgefaßt. Diese entsprechen einem Knoten des Fuzzy-Abhängigkeitsgraphen, wie er im theoretischen Modell definiert ist.

Was jedoch noch fehlt, sind die konkreten Realisierungen für Fuzzy-Facetten, die Fuzzy Konjunktive Normalformen aufnehmen sollen, sowie die praktische Umsetzung der Fuzzy-Abhängigkeitsgraphen mit den zugehörigen Operatoren. Dies ist Gegenstand des nächsten Kapitels.

Kapitel 13

Die Fuzzy-Bibliothek

Generally, people can only argue about things they understand.
Thus, a bunch of business executives will approve a $1 billion nuclear power plant without
any debate, but argue for hours about whether the company should spend $500 on a party.
They've all thrown parties before, so they have an idea that
maybe paper cups can be purchased for less than $2.75 per 100.
But none of them has any personal experience purchasing cooling towers.
Analogously, if you show a super-hairy transactional Web service to
a bunch of folks in positions of power at a big company, they aren't going to say,
"I think you would get better linguistics performance if you kept a
denormalized copy of the data in the Unix file system and indexed it with PLS."
Nor are they likely to opine that "You should probably upgrade to Solaris 2.6
because Sun rewrote the TCP stack to better handle 100-plus simultaneous threads."
Neither will they look at your SQL queries and say, "You could clean this up
by using the Oracle tree extensions; look up CONNECT BY in the manual."
What they will say is "I think that page should be a lighter shade of mauve."

Philip Greenspun, „Philip and Alex's Guide to Web Publishing"

Mit dem im letzten Kapitel vorgestellten Annotations-Entwurfsmuster haben wir jetzt die Möglichkeit, imperfekte Daten in ein objektorientiertes Informationssystem einzubetten. Wir sind jedoch noch den Nachweis der Realisierbarkeit für die in Teil III entwickelten theoretischen Konzepte schuldig.

Dieses Kapitel widmet sich daher der Implementierung dieser theoretischen Konzepte. Dazu gehört die Realisierung von Fuzzy-Mengen (Abschnitt 13.1 auf der nächsten Seite), Fuzzy Konjunktiven Normalformen (Abschnitt 13.2 auf Seite 187), sowie der Fuzzy-Abhängigkeitsgraphen (Abschnitt 13.3 auf Seite 190). Bei der Beschreibung verwenden wir eine an das Schema zur Beschreibung von Entwurfsmustern angelehnte Gliederung. Für die Klassendiagramme wird durchgängig UML [FS00] verwendet.

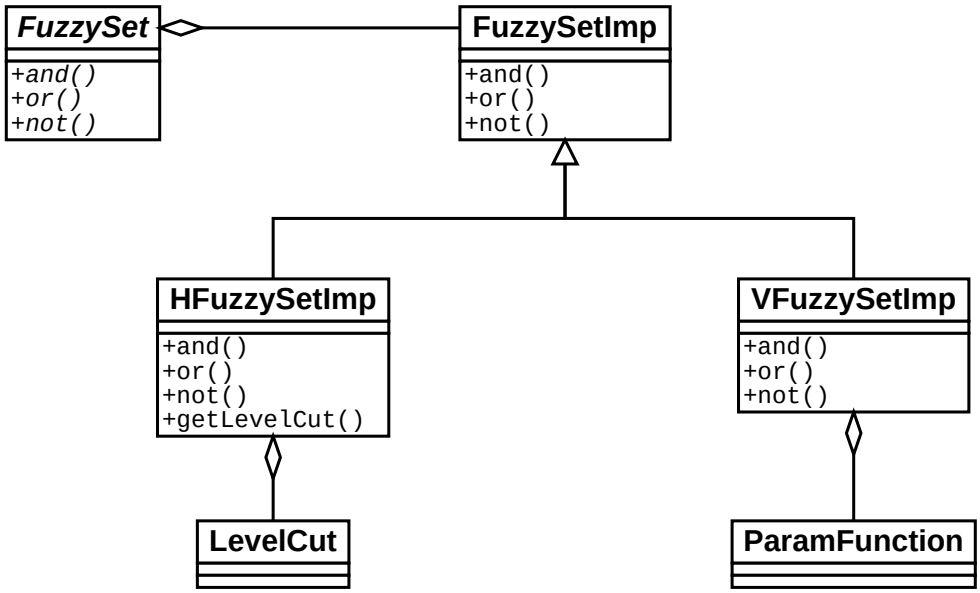


Abbildung 13.1: Realisierung von Fuzzy-Mengen

13.1 Fuzzy-Mengen

Der Grundbaustein des theoretischen Modells, und damit auch der Implementierung, ist die Fuzzy-Menge.

Bei ihrer Implementierung gibt es eine Reihe von Randbedingungen. Wie bereits in Abschnitt 6.2 auf Seite 52 diskutiert, gibt es verschiedene Möglichkeiten zur Repräsentation von Fuzzy-Mengen. Der Entwurf soll dabei sowohl die horizontale, als auch die vertikale Repräsentationsform unterstützen. Für einen Nutzer der Klasse soll es dabei jedoch transparent bleiben, welche Form intern eingesetzt wird.

Eine solche Trennung von Schnittstelle und Implementierung erlaubt das hier eingesetzte Entwurfsmuster *Brücke* (Bridge).

13.1.1 Struktur und Teilnehmer

Die Modellierung einer Fuzzy-Menge als UML-Diagramm zeigt Abbildung 13.1. Sie besteht aus den folgenden Klassen:

FuzzySet definiert die Abstraktion der Schnittstelle für Fuzzy-Mengen. Dabei hält sie eine Referenz auf das Implementierer-Objekt `FuzzySetImp`.

FuzzySetImp definiert die Schnittstelle für die Implementierungsklassen von Fuzzy-Mengen. Man beachte, daß das verwendete Muster Brücke es erlaubt, innerhalb einer Implementierung eine andere Schnittstelle als die der Klasse `FuzzySet` zu benutzen.

HFuzzySetImp bietet eine konkrete Implementierung von Fuzzy-Mengen in horizontaler Repräsentation.

LevelCut stellt einen α-Schnitt bereit, der innerhalb der horizontalen Repräsentation verwendet wird.

VFuzzySetImp definiert eine konkrete Implementierung von Fuzzy-Mengen in vertikaler Repräsentation.

ParamFunction ist die abstrakte Oberklasse für Fuzzy-Mengen in vertikaler Repräsentation. Ihre Unterklassen realisieren konkrete Funktionen, wie sie in Abschnitt 6.2.1 auf Seite 53 beschrieben sind.

13.1.2 Konsequenzen

Durch den Einsatz des Entwurfsmusters Brücke wird eine Entkopplung der Schnittstelle einer Fuzzy-Menge von ihrer Implementierung erreicht. Dabei ist eine Implementierung nicht fest an eine Schnittstelle gebunden; es ist für eine Fuzzy-Menge sogar möglich, zur Laufzeit ihre Implementierung zu ändern. Dies ist insbesondere erforderlich, wenn Fuzzy-Mengen in vertikaler Repräsentation kombiniert werden. Beispielsweise ergibt die Verknüpfung zweier dreiecksförmiger Fuzzy-Mengen (vergleiche Definition 6.2.1 auf Seite 53) üblicherweise keine neue dreiecksförmige Fuzzy-Menge.

Zukünftige Änderungen oder Erweiterungen lassen sich getrennt sowohl an der abstrakten Schnittstelle als auch an der Implementierung durchführen. Ein mögliche Erweiterung wäre beispielsweise die Implementierung von intervallwertigen Fuzzy-Mengen.

Die Details der Implementierung bleiben dem Klienten verborgen, da dieser normalerweise nur auf die Abstraktion zugreifen muß. Das Ändern einer Implementierungsklasse von Fuzzy-Mengen erfordert so keine Neuübersetzung der Abstraktionsklasse oder deren Klienten. Diese Eigenschaft ist wichtig, wenn Binärkompatibilität zwischen verschiedenen Versionen der Fuzzy-Bibliothek sichergestellt werden muß.

13.1.3 Beispielcode

Als Beispiel betrachten wir eine Fuzzy-Menge, wie sie im geschilderten Anwendungsfall des Reengineering auftreten kann (siehe Abbildung 13.2 auf der nächsten Seite).

Gespeichert werden soll die Fuzzy-Menge in horizontaler Darstellung, also in Form von α-Schnitten (vergleiche Abschnitt 6.2.2 auf Seite 55). Wichtige Parameter bei der technischen Umsetzung sind dafür die Anzahl und Werte der Niveaumengen $[\mu]_\alpha$,

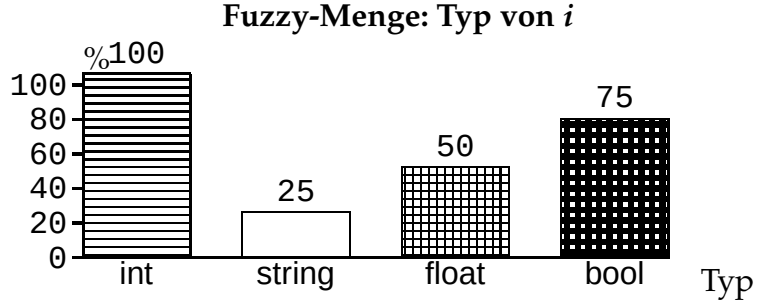


Abbildung 13.2: Beispiel für eine Fuzzy-Menge aus dem Anwendungsfall Reengineering

sowie die Angabe, ob die Speicherung der Fuzzy-Menge in Form von strengen α-Schnitten erfolgen soll.

Der folgende Programmausschnitt zeigt die Speicherung der Fuzzy-Menge in einem Objekt `typFuzzySet`. Nach der Definition der erforderlichen Parameter wird mit Hilfe eines Konstruktors ein Implementierungsobjekt `typImp` zur Aufnahme der Fuzzy-Menge in horizontaler Repräsentation erzeugt. Danach werden die Werte der einzelnen α-Schnitte eingetragen:

```
double[] typLevels = {0.25, 0.5, 0.75, 1.0};
boolean typStrict = true;
HFuzzySetImp typImp = new HFuzzySetImp( typLevels, typStrict);

// Werte der α-Schnitte eintragen
typImp.multipleAddValue( 1.0, 'int');
typImp.multipleAddValue( 0.25, 'string');
typImp.multipleAddValue( 0.5, 'float');
typImp.multipleAddValue( 0.75, 'bool');

// Fuzzy-Mengen-Objekt mit Implementierung
// in horiontaler Repräsentation erzeugen
FuzzySet typFuzzySet = new FuzzySet( typImp );
```

Mit dieser Fuzzy-Menge lassen sich durch Anwendung der Operatoren neue Fuzzy-Mengen bilden. Die Berechnung der Negation dieser Fuzzy-Menge und die Oder-Verknüpfung mit einer zweiten Fuzzy-Menge beispielsweise sieht folgendermaßen aus:

```
// Negation der Fuzzy-Menge "typFuzzySet" bestimmen
FuzzySet notTypFuzzySet = typFuzzySet.not();

// Oder-Verknüpfung der Fuzzy-Menge "typ" mit
```

```
// einer Fuzzy-Menge "typ2FuzzySet" berechnen
FuzzySet oderFuzzySet = typFuzzySet.or( typ2FuzzySet );
```

13.2 Fuzzy Konjunktive Normalformen

Fuzzy-Mengen bilden die Grundlage für die Bildung von komplexeren Strukturen wie Fuzzy-Literalen, Fuzzy-Klauseln und Fuzzy-Formeln (vergleiche Abschnitt 8.1 auf Seite 71). Die Komposition einer solchen komplexen Aussage aus den einzelnen Bausteinen ergibt eine Baumstruktur, die eine Teil-Ganzes-Hierarchie repräsentiert. Das Entwurfsmuster *Kompositum* (Composite) kann zur Modellierung solcher Hierarchien eingesetzt werden. Es erlaubt dabei dem Klienten, sowohl einzelne Objekte (wie eine Fuzzy-Menge) als auch Kompositionen dieser Objekte (wie eine Fuzzy-Formel) einheitlich zu behandeln.

13.2.1 Struktur und Teilnehmer

Abbildung 13.3 auf der nächsten Seite zeigt die Modellierung von Fuzzy-Literalen, -Klauseln und -Formeln als UML-Klassendiagramm.

Im Kompositionsmuster deklariert dabei die abstrakte Klasse `FuzzyComponent` die Schnittstelle für Objekte in der zusammengesetzten Struktur.

Interessant an diesem Einsatz des Entwurfsmusters ist die abstrakte Klasse `FuzzyComposite`, die die Oberklasse für mögliche zusammengesetzte Objekte bildet. Ihre konkreten Unterklassen sind allerdings nicht völlig homogen in ihrer Bedeutung. Ein `Literal` (Fuzzy-Literal) hat zum Beispiel genau eine zugeordnete `FuzzyComponent` und nicht beliebig viele wie eine `Formula` (Fuzzy-Formel). Trotzdem ist das Entwurfsmuster angebracht, um die vorhandenen Gemeinsamkeiten zu modellieren: alle Fuzzy-Komponenten besitzen eine zugehörige Interpretation als Fuzzy-Menge und alle zusammengesetzten Klassen bieten dieselbe Schnittstelle zur Verwaltung ihrer Kinder. Die Modellierung betont damit die vielen strukturellen Gemeinsamkeiten und nicht die wenigen Unterschiede.

FuzzyComponent definiert die Schnittstelle für alle Objekte der Komposition. Sie implementiert das Standardverhalten für die Schnittstelle: zum Beispiel ist die Methode `getFuzzySet()` für alle Unterklassen gleich und wird deshalb hier implementiert. Ebenfalls an dieser Stelle ist die Schnittstelle zum Zugriff auf und zur Verwaltung von Kindobjektkomponenten definiert.

FuzzySet repräsentiert das Blatt-Objekt der Komposition; in unserem Modell ist dies eine einzelne Fuzzy-Menge, wie sie in Abschnitt 13.1 auf Seite 184 modelliert wurde.

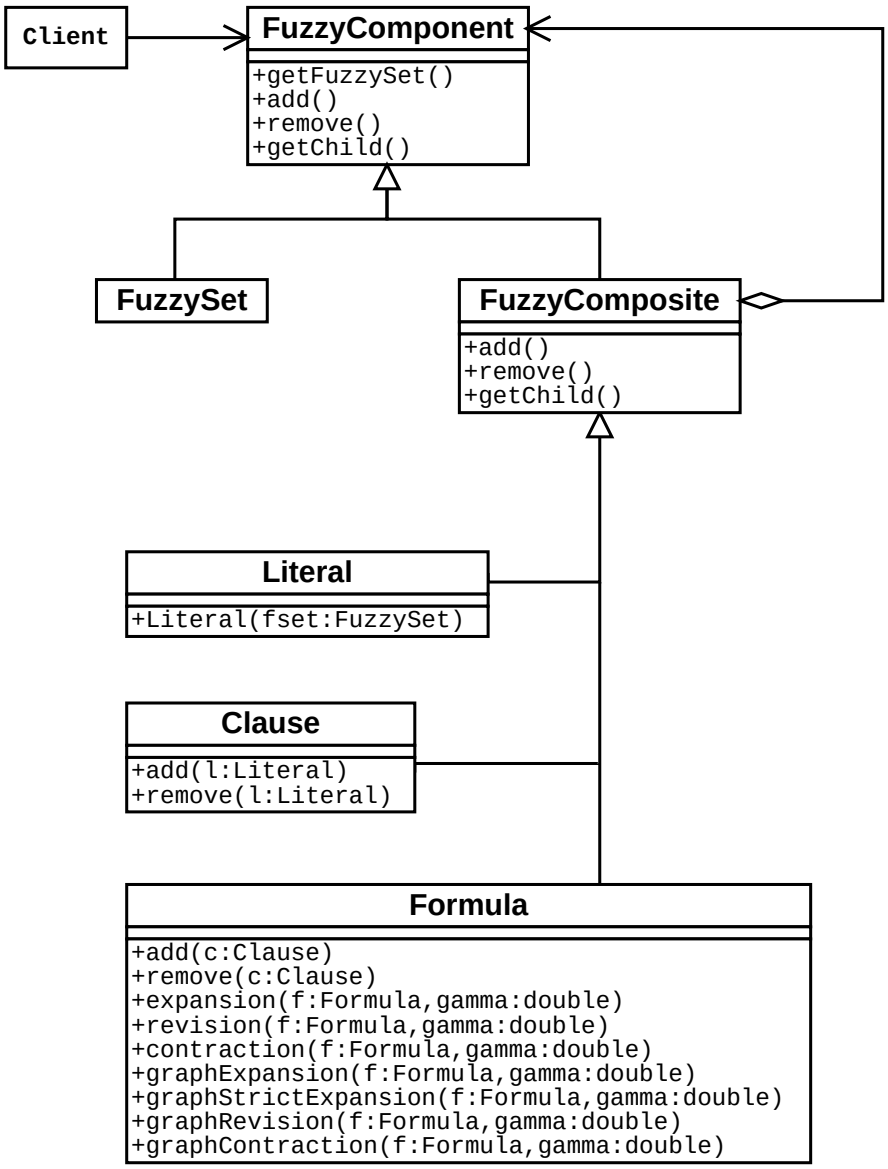


Abbildung 13.3: Modellierung von Fuzzy Konjunktiven Normalformen

FuzzyComposite definiert das Verhalten für Komponenten, die ihrerseits Kindobjekte haben können. Ein `FuzzyComposite` speichert diese Kindobjektkomponenten und implementiert die kindobjektbezogenen Operationen der Schnittstelle von Komponenten. `FuzzyComposite` ist eine abstrakte Klasse; konkrete Kompositionsobjekte sind in deren Unterklassen definiert.

Literal definiert das Verhalten für ein Fuzzy-Literal (vergleiche Definition 8.1.3 auf Seite 74) als ein konkretes Kompositum. Ein Fuzzy-Literal hat genau eine Fuzzy-Menge als Kindkomponente.

Clause definiert das Verhalten für eine Fuzzy-Klausel (vergleiche Definition 8.1.4 auf Seite 75). Eine Fuzzy-Klausel hat beliebig viele Fuzzy-Literale als Kindkomponenten.

Formula definiert das Verhalten für eine Fuzzy-Formel (vergleiche Definition 8.1.7 auf Seite 77). Eine Formel hat beliebig viele Klauseln als Kindkomponenten.

Hier sind auch die konsistenzerhaltenden Operationen γ-Expansion, γ-Revision und γ-Kontraktion definiert (vergleiche Kapitel 10). Überdies implementiert die Klasse `Formula` die entsprechenden Operationen für Fuzzy-Abhängigkeitsgraphen (siehe auch Abschnitt 13.3).

Fuzzy-Atome sind hier nicht explizit modelliert, sondern aus Effizienzgründen mit der Realisierung von Fuzzy-Literalen zusammengefaßt.

13.2.2 Konsequenzen

Das Kompositionsmuster definiert Klassenhierarchien, bestehend aus primitiven und zusammengesetzten Objekten. An jeder Stelle, an der ein Klient ein primitives Objekt erwartet, kann auch ein zusammengesetztes Objekt erscheinen. Dadurch wird die Implementierung von Klienten vereinfacht: sie können zusammengesetzte Strukturen und individuelle Objekte einheitlich behandeln.

Ebenso ermöglicht es diese Modellierung, einfach neue Arten von Komponenten hinzuzunehmen. Neu definierte Unterklassen arbeiten automatisch mit existierenden Strukturen und Klienten zusammen.

Die Beschränkung der Kindobjektkomponenten eines zusammengesetzten Objektes auf bestimmte Typen von Kompositionen wird allerdings schwieriger. Zum Beispiel akzeptiert eine Fuzzy-Formel nur Fuzzy-Klauseln als Kinder. Zur Überprüfung muß man daher Laufzeit-Tests einführen.

13.2.3 Beispielcode

Als Einsatzbeispiel betrachten wir Anforderungen zur Auswahl eines Autotyps, wie wir sie später in unserem Fuzzy-Entscheidungshilfesystem verwenden werden. Wie schon zuvor soll der Motor eines auszuwählenden Fahrzeuges die folgenden Eigenschaften erfüllen:

$$\text{leistungsstark} \wedge (\text{sparsam} \vee \text{biologisch})$$

Als Grundlage stehen bereits Objekte vom Typ `FuzzySet` zur Modellierung der Konzepte *leistungsstark* (hohe Motorleistung), *sparsam* (im Verbrauch) und *biologisch* (arbeitet mit Biotreibstoff) zur Verfügung.

Aus diesen Bausteinen soll nun eine Fuzzy-Formel aufgebaut werden, die die oben gezeigte Bedingung repräsentiert. Das folgende Programmfragment erzeugt die notwendigen Objekte und Strukturen:

```
// die Literale zu den Fuzzy-Mengen erzeugen
Literal myLiteral1 = new Literal( leistungsstarkFuzzySet );
Literal myLiteral2 = new Literal( sparsamFuzzySet );
Literal myLiteral3 = new Literal( biologischFuzzySet );

// Klauseln erzeugen
Clause myClause1 = new Clause();
Clause myClause2 = new Clause();

// Literal in die erste Klausel einfügen
myClause1.add( myLiteral1 );
// Literale in die zweite Klausel einfügen
myClause2.add( myLiteral2 );
myClause2.add( myLiteral3 );

// Formel erzeugen
Formula myFormula = new Formula();
// Klauseln in die Formel einfügen
myFormula.add( myClause1 );
myFormula.add( myClause2 );
```

Als Ergebnis steht die Bedingung in dem Fuzzy-Formelobjekt `myFormula` zur Verfügung.

13.3 Fuzzy-Abhängigkeitsgraphen

Das nächste, der Fuzzy-Formel übergeordnete, Konzept ist der Fuzzy-Abhängigkeitsgraph (vergleiche Abschnitt 8.5.1 auf Seite 85). In einem objektorientierten Fuzzy-

Informationssystem entspricht ein Knoten eines solchen Abhängigkeitsgraphen einer annotierbaren Komponente des objektorientierten Datenmodells (vergleiche Abschnitt 11.2.5 auf Seite 164).

Tatsächlich besitzen wir mit den oben vorgestellten Entwürfen bereits die komplette Infrastruktur, um Abhängigkeitsgraphen zu verwalten: sie können einfach als weitere Facette einer annotierbaren Komponente gespeichert werden, die eine Menge der von dieser Annotation abhängigen Knoten aufnimmt. Der Graph wird so in Adjazenzlistendarstellung repräsentiert.

Diese Modellierung ist insbesondere deshalb vorteilhaft, weil bei der Verarbeitung eines Knotens die benötigten Informationen zur Bestimmung der Abhängigkeiten sofort aus der gleichen Annotation geholt werden können. Bei einer Graph-γ-Revision eines Objekt-Attributes beispielsweise würde man neben der benötigten Fuzzy-Facette gleich die Abhängigkeits-Facette aus der Annotation dieses Attributes lesen, um damit die Operation durchzuführen.

13.3.1 Beispielcode

Der folgende Programmcode zeigt beispielhaft, wie eine Abhängigkeit von einem Knoten, der Annotation auf dem Attribut `typ`, zu einem zweiten Knoten, der Annotation auf dem Attribut `zweck` hergestellt werden kann (siehe auch das Beispiel in Abschnitt 8.5 auf Seite 85). Wir führen dazu das im letzten Kapitel gezeigte Beispiel (vergleiche Abschnitt 12.2.9 auf Seite 179) fort:

```
VarInfo varInfo = new VarInfo();

// Annotiere das skalare Attribut "typ"
singleValued typAnnotation = new singleValued( varInfo, 'typ' );
varInfo.addAnnotation( typAnnotation );

// Annotiere das objektwertige Attribut "zweck"
objectValued zweckAnnotation =
    new objectValued( varInfo, 'zweck' );
varInfo.addAnnotation( zweckAnnotation );

// Erzeuge Facette "dependencies" zur Verwaltung
// der Abhängigkeiten vom Knoten "typ"
DependencyFacet typDependencies = new DependencyFacet(
    typAnnotation, 'Dependencies', NULL));

// Füge Abhängigkeit "typ" → "zweck" hinzu
typDependencies.addNode( zweckAnnotation );
// Füge Abhängigkeits-Facette zu "typ" hinzu
varInfo.addAnnotation( typDependencies );
```

Abbildung 13.4: Realisierung der globalen Informationen

13.4 Globale Informationen

Innerhalb eines Fuzzy-Informationssystems gibt es eine Reihe von Informationen, die über die einzelnen Stufen und Komponenten hinweg benötigt werden. Dazu gehören beispielsweise voreingestellte Werte für die Anzahl und Grade der α-Schnitte von Fuzzy-Mengen in horizontaler Repräsentation.

Solche Informationen sollten in einheitlicher Weise zur Verfügung stehen. Wir bieten daher eine zusätzliche Klasse an, die die Verwaltung solcher *globaler Information* übernimmt. Zur Realisierung dieser globalen Informationen wird das Entwurfsmuster *Singleton* eingesetzt. Es garantiert, daß innerhalb einer Anwendung nur eine Objekt-Instanz der jeweiligen Singleton-Klasse existiert.

Abbildung 13.4 zeigt die Klasse `GlobalInfo` zur Verwaltung dieser Informationen. Werden für eine Anwendung zusätzliche Informationen benötigt, die systemweit zur Verfügung stehen sollen, läßt sich dies durch die Definition einer Unterklasse erreichen. Der Zugriff auf die globalen Informationen erfolgt dabei ausschließlich über die `Exemplar()`-Methode. Damit sichergestellt ist, daß zur Laufzeit nur ein Objekt dieser Klasse erzeugt werden kann, muß der Konstruktor *geschützt* (protected) sein.

13.5 Implementierung

Eine Beispielimplementierung dieser Konzepte erfolgte in der Programmiersprache *Java*. Das gesamte Projekt (GURU v3) unterteilt sich dabei in die folgenden Einzelprojekte:

FuzzyLib enthält die Pakete zur Realisierung des im letzten Kapitel vorgestellten Annotations-Entwurfsmusters, sowie die in diesem Kapitel vorgestellte Fuzzy-Bibliothek.

FuzzyGUI enthält Pakete mit Komponenten zur Entwicklung von graphischen Oberflächen für Fuzzy-Anwendungen. Diese werden benötigt, um die hier vorgestellten Konzepte einen Anwender präsentieren zu können. Auf sie wird im nächsten Teil dieser Arbeit eingegangen.

FuzzyTools enthält eine Reihe von Werkzeugen zur Entwicklung von Fuzzy-Informationssystemen, auf die wir ebenfalls im nächsten Teil dieser Arbeit noch eingehen werden.

FuzzyApp enthält komplette Fuzzy-Informationssysteme, die mit dem hier vorgestellten Modell entwickelt wurden. Auf zwei Anwendungen werden wir in Teil VI dieser Arbeit eingehen.

Zu einzelnen Teilen dieser Implementierung sind weitere Informationen in den vom Autor betreuten Studienarbeiten [Lan97] und [Chr97] zu finden.

Dies beendet Teil IV dieser Arbeit. Im nächsten Teil betrachten wir die Gesamtarchitektur einsatzfähiger Fuzzy-Informationssysteme (siehe Abbildung 13.5 auf der nächsten Seite).

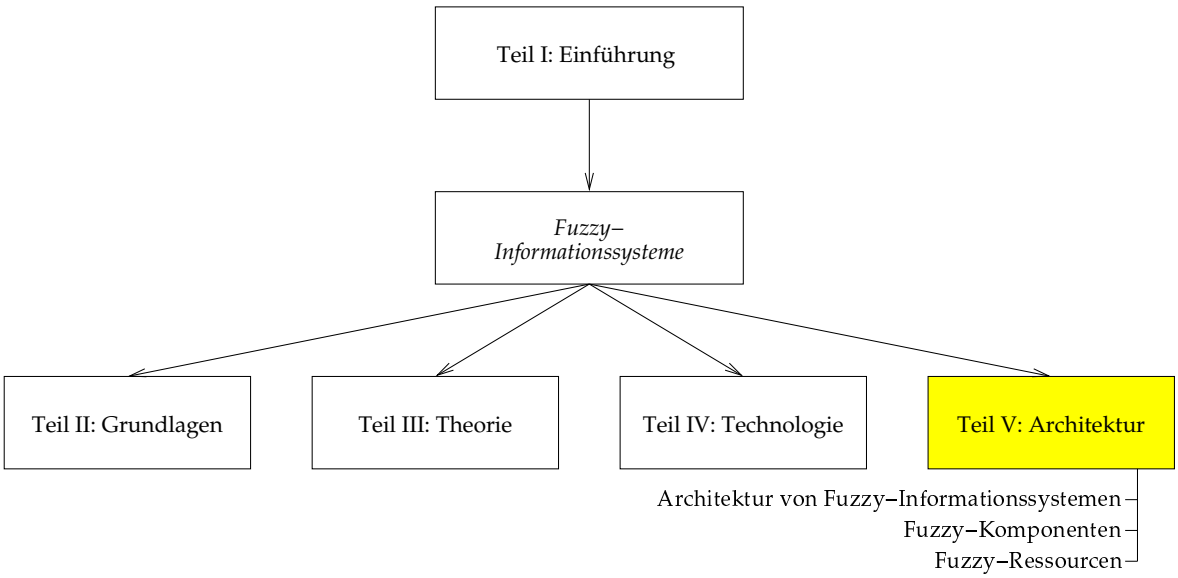


Abbildung 13.5: Gliederung Teil V

Teil V

Architektur von Fuzzy-Informationssystemen

Kapitel 14

Architektur von Fuzzy-Informationssystemen

I didn't build it.
All I did was draw the architectural models
[. . .] an architect doesn't do an engineer's work.

Roger Taillebert, architect of the Montréal Olympic Stadium,
in a press conference after a 55-ton concrete beam fell off

Die Komplexität von Informationssystemen wächst ständig. Wie schon in der Einleitung erwähnt wurde, spiegelt sich diese Komplexitätssteigerung auch in der Notwendigkeit einer zunehmenden Abstraktion bei der Systemkonzeption wider. Erreicht wird dies heute durch die Definition sogenannter Referenzarchitekturen, die es durch ihren hohen Abstraktionsgrad ermöglichen, die steigende Komplexität der Anwendungsentwicklung beherrschbar zu machen.

In dieser Arbeit wird ein Hauptschwerpunkt auf die praktische Einsatzfähigkeit eines Fuzzy-Modells gelegt. Wir haben im letzten Teil dieser Arbeit gezeigt, wie ein objektorientiertes Datenmodell systematisch zur Aufnahme von Fuzzy-Informationen erweitert werden kann. Folglich müssen wir auch für die Entwicklung von Fuzzy-Informationssystemen eine geeignete Referenzarchitektur anbieten.

Dieser Teil der Arbeit widmet sich daher der Frage, wie auf der Ebene der Referenzarchitekturen systematisch Erweiterungen zur Entwicklung von Fuzzy-Informationssystemen angeboten werden können, um damit sowohl unsere Anforderung an ein DURCHGÄNGIGES VERFAHREN (vergleiche Abschnitt 3.2.2 auf Seite 28), als auch an eine MODERNE ARCHITEKTUR (siehe Abschnitt 3.2.1 auf Seite 27) zu erfüllen.

Unsere Vorgehensweise ist, zunächst eine abstrakte Referenzarchitektur für Informationssysteme aufzustellen. Hierfür stellen wir im nächsten Abschnitt zunächst die grundlegenden Architekturprinzipien vor. Darauf aufbauend erfolgt der Entwurf

unserer Referenzarchitektur. Wir zeigen überdies, mit welchen konkret verfügbaren Technologien dieser abstrakte Architekturrahmen ausgefüllt werden kann. Anschließend untersuchen wir anhand dieser Referenzarchitektur die für Fuzzy-Informationssysteme notwendigen Architekturänderungen und fassen diese in einer erweiterten Fuzzy-Referenzarchitektur zusammen. Ein Abschnitt über den systematischen Entwurf der konkreten Architektur eines Fuzzy-Informationssystems schließt dieses Kapitel ab; die beiden nachfolgenden Kapitel widmen sich dann der technischen Umsetzung der angesprochenen Erweiterungen.

14.1 Referenzarchitekturen

Der Begriff der *Architektur* von Informationssystemen hat in den letzten Jahren immer mehr an Bedeutung gewonnen. Während es schon länger üblich ist, den konkreten Aufbau eines Systems in Form einer abstrakten Architektur zu beschreiben, ist der neue Aspekt hier der methodische Entwurf einer solchen Architektur [HNS00]:

> *Software architecture is a recently emerged technical field, but it's not a new activity [...] However, now the consensus is that what these designers do is qualitatively different from other software engineering activities, and we've begun figuring out how they do it and how we can teach others to do it.*

Dabei wird der Entwurf von Software-Architekturen erstmals als eigenständiges Problem, losgelöst von einer konkreten Anwendung, betrachtet. Dies führt zu der Entwicklung von *Architekturrahmen* (auch *Referenzarchitekturen* oder *Blueprints* genannt), die dann für die Definition der konkreten Architektur eines Systems herangezogen werden.

Im Rahmen der Softwareentwicklung läßt sich die Entwicklung solcher Referenzarchitekturen als weiterer Abstraktionsschritt verstehen, ähnlich wie Entwurfsmuster dies bei der objektorientierten Modellierung erreicht haben, indem sie erfolgreiche Lösungen in abstrahierter Form dokumentieren.

Insbesondere für den Einsatz und die Entwicklung von Informationssystemen in Unternehmen hat der Begriff der Referenzarchitektur in den letzten Jahren eine große Bedeutung gewonnen. Denn hier spielt nicht nur der Aspekt eine Rolle, daß durch die Vorgabe eines Architekturmodells das Wissen erfahrener Architekten in einzelne Projekte getragen werden kann. Ein weiterer Vorteil ist, daß sich auf diese Weise eine Standardisierung erreichen läßt, so daß nicht jedes neue Projekt andere Technologien und Produkte in ein Unternehmen einbringen kann.

Mit anderen Worten, die im Rahmen einer Anwendungsentwicklung klassischerweise entwickelte Spezifikation reicht nicht dazu aus, ein Informationssystem in allen Dimensionen festzulegen, da verschiedene Architekturen zu seiner Implementierung

verwendet werden können. Gerade diese Architektur erweist sich jedoch mit steigender Systemkomplexität als kritischer Punkt bei der Realisierung [Edw99]:

> *Today, we use* tiers *to describe the logical partitioning of an application across clients and servers. [...] This partitioning is a central design issue that makes a big difference in determining the success of mission-critical applications.*
> *Today, application designers and developers are making architectural mistakes that cost millions of dollars. Applications most often fail because of these errors — not because of coding problems. While you can readily fix bugs, a project can almost never recover from a fundamental architectural flaw.*

Während der Begriff „Architektur" in anderen Wissenschaften eine lang etablierte Tradition hat, steht das Gebiet der Software-Architektur aber noch in den Anfängen, wie etwa in [HNS00] bemerkt wird:

> *Architecture is one of the oldest arts and one of the most ancient engineering disciplines. [...] Although many architectural specializations (e.g., building architecture, naval architecture, aerospace architecture) are mature disciplines, software architecture is still in its nascency.*

Doch diese Neuheit darf keine Entschuldigung dafür sein, die Architektur eines Software-Systems nur unsystematisch anzugehen, wie die Autoren weiter ausführen:

> *The common excuses for software architecture's immaturity are its youth and uniqueness. Although these excuses may have played well a decade or two ago, they are less convincing today. The software industry is no longer callow, and the stakes are no longer trivial. The industry is half-century old, and much of our new "information age" economy relies on its products.*

Obwohl kein Mangel an Literatur zum Thema Software-Architektur besteht [BCK98, Edw99, OHE99, HNS00, RMB01, SSJt02], gibt es seitens der Informatik noch keine formale Behandlung dieses Gebietes. So existiert noch kein Konsens, was eine solche Architektur ausmacht und wie sie formalisiert werden könnte. Ein typisches Beispiel für eine von vielen pragmatischen Definitionen ist [BCK98]:

> *The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them.*

Wir verwenden im folgenden den Begriff *Architektur* für die Struktur eines konkreten Informationssystems und den Begriff *Referenzarchitektur* für eine Sammlung von Architekturbestandteilen, von der dann die Architektur einer konkreten Anwendung anhand ihrer Anforderungen abgeleitet werden kann.

Bevor wir jedoch unsere Referenzarchitektur für Fuzzy-Informationssysteme vorstellen, beginnen wir mit der Einführung der wichtigsten Prinzipien bei der Entwicklung von Referenzarchitekturen.

14.1.1 Architekturprinzipien

Dominantes Grundprinzip heutiger Informationssystem-Architekturen ist der Begriff des *Client/Server-Computing*. Dieses Prinzip dient der logischen und physikalischen Trennung und Entkopplung der einzelnen Bestandteile eines Systems. Einem Architekten bietet diese Entkopplung zwei wesentliche Vorteile: Zum einen ermöglicht sie neue Freiheitsgrade beim Entwurf eines Systems. Diese sind dringend notwendig, weil die heutigen Anforderungen an ein Informationssystem nicht nur immer anspruchsvoller werden, sondern dabei oft auch miteinander konkurrieren. Und zum anderen ermöglicht es diese Trennung, flexiblere und gleichzeitig robustere Architekturen zu entwerfen, da durch die Entkopplung nur noch genau identifizierbare Teile eines Systems von Anforderungsänderungen betroffen sind.

Dabei stellt sich jedoch die Frage, nach welchen Prinzipien man die einzelnen Bestandteile eines Systems voneinander trennen kann, und wie diese Einzelteile miteinander wechselwirken sollen, um die Gesamtfunktionalität zu bilden. Wir stellen die wichtigsten Architekturkonzepte in den nachfolgenden Unterabschnitten vor: Schichtenarchitekturen, Stufenarchitekturen und Komponentenarchitekturen.

Schichtenarchitekturen

Eine Schichtenarchitektur zerlegt ein komplexes System in eine Reihe einfacherer *Schichten* (layer), die jeweils eine genau definierte Teilaufgabe übernehmen [LKK93].

Abbildung 14.1 auf der nächsten Seite zeigt das Prinzip: jede Schicht bietet der ihr übergeordneten Schicht eine Reihe von Diensten an und nimmt dafür selbst Dienste der ihr untergeordneten Schicht in Anspruch. Dabei dürfen einzelne Schichten nicht übersprungen werden: Dies verhindert, daß sich Änderungen in einer Schicht unkontrolliert über die gesamte Architektur ausbreiten können. So wird es zum Beispiel möglich, die Implementierung einer Schicht komplett auszutauschen, ohne daß dies Auswirkungen auf andere als die unmittelbar benachbarten Schichten hat.

Die Aufgabe des Architekten ist hier, eine geeignete Zerlegung zu finden, die eine komplexe Aufgabe dergestalt auf eine Menge von Schichten abbildet, daß die resultierende Architektur möglichst wenig Schichten enthält (denn jede Schicht verursacht in der Implementierung Leistungseinbußen), dabei aber die einzelnen Schichten noch gut realisierbar bleiben, also nicht zu komplex sind.

Dies ist das klassische Schichtenprinzip nach akademischer Definition. In der Praxis finden sich häufig Abwandlungen von diesem Grundprinzip:

- Manche Architekturen erlauben es, Schichten zu überspringen. Oft ist es beispielsweise erlaubt, durch Umgehung der Schichtenhierarchie direkt auf eine Basisschicht zuzugreifen.

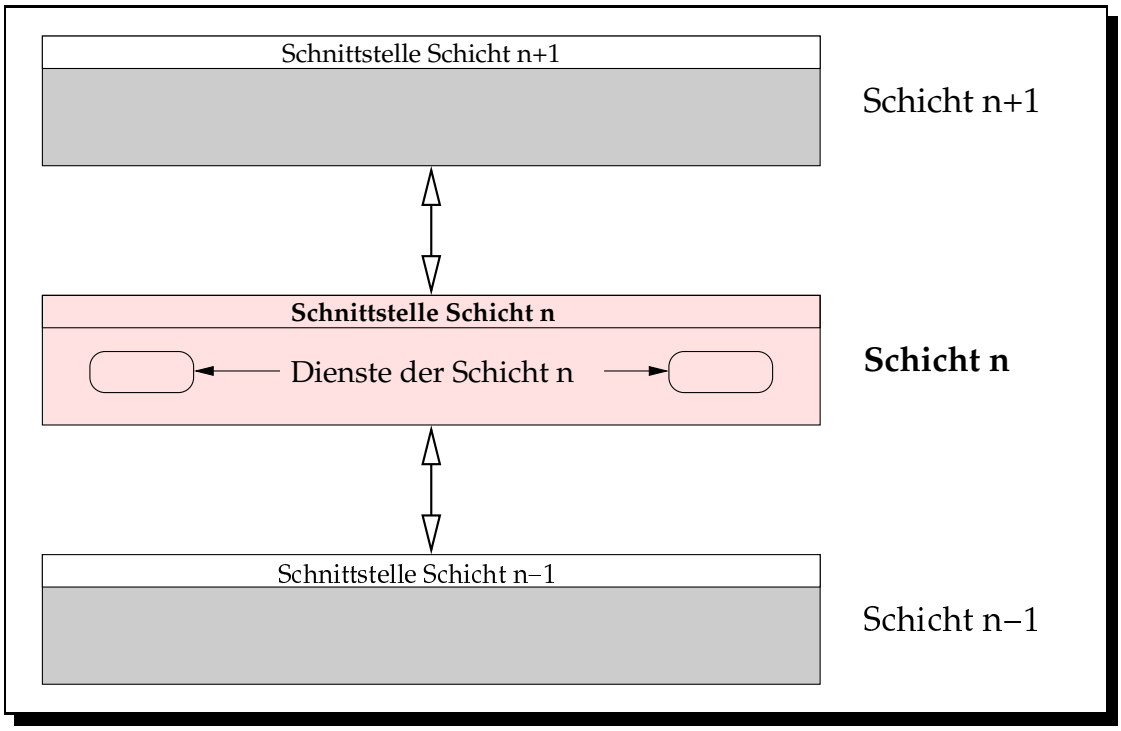


Abbildung 14.1: Aufbau einer Schicht

- Schichten können selbst wieder nach dem Schichtenprinzip strukturiert sein, also Unterschichten *(sub-layer)* beinhalten.
- Auf einer Schicht können parallel mehrere getrennte Schichtenhierarchien aufsetzen und so eine „Schornsteinarchitektur“ bilden. Dies geschieht, wenn funktional stark unterschiedliche Leistungen erbracht werden sollen, die auf einer Basisschicht aufsetzen, aber nicht sinnvoll in einer gemeinsamen Hierarchie untergebracht werden können.

Solche gewollten Verletzungen des Schichtenprinzips geschehen meist aus pragmatischen Gründen, etwa zur Leistungssteigerung eines Systems. Zum Teil verliert man jedoch dadurch die Vorteile, die einem die Schichtenarchitektur ursprünglich bietet.

Ein klassisches Beispiel für eine Schichtenarchitektur ist das Referenzmodell für Datenbankmanagementsysteme [LKK93]. Die dort anfallenden Aufgaben zur Umsetzung von deklarativen Anfragen auf einem Datenmodell bis hinunter zu physikalischen Lese-/Schreiboperationen auf Geräteebene erfordern einen enormen Abstraktionsgrad, der konzeptionell durch die Einführung mehrerer Schichten wie Dateiverwaltung, Segmentverwaltung und Anfragebearbeitung bewältigt wird.

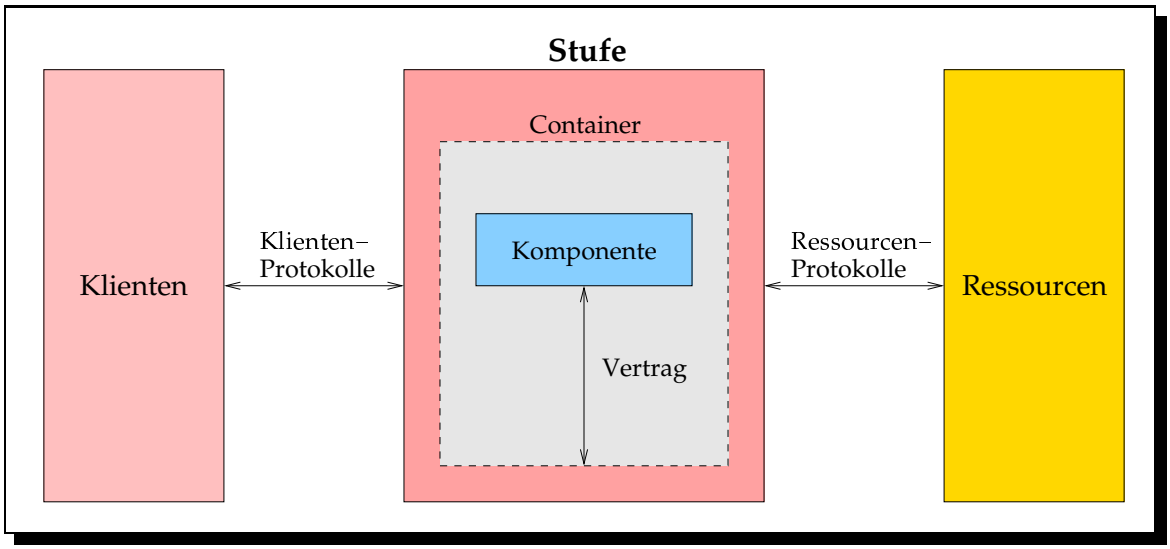


Abbildung 14.2: Aufbau einer Stufe

Stufenarchitekturen

In einer Stufenarchitektur findet eine vertikale Aufteilung der Funktionalität eines Systems auf mehrere Stufen statt. Zur Erbringung der Gesamtfunktionalität kommunizieren und kooperieren die einzelnen Stufen miteinander. Im Gegensatz zu Schichtenarchitekturen steht hier also nicht die Reduktion von Komplexität im Vordergrund, sondern vielmehr die Verteilung von Verantwortung durch Trennung inhaltlicher Aspekte.

Den prinzipiellen Aufbau einer *Stufe* (tier),[1] angelehnt an [SSJt02], zeigt die Abbildung 14.2. Jede Stufe bietet ihren Klienten in anderen Stufen eine Reihe von Diensten an, die über vereinbarte Protokolle in Anspruch genommen werden können, und benutzt ihrerseits eine Reihe von Ressourcen aus anderen Stufen.

Zur Erbringung der Dienste besitzt eine Stufe eine Reihe von Programmen, Klassen oder Modulen, die hier allgemein als *Komponente* (component) bezeichnet werden. Eine solche Komponente erhält Systemdienstleistungen von einem *Container*, mit dem sie überdies in einem Vertragsverhältnis steht.

Ein solcher *Vertrag* (contract) definiert die Anwendungsprogrammierschnittstelle (application programming interface, API) zwischen den Diensterbringern und der Systemumgebung. Sie kann dabei von einem einfachen Programmaufruf wie in einer

[1] Im Deutschen findet sich oft die Übersetzung *Schicht* sowohl für das englische „tier“ wie auch für „layer“. Eine *three-tier architecture* wird dadurch genauso zur Dreischichtenarchitektur wie eine *three-layer architecture*. Wir halten dies für äußerst unglücklich und schließen uns dieser Übersetzungsvariante daher nicht an: Der deutsche Begriff „Schicht“ bezeichnet in dieser Arbeit ausschließlich *layer*, während ein *tier* durchgängig mit „Stufe“ übersetzt wird.

CGI-Schnittstelle *(Common Gateway Interface)* bis hin zu komplexen Programmrahmen im Falle von EJB-Komponenten *(Enterprise JavaBeans)* reichen, bei denen der Container umfangreiche Dienste wie Transaktionen, Persistenz, Sicherheit und Prozeßkontrolle übernimmt. Die Abstraktion über einen Vertrag erlaubt es, Änderungen an Systemdiensten vorzunehmen, ohne daß eine Komponente notwendigerweise davon betroffen ist.

Die Kommunikation mit den Klienten oder Ressourcen einer Stufe erfolgt dabei über spezifische Protokolle. Beispielsweise kommunizieren Web-Klienten mit ihren Servern typischerweise über das *Hypertext Transmission Protocol* (HTTP).

Die Motivation für den Entwurf von Stufenarchitekturen ist, eine vertikale Trennung der Systemfunktionalität zu erhalten. Damit möchte man vor allem eine Aufgabenverteilung der Gesamtfunktionalität auf einzelne Stufen erreichen, um semantisch unabhängige Dienste soweit als möglich voneinander zu entkoppeln. Ein solcher Entwurf gibt einem Architekten eine Reihe von Freiheitsgraden, um Anforderungen wie Flexibilität (durch Änderung/Austausch einzelner Stufen), Skalierbarkeit (durch physikalische Trennung) oder Wiederverwendbarkeit (durch Nutzung der Dienste einer Stufe innerhalb einer anderen Anwendung) erfüllen zu können.

Ein typisches Beispiel einer solchen mehrstufigen Client/Server-Architektur, das wir unten noch genauer besprechen werden, zeigt Abbildung 14.3 auf Seite 206. Da die Trennung zwischen der zweiten und der dritten Stufe oft nur schwach oder gar nicht ausgeprägt ist, findet man diese Architektur oft unter der Bezeichnung *Dreistufenarchitektur* (three-tier architecture).

Stufenarchitekturen sind damit orthogonal zu Schichtenarchitekturen; dabei wird in der Praxis eine Stufe oft selbst wieder intern das Schichtenprinzip anwenden.

Komponentenarchitekturen

Ein im Zusammenhang mit der Beschreibung von Informationssystem-Architekturen immer populärer werdender Begriff ist die *Komponente* (component).

Die Grundidee bei der Definition von Komponenten ist, wiederverwendbare, funktional abgeschlossene Einheiten zu entwickeln, aus denen ein komplexes System bausteinhaft zusammengesetzt werden kann, wie es auch in anderen Ingenieursdisziplinen üblich ist [HNS00]:

> *We should also realize that the software industry has been able to leverage the knowledge in other architectural disciplines, most notably its hardware counterpart. For example, the "software IC" concept was borrowed from the hardware industry over a decade ago, although component-based architectures have not yet realized their potential.*

Der Wunsch nach mehr Wiederverwendbarkeit in der Softwareentwicklung ist nicht neu; über die letzten Jahrzehnte hinweg wurden immer neue Konzepte entwickelt,

die dies ermöglichen sollten: Bibliotheken, Module, Objektorientierung. Allen diesen Ansätzen ist gemein, daß sie in der Praxis zwar zu einer verbesserten Abstraktion führten, die damit gewonnen Vorteile aber immer schnell durch die ebenfalls steigende Systemkomplexität aufgezehrt wurden.

Bei der obigen Beschreibung der Stufenarchitektur wurden einer Komponente keine Einschränkungen auferlegt — sie kann in einer beliebigen Sprache realisiert sein und ihre Funktionalität auf beliebige Weise zur Verfügung stellen. Komponenten werden dann in einen *Container* eingestellt, über den der Aufruf einer Komponente erfolgt. Sie muß lediglich den Vertrag erfüllen, der von dem Container vorgegeben wird — im einfachsten Fall kann dies ein Programmaufruf mit der betriebssystemspezifischen Parameterübergabe sein.

Die Container stellen einer Komponente dabei Systemdienste zur Verfügung (siehe oben). Je mehr Systemdienste in einer konkreten Technologie von den Containern erbracht werden, desto mehr ist der Anwendungsentwickler von diesen Details entlastet und um so leichter fällt eine mögliche Wiederverwendung der entwickelten Komponenten im Rahmen anderer Anwendungen. Da eines der Ziele moderner Architekturen die vereinfachte Anwendungsentwicklung ist, und die einfachste Entwicklung darin besteht, gar nichts zu entwickeln, gewinnen solche Ansätze mehr und mehr an Bedeutung: Industriell vorgefertigte *(COTS, commercial off-the-shelf)* Komponenten sollen sich möglichst nahtlos mit Eigenentwicklungen zu einer kompletten Anwendung zusammenfügen lassen. Anwendungen wiederum sollen heutzutage nicht mehr starr für ihre Lebensspanne entworfen werden, sondern die Dynamik in Unternehmen widerspiegeln — etwa zur informationstechnischen Unterstützung virtueller Unternehmen, die nur für die Dauer eines Projektes gebildet werden. Um eine gute Aufteilung der Funktionalität auf Komponenten zu finden, reicht es daher nicht, nur die Anforderungen einer konkreten Anwendung zu berücksichtigen. Die Herausforderung liegt hier darin, Dienste zu identifizieren, die eine in sich abgeschlossene Aufgabe erfüllen und somit auch im Kontext einer anderen Anwendung eingesetzt werden können.

Um diesen Bausteingedanken zu unterstützen, spezifizieren moderne Komponentensysteme einen umfangreichen Vertrag, an den eine Komponente gebunden wird. Besonderes Merkmal solcher Technologien ist, daß die Entwicklung von Komponenten in einem vorgegebenen *Rahmenwerk* (framework) stattfindet. Bei diesem Ansatz muß eine Komponente eine Reihe vorgegebener Schnittstellen erfüllen; eine Komponente ruft also nicht selber Funktionen etwa einer Bibliothek auf, sondern wird ihrerseits von den vertraglich festgelegten Funktionen des Rahmens aufgerufen. Auf diese Weise läßt sich garantieren, daß eine Komponente, die einen spezifizierten Vertrag erfüllt, zusammen mit anderen Komponenten in einem beliebigen Container plaziert werden kann, der seinerseits den vereinbarten Vertrag honorieren muß.

Die technische Grundlage dafür liefern Anwendungsserver *(application server)*, die auf Komponententechnologie ausgelegt sind und dabei einen spezifizierten Komponentenstandard erfüllen. Prominentes Beispiel sind die *Enterprise JavaBeans* [MH00,

SSJt02], die sich zur Zeit als neuer Herstellerstandard etablieren.

14.2 Referenzarchitektur für Informationssysteme

Mit den Erkenntnissen aus dem letzten Abschnitt können wir jetzt mit der Definition einer Referenzarchitektur für klassische, scharfe Informationssysteme beginnen: Abbildung 14.3 auf der nächsten Seite zeigt unsere Interpretation. Diese Referenzarchitektur abstrahiert die wesentlichen Aufgaben eines Informationssystems und teilt sie auf die vier Stufen *Klienten, Präsentation, Geschäftsprozesse* und *Ressourcen* auf, die wir unten noch genauer beschreiben werden.

Orthogonal zu dieser Stufenarchitektur ist in der Praxis immer eine Schichtenarchitektur zu finden, die in der Abbildung aber nicht explizit gezeigt wird. Insbesondere die Kommunikation zwischen den einzelnen Stufen wird dabei über technologiespezifische, standardisierte Zwischenschichten geführt. Diese ermöglichen es, eine nachrichtenorientierte Kommunikation *(message passing)* oder entfernte Methodenaufrufe *(RMI, remote method invocation)* in standardisierter Weise durchführen zu können. Eingesetzte Technologien sind dabei etwa *Java/RMI, SOAP (Simple Object Access Protocol), IIOP (Internet Inter-ORB Protocol)* oder *XML-RPC*.

Insbesondere die Kommunikationsinfrastruktur zwischen verschiedenen Geschäftsprozeß-Servern, die sogenannte *Server-to-Server*-Infrastruktur, spielt in Unternehmen eine wichtige Rolle. Durch eine geeignete Standardisierung erhält man hier die Möglichkeit, innerhalb einer Organisation jede Anwendung mit allen anderen Anwendungen kommunizieren zu lassen, ohne dabei einen quadratischen Anstieg an Protokollumsetzungen in Kauf nehmen zu müssen. Eine solche Standardisierung der Kommunikationsschicht bildet daher eine wichtige Grundlage der sogenannten *Enterprise Application Integration (EAI)* [RMB01].

Man beachte, daß in der konkreten Architektur eines Informationssystems nicht alle Bestandteile enthalten sein müssen, die in einer Referenzarchitektur vorkommen. Vielmehr wird man genau diejenigen Teile auswählen, die zur Erfüllung der Anforderungen eines konkreten Systems notwendig sind. Ebenso zeigt eine solche Referenzarchitektur nur die *logische* Aufteilung eines Informationssystems. Eine *physikalische* Trennung der Bestandteile ist eine nachfolgende Aufgabe, die sich an der vorhandenen Infrastruktur und den Laufzeitanforderungen (wie Leistung, Skalierbarkeit, Redundanz) orientiert.

Ausgehend von dieser Architektur werden wir nun untersuchen, welche Änderungen und Erweiterungen sich für die Architektur eines Fuzzy-Informationssystems ergeben und eine angepaßte Referenzarchitektur präsentieren. Zunächst beschreiben wir jedoch die Aufgaben der einzelnen Stufen im Detail.

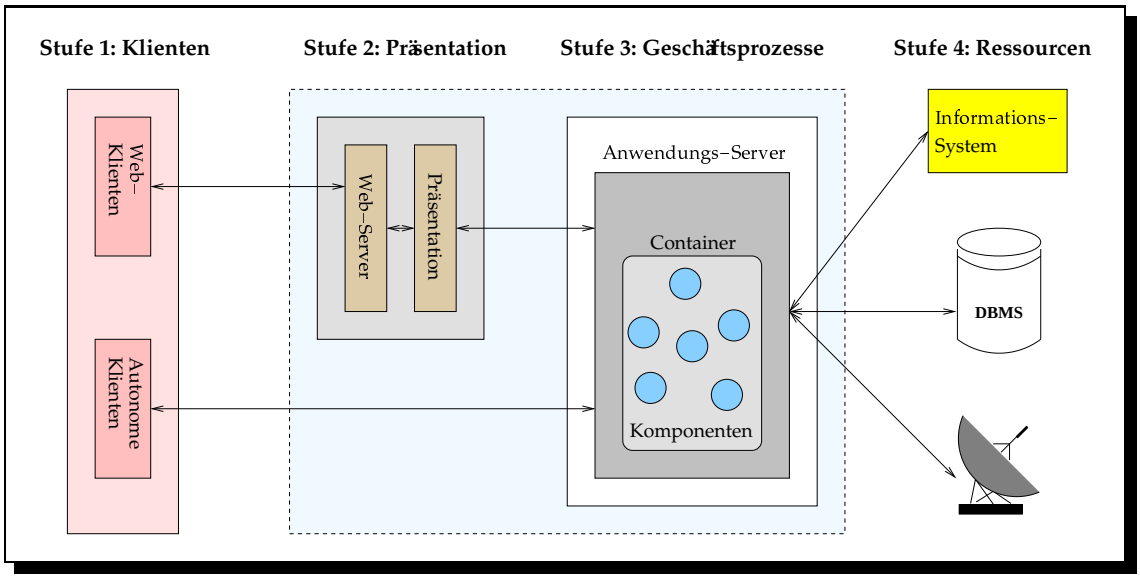


Abbildung 14.3: Mehrstufige Client/Server-Architektur

1. Stufe: Klienten

Die erste Stufe *(client tier)* beherbergt die Klienten eines Informationssystems. Typischerweise dienen Klienten dazu, einem Benutzer über eine graphische oder textuelle Schnittstelle den Zugang zu einem Informationssystem zu ermöglichen. Dabei werden Benutzeraktionen in Anfragen übersetzt, die über ein klientenspezifisches Protokoll an die nächste Stufe gesendet werden.

Mit welcher Stufe ein Klient kommuniziert, hängt dabei von der Art des Klienten ab: *Autonome Klienten* beinhalten alle notwendigen Fähigkeiten, um die von einem Anwendungs-Server gelieferten Informationen in eine Darstellung umzuwandeln, sind also nicht auf externe Hilfe zur Präsentation angewiesen. Sie können daher direkt mit der Systemstufe kommunizieren, die für die Abbildung der Geschäftsprozesse verantwortlich ist; in der hier vorgestellten Architektur also mit der dritten Stufe. Dadurch sind autonome Klienten in der Lage, zum einen die Leistung des Systems, auf dem sie ablaufen, für eine aufwendigere Präsentation und Interaktion auszunutzen, und zum anderen die notwendige Kommunikation mit dem Server durch intelligente, auf eine Anwendung abgestimmte Protokolle zu reduzieren.

Dagegen sind *nicht-autonome Klienten* auf eine Server-seitige Aufbereitung der Informationen angewiesen. Selbst verfügen sie nur über die Infrastruktur zur Anzeige *(rendering)* der Informationen für ein spezifisches Ausgabegerät und zur Durchführung der Interaktion mit einem Anwender. Die Abbildung der Informationen auf ein geeignetes Format, sowie die Steuerung der Interaktion mit dem Anwender, ist getrennt von dem Klienten in einer eigenen Präsentationsstufe realisiert. Diese Trennung wird typischerweise nach dem MVC-Entwurfsmuster *(Model-View-Controller)*

ausgelegt, das Datenrepräsentation, Datenpräsentation und Anwendungsverhalten voneinander entkoppelt (siehe [GHJV95, SSJt02]).

Neben den klassischen Web-Klienten kommen hier zunehmend weitere spezialisierte Klienten zum Einsatz, etwa für Handcomputer (Palmtops), Mobiltelefone oder sonstige industrielle Klienten wie Geldautomaten, Signalgeber, Steuerungssysteme und ähnliche *„embedded devices“*. Mischformen der Klientenarten sind ebenfalls möglich, etwa im Falle von Web-Klienten, die stellenweise eingebettete Programme (wie Java *Applets*) einsetzen. Aus Gründen der Übersichtlichkeit und Anschaulichkeit betrachten wir stellvertretend für die Klasse der nicht-autonomen Klienten im folgenden nur die *Web-Klienten*. Die gelieferten Argumente und Ergebnisse gelten jedoch analog auch für andere nicht-autonome Klienten.

Obwohl Klienten meist zur direkten Benutzerinteraktion verwendet werden, kann auch ein automatisiertes System als Klient zum Einsatz kommen, beispielsweise zur Durchführung von Regressionstests durch Simulation von Benutzereingaben.

2. Stufe: Präsentation

Die Präsentationsstufe übernimmt die Aufbereitung von Informationen in ein für einen nicht-autonomen Klienten geeignetes Ausgabeformat. Neben der reinen Darstellung der Informationen nimmt diese Stufe auch die Benutzereingaben entgegen und steuert die in komplexeren Anwendungen benötigte Ablauffolge *(screen flow)* der Eingabe. Die dafür benötigten Zustandsinformationen werden ebenfalls in dieser Stufe verwaltet.

Eine weitere Aufgabe der Präsentationsstufe ist die Umsetzung zwischen den Protokollen der Klienten und dem der Geschäftsprozeß-Komponenten. Für Web-Klienten wird diese Stufe dafür einen Web-Server enthalten, wie in Abbildung 14.3 gezeigt. Kommen andere nicht-autonome Klienten zum Einsatz, zum Beispiel WAP-Mobiltelefone, wird eine Architektur in analoger Weise spezialisierte Server zur Kommunikationssteuerung bereithalten.

3. Stufe: Geschäftsprozesse

Die Kernfunktionalität einer Anwendung residiert in der Geschäftsprozeß-Stufe. Hier erbringen die Komponenten die einzelnen Dienste einer Anwendung; gemeinsam werden sie in Containern verwaltet. Container wiederum werden von Anwendungs-Servern verwaltet, die Kommunikations- und Steuerungsaufgaben wahrnehmen.

Der in diesem Zusammenhang etablierte Begriff „Geschäftsprozeß“ stammt aus der Welt der betrieblichen Informationssysteme und läßt sich besser abstrahiert als Funktion verstehen. Die Architektur hat gleichermaßen Gültigkeit für ein technisches oder wissenschaftliches Informationssystem.

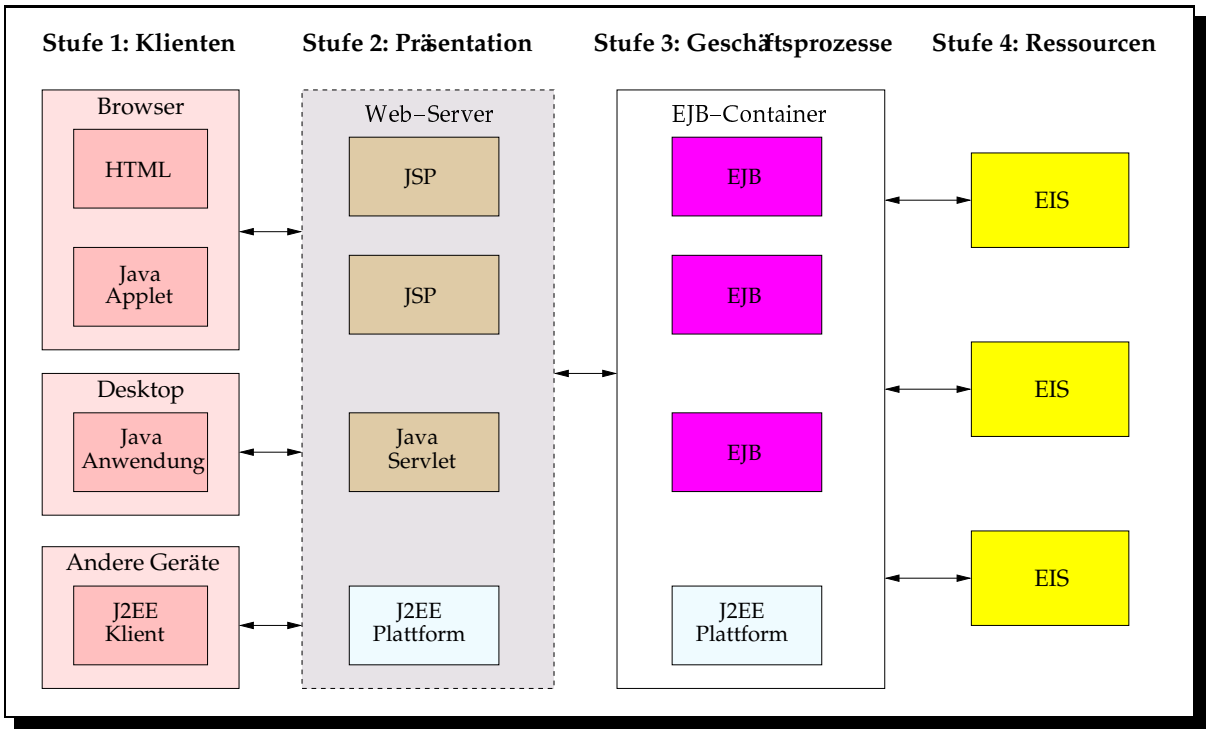


Abbildung 14.4: Technologie: die Java 2 Enterprise Edition (J2EE)

4. Stufe: Ressourcen

Zur Erfüllung ihrer Funktion greifen die Komponenten der 3. Stufe typischerweise auf verschiedene *Ressourcen* zurück. Diese Ressourcen stellen Daten oder Dienste bereit, sie können dabei aus der gesamten Bandbreite eingesetzter Systeme stammen.

Beispiele für solche Ressourcen sind Datenbankmanagementsysteme (DBMS) zur persistenten Datenspeicherung, Data Mining- und Data Warehousing-Systeme, Authentisierungs- und Authorisierungsdienste, oder ERP-Systeme für das *Enterprise Resource Planning*.

Insbesondere kann eine solche Ressource selbst wieder ein in mehrstufiger Architektur ausgeführtes Informationssystem darstellen; durch diese Rekursion kommt man zu dem Begriff einer n-stufigen *(n-tier)* Architektur.

14.2.1 Beispieltechnologie: die Java 2 Enterprise Edition

Eine Referenzarchitektur, wie sie oben vorgestellt wurde, muß für die Realisierung eines konkreten Informationssystems mit Technologien gefüllt werden. Wir betrachten hierzu im folgenden die *Java 2 Enterprise Edition* (J2EE) der Firma Sun Microsystems. Abbildung 14.4 zeigt die wichtigsten Bestandteile dieser Technologie und deren Aufteilung auf die zuvor beschriebenen Stufen.

Wir gehen im folgenden nur auf die Technologien ein, die bei der Betrachtung von Fuzzy-Informationssystemen eine Rolle spielen; für eine detaillierte Beschreibung der J2EE verweisen wir den Leser auf einschlägige Literatur [FFCM99, MH00, All00, SSJt02].

Die J2EE-Technologie vereinigt die oben beschrieben Konzepte und eignet sich daher vor allem für die Entwicklung moderner, netzbasierter Anwendungen. Sie findet seit einigen Jahren vor allem im Bereich der unternehmenskritischen Anwendungen eine immer stärkere Verbreitung; so setzen heute bereits viele Banken, Universitäten und Unternehmen die J2EE für ihre Anwendungsentwicklung ein.

Damit erfüllt diese Technologie unsere Anforderungen an eine MODERNE ARCHITEKTUR (vergleiche Abschnitt 3.2.1 auf Seite 27), bietet also eine ideale Grundlage für die Entwicklung von Fuzzy-Informationssystemen. Für den praktischen Einsatz verwenden wir im folgenden die J2EE-Technologie, um die hier beschriebenen Erweiterungen zu implementieren und so deren Realisierbarkeit nachzuweisen.

14.3 Fuzzy-Referenzarchitektur

Wir beginnen nun mit der Entwicklung unserer Referenzarchitektur für Fuzzy-Informationssysteme. Dabei soll die vorgestellte Informationssystem-Architektur soweit wie möglich beibehalten werden.

Folglich stellt sich die Frage, an welchen Stellen Änderungen an der in Abschnitt 14.2 vorgestellten Referenzarchitektur nötig werden. Dabei dürfen keine grundlegenden Änderungen vorgenommen werden, da sonst die Anforderung der INTEROPERABILITÄT (siehe Abschnitt 3.2.7 auf Seite 34) verletzt würde. Um diese Anforderung zu erfüllen, haben wir in dieser Arbeit durch die Ebenen der Theorie und Praxis hinweg die Prinzipien der Entkopplung und Abstraktion angewendet, um Fuzzy-Aspekte als zusätzliche Systemeigenschaften anzubieten. Dies hat sich bereits bei der Formulierung des objektorientierten Fuzzy-Datenmodells (Kapitel 11) sowie dessen Implementierung (Kapitel 12) ausgezahlt: beide konnten als strikt orthogonale Erweiterungen entwickelt werden. Das Ziel muß daher sein, auch die Architektur in gleichermaßen orthogonaler Weise zu erweitern.

Um zu klären, an welchen Stellen der Referenzarchitektur sich Änderungen für die Entwicklung von Fuzzy-Informationssystemen ergeben, die über den reinen Einsatz der im letzten Kapitel entwickelten Fuzzy-Bibliothek hinausgehen, gehen wir in den nachfolgenden Abschnitten die einzelnen Stufen durch. Dabei ist jeder der Unterabschnitte einheitlich gegliedert: zuerst erfolgt die Motivation der durchzuführenden Änderung, anschließend erfolgt die Definition der erweiterten Fuzzy-Architektur.

14.3.1 Klienten-Stufe

Klienten ermöglichen den Zugang zu einem Fuzzy-Informationssystem. Eine grundsätzliche Frage ist hier zunächst, ob ein Benutzer imperfekte Informationen zur direkten Systeminteraktion verwenden soll, oder ob diese nur systemintern eingesetzt werden und somit nicht an der Benutzerschnittstelle sichtbar sind.

Für den Fall, daß die imperfekten Informationen auf Klientenseite sichtbar werden sollen, muß zwischen der Art der Klienten unterschieden werden: Kommen nichtautonome Klienten (wie Web-Klienten) zum Einsatz, ergeben sich keine Änderungen an der Architektur, da die Präsentation von der 2. Stufe erzeugt wird.

Werden dagegen autonome Klienten eingesetzt, muß die Darstellung und Interaktion mit dem Anwender von dieser Stufe gesteuert werden. Für den Umgang mit Fuzzy-Informationen wird folglich eine Reihe neuer Bausteine benötigt, die es bereitzustellen gilt. Ebenso kann die Art der Benutzerinteraktion Einfluß auf alle nachfolgenden Stufen haben, wie wir unten noch zeigen werden.

Wesentliche Fragen für die Entwicklung dieser neuen Komponenten sind:

- Wie sollen die Informationen, die in den verschiedenen Facetten einer Annotation, also insbesondere in Fuzzy-Facetten, gespeichert sind, im Klienten angezeigt werden?
- In welcher Form sollen Fuzzy-Informationen präsentiert werden, so daß sie verständlich für den Benutzer sind?
- Wie können imperfekte Informationen von einem Benutzer eingegeben werden?

Die ersten beiden Punkte betreffen die *Präsentation* imperfekter Informationen. Da wir die Entwicklung von Benutzerschnittstellen auf der Basis einzelner Komponenten unterstützen wollen, müssen wir hierfür neue, auf unser Fuzzy-Modell spezialisierte Darstellungskomponenten anbieten.

Der dritte Punkt dagegen betrifft die *Eingabe:* also die Frage, wie imperfekte Informationen, repräsentiert durch die Bestandteile von Fuzzy-Formeln (vergleiche Kapitel 8), mit der Begriffswelt eines Anwenders verknüpft werden sollen. Nachdem wir bereits an mehreren Stellen auf die Notwendigkeit der Entkopplung einer imperfekten Informationen von ihrer konkreten Repräsentation als Fuzzy-Menge hingewiesen haben (vergleiche Abschnitt 8.4 auf Seite 82), können wir hier natürlich nicht verlangen, daß ein Benutzer Eingaben in Form einzelner Fuzzy-Mengen vornimmt. Vielmehr muß es einem Anwender möglich sein, in Form imperfekter Begriffe wie *schnell* oder *groß* zu kommunizieren (vergleiche Kapitel 7), die eine Anwendung dann intern in passende Fuzzy-Ausdrücke übersetzt.

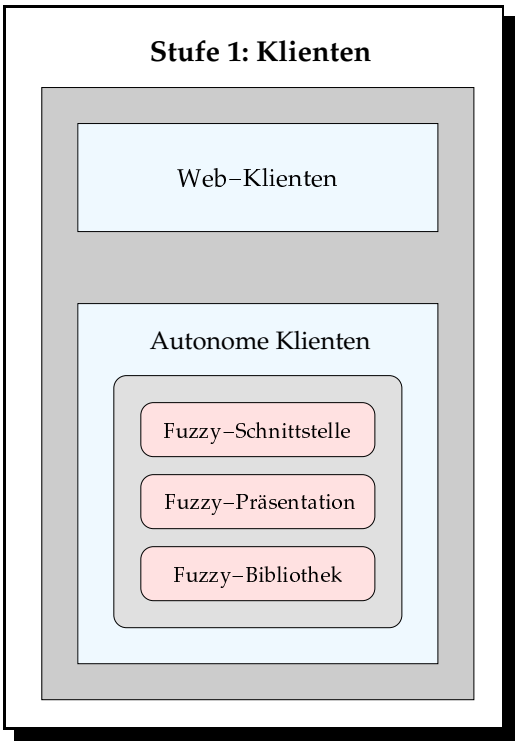


Abbildung 14.5: Klienten-Stufe in einem Fuzzy-Informationssystem

Es läßt sich bereits erkennen, daß diese Informationen auch in den weiteren Stufen eines Fuzzy-Informationssystems zur Verfügung stehen müssen: die Präsentationsstufe benötigt sie für nicht-autonome Klienten und die Geschäftsprozeßstufe schließlich muß sie bei Berechnungen berücksichtigen. Die konsequente Architekturentscheidung ist daher, für solche Systeme eine neue Ressource anzubieten, die diese Informationen bereithält: ein *Fuzzy-Lexikon*. Wir werden weiter unten in Zusammenhang mit der 4. Stufe genauer darauf eingehen und in Kapitel 16.1 seine Realisierung vorstellen.

Fuzzy-Architektur

Abbildung 14.5 zeigt den Aufbau der Klienten-Stufe in unserer Referenzarchitektur. Für die Entwicklung autonomer Klienten werden *Fuzzy-Präsentationskomponenten* angeboten, die eine graphische Ausgabe von Annotationen mit Facetten und deren Fuzzy-Informationen ermöglichen. Diese Komponenten wiederum benötigen die in Kapitel 13 (siehe Seite 183) vorgestellte Fuzzy-Bibliothek, um auf die in Facetten gespeicherten Fuzzy-Informationen zugreifen zu können.

Wir werden in Abschnitt 15.1 einige konkrete Komponenten vorstellen. Mit ihnen wird es möglich, einen autonomen Klienten schnell um die Fähigkeit der Darstellung

und Eingabe von Fuzzy-Informationen zu erweitern. Anwendungen, die spezielle Anforderungen an die Präsentation haben, können entweder auf diesen Komponenten aufbauen, oder komplett eigene Darstellungskomponenten einsetzen.

14.3.2 Präsentations-Stufe

An dieser Stufe kehren sich die für die Klienten-Stufe beschriebenen Aspekte gerade um: Änderungen werden nur für nicht-autonome Klienten benötigt. Hier gelten analoge Schlußfolgerungen wie oben ausgeführt.

Allerdings sind die Möglichkeiten zur Darstellung in dieser Stufe im Vergleich zu autonomen Klienten meist stark eingeschränkt: je nach Fähigkeit des Klienten können beispielsweise nur statische HTML-Seiten oder Seiten mit stark eingeschränkter Dynamik erzeugt werden.

Fuzzy-Architektur

Die Präsentations-Stufe übernimmt in unserer Referenzarchitektur die Aufbereitung und Kontrolle der Benutzerinteraktion. Da die Trennung zwischen der 2. und 3. Stufe oft nur schwach ausgeprägt ist, haben wir beide in Abbildung 14.6 auf der nächsten Seite zusammengefaßt.

Auch hier wird, wie in der Klienten-Stufe, die Fuzzy-Bibliothek zum Umgang mit den imperfekten Informationen benötigt. Die Darstellung wird im Falle von Web-Klienten typischerweise in einfachem HTML oder einer mächtigeren Sprache wie DHTML/XML stattfinden. Für die Umwandlung der Fuzzy-Formeln in HTML steht eine entsprechende Komponente bereit, die von der Fuzzy-Schnittstelle in dieser Stufe verwendet werden kann. Da die Möglichkeiten der direkten Interaktion im Vergleich zu autonomen Klienten aber beschränkt sind, wird man hier meist speziell auf die Anwendung zugeschnittene Darstellungskomponenten entwickeln. Entsprechende Folgerungen gelten für Architekturen, die andere nicht-autonome Klienten einsetzen.

Ein Beispiel für den Einsatz eines Web-Klienten in einem Fuzzy-Informationssystem zeigen wir in Kapitel 18.

14.3.3 Geschäftsprozeß-Stufe

Die Funktionalität einer Anwendung wird in dieser Stufe von den einzelnen Komponenten erbracht. Vor allem hier finden die Konzepte und Technologien Einsatz, die in den letzten Teilen dieser Arbeit entwickelt wurden: Fuzzy-Formeln, Abhängigkeitsgraphen, konsistenzerhaltende Verarbeitungsoperationen, Annotationen und Fuzzy-Facetten.

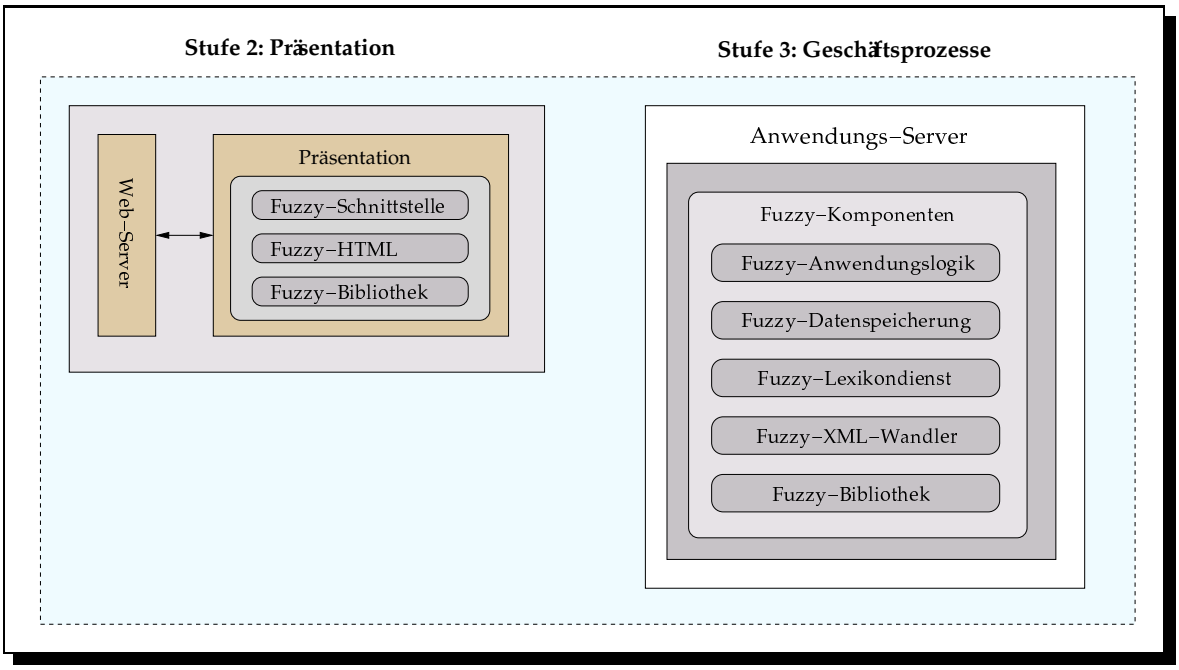


Abbildung 14.6: Aufbau der Präsentations- und Geschäftsprozeß-Stufe eines Fuzzy-Informationssystems

Weitere Änderungen an der Architektur ergeben sich in dieser Stufe dann, wenn annotierte Objekte mit der Ressourcen-Stufen kommunizieren müssen. Dies kann drei verschiedene Formen annehmen:

Speicherung Wenn imperfekte Informationen in einer Datenbank gespeichert werden sollen, muß die von der 3. Stufe durchgeführte Abbildung von Objekten auf das Datenmodell des Datenbanksystems erweitert werden, um auch die neuen Annotationen (vergleiche Abschnitt 12.2 auf Seite 173) mit den verschiedenen Facetten einzubeziehen.

Wir werden in Kapitel 16 zeigen, wie eine existierende relationale Datenbank systematisch zur Speicherung von Fuzzy-Informationen erweitert werden kann.

Dienst-Aufruf Wenn der Aufruf von externen Diensten notwendig ist, die nicht mit Fuzzy-Daten umgehen können, müssen diese zunächst defuzzifiziert werden (vergleiche Definition 9.3.6 auf Seite 100). An dieser Stelle sind mehrere anwendungsspezifische Strategien denkbar, um den mit der Defuzzifizierung einhergehenden Informationsverlust so gering wie möglich zu halten; beispielsweise könnte für einen Fuzzy-Parameter ein externes System mehrfach aufgerufen und die Einzelergebnisse durch Kombination in ein Fuzzy-Ergebnis rückübersetzt werden.

Datenaustausch Findet dagegen ein Datenaustausch, etwa zur Datenweiterleitung an ein anderes System oder zum Datenexport statt, müssen die Fuzzy-Daten entweder wie oben beschrieben defuzzifiziert, oder in ein geeignetes Austauschformat verpackt werden. Wir werden in Abschnitt 16.3 eine auf der Sprache XML basierende Möglichkeit zum Austausch von Fuzzy-Informationen vorstellen.

Fuzzy-Architektur

Den Aufbau der Geschäftsprozeß-Stufe zeigt ebenfalls Abbildung 14.6 auf der vorherigen Seite. In dieser Stufe wird die Anwendungslogik plaziert. Diese wird punktuell Fuzzy-Informationen einsetzen — so wie wir es in der Anforderung PARTIELLE UNSCHÄRFE (vergleiche Abschnitt 3.2.4 auf Seite 30) ausgeführt haben. Die technischen Grundlagen dafür liefert das Annotationskonzept und die Fuzzy-Bibliothek mit der Implementierung der Operatoren, auf der aufbauend die Fuzzy-Anwendungslogik realisiert werden kann.

Für spezielle Anwendungen, nämlich solche, bei denen ein Anwender mit imperfekten Begriffen mit einem System interagieren soll (vergleiche Abschnitt 14.3.1 auf Seite 210), wird in dieser Stufe ein neuer Dienst benötigt, um auf das Fuzzy-Lexikon zuzugreifen. Wir werden den Entwurf des Fuzzy-Lexikons zusammen mit seinem Zugriffsdienst ausführlich in Abschnitt 16.1 vorstellen.

Für den Aufruf externer Ressourcen wird, wie oben beschrieben, eine Defuzzifizierung benötigt; die dafür benötigten Operationen werden bereits durch die Fuzzy-Bibliothek bereitgestellt und können um anwendungsspezifische Strategien erweitert werden.

Für den Datenaustausch mit externen Systemen sehen wir eine Umwandlung der Fuzzy-Informationen in das XML-Format vor. Durch seinen hierarchischen Aufbau ist dieses Format prädestiniert für eine Trennung der scharfen Daten von ihren Facetten, die — in einer Unterhierarchie abgelegt — von Systemteilen übersprungen werden können, die auf ihre Verwendung nicht ausgelegt sind. Dies setzt natürlich voraus, daß die Systeme, mit denen kommuniziert werden soll, eine XML-Schnittstelle anbieten. Diese etabliert sich jedoch zunehmend als Standard. Wir werden die Umwandlung von Fuzzy-Informationen in ein XML-Format in Abschnitt 16.3 besprechen.

Der letzte Punkt betrifft die persistente Speicherung von Fuzzy-Daten. Die dabei eingesetzte Strategie hängt stark von der eingesetzten Technologie ab; grundsätzlich kann man hier unterscheiden zwischen der *anwendungsdefinierten Speicherung* und der *containerdefinierten Speicherung*. Im ersten Fall entscheidet eine Anwendung selbständig, welche Objekte gespeichert werden und zu welchem Zeitpunkt. Ebenso muß, falls relationale Datenbanken zum Einsatz kommen, die Abbildung der Objekte auf das Datenbankschema durchgeführt werden. Bei der containerbasierten Speiche-

rung dagegen übernimmt dies der Container, in dem die Anwendungskomponente abläuft. Die Entscheidung, wann und wie die Daten gespeichert werden, findet aber nicht während der Entwicklung, sondern erst beim *Einsatz* (deployment) einer Komponente statt.

Die zweite Variante kommt bei modernen Architekturen mehr und mehr zum Einsatz, da sie die Entwicklung der Komponenten nicht nur von einem konkreten Datenbankmanagementsystem, sondern auch von dem anwendungsspezifischen Datenmodell und Datenbankschema unabhängig macht; dies ist eine wichtige Voraussetzung, sollen Komponenten jemals als vorgefertigte Bausteine verwendbar sein. Im Beispiel der Java-Architektur entspricht dies der *containerverwalteten Persistenz* (container-managed persistence, CMP). Kommt bei der Speicherung eine relationale Datenbank zum Einsatz, was heute noch der Standard ist, muß aber für beide Fälle das Datenbankschema erweitert werden, um die zusätzlichen Fuzzy-Informationen aufnehmen zu können.

14.3.4 Ressourcen-Stufe

In dieser Stufe sind alle Ressourcen vereinigt, auf die ein Fuzzy-Informationssystem zur Erbringung seiner Dienste zurückgreifen muß.

Wenn ein existierendes Informationssystem als Diensterbringer eingesetzt wird, entsteht hier die Möglichkeit, es mit den in dieser Arbeit vorgestellten Methoden um Fuzzy-Technologien zu erweitern. Andernfalls müssen die Aufrufe aus der dritten Stufe entsprechend den Möglichkeiten des Systems angepaßt werden. Ähnliches gilt im Falle der Weiterleitung von Daten, wobei hier die Möglichkeit besteht, durch ein geeignetes Austauschformat Fuzzy-Daten zumindest als zusätzliche Information mitzugeben, damit sie nach einer zukünftigen Systemerweiterung zur Verfügung stehen.

Wenn die Ressource ein Datenbankmanagementsystem darstellt, also die persistente Speicherung von Daten als Dienst bereitstellt, muß das Schema der Datenbank gegebenenfalls angepaßt werden, um die Fuzzy-Informationen (und sonstige Facetten einer Annotation) transparent abspeichern zu können.

Eine neue Ressource stellt das oben angesprochene Fuzzy-Lexikon dar. Es wird von dem Lexikon-Dienst der 3. Stufe benutzt, um den anderen Stufen eines Systems Informationen über die Repräsentation imperfekter Informationen in Form von Fuzzy-Ausdrücken zugänglich zu machen.

Fuzzy-Architektur

Auf der Ressourcen-Stufe ergeben sich zwei wesentliche Änderungen: zum einen kommt als neue Ressource das bereits erwähnte Fuzzy-Lexikon hinzu. Seine Realisierung wird in Abschnitt 16.1 vorgestellt.

Für die persistente Speicherung muß die oben beschriebene Erweiterung für Fuzzy-Informationen durchgeführt werden. Wir gehen an dieser Stelle nur auf die für relationale Datenbanken benötigten Erweiterungen ein, da die Möglichkeit der Speicherung von Objekten in einer objektorientierten Datenbank nicht weiter untersucht werden muß; diese werden in Abschnitt 16.2 vorgestellt.

Die sonstigen Ressourcen können unverändert beibehalten werden — dies war ja auch die Vorgabe unserer Anforderung INTEROPERABILITÄT (vergleiche Abschnitt 3.2.7 auf Seite 34). Es kann jedoch sinnvoll sein, diese Ressourcen zu erweitern, damit auch sie von den neuen Möglichkeiten der Verarbeitung von Fuzzy-Informationen profitieren.

14.3.5 Systematischer Architekturentwurf

Die definierte Referenzarchitektur enthält eine Vielzahl von Bestandteilen, die je nach Vorgabe in einer konkreten Anwendung eingesetzt werden können. Es verbleibt aber die Aufgabe, für die Entwicklung eines Fuzzy-Informationssystems systematisch eine konkrete Architektur abzuleiten, auf der dann die Anwendungsentwicklung aufsetzen kann.

Zu diesem Zweck läßt sich die auf der nächsten Seite gezeigte Tabelle 14.1 heranziehen. Sie gibt zu jeder Anforderung an, welche Komponenten in den einzelnen Stufen benötigt werden:

Web-Klient? Wird für die Anwendung ein Web-Klient gefordert, der imperfekte Informationen anzeigen können muß?

Autonomer Klient? Wird für die Anwendung ein autonomer Klient gefordert, der imperfekte Informationen anzeigen können muß?

Persistente Speicherung? Müssen Fuzzy-Informationen persistent in einer Datenbank gespeichert werden?

Imperfekte Begriffe? Sollen linguistische Begriffe zur Beschreibung imperfekter Informationen eingesetzt werden?

Datenaustausch? Müssen imperfekte Informationen an andere Systeme weitergeleitet werden können?

Diese Anforderungen betreffen natürlich nur die zusätzlichen Aspekte, die bei der Betrachtung von Fuzzy-Informationssystemen hinzukommen. Anhand der Tabelle kann nun leicht entschieden werden, welche Teile der Fuzzy-Referenzarchitektur für ein konkretes System ausgewählt werden müssen.

Stufe	Komponente	Web-Klient	Autonomer Klient	Persistente Speicherung	Imperfekte Begriffe	Datenaustausch
1. Stufe	Web-Browser	✓				
	Fuzzy-Schnittstelle		✓			
	Fuzzy-Präsentation		✓			
	Fuzzy-Bibliothek		✓			
2. Stufe	Web-Server	✓				
	Fuzzy-Schnittstelle	✓				
	Fuzzy-HTML	✓				
	Fuzzy-Bibliothek	✓				
3. Stufe	Fuzzy-Anwendungslogik	✓	✓	✓	✓	✓
	Fuzzy-Datenspeicherung			✓		
	Fuzzy-Lexikondienst				✓	
	Fuzzy-XML-Wandler					✓
	Fuzzy-Bibliothek	✓	✓	✓	✓	✓
4. Stufe	Nicht-Fuzzy-System					✓
	Fuzzy-Lexikon				✓	
	Fuzzy-Datenbank			✓		
	Fuzzy-Bibliothek				✓	

Tabelle 14.1: Bestimmung der Architektur eines Fuzzy-Informationssystems

Beispiel (Architekturentwurf) Wir zeigen ein Beispiel für die konkrete Architektur eines Fuzzy-Informationssystems. Hierfür setzen wir das Einsatzbeispiel aus dem Reengineering fort: gesucht ist jetzt die Architektur für ein Fuzzy-Reengineering-System. Der Anwender soll dabei über eine Web-Schnittstelle mit dem System kommunizieren. Das System soll sowohl neue Fuzzy-Algorithmen zur Programmanalyse realisieren, als auch bestehende Systeme als externe Ressource einbinden. Zusätzlich soll die persistente Speicherung gewonnener imperfekter Informationen möglich sein.

Die resultierende Architektur zeigt Abbildung 14.7 auf der nächsten Seite. Man sieht, daß nur ein Teil der Fuzzy-Referenzarchitektur zum Einsatz kommt: In der Geschäftsprozeß-Stufe wird keine Komponente für das Fuzzy-Lexikon benötigt, da laut Anforderung unscharfe Ergebnisse der Berechnungen nur angezeigt, aber nicht eingegeben werden sollen. Auch für den Datenexport wird keine Komponente benötigt, da

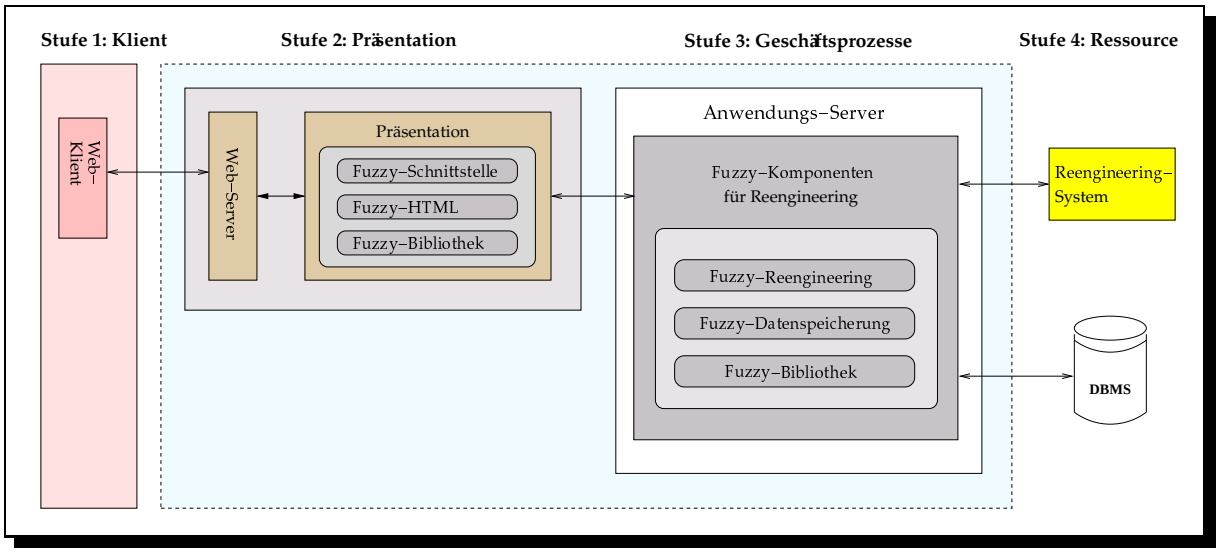


Abbildung 14.7: Architektur des Fuzzy-Reengineering-Systems

imperfekte Ergebnisse zwar in einer Datenbank gespeichert, aber sonstige externe Systeme nur lesend verwendet werden.

Kapitel 15

Fuzzy-Komponenten

Build a better mousetrap, and the world will beat a path to your door.
Emerson

Die im letzten Kapitel entwickelte Referenzarchitektur macht die Entwicklung einiger zusätzlicher Komponenten für Fuzzy-Informationssysteme notwendig. In diesem Kapitel stellen wir die Realisierung der Komponenten für Fuzzy-Klienten vor.

15.1 Komponenten für autonome Klienten

Autonome Fuzzy-Klienten müssen die Fähigkeit zur *Präsentation* der imperfekten Informationen und zur *Interaktion* mit einem Anwender aufweisen (vergleiche Abschnitt 14.3.1 auf Seite 210).

Hierbei ergeben sich die folgenden Einzelaspekte:

Annotationen und Facetten In unserem Fuzzy-Datenmodell können einzelne Objekte, Attribute oder Attributmengen mit Annotationen versehen sein, die Zusatzinformationen in Form einzelner Facetten aufnehmen (vergleiche Abschnitte 11.2.1 auf Seite 158 und 12.2 auf Seite 173). Diese müssen in geeigneter Weise präsentiert werden.

Fuzzy Konjunktive Normalformen Die Fuzzy Konjunktiven Normalformen (siehe Definition 8.1.7 auf Seite 77) müssen so dargestellt werden, daß sowohl ihre Struktur als auch ihre Interpretation als Fuzzy-Menge ersichtlich wird.

Grundsätzlich soll dabei ein komponentenorientierter Ansatz verfolgt werden. Ziel ist also nicht, eine komplette graphische Benutzeroberfläche speziell für einen Anwendungsfall zu entwickeln, wie es in vielen anderen Ansätzen [Sch95, Bos96] geschieht. Vielmehr sollen einzelne *Präsentationskomponenten* entwickelt werden, die

sich zur Realisierung einer konkreten Oberfläche, abhängig von deren Anforderungen, zusammensetzen lassen (vergleiche auch Abschnitt 20 auf Seite 203). Natürlich können diese Komponenten nur vergleichsweise abstrakte, universelle Sichten anbieten — eine Oberfläche für eine Fuzzy-Anwendung kann aber unter ihrer Verwendung schnell prototypisch realisiert werden; eine Verfeinerung durch die Entwicklung anwendungsspezifischer Sichten ist dann dank des Komponentenansatzes problemlos möglich.

Wir stellen im nächsten Abschnitt zunächst den Sichtenverwalter vor, der die Aufgabe der Auswahl und des Einsatzes einer passenden Präsentationskomponente für ein Objekt übernimmt. Anschließend werden die einzelnen, konkreten Sichtenkomponenten beschrieben.

15.1.1 Der Sichtenverwalter

Wie oben erläutert, soll sich eine komplette graphische Benutzeroberfläche aus einzelnen Komponenten bausteinhaft zusammensetzen lassen. Die Präsentation der Objekte einer Klasse erfolgt dabei nach dem MVC-Prinzip *(Model-View-Controller)* [GHJV95, SSJt02] in Form einzelner *Sichten* (Views), die jeweils eine spezielle Darstellung des *Modells* (model) anbieten. So kann es für eine Fuzzy-Menge beispielsweise eine Sicht mit einer graphischen Balkendarstellung und eine mit einer einfachen Auflistung der α-Schnitte geben, die beide auf das gleiche interne Modell zurückgreifen. Der *Controller* steuert dabei die Interaktion des Benutzers mit der Sicht.

Nun könnte man die Auswahl einer für ein Modell geeigneten Sicht manuell durchführen. Gerade für den oben beschriebenen Entwicklungszyklus mit einer laufenden Verfeinerung wäre es jedoch von Vorteil, wenn ein System selbständig die am besten passende Oberflächenkomponente bestimmen könnte. Wir schlagen daher vor, diese Auswahl einem *Sichtenverwalter* (view manager) zu überlassen.

Dieser Sichtenverwalter hat die Aufgabe, zu einem beliebigen Objekt eine geeignete Darstellungsklasse zu finden und zu instanziieren. Dazu greift er auf eine Baumstruktur mit den verfügbaren Sichten zu, die entsprechend den ihnen zugehörigen Klassen angeordnet sind. Ist für die angeforderte Klasse keine Sicht verfügbar, sucht der Sichtenverwalter in der Klassenhierarchie nach einer Sicht für deren Oberklasse, bis hin zu der Wurzelklasse *Objekt*, für die wir eine sogenannte universelle Sicht anbieten. Kommt im Laufe der Entwicklung eine neue, speziellere Sicht für eine Klasse hinzu, kann diese einfach zu der Konfiguration hinzugefügt werden; sie wird dann beim nächsten Start der Oberfläche automatisch ausgewählt.

Da wir eine universelle Sichtenkomponente anbieten, die jedes Objekt darstellen kann, findet der Sichtenverwalter immer mindestens eine passende Sicht für ein Objekt. Damit wird insbesondere die inkrementelle Programmentwicklung[1] unterstützt,

[1] Neuerdings auch modischer als *agile programming* (agile Programmentwicklung) bezeichnet.

wie sie von Methoden wie dem *Extreme Programming* [Bec00] propagiert wird, da so für jede Anwendung schon zu Beginn der Entwicklung eine lauffähige graphische Oberfläche verfügbar ist, die dann schrittweise durch anwendungsspezifische Sichten verfeinert werden kann.

15.1.2 Anzeige von Annotationen und Facetten

Unser Modell verwendet das Konzept der Annotationen und Facetten (vergleiche Kapitel 12), um Objekte und Attribute eines Systems zur Laufzeit um Fuzzy- und andere Zusatzinformationen erweitern zu können. Wir benötigen daher eine Möglichkeit, solche Annotationen mit ihren verschiedenen Facetten anzeigen und modifizieren zu können. Dafür wurde eine Präsentationskomponente namens `AnnotationView` entwickelt, die die folgenden Funktionen anbietet:

AnnotationTreeManager Der `AnnotationTreeManager` erzeugt und verwaltet eine Ansicht in Form einer Baumstruktur. Dabei bildet ein darzustellendes Objekt die Wurzel des Baumes. Die einzelnen Attribute des Objektes werden als Kindknoten dargestellt; hat eine annotierte Struktur eine Annotation, bildet diese einen weiteren Kindknoten. Dies setzt sich rekursiv über die einzelnen Facetten bis zu deren Inhalten als Blätter des Baumes fort.

FuzzyFacetManager Der `FuzzyFacetManager` erlaubt das Anlegen und Anzeigen von Fuzzy-Facetten, die Fuzzy-Formeln beinhalten.

StringFacetManager Der `StringFacetManager` erlaubt das Anlegen und Ändern von String-Facetten, die einfache textuelle Kommentare enthalten. Mit seiner Hilfe können Objekte eines Systems leicht um Zusatzinformationen erweitert werden.

Die Möglichkeit zur Verwaltung von String-Facetten wird zwar für Fuzzy-Informationssysteme nicht benötigt, wurde hier aber zusätzlich realisiert, um die Definition neuer Facetten-Typen und die parallele Verwaltung multipler Facetten zu einer annotierten Komponente anschaulich zu machen.

Beispiel (Präsentation von Annotationen mit Facetten) Ein Beispiel für die Anzeige von Annotationen mit einzelnen Facetten zeigt Abbildung 15.1 auf der nächsten Seite. Dort wird ein Objekt der Klasse `Automobil` mit seinen Attributen `Hersteller`, `Bauart`, `Antrieb`, ... gezeigt; das Attribut `Motor` verfügt über eine Annotation, in der drei Facetten angelegt sind: eine String-Facette, die einen einfachen textuellen Kommentar zu dem Attribut enthält *(„Verbrauch: Sollte nicht viel verbrauchen“)*; eine Fuzzy-Facette, die eine Fuzzy Konjunktive Normalform zur Repräsentation dieser Aussage in Form einer Fuzzy-Formel enthält, sowie eine Abhängigkeits-Facette, die eine Abhängigkeit der Attribute `Bauart` und `Modell` von diesem

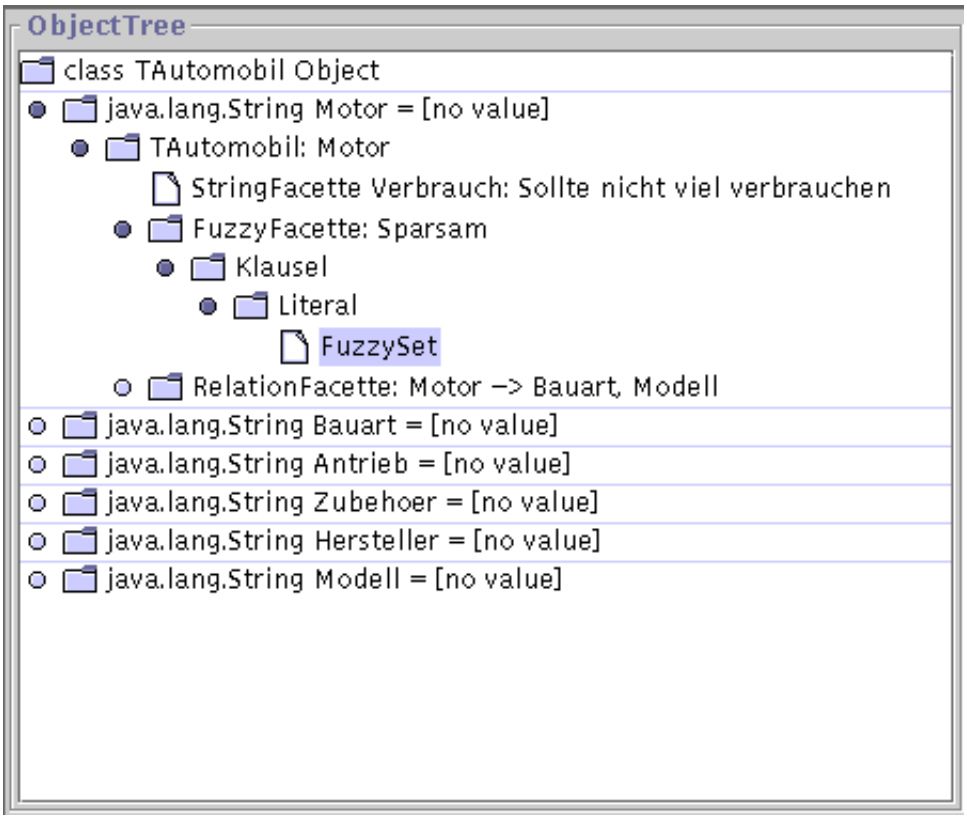


Abbildung 15.1: Anzeige von Annotationen mit Facetten

Attribut anzeigt, also die Menge der abhängigen Knoten des Abhängigkeitsgraphen im Adjazenzlistenformat enthält (vergleiche Abschnitt 13.3 auf Seite 190).

Zusätzlich enthält diese Präsentationskomponente ein Kontextmenü, mit dem die Methoden eines Objektes angewählt werden können.

15.1.3 Darstellung von Fuzzy-Informationen

Der `AnnotationTreeManager` erlaubt die anschauliche Darstellung von Fuzzy-Facetten mit den Fuzzy-Formeln und deren syntaktische Struktur in Form von Klauseln und Literalen. Zusätzlich zu der Syntax besitzt jede Fuzzy-Formel aber auch eine Semantik in Form einer Fuzzy-Menge. Auch diese sollen dem Benutzer in intuitiv verständlicher Weise präsentiert werden. Hierfür eignet sich besonders eine Darstellung als Balkendiagramm. Angeboten werden dazu zwei Darstellungskomponenten, die je eine zwei-, sowie eine dreidimensionale Darstellung ermöglichen.

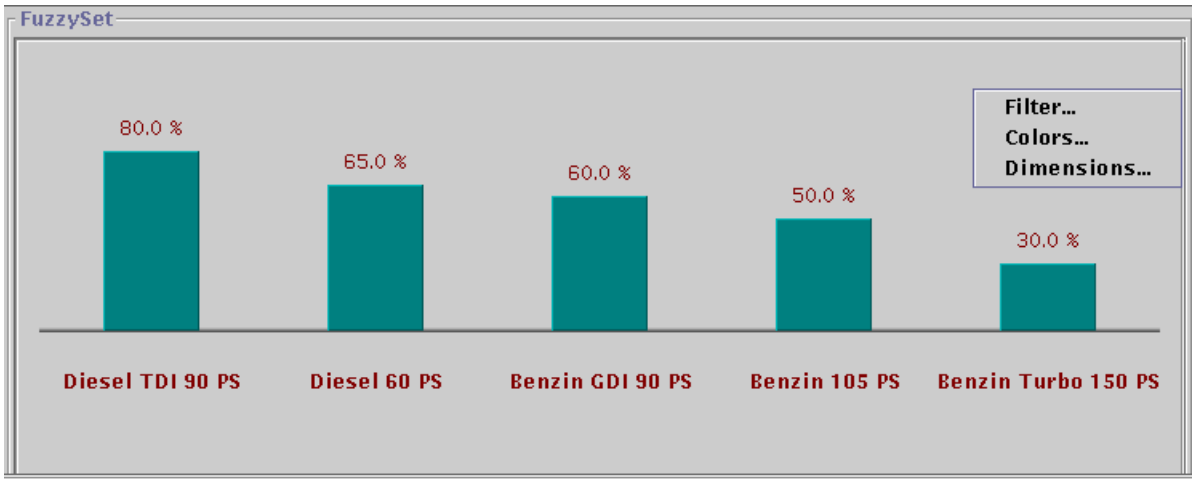


Abbildung 15.2: Balkendarstellung einer Fuzzy-Menge

2D-Darstellung

Bei der 2D-Darstellung werden die einzelnen α-Schnitte einer Fuzzy-Menge (vergleiche Abschnitt 6.2.2 auf Seite 55) als Balken über der Grundmenge Ω aufgetragen. Ein Beispiel zeigt Abbildung 15.2.

Bei dieser Darstellungskomponente ermöglicht ein Kontextmenü (in der Abbildung oben rechts) die Konfiguration der Sicht durch Einstellung von:

- Filterkriterien, so daß bei großen Domänen die Übersichtlichkeit erhöht werden kann, indem einzelne Elemente, deren Wert sich unterhalb einer definierbaren Schwelle befindet, aus der Domäne ausgeblendet werden;
- verschiedenen Farben für die einzelnen α-Schnitte; sowie
- den Abmessungen für Balkenbreite und -abstände in der Darstellung.

3D-Darstellung

Die Darstellung einer einzelnen Fuzzy-Menge in Form eines Balkendiagramms reicht jedoch nicht aus, wenn man einen Gesamtüberblick über viele Fuzzy-Mengen innerhalb eines Systems gewinnen möchte. Da die parallele Darstellung vieler einzelner Fuzzy-Mengen zu unübersichtlich würde, wurde hierfür eine experimentelle 3D-Darstellung entwickelt.

Die einzelnen Fuzzy-Mengen, die wieder in Form von Balkendiagrammen dargestellt sind, lassen sich damit zu Blöcken anordnen, die beispielsweise alle Literale

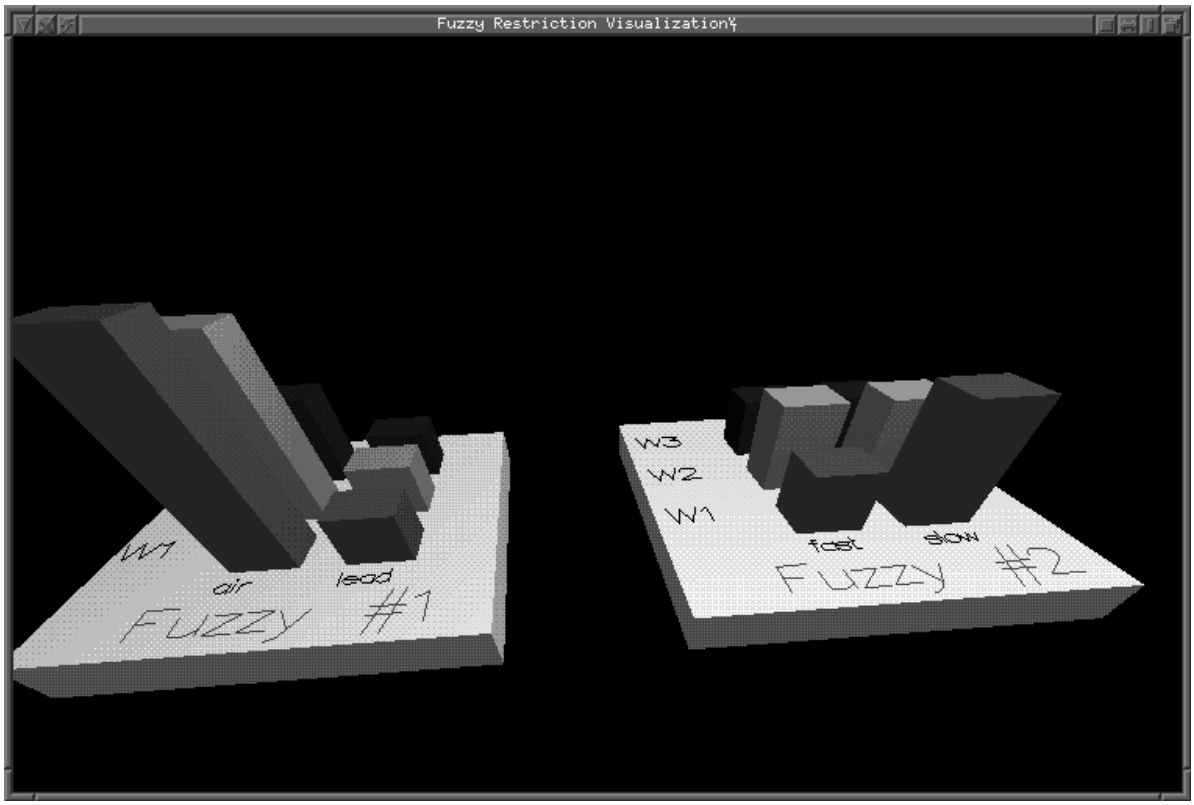


Abbildung 15.3: 3D-Darstellung von Fuzzy-Informationen

einer Klausel oder alle Klauseln einer Formel aufnehmen können. Diese Blöcke lassen sich dann in allen drei Dimensionen anordnen, um so alle Fuzzy-Informationen eines Objektes zusammen mit den Abhängigkeiten anzuzeigen. Durch diese Darstellung kann sich der Anwender dann frei bewegen (Rotation und Translation um alle Raumachsen, Vergrößerungsfaktor, Skalierung, Beleuchtungsänderung, Nebelwurf). Ein Beispiel zeigt Abbildung 15.3.

Die 3D-Darstellung wurde dabei unter Verwendung der Standard-Graphik-Bibliothek *OpenGL* und des *glmonitor*-Werkzeuges realisiert.

15.1.4 Zusammengesetzte Sichten

Die vorgestellten Sichten präsentieren jeweils einzelne Aspekte eines Modells. Für einen Fuzzy-Klienten wird man aus diesen einzelnen Sichten komplexere Oberflächen zusammensetzen. Eine solche zusammengesetzte Darstellungskomponente ist in Abbildung 15.4 auf der nächsten Seite gezeigt: dort sind die Anzeige der Beschreibung eines Konzeptes (oben) mit der Baumdarstellung der Fuzzy Konjunktiven Normalform (links) sowie der zu einem Baumknoten gehörenden Fuzzy-Menge (rechts)

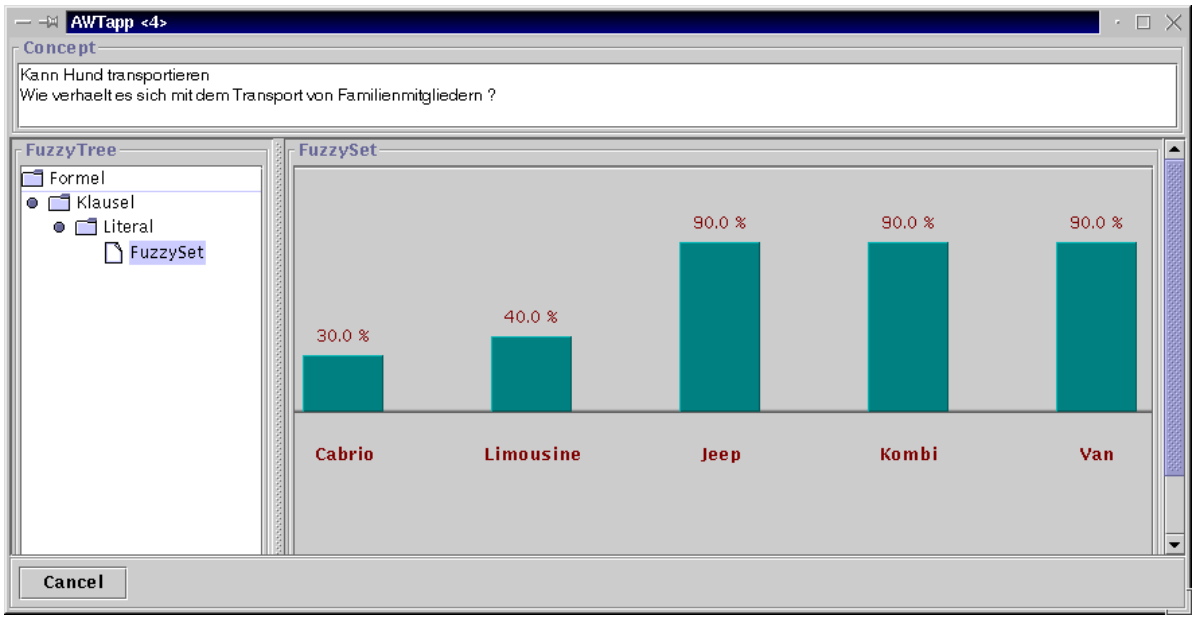


Abbildung 15.4: Zusammengesetzte Darstellungskomponente zur Anzeige eines Konzeptes mit seiner logischen Form und der resultierenden Fuzzy-Menge

kombiniert.

Die einzelnen Präsentationskomponenten sind dabei miteinander verknüpft, so daß beispielsweise nach Anwahl eines Knotens in der Baumstruktur (wie „FuzzySet“ in der Abbildung) die entsprechende Fuzzy-Interpretation in der Balkenkomponente (rechts) und dem zugehörigen Konzept (oben) angezeigt wird. Auf diese Weise ist leicht nachvollziehbar, wie sich etwa die Interpretation einer komplexen Fuzzy-Beschreibung aus den atomaren Bestandteilen zusammensetzt (vergleiche mit Abbildung 8.4 auf Seite 79).

15.1.5 Benutzerinteraktionen

Neben der reinen Darstellung von Fuzzy-Informationen muß die Möglichkeit geboten werden, mit einer Anwendung zu interagieren.

Für solche Benutzerinteraktionen bieten wir einen speziellen Dialog zur Steuerung der Expansions-, Revisions-, und Kontraktionsoperationen aus Kapitel 10 an. Dazu wird zunächst die annotierte Komponente aus der Baumdarstellung ausgewählt, die bearbeitet werden soll. Über ein Menü läßt sich dann — neben einer Reihe primitiver Operationen — ein spezieller Bearbeitungsdialog aufrufen (siehe Abbildung 15.5 auf der nächsten Seite).

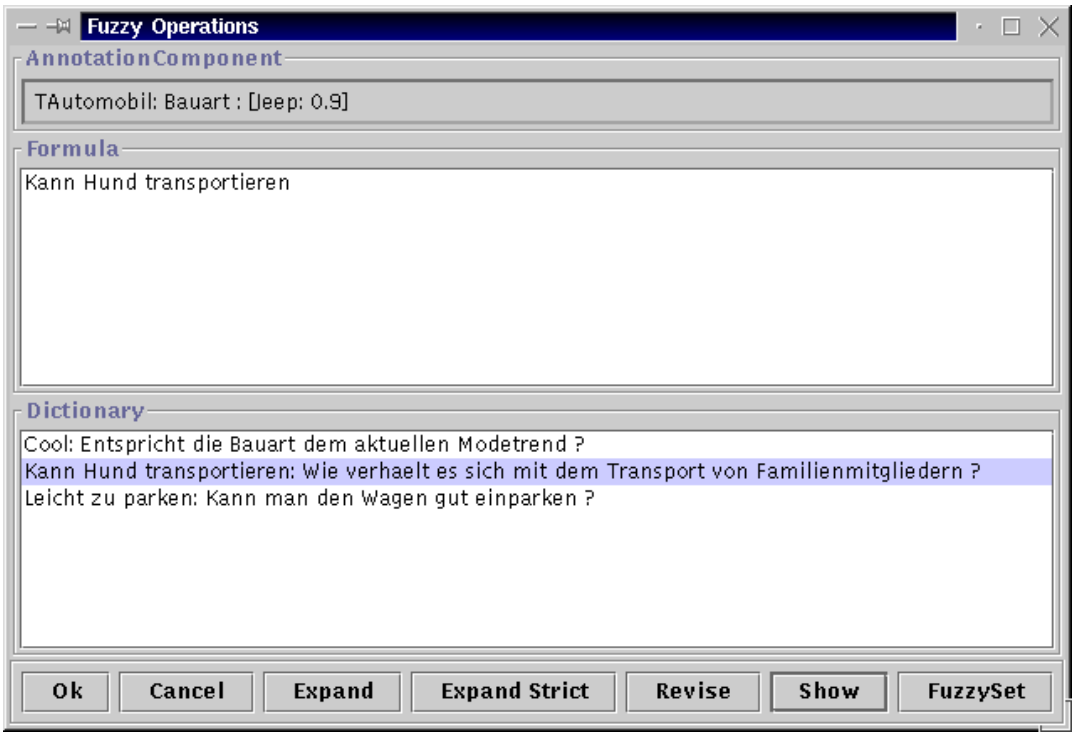


Abbildung 15.5: Steuerung von Operationen

Dieser Dialog erfüllt zwei Aufgaben:

- Zunächst werden aus dem Fuzzy-Lexikon alle Begriffe ausgelesen, die auf die ausgewählte annotierte Struktur angewendet werden können; diese werden dann in einer Liste zur Auswahl angeboten.
- Der Anwender kann jetzt einen Begriff auswählen und damit eine Operation durchführen (Expansion, Revision, Kontraktion, (strikte) Graph-Expansion, Graph-Revision, Graph-Kontraktion).

Schlägt die Operation fehl, weil beispielsweise bei einer Expansion der geforderte Konsistenzgrad nicht erreicht werden kann, bekommt der Benutzer dies über ein zusätzliches Dialogfenster angezeigt. Andernfalls wird die geänderte Liste von Konzepten angezeigt, die sich durch die Operation ergeben. Der Benutzer hat dann die Möglichkeit, weitere Operationen zu starten oder den Dialog zu verlassen.

15.1.6 Implementierung

Die Präsentationskomponenten wurden, wie die Fuzzy-Bibliothek, in der Sprache Java unter Verwendung der Oberflächenbibliothek *Swing* realisiert. Sie sind innerhalb des Paketes *FuzzyGUI* zusammengefaßt (vergleiche auch Abschnitt 13.5 auf

Seite 192). Die universelle Sicht geht zurück auf die vom Autor betreute Studienarbeit [Chr97].[2]

Damit stehen jetzt alle erforderlichen Komponenten für die Klienten-Stufe eines Fuzzy-Informationssystems zur Verfügung. Im nächsten Kapitel dieses Teils wenden wir uns nun der Ressourcen-Stufe zu.

[2]Eine solche universelle Sicht läßt sich nur für Programmiersprachen implementieren, die eine Introspektion ermöglichen, also das Erkunden der Objektstruktur und des Objektverhaltens zur Laufzeit.

Kapitel 16

Fuzzy-Ressourcen

I do not like this word 'bomb.'
It is not a bomb.
It is a device which is exploding.
Jacques Le Blanc, France's ambassador to
New Zealand, on his country's nuclear tests

Ein Fuzzy-Informationssystem benötigt eine Reihe von Ressourcen, wie sie bei der Beschreibung der entsprechenden Stufe in der Referenzarchitektur in Abschnitt 14.3.4 auf Seite 215 beschrieben wurden. In diesem Kapitel widmen wir uns nun der Realisierung dieser Ressourcen, namentlich dem Fuzzy-Lexikon, der Fuzzy-Erweiterungen für Datenbanken und dem Datenaustausch per Fuzzy-XML.

16.1 Fuzzy-Lexikon

Das *Fuzzy-Lexikon* verwaltet die anwendungsspezifischen imperfekten Begriffe und ermöglicht es so, in einem Fuzzy-Informationssystem die Interpretation dieser Begriffe in Form von Fuzzy-Mengen vor einem Anwender zu verstecken. Imperfekte Begriffe wurden bei der Definition der Fuzzy-Atome innerhalb des theoretischen Modells (siehe Abschnitt 8.1 auf Seite 71) eingeführt. Jeder solche Begriff, wie *groß* oder *umweltfreundlich*, ist formal ein Fuzzy-Atom, das Bestandteil der Atommenge eines Fuzzy-Repräsentationssystems (vergleiche Definition 8.1.1 auf Seite 72) ist.

Das objektorientierte Fuzzy-Modell (vergleiche Abschnitt 11.2 auf Seite 158) ermöglicht durch die Verbindung der Fuzzy-Formeln mit objektorientierten Konzepten den Einsatz imperfekter Begriffe in objektorientierten Programmiersprachen, und somit die Entwicklung von Fuzzy-Informationssystemen. Bei der Diskussion von Fuzzy-Klienten in Abschnitt 14.3.1 auf Seite 210 haben wir motiviert, warum das Fuzzy-Lexikon als eigenständige Ressource realisiert werden muß. An dieser Stelle beschreiben wir nun den Entwurf des Fuzzy-Lexikons im Detail.

Ein ähnliches Problem wird in [Bos96] in Form des sogenannten Begriffswörterbuches für Entwurfsanwendungen beschrieben. Allerdings lassen sich die dort vorgestellten Konzepte nicht auf unser Modell übertragen, da dieses Begriffswörterbuch direkt mit dem Datenbank-Schema einer Anwendung verknüpft ist. In unserem Architekturmodell ist ein solches Schema jedoch optional, die imperfekten Begriffe müssen vielmehr mit annotierbaren Konzepten verknüpft werden können, die sich beliebig zur Laufzeit ändern können. Auch die Art der annotierbaren Konzepte — wie Objekte, Attribute und Attributmengen — ist durch den Einsatz des Annotations-Entwurfsmusters beliebig erweiterbar; das Fuzzy-Lexikon soll dabei für jede Anwendung einsetzbar sein.

16.1.1 Entwurf

Wir leiten den kompletten Entwurf schrittweise her. Die Grundaufgabe des Fuzzy-Lexikons besteht in der Verwaltung der anwendungsspezifischen imperfekten Begriffe in Form von Fuzzy-Atomen. Während die Angabe des Fuzzy-Atoms für den Anwender zunächst ausreicht, benötigt ein Fuzzy-Informationssystem aber zusätzlich seine Semantik in Form einer Fuzzy-Menge. Im formalen Modell (vergleiche Definition 8.1.1 auf Seite 72) leistet dies die Funktion $\mu.$, die die Atome einer Begriffsmenge $\mathfrak{A}(\Omega)$ auf ihre jeweilige Fuzzy-Interpretation abbildet:

$$\mu. : \mathfrak{A}(\Omega) \rightarrow F(\Omega), \mathcal{A} \mapsto \mu_{\mathcal{A}}$$

Jedes Fuzzy-Atom steht dabei für einen imperfekten Begriff *(concept)*.

Im Fuzzy-Lexikon wird nun eine endliche Teilmenge von $\mathfrak{A}(\Omega)$ abgelegt, nämlich gerade die vordefinierten anwendungsspezifischen Begriffe. Diese *benannten* Atome, wie „groß“, „schweißbar“ oder „umweltfreundlich“, werden dann mit geeigneten Fuzzy-Interpretationen verknüpft und stehen so für Interaktionen mit Benutzern zur Verfügung.[1]

Die obige Definition muß nun auf eine geeignete objektorientierte Modellierung abgebildet werden, die sich implementieren und als Ressource in einem Fuzzy-Informationssystem einsetzen läßt. Dadurch ergibt sich eine Reihe zusätzlicher Anforderungen, denn für eine Anwendung ist es wichtig, die für eine annotierbare Struktur (Objekt, Attribut oder Attributmenge) passenden Begriffe ausfindig machen zu können. Um dies zu ermöglichen, führen wir das Konzept der *Anwendbarkeit* (applicability) ein. Zu jedem imperfekten Begriff ist eine Anwendbarkeits-Relation definiert, die festlegt, auf welchen Annotations-Komponenten (siehe Abschnitt 12.2 auf Seite 173)

[1] Jede Fuzzy-Menge, die beispielsweise bei Berechnungen innerhalb eines Systems entsteht, verfügt per Definition über ein syntaktisches Fuzzy-Atom. Dieses kann in einer beliebigen freien Syntax konstruiert sein, die die Semantik der Fuzzy-Menge wiederspiegelt: beispielsweise eine Mengenschreibweise, oder technischer gesprochen die Umwandlung einer Fuzzy-Menge in einen textuellen Bezeichner wie mit Hilfe der Java-Methode `toString()`. Solche Fuzzy-Atome müssen jedoch nicht im Fuzzy-Lexikon gespeichert werden.

ein Konzept angewendet werden kann. Dabei ist jedem imperfekten Begriff genau eine Fuzzy-Komponente zugeordnet (vergleiche Abschnitt 13.2 auf Seite 187). Diese Fuzzy-Komponente wird typischerweise ein Fuzzy-Atom sein, wir erlauben darüber hinaus aber auch, zusammengesetzte Fuzzy-Komponenten mit Begriffen zu versehen.

Damit sind die Grundanforderungen an ein Fuzzy-Lexikon erfüllt. Für den praktischen Einsatz benötigen wir jedoch noch die Modellierung zweier weiterer Eigenschaften: die Unterstützung von *Vererbung* sowie die Definition von *Kategorien*.

Kategorien ermöglichen es, imperfekte Begriffe in anwendungsspezifische Gruppen einzuteilen. Damit läßt sich die Menge der für eine Annotationskomponente zulässigen Begriffe nochmals feiner bestimmen. So können Anwendungen unterstützt werden, die einem Benutzer nur solche Begriffe präsentieren wollen, die einer spezifischen *Rolle* des Benutzers entsprechen — beispielsweise könnte man in dem Entscheidungshilfeszenario eine Trennung von Begriffen für Kunden und Berater einführen und so ermöglichen, daß ein Kunde zwar den Begriff „preiswert“ sieht, nicht aber den für Kundenberater definierten Begriff „hohe Marge“. Kategorien sind in einer Baumhierarchie angeordnet, die Wurzel bildet die Kategorie „alle Kategorien“. Modellieren läßt sich dies einfach als benannte Auto-Assoziation auf Begriffen.

Die Aufnahme einer Vererbungsbeziehung schließlich ermöglicht die Abbildung der Klassenhierarchie eines Fuzzy-Informationssystems auf eine Vererbungshierarchie der Begriffe im Fuzzy-Lexikon. So wird vermieden, daß für jede Unterklasse grundsätzlich neue Begriffe definiert werden müssen: finden sich zu einer gegebenen Annotationskomponente keine passenden Konzepte, kann die Suche entlang der Vererbungshierarchie bis zur Wurzel fortgesetzt werden. Dazu wird die Klassenhierarchie über eine weitere Auto-Assoziation auf die Annotationskomponenten abgebildet.

Den kompletten Entwurf des Fuzzy-Lexikons zeigt Abbildung 16.1 auf der nächsten Seite.

16.1.2 Realisierung

Das Fuzzy-Lexikon wurde mit einer Reihe von Java-Klassen realisiert, die in einem Paket zusammengefaßt sind. Dabei realisiert die Klasse `Lexicon` die Schnittstelle für den Anwender. Mit ihr können Begriffe angelegt sowie Anwendbarkeiten eingetragen und abgerufen werden. Sie dient der Verwaltung der einzelnen Konzepte sowie der Kategorie- und Vererbungshierarchie.

Die Klassen `AnnotationComponent` zur Verwaltung kompositer Fuzzy-Strukturen und `FuzzyComponent` zur Annotation wurden bereits in den Abschnitten 12.2 und 13.2 auf den Seiten 173 und 187 beschrieben.

Beispielcode Der folgende Programmcode zeigt beispielhaft, wie mit Hilfe des Fuzzy-Lexikons ein neues Konzept angelegt werden kann, wie eine neue Anwendbarkeit

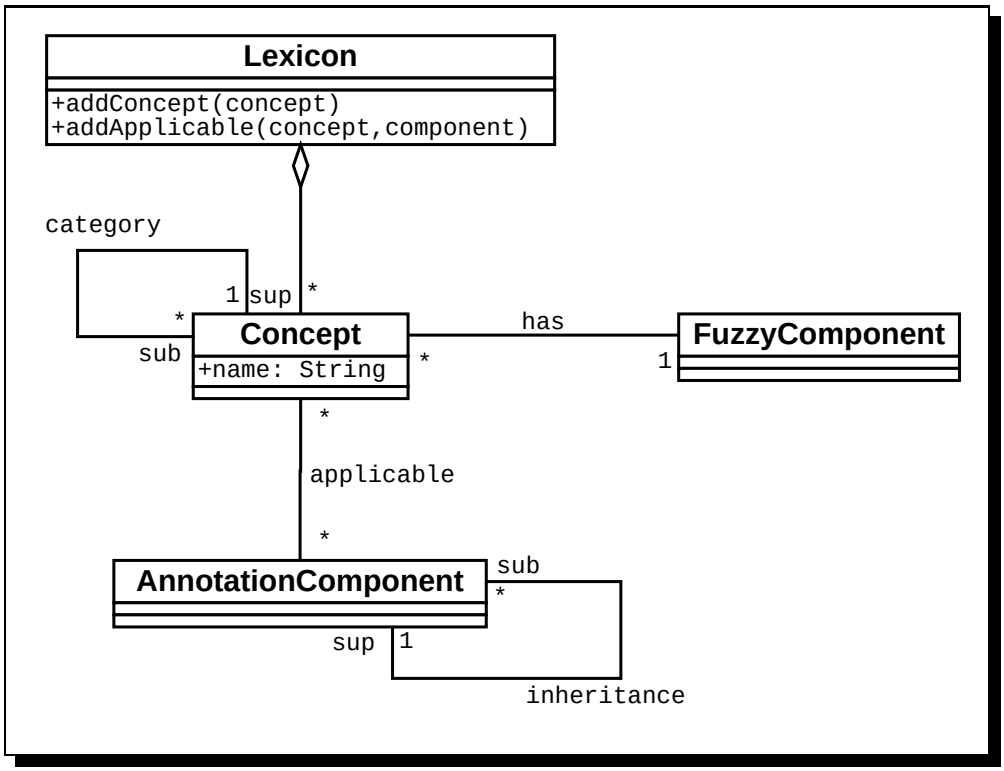


Abbildung 16.1: Entwurf des Fuzzy-Lexikons

eines Konzeptes eingetragen wird und wie sich die zu einer Annotationskomponente passenden Konzepte finden lassen:

```
// Erzeuge ein neues Konzept "groß" zusammen mit
// der Interpretation "myGroßAtom"
myConcept = new Concept( "groß", myGroßAtom );

// Trage das Konzept "groß" in das Lexikon ein
myLexicon.add( myConcept );

// Füge eine Anwendbarkeit des Konzeptes hinzu
myLexicon.addApplicable( myConcept, mySkalarAnnotation );

// Finde alle anwendbaren Konzepte für eine Annotation
Vector myConcepts = myLexicon.getConcepts( mySkalarAnnotation );
```

16.1.3 Werkzeug zur Lexikonverwaltung

Zum Anlegen, Verwalten und Ändern von Einträgen in einem anwendungsspezifischen Fuzzy-Lexikon existiert zusätzlich ein Werkzeug mit einer graphischen Ober-

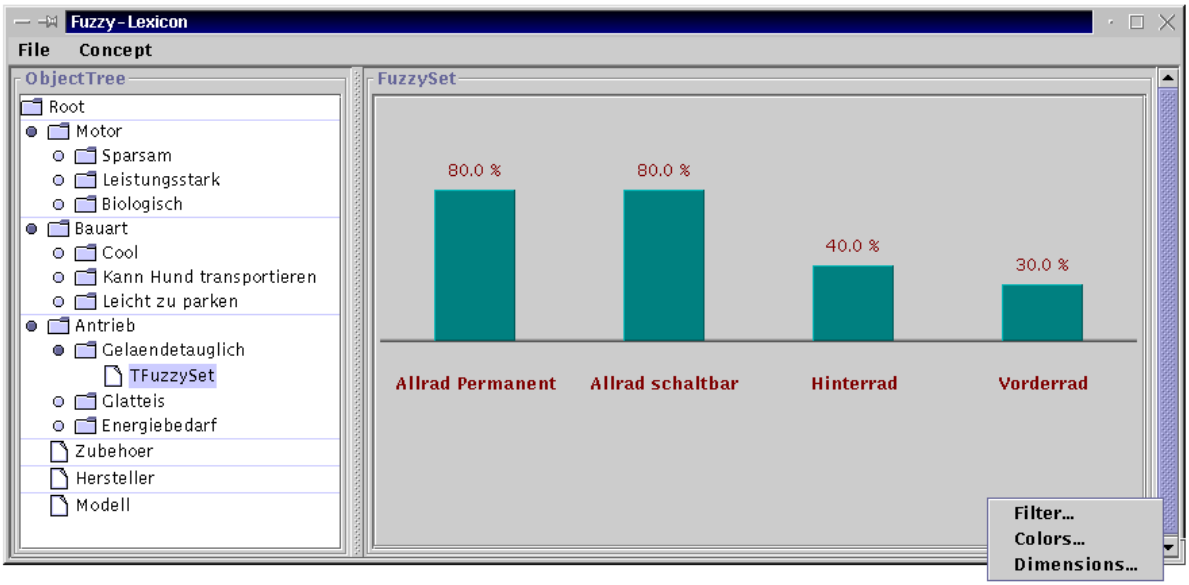


Abbildung 16.2: Fuzzy-Lexikon mit Objekt- und Fuzzy-Browser

fläche. Dieses enthält einen Objektbrowser, der es ermöglicht, die Klassen und Attribute einer Anwendung zu durchsuchen und mit Fuzzy-Begriffen zu verknüpfen. Existierende Begriffe können so einfach graphisch dargestellt und neue Konzepte angelegt werden.

Abbildung 16.2 zeigt ein Beispiel aus dem Fuzzy-Entscheidungshilfeszenario: dargestellt ist eine Auto-Klasse, die Attribute wie `Motor`, `Bauart` und `Antrieb` enthält. Auf diesen Attributen sollen nun imperfekte Begriffe wie „leistungsstark“ für den Motor oder „kann Hund transportieren“ für die Bauart mit passenden Fuzzy-Interpretationen definiert werden. Die Abbildung zeigt den für das Attribut `Antrieb` definierten imperfekten Begriff „geländetauglich“ (linke Seite) zusammen mit seiner Interpretation als Fuzzy-Menge (rechte Seite).

Zur Eingabe eines neuen Begriffes wählt man zuerst im Objektbrowser die zugehö-

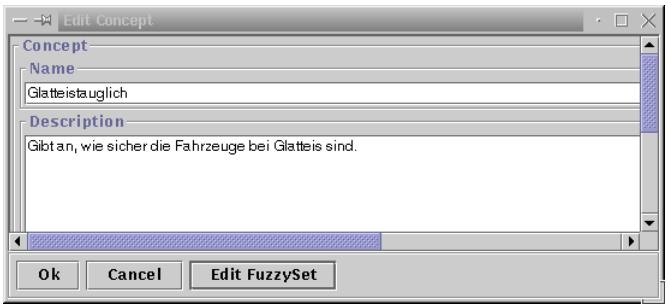


Abbildung 16.3: Anlegen eines neuen Konzeptes

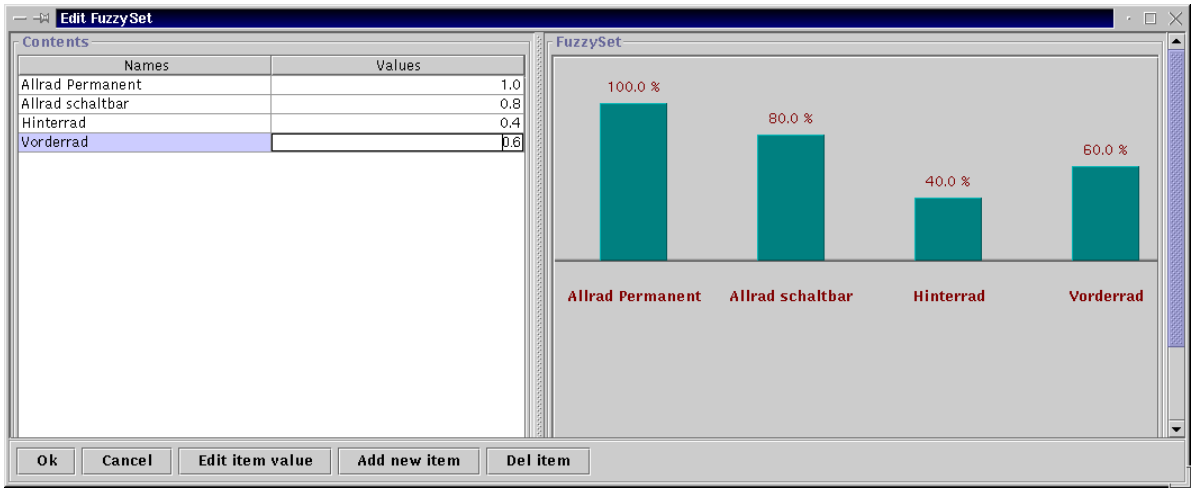


Abbildung 16.4: Eingabe der Fuzzy-Interpretation eines Konzeptes

rige Strukturkomponente aus. Anschließend kann ein bestehender Begriff verändert oder ein neuer angelegt werden; Abbildung 16.3 auf der vorherigen Seite zeigt als Beispiel den Begriff „glatteistauglich“ für den Antrieb.

Für einen solchen Begriff muß schließlich die Fuzzy-Interpretation festgelegt werden, dies geschieht in einer weiteren Eingabemaske, wie in Abbildung 16.4 zu sehen ist.

Bei der Entwicklung des Werkzeuges wurden die in Abschnitt 15.1 auf Seite 219 vorgestellten Oberflächenkomponenten eingesetzt; somit bieten sich hier die gleichen Möglichkeiten zur Beeinflussung der Darstellung, etwa zur Ausblendung bestimmter Werte, wie dort geschildert.

16.1.4 Verwaltung von Transformationsfunktionen

Als nächstes modellieren wir die Transformationsfunktionen für Fuzzy-Mengen, die wir im theoretischen Modell (siehe Abschnitt 9.2 auf Seite 96) eingeführt haben. Diese hängen zwar nicht direkt mit dem Fuzzy-Lexikon zusammen, sind aber aus Gründen der Übersichtlichkeit im gleichen Paket enthalten.

Die Erweiterung des Fuzzy-Lexikons um die Verwaltung dieser Transformationsfunktionen zeigt Abbildung 16.5 auf der nächsten Seite. Hierbei sind Fuzzy-Komponenten mit den Domänen ihrer Fuzzy-Interpretationen assoziiert, auf denen durch Auto-Assoziation Transformationsfunktionen definiert sein können. Jede Transformation *von* einer Domäne *zu* einer anderen ist mit einer Assoziationsklasse versehen, welche die entsprechende Transformationsfunktion implementiert. Hierbei werden zwei vordefinierte Funktionen angeboten, die eine einfache konstante beziehungsweise lineare Transformation realisieren und dafür mit den konkreten gegebenen Domänen parametrisiert werden. Zusätzlich können beliebige anwendungsspezifische

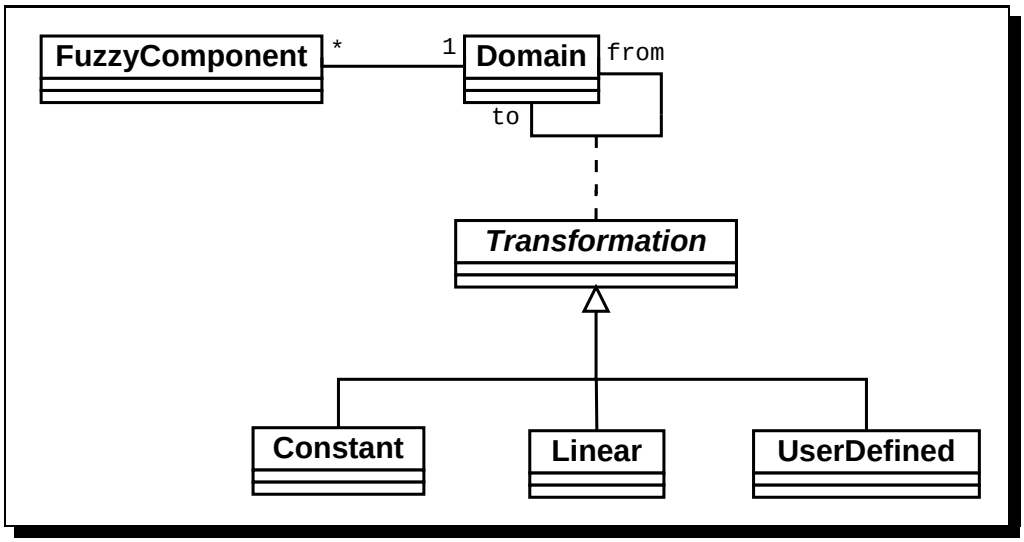


Abbildung 16.5: Modellierung von Transformationsfunktionen

Transformationsfunktionen als Unterklasse der abstrakten Oberklasse `Transformation` realisiert werden.

16.2 Fuzzy-Datenbanken

Die meisten Informationssysteme besitzen eine Ressource zur persistenten Speicherung von Daten, typischerweise in Form eines relationalen oder objektorientierten Datenbanksystems. In einem Fuzzy-Informationssystem fallen nun zusätzliche Daten an, die in Annotationen und Facetten (vergleiche Kapitel 12) strukturiert sind. Sollen auch diese persistent gespeichert werden, muß das Datenbankschema der Anwendung erweitert werden, wie wir in Abschnitt 14.3.4 auf Seite 215 ausgeführt haben. An dieser Stelle betrachten wir nur die für relationale Datenbanken erforderliche Abbildung, da die Möglichkeit der Speicherung von Objekten in einer objektorientierten Datenbank nicht weiter diskutiert werden muß.

Die Aufgabe ist hier also, ein existierendes relationales Datenbankschema einer Anwendung systematisch zur Aufnahme von Fuzzy-Informationen zu erweitern. Diese Änderung soll auch für ein solches Schema möglichst orthogonal erfolgen, damit existierende Anwendungen, die keine Fuzzy-Informationen nutzen wollen, unbeeinträchtigt weiter arbeiten können.

Man beachte, daß sich die hier vorgestellte Vorgehensweise zur Realisierung von „Fuzzy-Datenbanken“ damit erheblich von den bekannten Ansätzen unterscheidet, wie wir sie in Abschnitt 2.2 auf Seite 12 angerissen haben. Denn dort liegt der Fokus typischerweise auf der Erweiterung der relationalen Algebra um Fuzzy-Mengen. Wir

gehen jedoch von einem klassischen relationalen Datenbankmanagementsystem aus, wie es in freier und kommerzieller Form verfügbar ist, und das in die mehrstufige Architektur einer Anwendung eingebettet ist, wie in Kapitel 14 beschrieben wurde. Es wird also nicht das *Datenbankmanagementsystem* erweitert, sondern die konkrete *Datenbank* einer Anwendung, wie es auch von den Anforderungen MODERNE ARCHITEKTUR (Seite 27) und INTEROPERABILITÄT (Seite 34) gefordert wird.

Wir setzen ein existierendes Schema einer Anwendung voraus, das zur persistenten Speicherung der Anwendungsdaten erstellt wurde. Sollen nach einer Fuzzy-Erweiterung auch Teile der Fuzzy-Daten persistent gespeichert werden, muß man das Schema zuvor in geeigneter Weise erweitern. In den nachfolgenden Abschnitten wird nun gezeigt, wie diese Erweiterungen systematisch vorgenommen werden können.

16.2.1 Speicherung von Fuzzy-Mengen und Fuzzy-Formeln

Wir beginnen mit der Abbildung von Fuzzy-Mengen und den darauf aufbauenden komplexen Fuzzy-Literalen, -Klauseln und -Formeln auf relationale Tabellen.

Wir gehen hier davon aus, daß Fuzzy-Mengen in horizontaler Repräsentation gespeichert werden (vergleiche Abschnitt 6.2.2 auf Seite 55). Jede Fuzzy-Menge besteht so aus einer Menge von α-Schnitten, mit je einem Schnitt pro Niveau *(level)*:

```
FuzzySet( FuzzySetId, Level, AlphaCutId )
```

Hier definiert `FuzzySet` eine Datenbanktabelle mit den Spalten `FuzzySetId, Level` und `AlphaCutId`; unterstrichene Spalten geben den Primärschlüssel einer Tabelle an. Die Domänendefinitionen sind aus Gründen der Übersichtlichkeit nicht explizit gezeigt; wir gehen hier davon aus, daß diese in einem geeigneten Datenwörterbuch *(data dictionary)* definiert sind.

Jeder α-Schnitt ist wieder eine Menge von Werten, nämlich die Teilmenge der Grundmenge, die in dem α-Schnitt enthalten ist:

```
AlphaCut( AlphaCutId, value )
```

Damit können wir einfache Fuzzy-Mengen ablegen. Darauf aufbauend werden jetzt komplexe Fuzzy-Beschreibungen definiert. Ein Fuzzy-Atom (Definition 8.1.1 auf Seite 72) ist gerade ein imperfekter Begriff:

```
FuzzyAtom( FuzzyAtomId, ConceptId )
```

Ein Fuzzy-Literal (Definition 8.1.3 auf Seite 74) ist entweder ein Atom oder die Negation eines Atoms:

```
FuzzyLiteral( FuzzyLiteralId, FuzzyAtomId, Negated )
```

Fuzzy-Klauseln (Definition 8.1.4 auf Seite 75) sowie Fuzzy-Formeln (Definition 8.1.7 auf Seite 77) sind auf der syntaktischen Ebene reine Mengen und lassen sich so leicht auf das relationale Datenmodell abbilden:

```
FuzzyClause( FuzzyClauseId, FuzzyLiteralId )
FuzzyFormula( FuzzyFormulaId, FuzzyClauseId )
```

Man beachte, daß wir hier nicht vorsehen, die Fuzzy-Interpretationen der komplexen Fuzzy-Ausdrücke persistent zu speichern. Diese sollten in einer Anwendung berechnet (materialisiert) werden, sobald sie benötigt werden.

16.2.2 Speicherung von Annotationen und Facetten

Zur orthogonalen Erweiterung von Anwendungen um Fuzzy-Informationen haben wir das Konzept der Annotation und Facetten eingeführt (vergleiche Kapitel 12 ab Seite 171). Ein annotierbares Konzept wie ein Attribut, eine Attributmenge oder ein Objekt kann mit einer Annotation versehen werden, die beliebig viele Facetten mit Zusatzinformationen aufnehmen kann. Für eine relationale Datenbank wird man jeden Facettentyp auf eine eigene Tabelle abbilden. Fuzzy-Informationssysteme benutzen zumindest Facetten zur Speicherung von Fuzzy-Formeln:

```
FuzzyFacet( FuzzyFacetId, AnnotationId, FuzzyFormulaId)
```

sowie von Abhängigkeiten (vergleiche Abschnitt 13.3 auf Seite 190):

```
DependencyFacet( DependencyFacetId, AnnotationId,
                    DependsFromAnnotationId )
```

und optional String-Facetten:

```
StringFacet( StringFacetId, AnnotationId, Comment)
```

Schwieriger ist die Abbildung der Annotationen, da die annotierbaren Konstrukte des objektorientierten Systems auf vielfältige Weise auf relationale Tabellen abgebildet sein können: angefangen von einzelnen Spalten im Fall von Attributen bis hin zu mehreren, miteinander verknüpften Tabellen im Fall komplexer Objekte. Die einfachste Möglichkeit zur Abbildung des Annotationskonzeptes wäre, die entsprechende Datenbanktabelle um ein oder mehrere Spalten zur Aufnahme der `AnnotationId` zu erweitern. Im Fall eines Personen-Objektes mit den Attributen `Name`, `Größe` und `Alter` beispielsweise, bei dem die Größe und das Alter annotiert werden sollen, würde dies folgendermaßen aussehen;

```
Person( PersonId, Name, Größe, Alter,
          GrößeAnnotationId, AlterAnnotationId )
```

Dies ist jedoch keine orthogonale Erweiterung mehr, da in das Schema einer Anwendung direkt eingegriffen wird. Überdies ist diese Modellierung sehr unflexibel, da zur Laufzeit entstandene neue Annotationen nicht gespeichert werden können. Wünschenswert wäre also eine Modellierung, die eine Annotation direkt mit einer annotierbaren Struktur verknüpft:

```
Annotation( AnnotationId, AnnotatedStructureId )
```

Hier stößt man jedoch leider an die Grenzen der Ausdrucksmächtigkeit des relationalen Datenmodells. Eine annotierte Struktur kann ein einzelne Spalte sein, aber auch eine Kombination von Spalten, die aus mehreren Tabellen stammen können, oder eine komplette Tabelle. Dies läßt sich nur abbilden, wenn man die sogenannten Systemtabellen zu Hilfe nimmt, die in einem relationalen Datenbanksystem das Schema selbst in Form relationaler Tabellen enthalten. Annotation lassen sich dann als Baumstruktur abbilden, bei der ein Blatt entweder eine Zeilen/Spalten-Kombination oder eine ganze Tabelle darstellt:

```
AnnotatedStructure( AnnotatedStructureId,
                      ParentAnnotatedStructureId,
                      TableId, ColumnId, RowId )
```

Die Annotation eines einzelnen Attributes eines Objektes, das auf eine Spalte einer Tabelle mit einem bestimmten Schlüssel abgebildet wurde, findet sich dann in Form einer Zeile in der `AnnotatedStructure`-Tabelle, die über den Systemkatalog die Tabelle und die entsprechende Spalte des Attributes angibt, sowie die Zeile des Attributes in Form des Schlüssels der Tabelle. Die Annotation zweier Attribute läßt sich über eine Baumstruktur mit zwei Blättern abbilden, wobei jedes Blatt eines der beteiligten Attribute aufnimmt und auf die Wurzel verweist, die die Annotation als ganzes repräsentiert. Da der Schlüssel einer Tabelle selbst wieder aus mehreren Spalten zusammengesetzt sein kann, muß man in der `RowId` entweder grundsätzlich alle Spalten mit einem geeigneten Trennformat abspeichern, oder die Struktur der Schlüssel selbst nochmals unter Verweis auf Systemtabellen explizit ausmodellieren.

Diese Lösung kann beliebige Annotationen zur Laufzeit eines Systems aufnehmen. Man sollte sie allerdings nur wählen, wenn ein existierendes Schema aus den oben genannten Gründen nicht statisch erweitert werden kann, da man sich die hohe Flexibilität durch einen entsprechend hohen Aufwand beim Abbilden der Annotationen erkauft. Einen Ausweg bieten an dieser Stelle Datenbankmanagementsysteme mit objekt-relationalen Erweiterungen, was jedoch über den Rahmen dieser Arbeit hinausführt.

16.2.3 Speicherung des Fuzzy-Lexikons

Es verbleibt noch die Abbildung des oben definierten Fuzzy-Lexikons. Dazu müssen wir die imperfekten Begriffe mit den zugehörigen Kategorien und der Referenz auf die Fuzzy-Interpretation speichern:

```
Category( CategoryId, ParentCategoryId, CategoryName )
Concept( ConceptId, CategoryId, ConceptName, FuzzySetId )
```

Zusätzlich benötigen wir Informationen darüber, auf welche Annotationskomponenten ein Begriff anwendbar ist:

```
AnnotationComponent( AnnotationComponentId,
                     ParentAnnotationComponentId )
Applicable( ConceptId, AnnotationComponentId )
```

Die Tabelle `AnnotationComponent` benötigt noch eine zusätzliche Information zur Identifikation der (atomaren oder komplexen) annotierbaren Struktur im objektorientierten System. Diese ist jedoch implementierungsspezifisch, da sie sowohl von der abzubildenden Programmiersprache als auch der Abbildungsstrategie abhängt. Die Möglichkeiten reichen hier von einem einfachen Identifikationsstring bis hin zu einer expliziten Ausformulierung wie mit der oben gezeigten `AnnotatedStructure`-Tabelle. Im letzten Fall würde man die `AnnotatedStructureId` als Fremdschlüssel in die `AnnotationComponent`-Tabelle aufnehmen. Die auf Tabellen abgebildeten Annotationskomponenten würden dann in der `AnnotatedStructure`-Tabelle in Form zusätzlicher Zeilen gespeichert; in diesem Fall muß die `RowId` durch einen NULL-Wert anzeigen, daß der Eintrag nicht eine konkrete annotierte Instanz, sondern die grundsätzliche Annotierbarkeit der entsprechenden Struktur anzeigt.

16.3 Fuzzy-XML

Für den Datenaustausch zwischen der Geschäftsprozeß-Stufe und Systemen der Ressourcen-Stufe muß ein geeignetes Datenformat verwendet werden. Während hier in der Vergangenheit überwiegend anwendungsspezifische Binärformate dominierten, setzt sich seit einiger Zeit das textformatbasierte XML *(eXtensible Markup Language)* als Standard durch. Näheres zu der Definition und den Eigenschaften von XML findet sich etwa in [KST02].

Wir gehen daher an dieser Stelle davon aus, daß auch der Datenaustausch zwischen der Geschäftsprozeß-Stufe eines Fuzzy-Informationssystems und Systemen der Ressourcen-Stufe über XML erfolgt. Gezeigt werden muß also, wie ein anwendungsspezifisches XML-Dokument systematisch zur Aufnahme der neuen Fuzzy-Informationen erweitert werden kann.

```
<!-- Annotations and Facets -->
<!ELEMENT annotation (facet*)>
<!ATTLIST annotation
          foid IDREFS #IMPLIED>
<!ELEMENT fuzzyobjectid EMPTY>
<!ATTLIST fuzzyobjectid
          foid ID #REQUIRED>
<!ELEMENT facet (fuzzyfacet
                | dependencyfacet | stringfacet)>
<!ELEMENT fuzzyfacet (fuzzyformula)>
<!ELEMENT dependencyfacet EMPTY>
<!ATTLIST dependencyfacet
          foid IDREFS #IMPLIED>
<!ELEMENT stringfacet (#PCDATA)>

<!-- Fuzzy Formulas and Fuzzy Lexicon -->
<!ELEMENT fuzzyformula (fuzzyclause*)>
<!ELEMENT fuzzyclause (fuzzyliteral*)>
<!ELEMENT fuzzyliteral (fuzzyatom)>
<!ATTLIST fuzzyliteral
          negated (true | false) ”false”>
<!ELEMENT fuzzyatom EMPTY>
<!ATTLIST fuzzyatom
          conceptid IDREF #IMPLIED>
<!ELEMENT concept (name, fuzzyset)>
<!ATTLIST concept
          conceptid ID #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT fuzzyset ((level, value*)*)>
<!ELEMENT level (#PCDATA)>
<!ELEMENT value (#PCDATA)>
```

Abbildung 16.6: Dokumenttypdefinition (DTD) für Fuzzy-XML-Dokumente

16.3.1 Fuzzy-Dokumenttypdefinition

Für die Verarbeitung von XML-Dokumenten ist wichtig, daß diese *wohlgeformt* und *gültig* sind. Während die Wohlgeformtheit durch einfache Syntaxanalyse überprüft werden kann, benötigt man für die Überprüfung der Gültigkeit eine Beschreibung der legalen Dokumente, entweder in Form einer *Dokumenttypdefinition* (DTD) oder eines *XML-Schemas*.

Wir verwenden im folgenden Dokumenttypdefinitionen (DTDs), da diese die Min-

destvoraussetzung sind, um XML-Dokumente auf Gültigkeit zu überprüfen. Die gezeigten Ansätze zur Modellierung von Fuzzy-Informationen lassen sich jedoch gleichermaßen auf XML-Schemata übertragen.

Abbildung 16.6 auf der vorherigen Seite zeigt die komplette Dokumenttypdefinition für Fuzzy-XML-Dokumente. Dabei realisieren die Attribute `ID` und `IDREF` den Schlüssel/Fremdschlüsselmechanismus in XML.

16.3.2 Fuzzy-XML-Dokumente

Für die Übertragung eines Fuzzy-XML-Dokumentes müssen die Fuzzy-Daten mit ihren zugehörigen scharfen Daten kombiniert werden. Auch hier müssen die Anforderungen PARTIELLE UNSCHÄRFE (Seite 30) und INTEROPERABILITÄT (Seite 34) berücksichtigt werden. Auf die Aufgabe des Datenaustauschs übertragen heißt dies, daß eine Änderung des Empfängers nicht vorausgesetzt werden darf. Vielmehr müssen die Zusatzinformationen in einer Form vorliegen, die von einem fähigen Empfänger ausgewertet, von anderen aber ignoriert werden kann.

Diese Eigenschaft läßt sich durch die Verwendung sogenannter *Namensräume* (name spaces) erreichen. Ein XML-Namensraum ermöglicht es, Elemente eines XML-Dokumentes einen bestimmten, eindeutigen Bezeichner zuzuordnen. Ordnet man die scharfen Daten eines XML-Dokumentes einem anwendungsspezifischen und die Fuzzy- und sonstigen Zusatzdaten einem weiteren Namensraum zu, kann ein Empfänger genau diejenigen Teile eines Dokumentes selektieren, die ihn interessieren und die er verarbeiten kann. Ein Empfänger, der nicht mit Fuzzy-Daten umgehen kann, wird XML-Dokumentteile, die aus dem für ihn unbekannten Namensraum stammen, gar nicht sehen und somit in der Verarbeitung der bekannten Dokumentteile auch nicht behindert. Die Fuzzy-Daten stehen jedoch jederzeit zur Verfügung und können nach einer Erweiterung des Empfängers sofort verwendet werden.

Technisch wird dies durch das Einfügen einer *Namensraumdeklaration* in das auszutauschende XML-Dokument erreicht. Damit wird angezeigt, daß Elemente des Dokumentes aus verschiedenen Namensräumen stammen. Beide Namensräume müssen dabei durch einen eindeutigen Bezeichner, dem URI *(Uniform Resource Identifier)* deklariert werden; das folgende Beispiel zeigt die Kombination eines Namensraumes zum Austausch von Reengineering-Daten mit einem weiteren Namensraum für Fuzzy-Informationen:

```
<?xml version='1.0' encoding='iso-8859-1'?>
<fuzzy_reengineering
       xmlns:reengineering="uri_reengineering"
       xmlns:fuzzy="uri_fuzzy">
```

Hierdurch wird ein Präfix `"reengineering"`, sowie ein Präfix `"fuzzy"` definiert, die dann zur Auszeichnung der Elemente eines Dokumentes verwendet werden können:

```
<reengineering:typ>int</reengineering:typ>
<fuzzy:fuzzyformula>
            <fuzzy:fuzzyclause>
            ...
            </fuzzy:fuzzyclause>
</fuzzy:fuzzyformula>
```

Damit sind wir schon fast am Ziel, jedoch sind jetzt alle Standardelemente (wie `typ` im obigen Beispiel) mit einem zusätzlichen Namensraumbezeichner versehen. Dies läßt sich aber umgehen, indem der anwendungsspezifische Namensraum als Standard-Namensraum deklariert wird. Damit müssen nur noch die Fuzzy-Elemente gesondert ausgezeichnet werden, alle anderen Elemente werden automatisch dem Standard-Namensraum zugeschlagen:

```
<?xml version='1.0' encoding='iso-8859-1'?>
<reengineering xmlns="uri_reengineering">
<typ>int</typ>
<fuzzy:fuzzyformula xmlns:fuzzy="uri_fuzzy">
       <fuzzy:fuzzyclause>
       ...
       </fuzzy:fuzzyclause>
</fuzzy:fuzzyformula>
```

Falls innerhalb eines Systems nicht DTDs zur Beschreibung der XML-Daten, sondern XML-Schema verwendet wird, läßt sich die hier beschriebene Trennung nach dem gleichen Prinzip durchführen. Man gewinnt dabei jedoch noch zusätzliche Möglichkeiten, wie die direkte Erweiterung eines existierenden Schemas um Fuzzy-Informationen mit Hilfe von Vererbung.[2]

Beispiel (Fuzzy-XML-Dokument) Wir führen das Beispiel des Fuzzy-Reengineering fort. Die in Abbildung 14.7 auf Seite 218 entwickelte Architektur soll dabei um einen Datenaustausch mit externen Systemen erweitert werden, die mit dem hier vorgestellten Fuzzy-XML-Format erfolgen soll. Wir zeigen ein Beispiel-XML-Dokument, das Informationen über den ermittelten Typ und Verwendungszweck einer Variablen i an ein externes System übermittelt; dabei verwenden wir das Ergebnis des Berechnungsbeispiels aus Abschnitt 10.5.2. Ein XML-Dokument, welches nur das scharfe Ergebnis überträgt, könnte folgendermaßen aussehen:

```
<?xml version='1.0' encoding='iso-8859-1'?>
<variable>
    <name>i</name>
    <typ>int</typ>
```

[2] In der Praxis bereitet die XML-Verifikation bei der gleichzeitigen Verwendung von Namensräumen bei vielen Implementierungen Schwierigkeiten, was ein weiterer Grund für die Verwendung von XML-Schema sein kann.

```
        <verwendung>index</verwendung>
    </variable>
```

Dieses XML-Dokument soll erweitert werden, um auch die gewonnen Fuzzy-Daten an die externen Ressource zu übertragen. Hierfür nehmen wir ebenfalls die Ergebnisse des oben erwähnten Beispiels (vergleiche Abbildung 10.9 auf Seite 142). Dies führt zu folgendem, nur geringfügig längeren XML-Dokument:

```
<?xml version='1.0' encoding="iso-8859-1"?>
<reengineering xmlns="uri_reengineering">
<typ>int</typ>
<variable>
    <name>i</name>
    <typ>int
        <fuzzy:fuzzyobjectid foid="fo1" xmlns:fuzzy="uri_fuzzy"/>
        <fuzzy:annotation>
               <fuzzy:fuzzyobjectid foid="fo2"/>
               <fuzzy:fuzzyfacette>
                   <fuzzy:fuzzyformula>
                       <fuzzy:fuzzyclause>
                           <fuzzy:fuzzyliteral>
                               <fuzzy:fuzzyatom conceptid="c01"/>
                           </fuzzy:fuzzyliteral>
                       </fuzzy:fuzzyclause>
                       <fuzzy:fuzzyclause>
                           <fuzzy:fuzzyliteral>
                               <fuzzy:fuzzyatom conceptid="c02"/>
                           </fuzzy:fuzzyliteral>
                       </fuzzy:fuzzyclause>
                   </fuzzy:fuzzyformula>
               </fuzzy:fuzzyfacette>
               <fuzzy:dependencyfacette foid="fo4"/>
        </fuzzy:annotation>
    </typ>
    <verwendung>index
        <fuzzy:fuzzyobjectid foid="fo3"/>
        <fuzzy:annotation>
            <fuzzy:fuzzyobjectid foid="fo4"/>
            <fuzzy:fuzzyfacette>
                <fuzzy:fuzzyformula>
                    <fuzzy:fuzzyclause>
                        <fuzzy:fuzzyliteral>
                            <fuzzy:fuzzyatom conceptid="c03"/>
                        </fuzzy:fuzzyliteral>
                    </fuzzy:fuzzyclause>
                    <fuzzy:fuzzyclause>
```

```
                        <fuzzy:fuzzyliteral>
                            <fuzzy:fuzzyatom conceptid="c04"/>
                        </fuzzy:fuzzyliteral>
                    </fuzzy:fuzzyclause>
                </fuzzy:fuzzyformula>
            </fuzzy:fuzzyfacette>
        </fuzzy:annotation>
    </verwendung>
    <fuzzy:annotation foid="fo1_fo2">
        <fuzzy:fuzzyobjectid foid="fo5"/>
        <fuzzy:stringfacette>
            "Auch laut Dokumentation (Stand 20.06.1969)
            eine zur Berechnung verwendete Index-Variable"
        </fuzzy:stringfacette>
    </fuzzy:annotation>
</variable>

<!-- Nun folgt die Definition der Konzepte
     mit Name und Fuzzy-Interpretation -->
<fuzzy:concept conceptid="c01">
    <fuzzy:name>Verwendung von i nach Quellcodeanalyse</fuzzy:name>
    <fuzzy:fuzzyset>
        <fuzzy:level>50</fuzzy:level>
        <fuzzy:value>string</fuzzy:value>
        <fuzzy:level>75</fuzzy:level>
        <fuzzy:value>float</fuzzy:value>
        <fuzzy:level>90</fuzzy:level>
        <fuzzy:value>int</fuzzy:value>
    </fuzzy:fuzzyset>
</fuzzy:concept>
<fuzzy:concept conceptid="c02">
    <fuzzy:name>Verwendung von i nach Datenflußanalyse</fuzzy:name>
    <fuzzy:fuzzyset>
        <fuzzy:level>10</fuzzy:level>
        <fuzzy:value>float</fuzzy:value>
        <fuzzy:level>50</fuzzy:level>
        <fuzzy:value>string</fuzzy:value>
        <fuzzy:level>90</fuzzy:level>
        <fuzzy:value>int</fuzzy:value>
    </fuzzy:fuzzyset>
</fuzzy:concept>
<fuzzy:concept conceptid="c03">
    <fuzzy:name>Verwendung von i nach Datenbankanalyse</fuzzy:name>
    <fuzzy:fuzzyset>
        <fuzzy:level>10</fuzzy:level>
        <fuzzy:value>eingabe</fuzzy:value>
```

```
        <fuzzy:level>20</fuzzy:level>
        <fuzzy:value>steuerung</fuzzy:value>
        <fuzzy:level>50</fuzzy:level>
        <fuzzy:value>ausgabe</fuzzy:value>
        <fuzzy:level>75</fuzzy:level>
        <fuzzy:value>berechnung</fuzzy:value>
        <fuzzy:level>100</fuzzy:level>
        <fuzzy:value>index</fuzzy:value>
    </fuzzy:fuzzyset>
</fuzzy:concept>
<fuzzy:concept conceptid="c04">
    <fuzzy:name>Verwendung von i nach Transformation</fuzzy:name>
    <fuzzy:fuzzyset>
        <fuzzy:level>10</fuzzy:level>
        <fuzzy:value>berechnung</fuzzy:value>
        <fuzzy:level>50</fuzzy:level>
        <fuzzy:value>ausgabe</fuzzy:value>
        <fuzzy:value>eingabe</fuzzy:value>
        <fuzzy:level>90</fuzzy:level>
        <fuzzy:value>index</fuzzy:value>
    </fuzzy:fuzzyset>
</fuzzy:concept>
```

In diesem Beispiel haben wir zur Veranschaulichung der Referenzstrukturen noch eine Annotation hinzugefügt, die `typ` und `verwendung` zusammen annotiert und eine String-Facette mit einem Kommentar des Reverse Engineers enthält. Hier wurden nur die Fuzzy-Interpretationen der atomaren Konzepte übertragen, nicht die Interpretation der komplexen Fuzzy-Beschreibungen, da diese sich leicht aus den atomaren Fuzzy-Mengen berechnen lassen. Grundsätzlich erlaubt die DTD aber auch die zusätzliche Übertragung der materialisierten Fuzzy-Interpretation.

Mit diesem Teil ist das Architekturmodell für Fuzzy-Informationssysteme komplett. Im nächsten und letzten Teil widmen wir uns nun dem praktischen Einsatz des Modells anhand zweier Fallstudien (siehe Abbildung 16.7 auf der nächsten Seite).

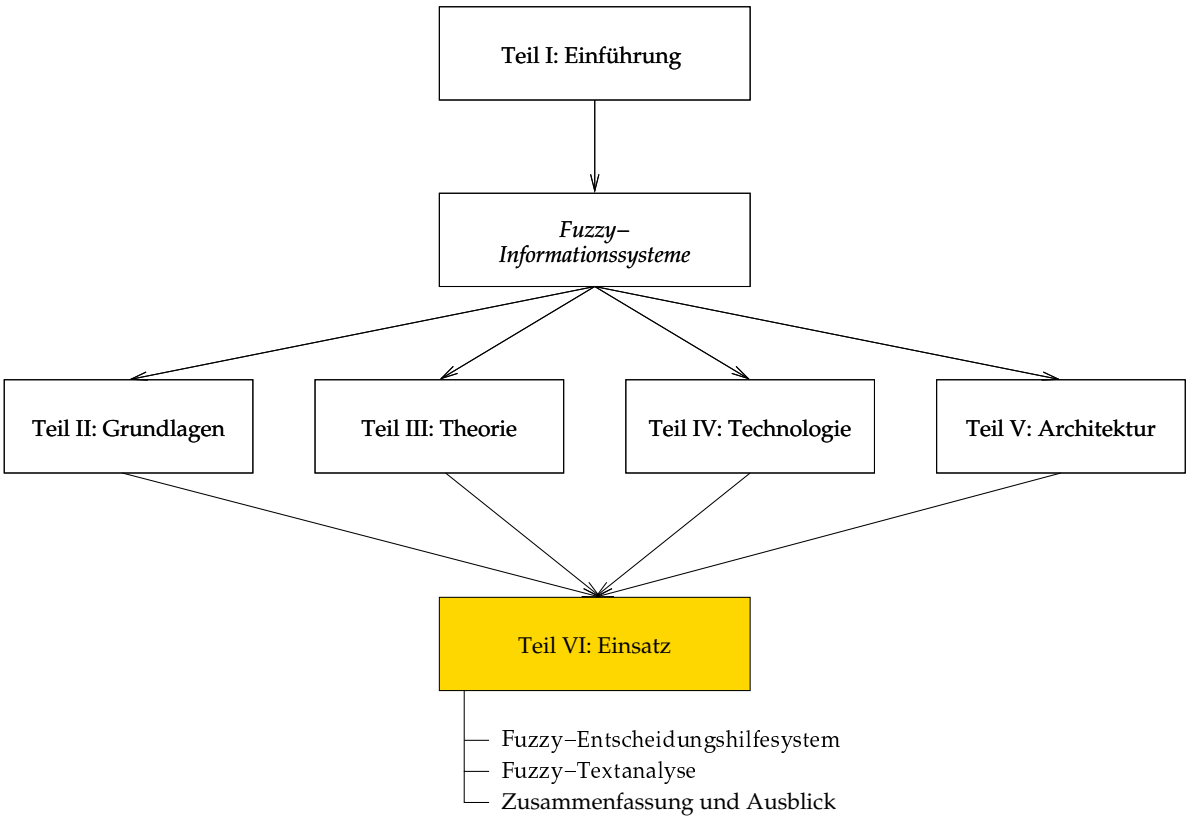


Abbildung 16.7: Gliederung Teil VI

Teil VI

Fuzzy-Informationssysteme im Einsatz

Kapitel 17

Fuzzy-Entscheidungshilfesystem

Dort gewinne ich eine Giraffe
eine große lange, aber schlaffe
weil der Fernseher stört, ist die Giraffe sauer
so ist auch diese Freude nur von kurzer Dauer.
Les Reines Prochaines, „Schöner Sonntag“

In diesem Teil der Arbeit demonstrieren wir, daß sich die entwickelten Konzepte wie geplant zur Realisierung von Fuzzy-Informationssystemen eignen. Dabei gehen wir für zwei Beispielsysteme den gesamten Entwicklungsprozeß durch.

Wir beginnen in diesem Kapitel mit der Entwicklung eines Systems zur Unterstützung von Auswahl- und Entscheidungsprozessen, eines sogenannten *Entscheidungshilfesystems* (Decision Support System). Als konkretes Anwendungsbeispiel modellieren wir die Auswahl eines Fahrzeuges, nämlich eines Autos, anhand vager Kriterien. Das System kann jedoch in gleicher Form für andere Entscheidungsszenarien eingesetzt werden.

Hierbei liegt der Schwerpunkt auf dem Aspekt, wie man die in dieser Arbeit entwickelten Konzepte einsetzt, um ein Fuzzy-Informationssystem von Grund auf zu entwickeln. Im nächsten Kapitel demonstrieren wir dann die Erweiterung eines existierenden scharfen Systems um Fuzzy-Technologien.

17.1 Szenario

Das Problem, zu einer Menge gegebener Anforderungen eine passende Lösung zu finden, die alle oder zumindest möglichst viele der Anforderungen erfüllt, hat eine lange Tradition in der Informatik. Üblicherweise sind Anforderungen dabei als schar-

fe Kriterien definiert. In vielen Fällen ist es jedoch für einen Anwender einfacher, die Anforderungen als vage Bedingungen zu formulieren.

Überdies ist es meist nicht realistisch, alle Anforderungen einhundertprozentig erfüllen zu wollen. Viel interessanter ist daher in der Praxis die Frage, ob sich eine Lösung finden läßt, die möglichst viele der Anforderungen zu einem möglichst hohen Grad befriedigt. Bespielsweise wird ein Anwender, der fünf Anforderungen stellt, die aber nicht alle gleichzeitig erfüllt werden können, eine Lösung bevorzugen, bei der vier Anforderungen zu 90% erfüllt werden, als mit einer hundertprozentigen Lösung zu leben, die aber nur zwei der Anforderungen erfüllt.

In diesem Beispiel betrachten wir als konkretes Einsatzszenario die Auswahl eines Fahrzeuges, für dessen Auswahl eine Reihe von vagen Anforderungen erfüllt werden sollen. Dieses Entscheidungshilfesystem soll in der Lage sein, aus der Menge der im System gespeicherten Automodelle aufgrund der Anforderungen des Anwenders diejenigen auszuwählen, die alle Anforderungen zu mindestens einem bestimmten vorgegebenen Grad erfüllen. Dabei sollen widersprüchliche Anforderungen dem Anwender aufgezeigt und gegebenenfalls durch Entfernen einzelner Bedingungen aufgelöst werden.

Wir bestimmen zunächst im nächsten Abschnitt die Architektur des Gesamtsystems, bevor wir uns dem Entwurf und der Realisierung der einzelnen Stufen zuwenden.

17.2 Architektur

Zur Bestimmung der Anwendungsarchitektur verwenden wir das in Abschnitt 14.3.5 auf Seite 216 vorgestellte Schema.

Für diese Anwendung soll ein autonomer Klient zur Visualisierung der im System befindlichen Fuzzy-Mengen zum Einsatz kommen. Entsprechend benötigen wir in der Präsentationsstufe neben der Fuzzy-Bibliothek die in Kapitel 15 vorgestellten Fuzzy-Komponenten. Da kein zusätzlicher Web-Klient gefordert wird, entfällt die zweite Stufe in der Anwendungsarchitektur.

Die Anwendungslogik, zur Verarbeitung der einzelnen Anforderungen, steht auch für diese Anwendung in der 3. Stufe. Da der Benutzer direkt mit vagen Begriffen interagieren soll, benötigen wir hier zusätzlich den Lexikondienst zum Abruf dieser Begriffe. Eine persistente Speicherung ist dagegen nicht gefragt, da die Ergebnisse der Benutzerinteraktionen nur für die Dauer einer Anwendungssitzung vorliegen müssen.

In der Ressourcen-Stufe schließlich muß aufgrund der Anforderungen das Fuzzy-Lexikon zur Verfügung stehen. Ein Datenaustausch mit anderen Systemen ist nicht gefordert. Die resultierende Architektur zeigt Abbildung 17.1 auf der nächsten Seite.

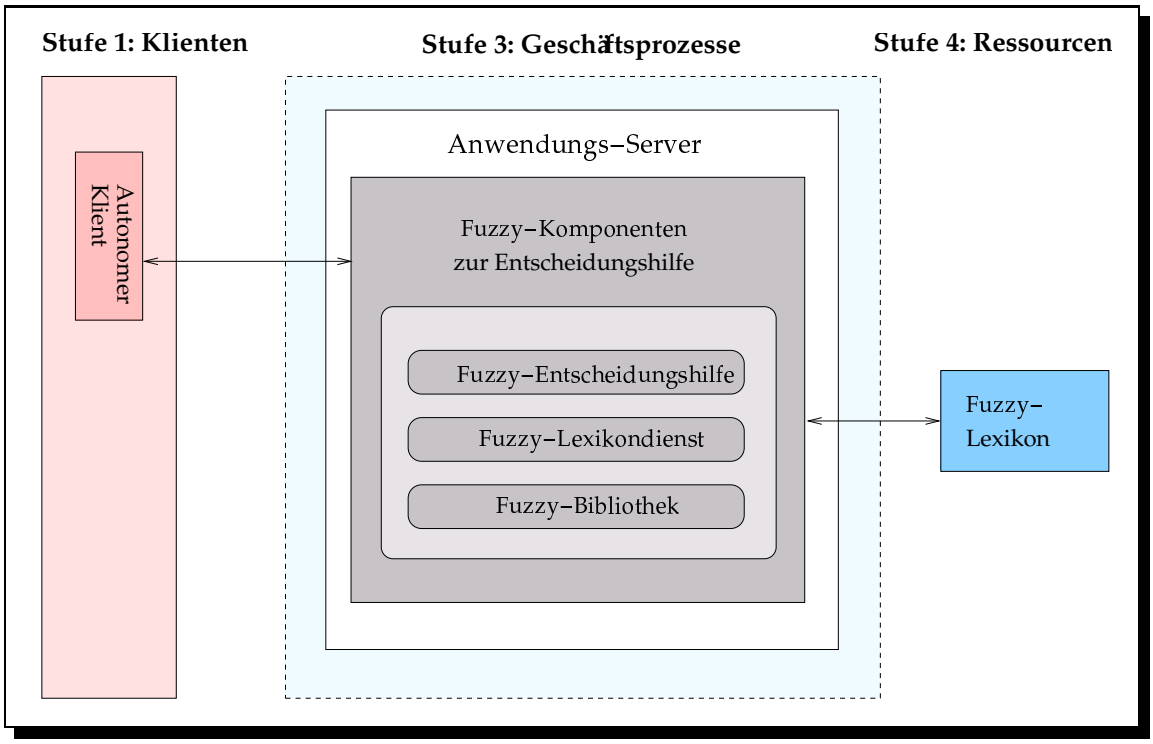


Abbildung 17.1: Architektur des Fuzzy-Entscheidungshilfesystems

17.3 Geschäftsprozeß-Stufe

Die Geschäftsprozeß-Stufe beherbergt den Kern des Entscheidungshilfesystems: sein Modell und die Anwendungslogik in Form des Entscheidungsalgorithmus.

Das System soll einen Anwender helfen, ein Auto passend zu einer Reihe vager Anforderungen zu finden. Modelliert werden muß also ein solches `Automobil` zusammen mit den Attributen, auf denen vage Anforderungen definiert werden sollen. Für dieses Beispiel erlauben wir Anforderungen auf der `Bauart` *(„familientauglich“?)*, dem `Motor` *(„fährt mit Biodiesel“?)*, dem `Antrieb` *(„geländetauglich“?)* sowie einigen weiteren Attributen, die zusammen in Abbildung 17.2 auf der nächsten Seite gezeigt sind. Dabei nimmt das Attribut `Modell` die konkrete Modellbezeichnung eines Fahrzeuges auf, was dem gesuchten Ergebnis entspricht.

Es verbleibt die Definition eines geeigneten Algorithmus zur Auswahl eines Ergebnisses, das mit den Anforderungen konsistent ist. Der Anwender muß die Möglichkeit haben, eine neue Anforderung hinzuzufügen. Ist diese konsistent zu den existierenden Anforderungen innerhalb eines eingestellten Grades γ, soll sie in die Anforderungsmenge aufgenommen und ein mögliches (scharfes) Ergebnis berechnet werden. Ergibt sich dagegen ein Konflikt mit den bestehenden Anforderungen, soll dies dem Anwender angezeigt werden. Er muß dann die Möglichkeit haben, die neue Bedin-

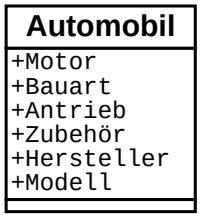


Abbildung 17.2: Modellierung des Automobils

gung zu *erzwingen*, indem die mit ihr in Konflikt stehenden Anforderungen entfernt werden.

Genau dies leisten die in Kapitel 10 definierten konsistenzerhaltenden Operationen γ-Expansion und γ-Revision. Die Menge der Anforderungen wird dabei über eine Fuzzy-Formel repräsentiert. Nach Einstellung eines geforderten Konsistenzgrades γ kann ein Anwender eine neue Anforderung über eine γ-Expansion einbringen. Ist diese mit den vorhandenen Anforderungen konsistent, wird die Fuzzy-Formel des ausgewählten Attributes um die neue Anforderung erweitert, seine Fuzzy-Interpretation wird sich also entsprechend der neuen Bedingung einschränken. Um den Wert für das zugehörige scharfe Attribut zu berechnen, wird danach eine Defuzzyfizierung mit der Maximum-Methode (vergleiche Abschnitt 6.3.1 auf Seite 58) angestoßen. Ist die neue Anforderung aber inkonsistent mit den vorhandenen Anforderungen, wird sie die Expansionsoperation zurückweisen. Um die neue Anforderung dennoch aufzunehmen, muß dieser Konflikt aufgelöst werden, indem eine oder mehrere der existierenden Anforderungen entfernt werden. Genau dies wird durch eine γ-Revision erreicht.

Damit läßt sich der Entscheidungshilfealgorithmus auf die Ausführung dieser beiden Operationen zurückführen. Benötigt wird jetzt noch eine epistemische Relevanzordnung für die einzelnen Anforderungen (vergleiche Definition 10.2.9 auf Seite 119). Für diese Anwendung eignet sich eine einfache Ordnung nach dem Einfügezeitpunkt. Bei einem Konflikt genießen also die zeitlich jüngeren Anforderungen Priorität, dies spiegelt gut ein exploratives Verhalten des Anwenders wider.

Damit bei einer Expansion oder Revision auf den Anforderungen eines Attributes wie `Motor` oder `Bauart` auch das zugehörige `Modell` geändert wird, müssen die Graph-Varianten dieser Operationen eingesetzt und eine Abhängigkeitsbeziehung zwischen diesen Attributen und dem Automodell hergestellt werden. Damit setzen sich Anforderungsänderungen automatisch auf das `Modell` fort. Auf diese Weise kann fortlaufend kontrolliert werden, welches Fahrzeug aufgrund der aktuellen Anforderungen in Frage kommt.

17.4 Klienten-Stufe

Der Anwender soll mit dem System über einen autonomen Klienten agieren können, der mit einer graphischen Oberfläche ausgestattet ist.

Nach einer Auswahl des zu bearbeitenden Attributes sollen die möglichen vagen Anforderungen zusammen mit den bereits existierenden Bedingungen angezeigt werden. Zusätzlich soll der durch Defuzzifizierung ermittelte mögliche scharfe Wert angegeben werden. Der Benutzer soll dann die Möglichkeit haben, eine neue Anforderung auszuwählen und durch Expansion oder Revision hinzuzufügen.

Zusätzlich soll es das System erlauben, den aktuellen Zustand der Attribute mit ihren Annotationen und Facetten anzuzeigen; dazu gehören sowohl die einzelnen Fuzzy-Formeln mit ihrer syntaktischen Struktur als auch deren zugehörige Interpretation als Fuzzy-Menge.

17.5 Ressourcen-Stufe

In der Ressourcen-Stufe wird nach der obigen Architekturbestimmung ein Fuzzy-Lexikon benötigt, das die anwendungsspezifischen vagen Begriffe festhält, damit diese dann als Anforderungen auf den einzelnen Attributen eingesetzt werden können.

Zusätzlich müssen für die oben beschriebenen Abhängigkeiten die einzelnen Transformationsfunktionen definiert werden, damit die für Graph-Operationen benötigte automatische Umrechnung von Fuzzy-Mengen durchgeführt werden kann.

17.6 Implementierung

Wir beginnen mit der Beschreibung der Implementierung des Geschäftsprozeß-Servers, der das `Automobil` mit seinen Operationen verwaltet. Die Implementierung erweist sich als sehr einfach, da praktisch alle beschriebenen Funktionalitäten bereits durch die entwickelten Komponenten abgedeckt sind. Es muß lediglich die in Abbildung 17.2 auf der vorherigen Seite gezeigte Klasse implementiert werden. Seine scharfen Attribute müssen jetzt noch um die Möglichkeit erweitert werden, imperfekte Zustände in Form von Fuzzy-Formeln anzunehmen. In unserem Modell wird dies über das Annotationskonzept erreicht, das solche Fuzzy-Informationen in speziellen Facetten ablegt (vergleiche Abschnitt 12.2 auf Seite 173). Abbildung 17.3 auf der nächsten Seite zeigt die fertige Implementierung. Man sieht die modellierten Attribute, auf denen die Lösungssuche ablaufen soll. Durch die Vererbungsbeziehung mit der Klasse `AnnotatableObject` erhält die Klasse `Automobil` die Fähigkeit, mit Annotationen und Fuzzy-Facetten ausgestattet zu werden. Die benötigten Operationen Expansion und Revision schließlich sind bereits in den zugehörigen Klassen der

```
import fuzzy.aobject.*;
import fuzzy.acomp.*;

public class Automobil extends AnnotatableObject {

        public String Motor;
        public String Bauart;
        public String Antrieb;
        public String Zubehoer;
        public String Hersteller;
        public String Modell;

        public Automobil() {
                super();
        }
}
```

Abbildung 17.3: Server-Quellcode des Fuzzy-Entscheidungshilfesystems

Fuzzy-Bibliothek realisiert (vergleiche Abschnitt 13.2 auf Seite 187) und stehen somit nach dem Anlegen einer Fuzzy-Facette automatisch zur Verfügung.

Als nächstes betrachten wir die Realisierung des Klienten mit seiner graphischen Oberfläche. Diese soll in der Lage sein, die einzelnen Attribute inklusive der Fuzzy-Informationen anzuzeigen und Anforderungen zu verwalten. Hierfür setzen wir die in Abschnitt 15.1 (Seite 219) vorgestellten Oberflächenkomponenten ein. Für die Anzeige der einzelnen Attribute mit ihren Fuzzy-Formeln und deren Interpretation wird eine kombinierte Sicht aus der Annotations-Präsentationskomponente und der 2D-Darstellung von Fuzzy-Mengen verwendet (vergleiche Abschnitte 15.1.2 auf Seite 221 und 15.1.3 auf Seite 222). Den wesentlichen Teil des Klientenprogramms (abzüglich einiger Initialisierungen) zeigt Abbildung 17.4 auf der nächsten Seite. Die Implementierung des Klienten beschränkt sich also darauf, ein Objekt der Klasse `Automobil` zu instanziieren und eine geeignete Sicht dafür zu suchen, was durch den Einsatz des Sichtenverwalters (siehe Abschnitt 15.1.1 auf Seite 220) erledigt wird. Die gefundene Sicht wird dann geöffnet und kann von einen Anwender zur Interaktion verwendet werden.

Damit das System einsetzbar wird, muß schließlich noch, wie oben skizziert, ein passendes Fuzzy-Lexikon mit den vagen Begriffen und deren Abhängigkeiten aufgebaut werden. Für dieses System wurden dazu mit Hilfe des Lexikon-Werkzeuges (siehe Abschnitt 16.1.3 auf Seite 232) einige Beispielbegriffe definiert.

```
import Views.View;
import System.ViewManager;
import java.lang.reflect.*;

...

public class autoDemo {

  public Automobil auto;

  class internalWindowAdapter extends WindowAdapter {
        public void windowClosed(WindowEvent e) {
               System.exit(0);
        }
  }

  public static void main(String args[]) {
        autoDemo mobil = new autoDemo();

        ViewManager vm = ViewManager.getManager();
        View v = vm.findView(mobil);

        if(v instanceof View) {
          v.addListener(mobil.new internalWindowAdapter());
          v.setValue(mobil);
          v.showView();
        }
  }
}
```

Abbildung 17.4: Klienten-Quellcode des Fuzzy-Entscheidungshilfesystems

17.7 Beispiellauf

Wir illustrieren die Funktionsweise des Systems anhand einer Beispielsitzung.

Hier stellt der Benutzer zuerst die Anforderung, daß der Motor *leistungsstark* sein soll (siehe Abbildung 17.5 auf der nächsten Seite). Die für die ausgewählte Komponente passenden vagen Begriffe sind dabei in der unteren Box aufgeführt. Nachdem ein Begriff selektiert wurde, kann der Anwender versuchen, ihn mit einer der angebotenen Operationen aufzunehmen; im Erfolgsfall erscheint der Begriff in der mittleren Box. Da dies die erste Anforderung ist, kann sie problemlos aufgenommen werden.

In dieser Abbildung sieht man in der oberen Zeile überdies den aktuellen scharfen Wert, der sich nach der Aufnahme der Anforderung durch Defuzzifizierung nach der Maximum-Methode (vergleiche Abschnitt 6.3 auf Seite 57) ergibt.

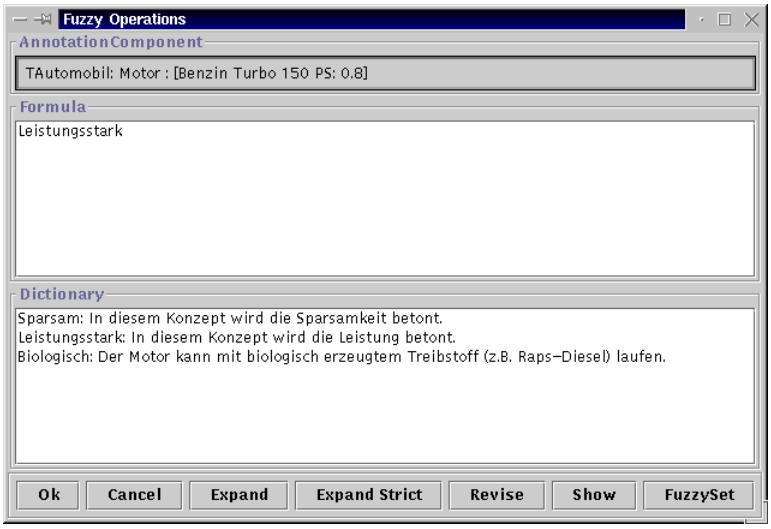


Abbildung 17.5: Aufnahme der Anforderung „leistungsstark“

Für diese Anforderung stehen mehrere Fahrzeugmodelle zur Verfügung, wie Abbildung 17.6 zeigt.

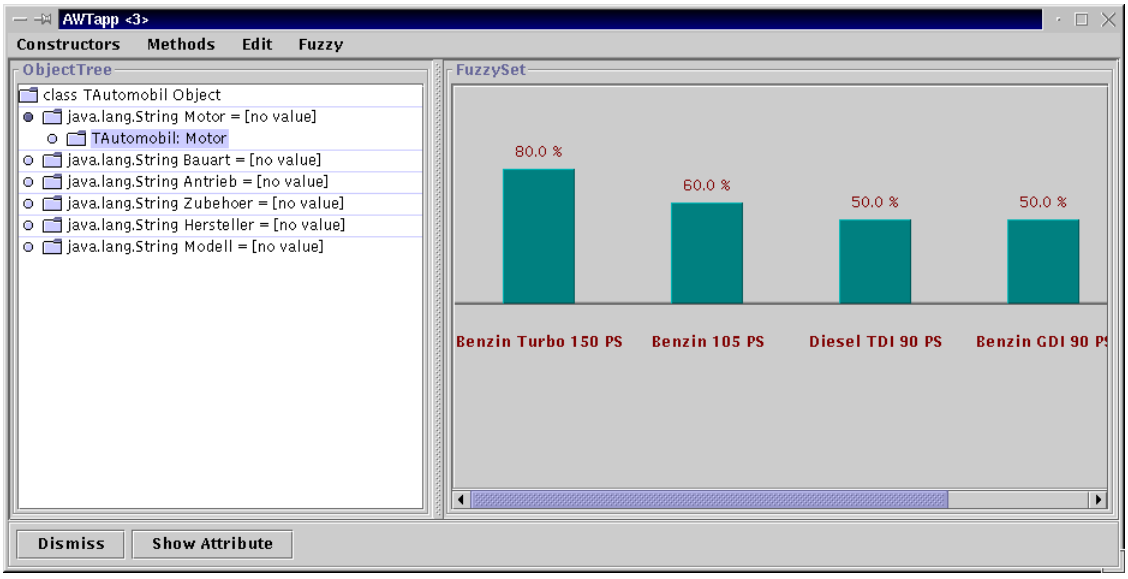


Abbildung 17.6: Ergebnis-Motoren nach Aufnahme der Anforderung „leistungsstark“

Als nächstes wünscht sich der Benutzer eine Bauart, mit der er seinen Hund transportieren kann (siehe Abbildung 17.7).

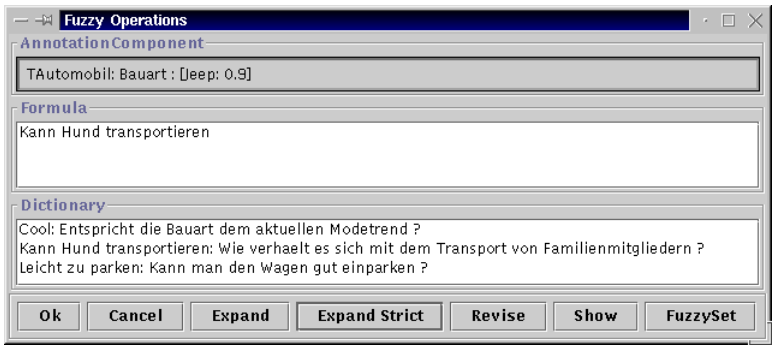


Abbildung 17.7: Aufnahme der Anforderung „kann Hund transportieren“

Auch diese Anforderung läßt sich problemlos aufnehmen und führt zu der in Abbildung 17.8 gezeigten Ergebnismenge möglicher Bauarten. Man kann erkennen, daß vor allem große Fahrzeuge zum Hundetransport geeignet sind, während ein Cabrio als eher unpraktisch eingestuft wird.

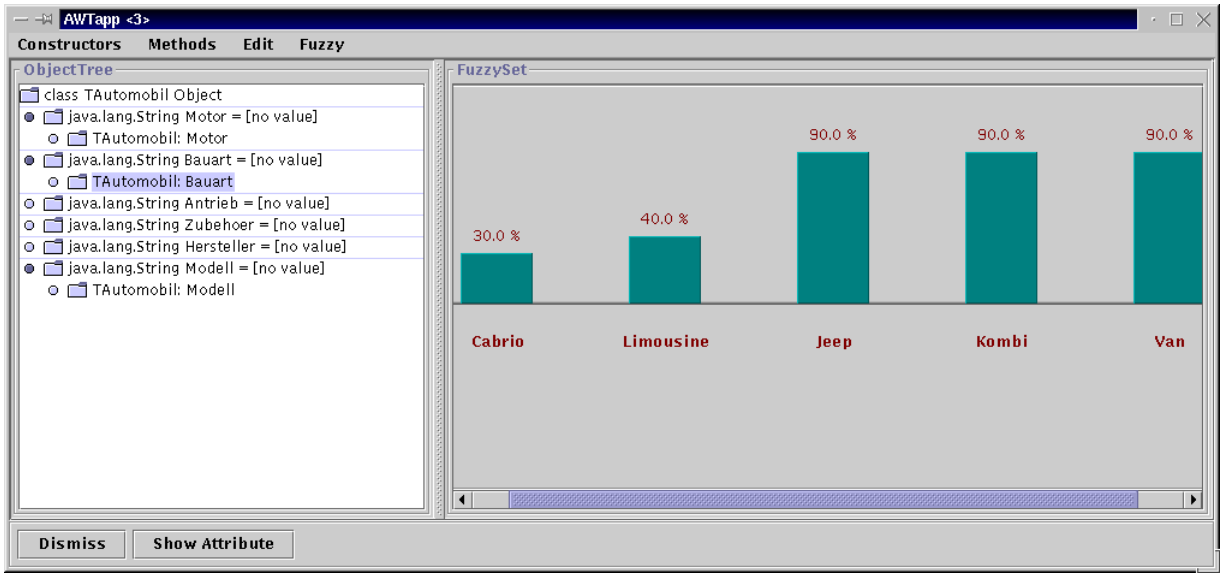


Abbildung 17.8: Ergebnis-Bauarten, die Hunde transportieren können

Beide Anforderungen werden durch die Graph-Operationen auf das Attribut `Modell` fortgesetzt, wie oben beschrieben. Das führt zu der in Abbildung 17.9 (Seite 258) gezeigten Ergebnismenge der Automodelle. Man sieht, daß das System einen Kombi „VW Passat“ als besonders geeignet empfiehlt.

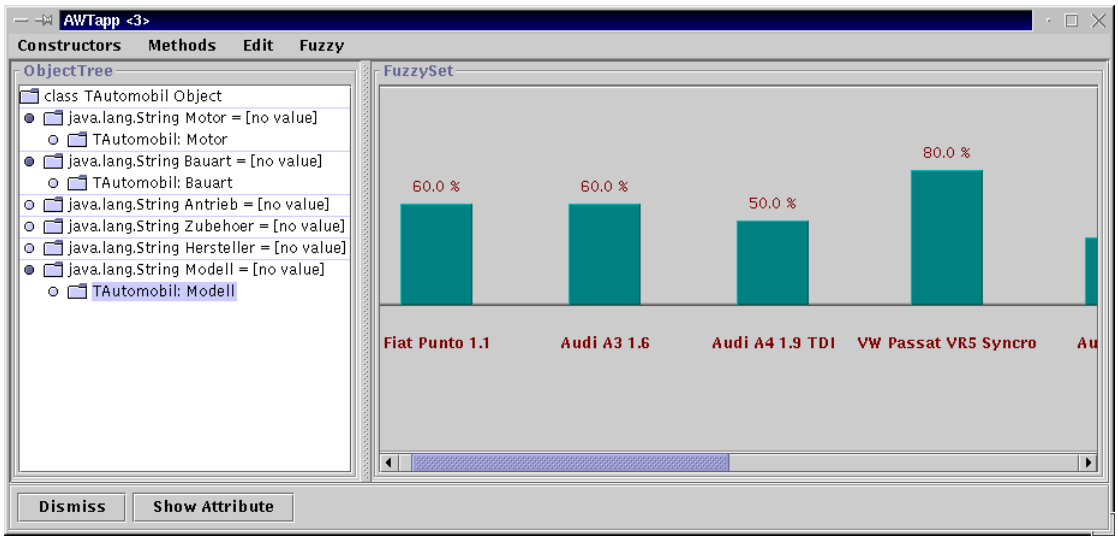


Abbildung 17.9: Leistungsstarke Automodelle, die Hunde transportieren können

Versucht der Benutzer jetzt allerdings, die Anforderung „fährt mit Biotreibstoff“ aufzunehmen, führt dies zu einem Konflikt, wie in Abbildung 17.10 (Seite 258) zu sehen ist (denn es gibt nur wenige solche Motoren, die alle nur zu einem sehr geringen Grad „leistungsstark“ sind).

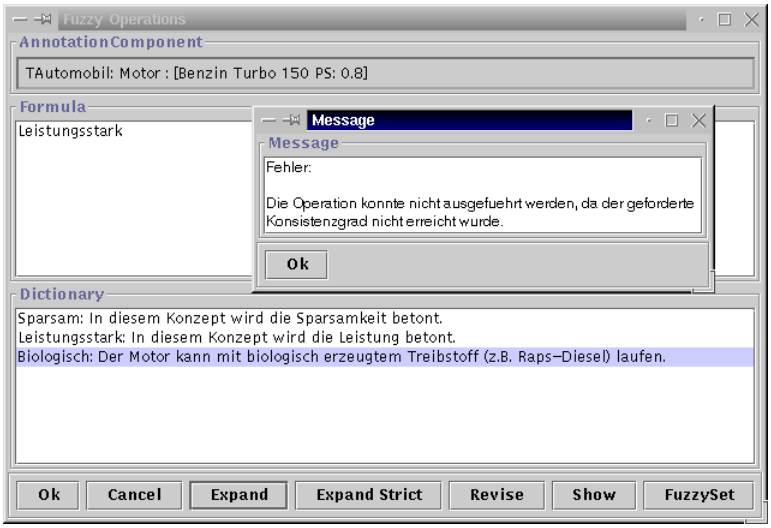


Abbildung 17.10: Aufnahme der zusätzlichen Anforderung „biologisch“ (fährt mit Biotreibstoff) führt zum Konflikt

Besteht der Benutzer auf dieser Anforderung, kann er eine Revision anstoßen. Wie in

Abbildung 17.11 (mittlere Box) zu sehen ist, muß dafür aber die vorhandene Anforderung „leistungsstark“ entfernt werden (vergleiche mit Abbildung 17.5 auf Seite 256).

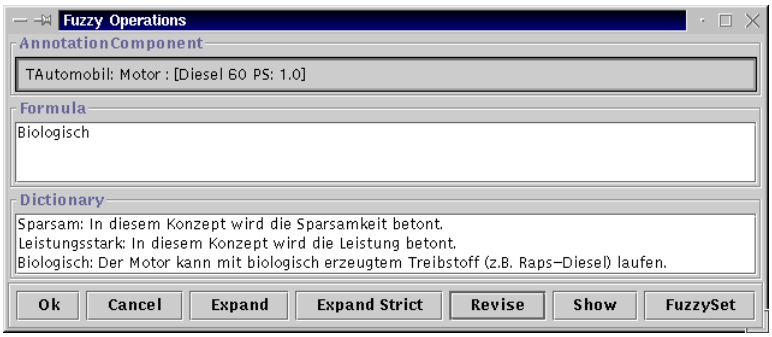


Abbildung 17.11: Aufnahme der Anforderung „biologisch“ durch Revision, Anforderung „leistungsstark“ wurde entfernt

Als Ergebnis kommen dadurch nur noch wenige Fahrzeuge in Frage, wie man in Abbildung 17.12 sehen kann: für diese Anforderungsmenge wird jetzt der (fiktive) „Jeep Wrangler“ empfohlen.

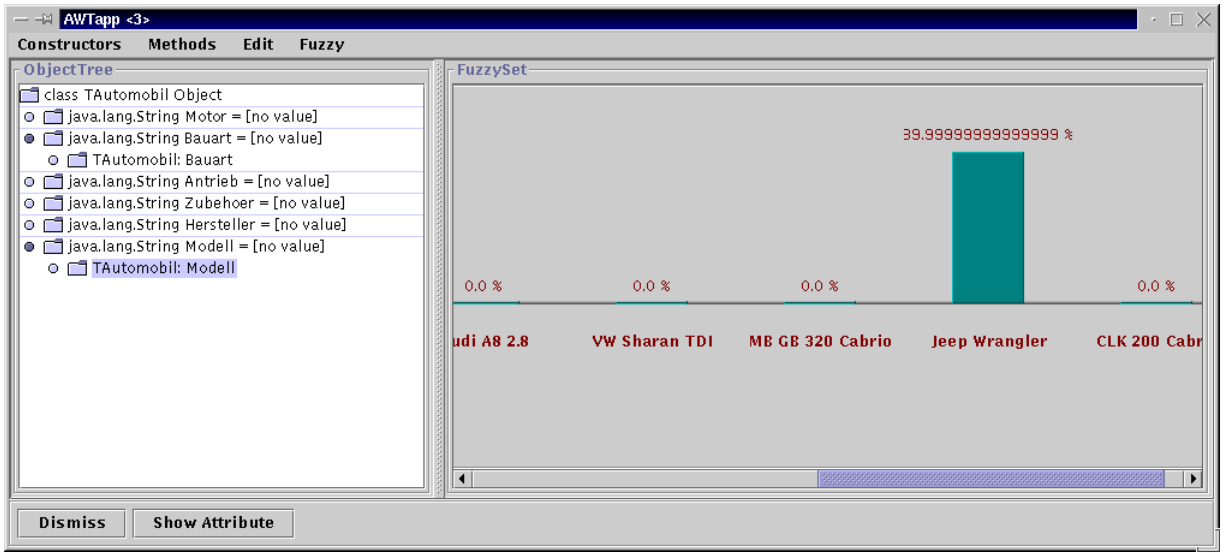


Abbildung 17.12: Revidierte Ergebnis-Modelle, die Hunde transportieren können und mit Biotreibstoff fahren

17.8 Fazit

Das Beispiel ist bewußt einfach gehalten, um die wesentlichen Aspekte beim Einsatz der Fuzzy-Technologien herauszuarbeiten. Es läßt sich aber schon erkennen, daß der Aufwand für die Realisierung eines solchen Fuzzy-Systems im Vergleich zu einer klassischen Lösung erheblich vermindert ist, da durchgängig auf ein existierendes Modell mit zugehörigen Operatoren zurückgegriffen werden kann, dieses also nicht im Rahmen der Anwendungsentwicklung mit realisiert werden muß.

Ein Anwendungsentwickler kann überdies mit vertrauten Konzepten arbeiten und benötigt keine besonderen Kenntnisse von Fuzzy-Mengen und deren Realisierung, um ein solches System erstellen zu können. Lediglich für die Definition des Fuzzy-Lexikons entsteht zusätzlicher Aufwand. Dieses ist jedoch klar von der Anwendungslogik getrennt und kann so potentiell von anderen Systemen wiederverwendet werden.

Für den praktischen Einsatz wird man einige Verfeinerungen vornehmen; so etwa sollten stark subjektive Begriffe wie „preiswert“ durch einen Anwender parametrisierbar sein. Dennoch zeigt das Beispiel anschaulich unsere Vorstellung von einem Fuzzy-Informationssystem, wie es beispielsweise als Web-Anwendung im Internet eingesetzt werden könnte. Dort ist bereits eine Vielzahl von Systemen im Einsatz, etwa im Bereich „E-Commerce“, die einem Benutzer die Auswahl eines Produktes erleichtern sollen, dabei aber nur mit — oft wenig hilfreichen — scharfen Kriterien arbeiten. Typisches Beispiel ist die Auswahl einer Digitalkamera: auch bei diesem Szenario ist es viel benutzerfreundlicher, wenn ein Anwender mit vagen Begriffen operieren kann, denn ihn interessieren weniger genaue Pixelzahlen (für die Auflösung), Milliamperestunden (für den Akku) und Millimeter (für die Abmessungen) , sondern vielmehr vage Kriterien wie „große Abzüge möglich“, „lange Laufzeit“ und „paßt in die Hosentasche“.

Kapitel 18

Fuzzy-Textanalyse

Rose is a rose is a rose is a rose, is a rose
Gertrude Stein

In diesem Kapitel zeigen wir ein weiteres Einsatzbeispiel für das hier entwickelte Modell. Der Anwendungsfall stammt aus dem Bereich der Computerlinguistik und behandelt das Problem der sogenannten *Resolution von Nominalkoreferenz.*

Ein wesentlicher Unterschied zum letzten Beispiel liegt darin, daß hier bereits ein fertiges, auf scharfer Basis arbeitendes System vorlag. Ein neuer Aspekt ist daher die Erweiterung eines existierenden klassischen Systems um Fuzzy-Technologien.

Darüber hinaus zeigt dieses Einsatzbeispiel eine grundsätzlich andere Anwendungsmöglichkeit unseres Fuzzy-Modells: während im Entscheidungshilfesystem ein Benutzer imperfekte Begriffe zur direkten Systeminteraktion verwendet, werden hier komplexe Fuzzy-Ausdrücke vorwiegend für interne Berechnungen verwendet und dafür automatisch erzeugt.

Im folgenden Abschnitt liefern wir zunächst eine kurze Einführung in das Szenario und das existierende Analysesystem ERS.

18.1 Einführung

Mit der zunehmenden Online-Verfügbarkeit von Zeitungen, Büchern, Aufsätzen und sonstigen Informationen wird die Frage nach sinnvollen Recherche- und Analyseverfahren immer dringender. Stand der Technik ist heute immer noch die einfache Schlagwortsuche, die in mehr oder weniger verfeinerter Form in Suchmaschinen Verwendung findet. Inhaltliche Aspekte spielen dabei keine Rolle, entsprechend schlecht sind die so erzielbaren Ergebnisse, da meist viele unrelevante Treffer erzielt werden,

die mühsam von Hand ausgefiltert werden müssen. Um hier substantielle Verbesserungen zu erreichen, benötigt man Metainformationen über den Aufbau und den Inhalt eines Textes. Da eine manuelle Erstellung solcher Metainformationen nur für eine verschwindend kleine Menge der verfügbaren Inhalte möglich ist, müssen automatisierte Verfahren entwickelt werden.

Die maschinelle Textanalyse ist daher ein zentrales Gebiet der Computerlinguistik. Aus pragmatischen Gründen gewinnt dabei die sogenannte *partielle Textanalyse* an Interesse. Man versucht hierbei nicht mehr, einen Text in seiner Gesamtheit zu analysieren oder zu verstehen, sondern konzentriert sich auf Teilanalysen, die automatisch durchführbar sind. Dazu gehört beispielsweise die Erstellung von Textprofilen, die Informationen über die Struktur eines Textes enthalten, welche dann bei einer Suche oder zur Weiterverarbeitung herangezogen werden [Ber95].

Solche Profile können Informationen über beteiligte Objekte *(Firmen, Personen)*, Aktivitäten *(Aktienübernahmen, Erfindungen)* oder Argumentationsstrukturen *(In welcher Beziehung stehen Objekte zueinander? Unterstützen sie sich argumentativ?)* enthalten. Eine Recherche kann unter Verwendung solcher Textprofile wesentlich genauere Ergebnisse liefern als eine einfache Schlagwortsuche. So kann beispielsweise eine Suche nach Firmenübernahmen, an denen bestimmte Personen beteiligt sind, auch Artikel finden, in denen das Wort „Übernahme“ selbst nicht vorkommt, aber zum Beispiel durch den „Kauf von 60% der Aktienanteile“ impliziert wird.

In diesem Szenario betrachten wir nun einen wichtigen Teilschritt der automatischen Textanalyse: die sogenannte Resolution von Nominalkoreferenz.

18.1.1 Resolution von Nominalkoreferenz

Die Aufgabe bei der Nominalkoreferenz-Resolution *(Noun Phrase Coreference Resolution)* ist, die in einem Text vorkommenden Objekte zu identifizieren. Grammatikalisch zeichnen sich diese durch sogenannte Nominalphrasen *(noun phrases, NP)* aus. Bei der Resolution der Nominalkoreferenz wird nun festgestellt, welche Objekte in einem Text auftauchen und mit welchen textuellen Bezeichnern jedes Objekt referenziert wird. Diese sind nicht eindeutig, da ein Objekt, wie eine Firma, innerhalb eines Textes meist unterschiedlich benannt wird: angefangen von der Nennung des kompletten Firmennamens inklusive Rechtsform und Sitz bis hin zum einfachen Bezug auf „die Firma“. Deshalb ist es notwendig, diese unterschiedlichen Referenzen aufzulösen, damit beispielsweise bei einer Suche alle relevanten Textstellen gefunden werden können, selbst wenn die dort verwendete Beschreibung unterschiedlich vom Suchstring ist.

Der in Abbildung 18.1 auf der nächsten Seite gezeigte Beispieltext aus dem *Wall Street Journal* macht das Problem anschaulich. In diesem Artikel koreferieren unter anderem die folgenden Nominalphrasen:

- *its vice president for manufacturing*

> Telxon Corp. said its vice president for manufacturing resigned and its Houston work force has been trimmed by 40 people, or about 15%. The maker of hand-held computers and computer systems said the personnel changes were needed to improve the efficiency of its manufacturing operation. The company said it hasn't named a successor to Ronald Bufton, the vice president who resigned. Its Houston work force now totals 230.

Abbildung 18.1: Beispielartikel aus dem *Wall Street Journal*

- *Ronald Bufton*
- *the vice president who resigned*

Während einem menschlichen Leser sofort klar ist, worauf sich *„its“* im ersten Satz bezieht (nämlich *Telxon Corp.*), entgeht dieser Zusammenhang bei einer maschinellen Bearbeitung zunächst völlig. Als Konsequenz gehen wichtige Informationen über ein Objekt (wie hier die „Telxon Corporation“) verloren, die dann beispielsweise eine Schlagwortsuche nicht mehr finden kann. Komplexere Anfragen *(„Welche Firma hat im vergangenen Jahr mehr als ein Drittel ihrer Mitarbeiter entlassen?“)* lassen sich so nicht mehr erfüllen — wichtig beispielsweise für Wirtschaftsanalytiker, die aus großen Artikelbeständen signifikante Trends ableiten wollen.

Die Bestimmung dieser Koreferenzen, die sogenannte Koreferenzbestimmung *(coreference resolution)*, ist daher ein wichtiger, zentraler Schritt in Sprachverarbeitungssystemen, auf dem dann höherwertige Funktionen aufbauen. An obigem Beispiel wird schon deutlich, welche Möglichkeiten dies für die Recherche über einen oder viele Texte hat; gezielte Angaben auf Fragen wie *„Welche Tätigkeiten übte Ronald Bufton aus?“*, *„Welche Vizepräsidenten hatte Telxon Corp.?“* oder *„Für welche Firmen arbeitete Ronald Bufton?“* werden so erst möglich.[1]

Viele weitere Analyseschritte, wie die Erstellung von Textprofilen, die Informationsextraktion oder eine automatische Textzusammenfassung, bauen auf den gefundenen Koreferenzen auf. Eine möglichst genaue, effiziente und vollständige Erfassung dieser Koreferenzen ist daher von großem Interesse in der Sprachverarbeitung und im Information Retrieval. Denn selbst kleine Verbesserungen an dieser Stelle wirken sich grundlegend auf die Qualität der Ergebnisse übergeordneter Verarbeitungsverfahren aus.

[1] Man beachte, daß ein *Textverständnis* damit noch nicht gegeben ist — aus dem obigen Beispieltext folgt zum Beispiel, daß „Ronald Bufton“ jetzt gerade **nicht** mehr Vizepräsident von „Telxon Corp.“ ist. Dies stellt eine Herausforderung für die nachfolgenden Verarbeitungsschritte dar.

18.2 Das AETNA-Projekt und ERS

Das vorliegende Anwendungsbeispiel entstand in freundlicher Zusammenarbeit mit dem von Prof. Dr. Bergler[2] geleiteten AETNA Projekt *(Analysis of English Texts from Newspaper Articles)*.

Als Textkorpus werden in diesem Projekt hauptsächlich Artikel aus dem *Wall Street Journal* (WSJ) herangezogen, das auch zentraler Untersuchungsgegenstand in vielen anderen Projekten und Konferenzen ist. Dies hat zwei wesentliche Gründe: erstens ist diese Zeitung für ihren kreativen Gebrauch der englischen Sprache bekannt und bietet daher viele Herausforderungen bei der Systementwicklung und -evaluierung; und zweitens besteht ein großes, auch kommerzielles Interesse an einer automatischen Aufbereitung der Artikel mehrerer Jahrgänge, um eine bessere Informationsextraktion zu ermöglichen.

Ein Teilprojekt von AETNA ist ERS, das *Experimental Resolution System* zur Bestimmung von Nominalphrasen-Koreferenzen. Eines der Ziele bei der Arbeit mit ERS ist es, präzise diejenigen Ressourcen identifizieren zu können, die für eine qualitativ hochwertige Koreferenzbestimmung notwendig sind.

ERS beginnt seine Arbeit nach einer partiellen syntaktischen Textanalyse, die zur Isolation der Nominalphrasen durchgeführt wird. Im wesentlichen besteht es aus zwei Komponenten: einer Sammlung von Heuristiken, die mögliche Koreferenzen bestimmen, sowie einem Algorithmus, der aus den Ergebnissen dieser Heuristiken sogenannte *Referenzketten* (reference chains) bildet.

Wir beschreiben zunächst kurz das existierende System und die bis jetzt noch offenen Anforderungen. Anschließend zeigen wir, wie diese Anforderungen mit Hilfe unseres Fuzzy-Ansatzes erfüllt werden können und stellen den Entwurf und die Realisierung von Fuzzy-ERS vor.

18.2.1 Referenzketten von Nominalphrasen

Als wesentliches Ergebnis liefert ERS sogenannte *Referenzketten von Nominalphrasen* (noun phrase coreference chains). Jede dieser Ketten enthält dabei eine Liste der Nominalphrasen eines Textes, die das gleiche Objekt referenzieren.

Die Referenzketten für den Beispieltext nach [Ber97] zeigt Abbildung 18.2. Die Koreferenzen sind dabei nach der sogenannten Lancaster-Notation gekennzeichnet; die Bezeichnung $_{(2.1<\text{ref}=1}$ bedeutet, daß die Nominalphrase 2.1 (eine Subnominalphrase von Nominalphrase 2) an diesem Punkt beginnt und die Nominalphrase 1 referenziert.

[2] Computer Science Department, Concordia University, Montréal, Québec, Canada

(1 Telxon Corp. 1) said (2(2.1<ref=1 its 2.1) vice president for manufacturing 2) resigned and (3(3.1<ref=1 its 3.1) Houston work force 3) has been trimmed by (4 40 people 4), or about (5<ref=4 15% 5).
(6<ref=1 The maker of hand-held computers and computer systems 6) said (7 the personnel changes 7) were needed to improve (8 the efficiency 8) of (9(9.1<ref=6 its 9.1) manufacturing operation 9).
(10<ref=1 The company 10) said (11<ref=10 it 11) hasn't named (12 a successor 12) to (13>ref=14 Ronald Bufton 13), (14<ref=2 the vice president 14) who resigned.
(15<ref=3(15.1<ref=1 Its 15.1) Houston work force 15) now totals (16 230 16).

Abbildung 18.2: Manuell bestimmte Nominalphrasen-Koreferenzen für den Beispielartikel aus Abbildung 18.1

18.2.2 Definitionen

Um ERS, sowie das neue Fuzzy-ERS präzise beschreiben zu können, definieren wir zunächst die verwendeten Begriffe:

Wort Ein Wort $w \in W$ ist eine Zeichenkette *(String)*.

Satz Ein Satz $s \in S$ ist ein Vektor $\langle w_1, \ldots, w_n \rangle$ von Wörtern.

Text Ein Text $t \in T$ ist ein Vektor $\langle s_1, \ldots, s_n \rangle$ von Sätzen.

Nominalphrase Eine Nominalphrase *(noun phrase, NP)* np $\in$ NP ist ein Vektor von benachbarten Wörtern $\langle w_i, \ldots, w_j \rangle$, $1 \leq i \leq j \leq n$ innerhalb eines Satzes $s \in S$, der auf Koreferenz untersucht wird. Dazu gehören die folgenden grammatikalischen Konstrukte:

- Substantive *(nouns)*
- Nominalphrasen *(noun phrases)*
- Pronomen *(pronouns)*
- Datumsangaben („24.12.1998“, „20. Juni“)
- Währungsausdrücke („$1.2 billion“)
- Prozentangaben („10%“)

Nicht berücksichtigt werden dagegen substantivische Verbformen *(gerunds)*. Beispiele für Texte von Nominalphrasen sind: *„John“*, *„the president“*, *„he“*, *„a titanium disk engine“*.

Jede Nominalphrase erhält zusätzlich eine Angabe über die Textstelle, an der sie gefunden wurde, bildet also ein Tupel bestehend aus dem oben definierten Wortvektor und seiner Position innerhalb des Textes. Im folgenden sprechen wir jedoch vereinfachend nur von „Nominalphrase“, wenn aus dem Kontext klar ist, ob alleine der Text oder das gesamte Tupel gemeint ist.

Koreferenz Zwei Nominalphrasen koreferieren, wenn sie sich auf das gleiche Objekt, beispielsweise die gleiche Menge, Funktion oder Aktivität beziehen. Die Koreferenz-Relation ist reflexiv, symmetrisch und transitiv.

Referenzkette Eine Referenzkette $c \in C$ ist eine Menge von Nominalphrasen (NPs): $c \subseteq \mathrm{NP}$.

Anhand der textweit eindeutigen Positionsangabe läßt sich jede Nominalphrase eindeutig referenzieren und so insbesondere von anderen Nominalphrasen mit dem gleichen textuellen Bezeichner unterscheiden. Dies ist wichtig für die Koreferenz-Relation, da eine Nominalphrase immer mit sich selbst koreferieren muß (Reflexivität), aber zwei textuell gleiche Nominalphrasen nicht notwendigerweise das gleiche Objekt bezeichnen, was insbesondere bei Pronomen der Fall ist.

18.2.3 Architektur von ERS

Die Architektur von ERS verwendet einen modularen Ansatz und unterscheidet sich damit grundsätzlich von den traditionell monolithisch strukturierten Sprachverarbeitungssystemen [Ber97]. Zur Bestimmung der Koreferenzen werden für ERS separate Heuristiken entwickelt, die jeweils ein spezifisches linguistisches Phänomen abdecken. Diese Vorgehensweise basiert auf den Ergebnissen einer manuellen Korpusanalyse, bei der sich gezeigt hat, daß ein Großteil der Koreferenzen durch relativ einfache, kostengünstige Verfahren bestimmt werden können [BK96].

Jede dieser Heuristiken hat dabei einen genau definierten Satz an Ressourcen, wie Syntaxbaum, Wörterbuch oder Thesaurus, den sie zur Ausführung benötigt. Ist eine bestimmte Ressource nicht verfügbar, kann das System trotzdem weiter arbeiten, indem die auf der fehlenden Ressource basierende Heuristik herausgenommen wird. Dadurch ist ERS in der Lage, auch dann noch eine Analyse zu liefern, wenn bestimmte Informationen nicht verfügbar sind. Lediglich die Sicherheit des Ergebnisses verschlechtert sich möglicherweise, abhängig davon, wie viele und welche Heuristiken ausfallen.

Im Rahmen der automatischen Analyse von großen Informationsmengen ist dies jedoch vorteilhafter als ein kompletter Abbruch, da ein solches System nie über alle Informationen verfügen kann, die in natürlichsprachigen Texten vorkommen können:

Grundsätzlich ist immer davon auszugehen, daß Wörter nicht im verwendeten Wörterbuch zu finden sind, Namen unbekannt sind oder grammatikalische Konstrukte nicht komplett erkannt werden. Für den Einsatz unter realen Bedingungen ist diese Robustheit daher eine Grundvoraussetzung.

Der modulare Aufbau von ERS macht es überdies möglich, billigere Heuristiken (im Sinne von Ausführungszeit) vorzuziehen und teure Heuristiken (wie eine komplette Syntaxanalyse) nur dann auszuführen, wenn mit einfachen Heuristiken kein sicheres Ergebnis erreicht werden kann.

Wir werden die einzelnen Heuristiken weiter unten in Abschnitt 18.3.1 vorstellen, wo wir deren Umwandlung in Fuzzy-Heuristiken diskutieren.

18.2.4 Der Kettenbildungs-Algorithmus

Die beschriebenen Heuristiken sind voneinander unabhängig, lassen sich also als separate Module realisieren. Die benötigten Ressourcen sind dabei sehr unterschiedlich, sie reichen von einfachen String-Vergleichen bis hin zu Lexika und semantischen Netzwerken.

Um jedoch die oben beschriebenen Nominalphrasen-Referenzketten liefern zu können, wird ein Algorithmus benötigt, der diese Heuristiken gezielt anwendet und aus deren Ergebnissen die Referenzketten berechnet.

In ERS wird dazu der folgende Algorithmus verwendet [Ber97]:

1. Bestimme zunächst mögliche Kandidaten (Nominalphrasen) im gleichen und im vorhergehenden Satz.

2. Für jeden dieser Kandidaten gilt: Falls die Nominalphrase ein Pronomen ist, verwende die Pronomen-Heuristik. Ansonsten wende nacheinander die Identitäts-, CommonHead-, Synonym/Hypernym-, Appositions- und Akronym-Heuristiken an. Sobald eine Heuristik erfolgreich ist, also eine Koreferenz feststellt, werden keine weiteren Heuristiken mehr aufgerufen.

3. Wenn eine Koreferenz gefunden wurde, füge die Nominalphrase in die entsprechende Kette ein. Ansonsten beginne eine neue Kette mit dieser Nominalphrase.

18.2.5 Neue Anforderungen

ERS verwirklicht eine Reihe neuer und innovativer Ideen zur Koreferenzbestimmung. Erste Erfahrungen mit dem realisierten System haben aber zu neuen Anforderungen geführt, die sich mit der vorhandenen Architektur nicht ohne weiteres erfüllen lassen:

Multiple Referenzen

ERS wurde so entworfen, daß es aus Effizienzgründen nach der ersten gefundenen Referenz keine weiteren Heuristiken mehr anwendet. Dies hat sich aber als problematisch herausgestellt, da es so dazu kommen kann, daß mehrere getrennte Einzelketten für ein einzelnes Objekt gebildet werden. Ziel ist es aber immer, möglichst alle Referenzen auf ein Objekt in einer Kette zu bündeln.

Dieses Ziel ließe sich besser erreichen, wenn man nach einer erfolgreich erkannten Koreferenz mit den restlichen Heuristiken nach weiteren Verbindungen suchen könnte. Dies wäre zwar mit der Architektur von ERS grundsätzlich möglich, würde jedoch sofort zu einem weiteren konzeptionellen Problem führen: Es gibt keine Möglichkeit um festzustellen, ob es sich bei einer zweiten gefundenen Koreferenz um eine alternative Klassifikation (nur eine der erkannten Referenzen ist korrekt) oder um eine zusätzliche Referenz handelt (beide gehören in eine Kette). Es gibt auch keine einfache Möglichkeit, alternative Ketten parallel zu halten, um dann später eine auszuwählen — denn dafür müßten zusätzliche Auswahlstrategien entwickelt werden.

So gehen aber Informationen verloren. Die Forderung ist also, während der Verarbeitung multiple Ergebnisketten zuzulassen, um am Ende der Verarbeitung mehr Informationen zur Berechnung des Gesamtergebnisses zur Verfügung zu haben.

Unterschiedliche Referenzsicherheiten

Heuristiken fällen binäre Entscheidungen: zwei Nominalphrasen koreferieren entweder „ganz oder gar nicht“. Wie schon in Abschnitt 7.1 auf Seite 61 erläutert, erfordern solche scharfen Verfahren daher eine hohe Sorgfalt bei der Bestimmung dieser Grenze, damit sie keine willkürlichen und sachlich falschen Ergebnisse zur Folge haben. Gerade in der Computerlinguistik ist man aber ständig mit einer Grauzone konfrontiert, da selbst für einen menschlichen Leser oft nicht eindeutig klar ist, ob eine Koreferenz vorliegt.

Muß man viele Fälle in dieser Grauzone modellbedingt ausschließen, verliert man gleichzeitig Informationen, die zur Verbesserung des Ergebnis beitragen könnten. Die Forderung ist daher, bei Koreferenzen die Sicherheit einer Referenz festhalten zu können.

Beliebige Referenzdistanz

ERS beschränkt sich bei seiner Suche nach Koreferenzen auf den aktuellen, den vorhergehenden und nachfolgenden Satz. Somit entgehen ERS alle Referenzen eines Textes, die über größere Distanzen gehen. Diese Einschränkung ist im Grunde unerwünscht, da durchaus Referenzen über mehr als einen Satz hinweg auftreten, und keine scharfe Grenze über die Entfernung gezogen werden kann.

Würde man diese Einschränkung aber komplett aufheben, gäbe es ein neues Problem: kann die echte Referenz nicht mit den eingesetzten Heuristiken gefunden werden, durchsucht das System den gesamten Text nach passenden Koreferenzen — gerade bei längeren Texten wird sich so praktisch immer ein Kandidat finden, der zwar die Bedingungen einer Heuristik erfüllt, aber tatsächlich keine echte Referenz darstellt. Falsche Referenzen sind jedoch mindestens genauso unerwünscht wie fehlende Referenzen.

Dieses Problem hängt mit dem zuletzt geschilderten zusammen: eine Referenz, die über eine lange Textdistanz gefunden wird, ist für ERS genauso sicher wie eine, die im gleichen Satz steht, was aber nicht der Realität entspricht. Zwar wird die Suche immer im aktuellen Satz begonnen, womit satzinterne Referenzen implizit Vorrang gewinnen, aber dies ersetzt keine explizite Modellierung der Referenzsicherheit.

Die Forderung ist also, den Kettenbildungsalgorithmus dergestalt zu erweitern, daß er mit Referenzen über beliebigen Distanzen umgehen kann.

Stabilere Heuristikausführung

Eine der Grundideen bei der Architektur von ERS ist, für verschiedene linguistische Phänomene separate Heuristiken zu entwickeln, deren Ergebnisse dann untereinander kombiniert werden. So wird die Robustheit erhöht, da der Ausfall einer Ressource nicht mehr zu einem kompletten Systemabbruch führt. Auch möchte man leicht eine neue Heuristik in das System einbringen können, um mit ihr einen isolierten Spezialfall abzudecken.

Nun muß aber ERS die zur Verfügung stehenden Heuristiken in einer fest definierten Reihenfolge anwenden. Die Reihenfolge ist besonders wichtig, da nach der ersten gefundenen Referenz aus den oben beschriebenen Gründen abgebrochen wird.

Grundsätzlich würde man erwarten, daß sich eine neue, möglicherweise experimentelle Heuristik schnell in das System integrieren läßt. Dabei sollte sie keine unerwünschten Seiteneffekte — über die Abdeckung ihres Spezialfalles hinaus — auf das Gesamtergebnis haben. Tatsächlich ist jedoch bei ERS ihre Plazierung in der Ausführungsreihenfolge der Heuristiken entscheidend für das Ergebnis: Stellt man sie an das Ende der Reihenfolge, kommt sie möglicherweise nicht zum Tragen, der Spezialfall würde also immer noch nicht abgedeckt. Plaziert man sie weiter vorne, kann sie als neue und somit potentiell unausgereifte Heuristik zu massiven Ergebnisänderungen führen.

Mit anderen Worten, selbst wenn die Heuristiken orthogonal zueinander formuliert sind, ist es ihre Ausführung nicht. Die Architektur von ERS zeigt sich in diesem Aspekt sehr instabil und erfordert große Sorgfalt bei der Auswahl und der Anwendung der verwendeten Heuristiken — es ist nicht ohne weiteres möglich, eine neue Heuristik zu realisieren, die einen bekannten Spezialfall abdeckt, und diese schnell in das System zu integrieren. Dies wurde bewußt in Kauf genommen, um eine hohe

Effizienz zu erreichen. Bei der Erweiterung verkehrt sich dies aber gerade ins Gegenteil.

Die Forderung ist daher, einen stabilen Rahmen zur Ausführung der einzelnen Heuristiken zu entwickeln, der ihre Ergebnisse unabhängig von deren Ausführung integrieren kann.

Steuerbares Ergebnis

Bei der Ergebnisberechnung schließlich entsteht immer ein Spannungsfeld zwischen der Forderung, nur korrekte Referenzketten zu bilden, die also keine falsch zugeordneten Ergebnisse enthalten, und dem Wunsch, möglichst vollständige Ketten zu berechnen, die alle potentiellen Koreferenzen enthalten.

Nun dient die Resolution von Nominalkoreferenz, wie oben beschrieben, als Grundbaustein für nachfolgende Verarbeitungsschritte. Diese können jedoch durchaus unterschiedliche Anforderungen an die Qualität der Referenzketten haben: während die eine Anwendung (wie eine Textprofilerstellung) nur absolut sichere Referenzen in den Ergebnisketten wissen möchte, könnte eine andere Anwendung (zum Beispiel eine Suchmaschine) möglichst lange Ketten anfordern, auch wenn darin möglicherweise vereinzelte falsche Zuordnungen enthalten sind. Mit der Architektur von ERS ist es aber nicht möglich, so eine steuerbare Ergebnisqualität anzubieten.

Die Forderung ist daher, eine solche Möglichkeit der Steuerung des Ergebnisses anzubieten.

18.3 Fuzzy-Nominalkoreferenzbestimmung

Die oben beschriebenen Anforderungen lassen sich mit unserem Fuzzy-Modell erfolgreich erfüllen, was nachfolgend gezeigt wird. Wir führen dazu Änderungen und Erweiterungen auf den Gebieten der Modellierung, Verarbeitung und Architektur von ERS durch:

Fuzzy-Heuristiken Wir zeigen zuerst, wie das Prinzip der Heuristiken auf *Fuzzy-Heuristiken* erweitert werden kann (Abschnitt 18.3.1).

Fuzzy-Verarbeitung Die Einzelergebnisse der Fuzzy-Heuristiken müssen dann, unter Berücksichtigung der neuen Anforderungen, zu einem Gesamtergebnis kombiniert werden. Wir zeigen dafür die Berechnung und Defuzzifizierung von *Fuzzy-Referenzketten*, aufbauend auf unserem Fuzzy-Modell (Abschnitt 18.3.2).

Fuzzy-ERS Zur Realisierung dieser Ideen entstand *Fuzzy-ERS*. Wir bestimmen zunächst dessen Architektur, die das existierende ERS in geeigneter Weise einbinden muß, und beschreiben anschließend seine Implementierung (Abschnitt 18.4).

Ein Beispiel zur Illustration der Anwendung des neuen Systems, sowie eine Zusammenfassung der geleisteten Arbeit schließen dieses Kapitel ab.

18.3.1 Fuzzy-Heuristiken

Wie oben ausgeführt, sind die verwendeten Heuristiken stets mit Unsicherheit behaftet: keines der vorgestellten Verfahren kann die Koreferenz von Nominalphrasen mit hundertprozentiger Sicherheit feststellen oder ausschließen. Fuzzy-Heuristiken ermöglichen es nun, die Sicherheit einer Koreferenz explizit zu modellieren und sie somit bei einer Weiterverarbeitung zu berücksichtigen. Wir zeigen im folgenden, wie die scharfen Heuristiken zu Fuzzy-Heuristiken erweitert werden können.

Eine scharfe Heuristik, wie sie derzeit in ERS Verwendung findet, nimmt als Parameter zwei Nominalphrasen $\mathrm{np}_i, \mathrm{np}_j$ und liefert als Ergebnis einen boolschen Wahrheitswert, der angibt, ob diese beiden Nominalphrasen koreferieren:

$$H : \mathrm{np}_i, \mathrm{np}_j \mapsto \{\text{true}, \text{false}\}$$

Das Ergebnis *true* bedeutet, daß nach Ansicht der Heuristik die beiden Nominalphrasen koreferieren. Jede Heuristik ist dabei spezialisiert auf ein bestimmtes linguistisches Phänomen und hat einen genau definierten Satz an Ressourcen, wie Syntaxbaum, Lexikon oder Thesaurus, den sie benötigt, um zwei Nominalphrasen auf Koreferenz untersuchen zu können.

Wegen der erwähnten Unsicherheit wird man bei einer scharfen Heuristik Grenzfälle, die eine Koreferenz vermuten, aber nicht mit Sicherheit bestätigen lassen, als nicht koreferierend kennzeichnen — andernfalls würde man Gefahr laufen, viele falsche Referenzen in das Ergebnis aufzunehmen.

Mit Hilfe von Fuzzy-Heuristiken läßt sich diese Einschränkung elegant umgehen, indem man als Ergebnis einen Sicherheitsgrad für die Koreferenz zweier Nominalphrasen bestimmt:

$$\mathcal{H} : \mathrm{np}_i, \mathrm{np}_j \mapsto [0, 1]$$

Abhängig von der Art der Heuristik werden die Sicherheitsgrade entweder aufgrund linguistischer Analysen berechnet, oder von einem Linguisten für ein spezifisches Phänomen vergeben. Für den letzten Fall ist es wichtig, eine geeignete Auswahl an Zugehörigkeitsgraden zu treffen, damit sich die Fuzzy-Heuristiken möglichst intuitiv formulieren lassen (vergleiche Definition 6.1.9 auf Seite 52). In dem hier entwickelten System werden dafür die folgenden fünf Grade verwendet: zwei Nominalphrasen koreferieren *sicher, üblicherweise, möglicherweise, selten* oder *nicht*. Diese entsprechen den α-Schnitten $\{1.0, 0.75, 0.5, 0.25, 0.0\}$.

Weiterhin soll es möglich sein, zwischen generellen Heuristiken und *Ausnahmen* von diesen zu unterscheiden, um die Übersichtlichkeit innerhalb der einzelnen Heuristiken zu verbessern. Im hier vorgestellten Fuzzy-ERS werden Ausnahmen mit Hilfe

zusätzlicher Heuristiken formuliert, die dann über eine konsistenzerhaltende Kontraktionsoperation (siehe Abschnitt 10.3 auf Seite 123) eingebracht werden. Mit dieser Trennung verbessert man zum einen die Übersicht innerhalb der Heuristiken, da eine einfache Regel nicht mit vielen Ausnahmen überfrachtet wird; zum anderen kann eine solchermaßen gekapselte Ausnahme problemlos in Kombination mit einer anderen Heuristik wiederverwendet werden.

Im folgenden stellen wir die Fuzzy-Heuristiken im einzelnen vor.

18.3.1.1 Fuzzy-Identitäts-Heuristik

Die einfachste, aber sehr oft vorkommende Form von Koreferenz ist die *Identität*, bei der die beiden beteiligten Nominalphrasen zeichenweise übereinstimmen. Dies tritt am häufigsten bei Namen auf. Diese Heuristik funktioniert jedoch nicht in allen Fällen: für Pronomen darf sie nicht angewendet werden, ebenso führt eine Anwendung auf Zahlen oder Beträge zu falschen Ergebnissen.

Die Identitäts-Heuristik ist sicherlich die einfachste Heuristik in ERS. Die Umsetzung in eine entsprechende Fuzzy-Heuristik ist jedoch nicht offensichtlich: Eine einfache Möglichkeit wäre, zwei Nominalphrasen genau dann als koreferierend zu kennzeichnen, wenn ihre Zeichenketten exakt übereinstimmen. So ergeben sich die beiden Sicherheitsgrade *nicht* und *sicher* für mögliche Koreferenzen. Diese Heuristik entspricht der scharfen Identitäts-Heuristik, ist jedoch unbefriedigend, da sie keinen Nutzen aus der Fuzzy-Modellierung zieht.

Die Fuzzy-Identität berechnet daher den Sicherheitsgrad der Koreferenz zweier Nominalphrasen anhand ihrer textuellen Repräsentation. Dabei wird betrachtet, mit welchem Grad die Zeichenketten zweier Nominalphrasen np_i und np_j übereinstimmen: Sind sie identisch, ergibt sich ein Sicherheitsgrad von 1,0. In vielen Fällen gibt es jedoch kleine Abweichungen, bei denen zwar ein großer Teil der Beschreibung wiederholt, aber nicht exakt kopiert wird, wie bei den beiden Nominalphrasen *„Telxon Corp.“* und *„Telxon“*. Erfahrungsgemäß ist eine Koreferenz um so sicherer, je mehr Teile der Zeichenketten übereinstimmen. Auf dieser Basis läßt sich eine einfache Fuzzy-Heuristik formulieren:

$$\mathcal{H}_{ident}(\mathrm{np}_i, \mathrm{np}_j) := \frac{|substr(\mathrm{np}_i, \mathrm{np}_j)|}{|\mathrm{np}_j|}$$

Der Sicherheitsgrad berechnet sich also aus der Länge des gefundenen Unterstrings von np_j in np_i, geteilt durch die Länge von np_i. Dabei wird gefordert, daß np_j komplett in np_i auftritt; die Heuristik arbeitet also nicht symmetrisch.

Ein Beispiel zeigt Abbildung 18.3 auf der nächsten Seite. Man sieht, daß ein reiner Zeichenkettenvergleich durchgeführt wird, dadurch erhält auch das Pronomen „he“ noch einen Referenzgrad von 0,04, da es im Wort „the“ gefunden wird.

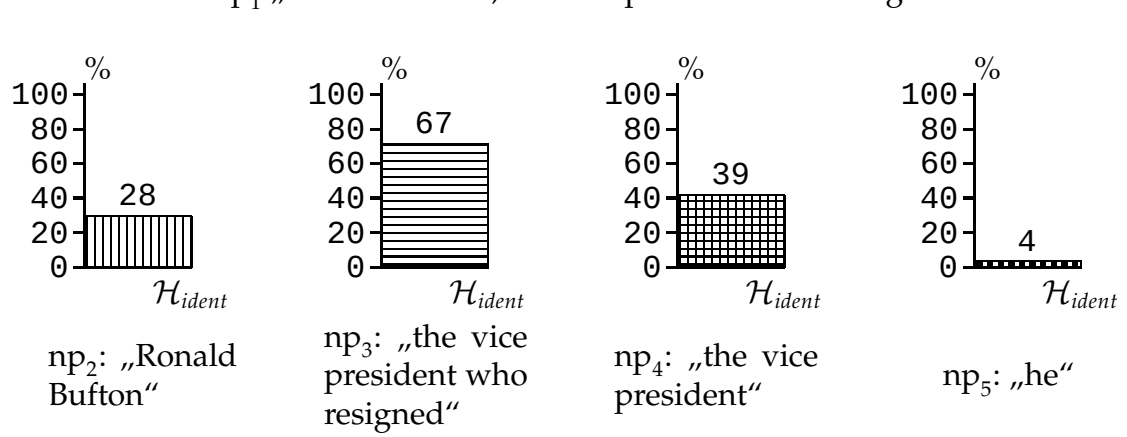


Abbildung 18.3: Beispiel zur Fuzzy-Identitäts-Heuristik

Umgang mit Ausnahmen: Fuzzy-Anti-Identität Wie zuvor erwähnt, gibt es für die meisten Heuristiken eine Reihe von Ausnahmen, die ihre Anwendung einschränken. Für die Identitäts-Heuristik etwa gilt, daß sie nicht auf Pronomen angewendet werden darf. Ebenso läßt sich die Koreferenz von Zahlen oder Datumsangaben nicht zuverlässig über einen Vergleich der Zeichenketten bestimmen. Auch diese Ausnahmen sind unscharf, da genau wie im positiven Fall eine Koreferenz nie absolut ausgeschlossen werden kann.

Diese Ausnahmen sind wiederum in einzelnen Heuristiken formuliert. Im Unterschied zu einer normalen Heuristik zeigt das Ergebnis aber an, mit welcher Sicherheit die beiden Nominalphrasen *nicht* koreferieren.

Wie diese Ausnahmen in die Berechnung des Gesamtergebnisses einfließen, zeigt Abschnitt 18.3.2.2 auf Seite 280.

18.3.1.2 Fuzzy-CommonHead-Heuristik

Oft sind Nominalphrasen nicht identisch, haben aber einzelne Wörter gemein. Wichtig ist insbesondere der sogenannte *Head* einer Nominalphrase, dies ist das Substantiv, das Geschlecht und Anzahl festlegt. Nach der *CommonHead*-Heuristik koreferieren Nominalphrasen, bei denen dieser Head übereinstimmt. Auch bei dieser Heuristik gibt es Ausnahmen, insbesondere bei Beträgen darf sie nicht angewendet werden.

Nach der scharfen CommonHead-Heuristik koreferieren beispielsweise die folgenden drei Nominalphrasen (der Head ist jeweils fett gedruckt): *„the Merc's* ***board****“*, *„the* ***board****“*, *„the* ***board*** *of directors“*.

Um diesen Head zu bestimmen, muß eine zumindest partielle syntaktische Analyse des Satzes vorgenommen werden. Der Sicherheitsgrad einer Koreferenz entspricht

dann der Sicherheit, mit der eine Teilnominalphrase als Head identifiziert wurde. Da die Syntaxanalyse momentan noch keine Fuzzy-Techniken einsetzt, ergibt sich an dieser Stelle als Ergebnis eine Koreferenz vom Grad *sicher* oder *nicht*. Eine Verbesserung durch das Fuzzy-Modell kann hier also erst erreicht werden, wenn auch die Syntaxanalyse genauere Informationen über die Sicherheit der erkannten Konstrukte liefert.

Ausnahmen: Fuzzy-Anti-CommonHead Zwei Nominalphrasen koreferieren *nicht*, wenn ihr Head einen Betrag kennzeichnet. Dies wird anhand einer Liste von Substantiven bestimmt, die einschlägige Wörter wie *„millions“* oder *„shares“* enthält.

18.3.1.3 Fuzzy-Pronomen-Heuristik

Die Pronomen-Heuristik dient zur Aufdeckung der Koreferenz zwischen Pronomen und nicht-pronominalen Nominalphrasen. Ihre Erkennung erfordert zum einen eine syntaktische Analyse des Satzes und zum anderen ein Lexikon, das Informationen über Geschlecht und Anzahl bereithält. Da Pronomen knapp 20% aller Koreferenzen ausmachen, muß ihre Überprüfung mit besonderer Sorgfalt durchgeführt werden.

Die Bestimmung von Koreferenz im Falle von Pronomen erfolgt selbst wieder in einer Reihe von Einzelschritten. Grundlage ist auch hier die partielle syntaktische Analyse des jeweiligen Satzes.

Überprüft wird dabei im einzelnen auf:

Pronomen? Zuerst wird mit Hilfe eines Lexikons bestimmt, ob die aktuelle Nominalphrase ein Pronomen darstellt. Falls nicht, kommt diese Heuristik nicht zum tragen und ein Sicherheitsgrad von *nicht* wird zurückgegeben.

Pleonastisches Pronomen? Im Falle eines Pronomens wird zuerst überprüft, ob ein pleonastisches Pronomen[3] vorliegt. Für solche Pronomen wird keine Koreferenz bestimmt, das Ergebnis der Heuristik ist also ein Grad von *nicht*.

Possessiv-Pronomen? Im Fall eines Possessivpronomens muß zunächst das zugehörige nicht-Possessivpronomen[4] gefunden werden. Mit diesem wird dann die Fuzzy-Pronomen-Heuristik rekursiv gestartet und das Ergebnis zurückgegeben.

Vergleich mit Nominalphrase Schließlich wird die zweite Nominalphrase, die selbst kein Pronomen sein darf, auf Übereinstimmung mit dem Pronomen überprüft.

[3] Ein Pronomen ohne Referenz, zum Beispiel: „Es ist Zeit, die Aktien abzustoßen“.

[4] Zum Beispiel würde in dem Satz *„He sold his shares.“* dem Possessivpronomen *„his“* das Pronomen *„he“* zugeordnet werden.

Da die einzelnen Schritte zum Teil sehr komplex sind, gehen wir an dieser Stelle nur auf die Änderungen und Neuerungen der fuzzyfizierten Verfahren ein.

Beim Vergleich mit der nichtpronominalen Nominalphrase wird überprüft, ob eine Übereinstimmung mit dem Pronomen hinsichtlich Anzahl und Geschlecht besteht. Hierfür wird, ähnlich wie bei der CommonHead-Heuristik, der Head der Nominalphrase herangezogen. Ergibt sich eine Übereinstimmung, wird für den Sicherheitsgrad der Koreferenz, wie weiter unten beschrieben, die Distanz zwischen Pronomen und Nominalphrase zugrundegelegt. Kann die Nominalphrase nicht im Lexikon gefunden werden, wird im klassischen ERS von keiner Koreferenz ausgegangen. Da eine Koreferenz in diesem Fall aber auch nicht ausgeschlossen werden kann, geht die Fuzzy-Heuristik hier von einer immerhin noch *möglichen* Koreferenz aus.

Gleichfalls kann es vorkommen, daß das Pronomen und die Nominalphrase *nicht* hinsichtlich Anzahl und Geschlecht übereinstimmen, aber trotzdem eine Koreferenz vorliegt. Ein Beispiel sind die beiden Sätze: *„The Band was formed in 1977. They had one tour and disbanded the next year."* Hier koreferieren, trotz fehlender Übereinstimmung, die Nominalphrase *„The Band"* (Singular) und das Pronomen *„They"* (Plural). Um solche Koreferenzen zu erkennen, benötigt man ein Lexikon, das Informationen über diese sogenannten Gruppen-Nominalphrasen enthält. Eine geplante Erweiterung der Pronomen-Heuristik kann dann auch solche Fälle abdecken, wobei ein geringerer Sicherheitsgrad als bei einer kompletten Übereinstimmung vergeben werden wird.

Selbst bei einer Übereinstimmung nimmt jedoch erfahrungsgemäß die Sicherheit der Koreferenz eines Pronomens mit einer Nominalphrase mit ihrer Entfernung zueinander ab. Dies läßt sich in natürlicher Weise als Fuzzy-Heuristik formulieren. Zunächst wird, wie oben beschrieben, die Koreferenz eines Pronomens mit einer Nominalphrase bestimmt; ergibt sich hier eine mögliche Koreferenz, wird der Sicherheitsgrad anhand der Distanz der Sätze, in denen die beiden Konstituenten auftreten, berechnet. Der Einfachheit halber wählen wir einen linearen Verlauf, wobei die Sicherheit der Koreferenz bei einer Distanz δ von 0 (beide stehen im gleichen Satz) genau 1,0 beträgt und bei einer parametrisierbaren maximalen Distanz $\delta_{\max}$ auf 0,0 sinkt. Sei nun c der auf die oben beschriebene Weise ermittelte Koreferenzgrad. Dann ergibt sich der Sicherheitsgrad der Pronomen-Heuristik $\mathcal{H}_{\text{pro}}$ eines Pronomens pro_i mit einer Nominalphrase np_j als:

$$\mathcal{H}_{\text{pro}}(\text{pro}_i, \text{np}_j) = c \cdot \max\left(0, \frac{\delta_{\max} - |\delta|}{\delta_{\max}}\right)$$

Ein Beispiel ist in Abbildung 18.4 auf der nächsten Seite zu sehen. Bei einem Koreferenzgrad von $c = 1{,}0$ und einer Distanz zwischen Pronomen und Nominalphrase von drei Sätzen ist das Ergebnis dieser Heuristik $\mathcal{H}_{\text{pro}} = 0{,}7$.

Innerhalb der einzelnen, immer noch scharfen, Pronomen-Subheuristiken bieten sich noch eine Reihe weiterer Möglichkeiten für zukünftige Verbesserungen durch unseren Fuzzy-Ansatz. Da dies aber den Rahmen dieser Arbeit sprengen würde, gehen wir darauf nicht weiter ein.

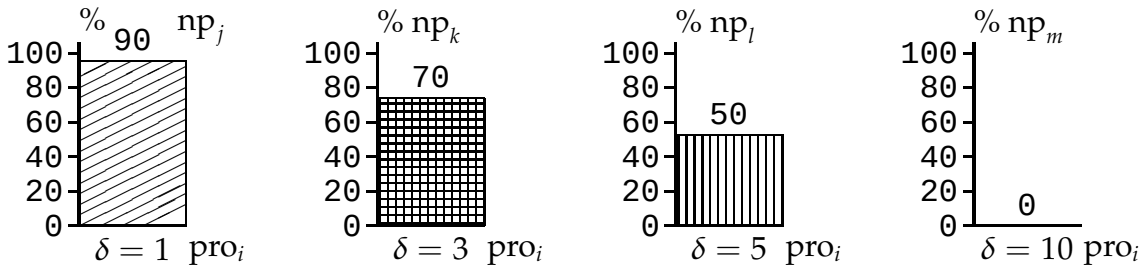


Abbildung 18.4: Beispiel für die Fuzzy-Pronomen-Heuristik mit Pronomen pro$_i$ und verschiedenen Koreferenz-Kandidaten np mit $\delta_{\max} = 10$ und $c = 1{,}0$

18.3.1.4 Fuzzy-Appositions-Heuristik

Eine Apposition ist die syntaktische Konstruktion [*NP, NP*]. Nach dieser Heuristik koreferieren zwei Nominalphrasen, wenn sie in einer solchen Apposition auftreten. Üblicherweise fügt dabei die zweite Nominalphrase beschreibende Informationen zur ersten hinzu. Ein Beispiel ist das Textfragment *„Ronald Bufton, the vice president“*, in der die beiden Nominalphrasen *„Ronald Bufton“* und *„the vice president“* eine Apposition bilden und somit koreferieren. Die Appositions-Heuristik funktioniert sehr zuverlässig, allerdings treten Appositionen nur selten auf: sie machen ca. 3% aller Nominalphrasen aus.

Voraussetzung für ein zuverlässiges Erkennen der Apposition ist eine partielle syntaktische Analyse. Der Fuzzy-Sicherheitsgrad dieser Heuristik entspricht somit der Sicherheit, mit der die Syntaxanalyse eine Apposition festgestellt hat. Momentan ist dies entweder ein Grad von *sicher* oder *nicht*, da auf der Ebene der Syntaxanalyse noch keine Fuzzy-Technologien eingesetzt werden.

18.3.1.5 Fuzzy-Synonym/Hypernym-Heuristik

Selbst wenn die beteiligten Nominalphrasen nicht textuell übereinstimmen, läßt sich eine mögliche Koreferenz oft anhand ihres semantischen Zusammenhangs bestimmen. Die dafür betrachteten Beziehungen sind hier *Synonyme* (zwei Begriffe sind bedeutungsgleich) und *Hypernyme* (ein Begriff ist ein Oberbegriff des anderen). Um diese Beziehung ermitteln zu können, wird ein Lexikon benötigt, das solche semantischen Verbindungen bereitstellt. In ERS kommt dafür *WordNet* [Fel98] zum Einsatz, dies ist ein Lexikon, das ein semantisches Begriffsnetzwerk enthält. Abbildung 18.5 zeigt als Beispiel die Synonyme und Hypernyme des Wortes *„workforce“*: die Synonyme stehen dabei auf der ersten Zeile und die Hypernyme darunter.

Da in einem solchen semantischen Lexikon keine komplexen Nominalphrasen abrufbar sind, beschränkt man sich auf den Vergleich des *Head* der beiden Nominalphrasen (vergleiche Abschnitt 18.3.1.2 auf Seite 273). Zwei Nominalphrasen koreferieren

```
Synonyms/Hypernyms (Ordered by Frequency) of noun workforce

1 sense of workforce

Sense 1
work force, workforce, manpower, hands, men
       => force, personnel
           => organization, organisation
               => social group
                   => group, grouping
```

Abbildung 18.5: Ausgabebeispiel von WordNet: Synonyme und Hypernyme für die Nominalphrase *„workforce“*

sicher, wenn der Head der zweiten Nominalphrase ein Synonym des Head der ersten Nominalphrase ist.

Ist dies nicht der Fall, wird untersucht, ob eine Nominalphrase ein Hypernym der anderen darstellt. Hierbei ist es wichtig, die Distanz der beiden Wörter in der Begriffshierarchie zu berücksichtigen, da sich fast alle Substantive auf eine handvoll Oberbegriffe wie *Konzept* oder *Objekt* zurückführen lassen. In der klassischen Variante dieser Heuristik mußte daher wieder eine scharfe Grenze gezogen werden: nur direkte Oberbegriffe werden als koreferierend angesehen. Die Fuzzy-Heuristik dagegen reduziert die Sicherheit einer Koreferenz abhängig von der Distanz in der Begriffshierarchie: bei einer direkten Ober-/Unterbegriffsbeziehung ergibt sich ein Sicherheitsgrad von 1,0, jede weitere Stufe verringert den Grad um einen einstellbaren Faktor (zur Zeit 10%).

18.3.1.6 Fuzzy-Akronym-Heuristik

Akronyme treten am häufigsten bei Firmennamen auf. Nach dieser Heuristik koreferieren Nominalphrasen, bei denen jeweils eine die Abkürzung der anderen darstellt, beispielsweise koreferieren „General Motors Corp.“ und „GMC“.

Die Bestimmung der Nominalkoreferenz bei Abkürzungen geschieht in zwei Schritten:

1. Zuerst wird anhand einer Abkürzungsdatenbank überprüft, ob eine Nominalphrase die Abkürzung der zweiten darstellt. Ist dies der Fall, wird eine Koreferenz als *sicher* angenommen.

2. Wurde keine Abkürzung gefunden, wird aus den Anfangsbuchstaben der ersten Nominalphrase eine mögliche Abkürzung gebildet und mit der zweiten Nominalphrase verglichen. Der Grad der Sicherheit einer Koreferenz berechnet sich dann, ähnlich wie bei der Fuzzy-Identitäts-Heuristik, anhand der Übereinstimmung dieser beiden Zeichenketten.

Vor allem beim zweiten Schritt lassen sich im Detail noch weitere Verbesserungen einbringen; Voraussetzung dafür ist jedoch noch eine genauere Analyse der Akronym-Referenzen, die falsch oder gar nicht erkannt werden.

18.3.2 Fuzzy-Referenzketten

Als Ergebnis der einzelnen Heuristiken haben wir nun die Sicherheit der Koreferenz von Nominalphrasen-Paaren. Das gewünschte Ergebnis sind jedoch Referenzketten, wie sie in Abschnitt 18.2.1 auf Seite 264 beschrieben wurden.

Wir zeigen nachfolgend, wie Fuzzy-Referenzketten mit Hilfe unseres Fuzzy-Ansatzes modelliert werden können, wie ihre Berechnung anhand der Ergebnisse der Fuzzy-Heuristiken erfolgt und wie ein scharfes Ergebnis aus einer solchen Fuzzy-Referenzkette abgeleitet werden kann.

18.3.2.1 Modellierung von Fuzzy-Referenzketten

Zur Bildung der imperfekten Referenzketten aus den Ergebnissen der einzelnen gibt es zwei naheliegende Modellierungsvarianten:

1. Jede Nominalphrase wird mit einer Fuzzy-Annotation versehen, die für jede mögliche Referenzkette ihren Zugehörigkeitsgrad enthält; das heißt, die Domäne der verwendeten Fuzzy-Mengen ist die Menge aller Referenzketten. Ein Beispiel zeigt Abbildung 18.6 auf der nächsten Seite, wobei hier für eine Nominalphrase ein Zugehörigkeitsgrad von 50, 20, ... für Kette 1, 2, ... berechnet wurde.

2. Jede Referenzkette wird mit einer Fuzzy-Annotation versehen, wobei die Fuzzy-Facette den Zugehörigkeitsgrad jeder Nominalphrase zu dieser Kette angibt; das heißt, die Domäne der verwendeten Fuzzy-Mengen ist die Menge aller Nominalphrasen. Abbildung 18.7 zeigt ein mögliches Beispiel: in einer Referenzkette sind die Nominalphrasen $np_1, np_2, \ldots$ zum Grad 50, 20, ... vertreten.

Beide Möglichkeiten lassen sich realisieren, im folgenden wird jedoch die zweite Variante verwendet, da sie einige Vorteile aufweist:

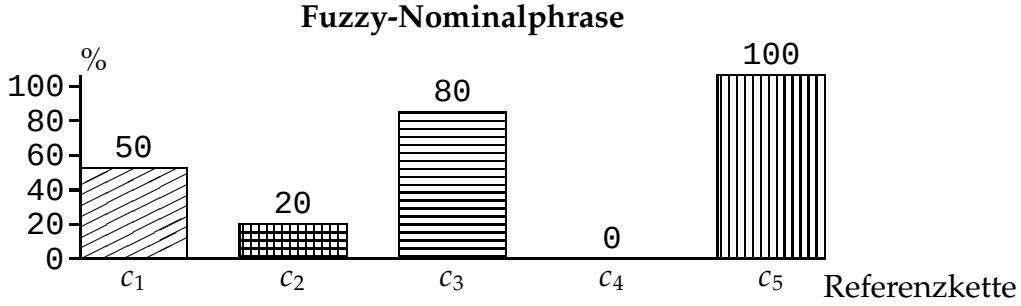


Abbildung 18.6: Fuzzy-Nominalphrase mit Zugehörigkeitsgraden zu Referenzketten

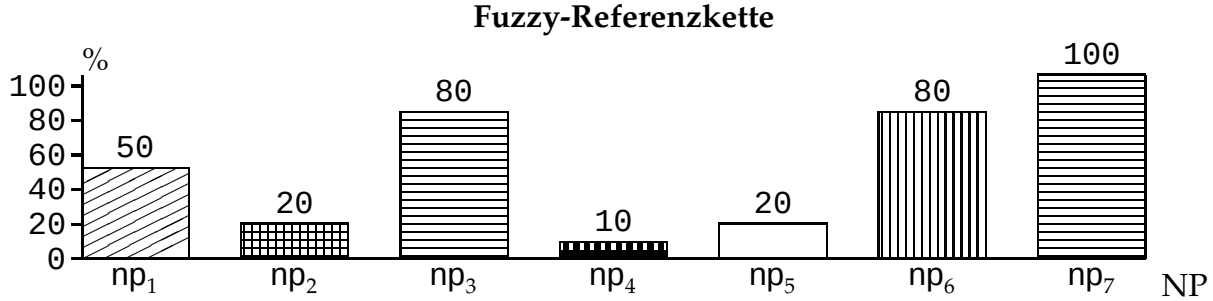


Abbildung 18.7: Fuzzy-Referenzkette mit Zugehörigkeitsgraden der Nominalphrasen

- Das gesuchte Ergebnis ist eine Menge von Referenzketten. Die zweite Lösung entspricht diesem Ergebnis direkt, so daß es nicht erst noch umgewandelt werden muß.
- Konsistenzbedingungen sind auf Ketten definiert, nicht auf Nominalphrasen. Mit Hilfe der zweiten Modellierung lassen sich diese einfacher umsetzen.
- In der ersten Lösung sind Ketten nur implizit definiert. Eine direkte Repräsentation der unscharfen Ketten wie in der zweiten Variante erlaubt es, komplexe Operationen wie das Verschmelzen zweier Ketten oder die Veränderung des Sicherheitsgrades einer Kette direkt anzuwenden.

Diese Fuzzy-Referenzketten, sowie deren scharfe Gegenstücke, werden in drei Schritten berechnet:

1. Zuerst werden die einzelnen Fuzzy-Heuristiken angewendet und aus deren Ergebnissen Fuzzy-Referenzketten gebildet, die jeweils die Koreferenz aus der Sicht einer Nominalphrase festhalten.
2. Da in einer Ergebniskette alle Nominalphrasen enthalten sein sollen, die miteinander koreferieren, werden diese Einzelketten miteinander *verschmolzen*.

3. Sofern ein nachfolgender Verarbeitungsschritt nicht direkt mit diesen Fuzzy-Ketten umgehen kann, wird es notwendig, sie durch Defuzzifizierung in scharfe Referenzketten umzuwandeln.

Diese Schritte werden nun in den nachfolgenden Abschnitten detailliert beschrieben.

18.3.2.2 Fuzzy-Kettenbildungsalgorithmus

Der erste Schritt ist die Bildung der einzelnen Fuzzy-Referenzketten, wie sie im letzten Abschnitt modelliert wurden, aus den Einzelergebnissen der Fuzzy-Heuristiken.

Dies geschieht mit dem folgenden Algorithmus: Gegeben sei ein Text $t \in T$ mit n Sätzen $\langle s_1, \ldots, s_n \rangle$, in denen insgesamt m Nominalphrasen $\langle \mathrm{np}_1, \ldots, \mathrm{np}_m \rangle$ vorkommen. Es kommen o Fuzzy-Heuristiken $\mathcal{H}_1, \ldots, \mathcal{H}_o$ zum Einsatz. Bestimme die Fuzzy-Referenzketten wie folgt:

1. Fuzzy-Heuristiken Jede Fuzzy-Heuristik $\mathcal{H}_i$ wird auf Paare von Nominalphrasen $\mathrm{np}_j, \mathrm{np}_k$ angewendet und liefert als Ergebnis ein Fuzzy-Atom. Dieses hat den Namen $\mathcal{A}^{\mathcal{H}_i}_{(\mathrm{np}_j, \mathrm{np}_k)}$ und als Fuzzy-Interpretation den Grad der Koreferenz der beiden Nominalphrasen. Dieses Ergebnis wird als Fuzzy-Literal $\mathcal{L}^{\mathcal{H}_i}_{(\mathrm{np}_j, \mathrm{np}_k)}$ festgehalten:

$$\mathcal{L}^{\mathcal{H}_i}_{(\mathrm{np}_j, \mathrm{np}_k)} := \mathcal{A}^{\mathcal{H}_i}_{(\mathrm{np}_j, \mathrm{np}_k)} = \mathcal{H}_i(\mathrm{np}_j, \mathrm{np}_k)$$

Zu jeder Fuzzy-Heuristik $\mathcal{H}_i$ kann es mehrere Ausnahmen $\widetilde{\mathcal{H}_i^1}, \widetilde{\mathcal{H}_i^2}, \ldots, \widetilde{\mathcal{H}_i^l}$ geben, von denen jede zu einem Fuzzy-Literal wird:

$$\widetilde{\mathcal{L}^{\mathcal{H}_i^x}_{(\mathrm{np}_j, \mathrm{np}_k)}} := \widetilde{\mathcal{A}^{\mathcal{H}_i^x}_{(\mathrm{np}_j, \mathrm{np}_k)}} = \widetilde{\mathcal{H}_i^x}(\mathrm{np}_j, \mathrm{np}_k)$$

Alle Ausnahmen zu einer Fuzzy-Heuristik $\mathcal{H}_i$ werden dann in einer Fuzzy-Klausel $\widetilde{\mathcal{K}^{\mathcal{H}_i}_{(\mathrm{np}_j, \mathrm{np}_k)}}$ vereinigt:

$$\widetilde{\mathcal{K}^{\mathcal{H}_i}_{(\mathrm{np}_j, \mathrm{np}_k)}} := \widetilde{\mathcal{L}^{\mathcal{H}_i^1}_{(\mathrm{np}_j, \mathrm{np}_k)}} \vee \widetilde{\mathcal{L}^{\mathcal{H}_i^2}_{(\mathrm{np}_j, \mathrm{np}_k)}} \vee \ldots \vee \widetilde{\mathcal{L}^{\mathcal{H}_i^l}_{(\mathrm{np}_j, \mathrm{np}_k)}}$$

2. Fuzzy-Nominalphrasen Die Ergebnisse aller Heuristiken zu einer Nominalphrase np_j werden gesammelt und in einer Fuzzy-Klausel $\mathcal{K}_{\mathrm{np}_j}$ festgehalten:

$$\begin{aligned} \mathcal{K}_{\mathrm{np}_j} := \{ \quad & \mathcal{L}^{\mathcal{H}_1}_{(\mathrm{np}_j, \mathrm{np}_1)}, \mathcal{L}^{\mathcal{H}_1}_{(\mathrm{np}_j, \mathrm{np}_2)}, \ldots, \mathcal{L}^{\mathcal{H}_1}_{(\mathrm{np}_j, \mathrm{np}_m)}, \\ & \mathcal{L}^{\mathcal{H}_2}_{(\mathrm{np}_j, \mathrm{np}_1)}, \mathcal{L}^{\mathcal{H}_2}_{(\mathrm{np}_j, \mathrm{np}_2)}, \ldots, \mathcal{L}^{\mathcal{H}_2}_{(\mathrm{np}_j, \mathrm{np}_m)}, \\ & \ldots, \\ & \mathcal{L}^{\mathcal{H}_o}_{(\mathrm{np}_j, \mathrm{np}_1)}, \mathcal{L}^{\mathcal{H}_o}_{(\mathrm{np}_j, \mathrm{np}_2)}, \ldots, \mathcal{L}^{\mathcal{H}_o}_{(\mathrm{np}_j, \mathrm{np}_m)} \} \end{aligned}$$

Analog werden die Ausnahmen der einzelnen Fuzzy-Heuristiken in einer Fuzzy-Klausel $\widetilde{\mathcal{K}_{\mathrm{np}_j}}$ gebündelt:

$$\begin{aligned}\widetilde{\mathcal{K}_{\mathrm{np}_j}} \quad := \quad & \widetilde{\mathcal{K}^{\mathcal{H}_1}_{(\mathrm{np}_j,\mathrm{np}_1)}} \cup \widetilde{\mathcal{K}^{\mathcal{H}_1}_{(\mathrm{np}_j,\mathrm{np}_2)}} \cup \ldots \cup \widetilde{\mathcal{K}^{\mathcal{H}_1}_{(\mathrm{np}_j,\mathrm{np}_m)}} \cup \\ & \widetilde{\mathcal{K}^{\mathcal{H}_2}_{(\mathrm{np}_j,\mathrm{np}_1)}} \cup \widetilde{\mathcal{K}^{\mathcal{H}_2}_{(\mathrm{np}_j,\mathrm{np}_2)}} \cup \ldots \cup \widetilde{\mathcal{K}^{\mathcal{H}_2}_{(\mathrm{np}_j,\mathrm{np}_m)}} \cup \\ & \ldots \cup \\ & \widetilde{\mathcal{K}^{\mathcal{H}_o}_{(\mathrm{np}_j,\mathrm{np}_1)}} \cup \widetilde{\mathcal{K}^{\mathcal{H}_o}_{(\mathrm{np}_j,\mathrm{np}_2)}} \cup \ldots \cup \widetilde{\mathcal{K}^{\mathcal{H}_o}_{(\mathrm{np}_j,\mathrm{np}_m)}}\end{aligned}$$

3. Fuzzy-Referenzketten Nun werden m Referenzketten gebildet; jede Referenzkette c_j wird dabei über eine Fuzzy-Formel $\mathcal{F}_{c_j}$ repräsentiert, die als Interpretation eine Fuzzy-Referenzkette wie oben beschrieben aufweist. Um dabei die Ausnahmen in das Gesamtergebnis einzubringen, wird eine γ-Kontraktionsoperation eingesetzt:

$$\mathcal{F}_{c_j} := \{\mathcal{K}_{\mathrm{np}_j}\} \ominus_\gamma \{\widetilde{\mathcal{K}_{\mathrm{np}_j}}\}$$

Dieses Verfahren liefert also zunächst die maximal mögliche Anzahl von Referenzketten, da immer so viele Referenzketten gebildet werden, wie es Nominalphrasen in einem Text gibt. Dabei kann eine Nominalphrase durchaus mehr als einer Kette zugeordnet sein.

18.3.2.3 Kettenverschmelzung

Der obige Kettenbildungsalgorithmus legt zunächst für jede Nominalphrase eine eigene Kette an. Jede Kette c_i enthält dabei die Koreferenzen „aus der Sicht“ der jeweiligen Nominalphrase np_i. Als Ergebnis interessieren jedoch die Gesamtinformationen, die über alle Ketten hinweg über die Koreferenzen der Nominalphrasen ermittelt wurden. Beispielsweise könnte die erste Kette die Information beinhalten, daß np_1 und np_3 (zu einem bestimmen Grad) koreferieren. Hält die zweite Kette zusätzlich eine Koreferenz von np_2 und np_3 (wieder zu einem bestimmten Grad) fest, wissen wir aufgrund der Symmetrieeigenschaft der Koreferenz (vergleiche Abschnitt 18.2.2 auf Seite 265), daß auch np_1 und np_3 koreferieren; alle drei Nominalphrasen gehören also in eine Ergebniskette.

Der nächste Verarbeitungsschritt ist daher das Verschmelzen *(merging)* der einzelnen Referenzketten zu einer Reihe *konsistenter Referenzketten*.

Wir definieren die Konsistenz einer Referenzkette über den Konsistenzgrad der zugehörigen Fuzzy-Formel:

Definition 18.3.1 (Kettenkonsistenz) *Der Konsistenzgrad $C(c_i)$ einer Referenzkette $c_i \in C$ ist definiert als der Konsistenzgrad der zugehörigen Fuzzy-Formel $C(\mathcal{F}_{c_i})$.*

Algorithmus 18.3.1 Kettenverschmelzung

```
1  VECTOR TEXT.MERGECHAINS( Vector chains, degree γ )
2  begin
3     Chain c1, c2;
4     Vector mergedChains := new Vector();
5
7     while (chains.size() > 0) do
8           c1 := chains.remove[1];
9
11          // prüfe Kette auf Kompatibilität mit den bereits verschmolzenen
12          for (Iterator i := mergedChains.iterator(); i.hasNext(); ) do
13             c2 := i.next();
14             if (c1.merge(c2, γ))
15                // Kette c2 wurde mit c1 verschmolzen, kann also entfernt werden
16                then i.remove();
17             fi
18          od
19
21          // überprüfe nun die verbliebenen Ketten
22          for (Iterator i := chains.iterator(); i.hasNext(); ) do
23             c2 := i.next();
24             if (c1.merge(c2, γ))
25                // Kette c2 wurde mit c1 verschmolzen, kann also entfernt werden
26                then i.remove();
27             fi
28          od
29          mergedChains.add(c1);
30    od
31
33    return(mergedChains);
34 end
```

Damit eine Referenzkette c_i also einen Konsistenzgrad von γ erreichen kann, muß es mindestens eine Nominalphrase np_j in dieser Kette geben, für die $\mu_{\mathcal{F}_{c_i}}(\text{np}_j) \geq \gamma$ gilt. Für die durch den obigen Kettenbildungsalgorithmus erzeugten Referenzketten gilt dies immer, da jede Nominalphrase per Definition mit sich selbst zum Grad von 1,0 koreferiert.

Beim Verschmelzen der Ketten werden die Informationen der Einzelketten kombiniert, indem die Eigenschaften der Symmetrie und Transitivität der Koreferenz ausgenutzt werden. Dabei entsteht allerdings ein Spannungsfeld zwischen dem Wunsch, möglichst vollständige, also lange Referenzketten zu bestimmen und der Forderung, eine möglichst hohe Präzision des Ergebnisses zu liefern, also keine falschen Nomi-

nalphrasen in eine Kette aufzunehmen.

Der Fuzzy-Ansatz bietet auch hier eine elegante Möglichkeit, zwischen den beiden Extrempunkten zu vermitteln, indem eine Verschmelzung basierend auf der Kettenkonsistenz durchgeführt wird: Dabei wird zunächst überprüft, ob der Konsistenzgrad der Vereinigung zweier Ketten c_i, c_j einen vorgegebenen Schwellwert γ erreicht. Ist dies der Fall, werden die Fuzzy-Interpretationen der beiden Ketten vereinigt, die Ergebnisse der Heuristiken also kombiniert:

$$\textbf{wenn } C(\mathcal{F}_{c_i} \cup \mathcal{F}_{c_j}) \geq \gamma, \textbf{ dann } \mu_{\mathcal{F}_{c_{(i,j)}}} := \mu_{\mathcal{F}_{c_i}} \cup \mu_{\mathcal{F}_{c_j}};$$

andernfalls wird mit der nächsten, nicht-verschmolzenen Kette fortgefahren. Damit kommt man zu dem auf der vorherigen Seite gezeigten Algorithmus 18.3.1.

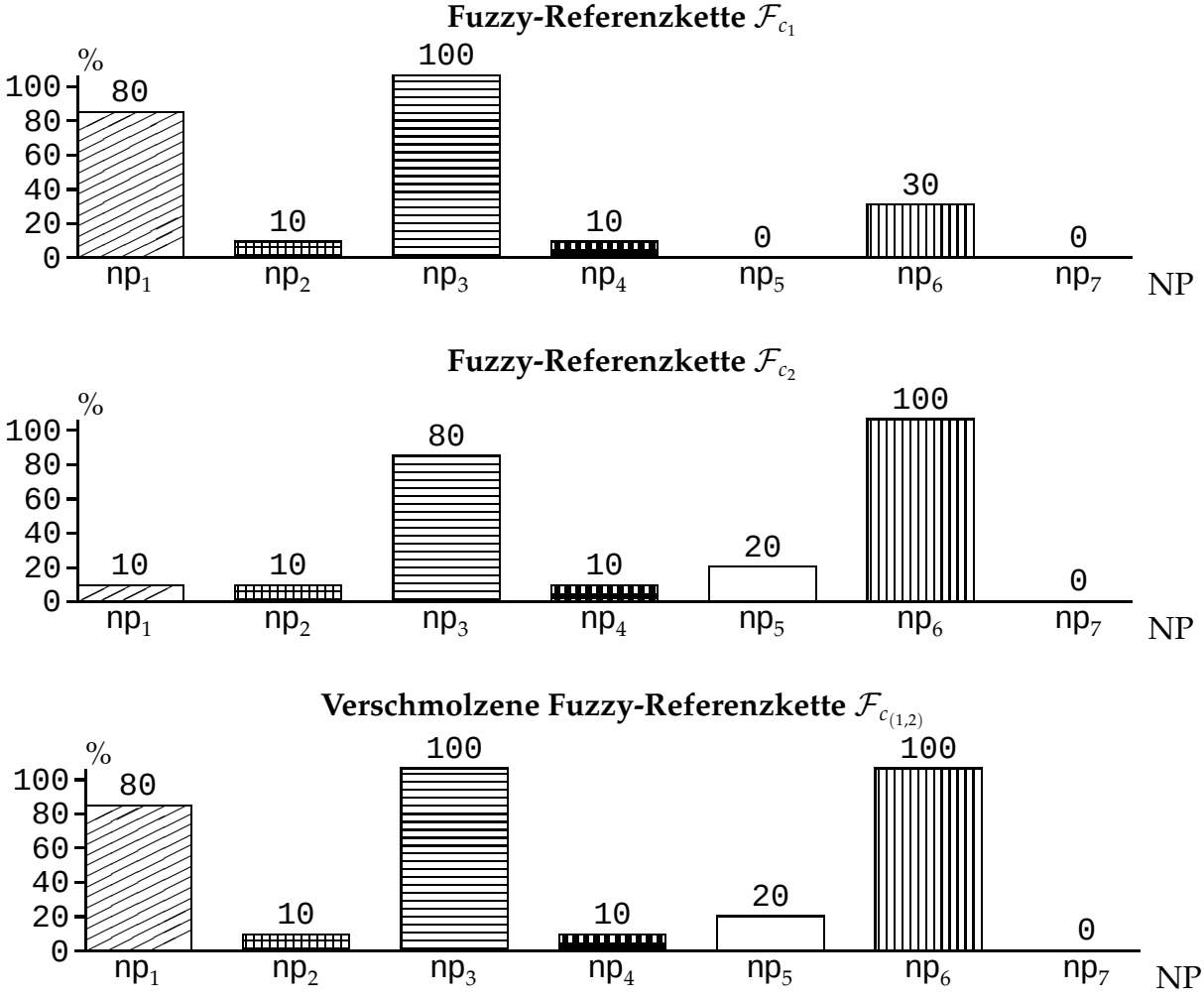


Abbildung 18.8: Verschmelzung zweier Fuzzy-Referenzketten mit $\gamma = 0{,}75$

Beispiel (Kettenverschmelzung) Ein Beispiel für die Verschmelzung zweier Ketten zeigt Abbildung 18.8; dort wurde bei einem Grad von $\gamma = 0{,}75$ aus den beiden einzel-

nen Ketten $\mathcal{F}_{c_i}$ und $\mathcal{F}_{c_j}$ (oben) eine neue Fuzzy-Referenzkette $\mathcal{F}_{c_{(i,j)}}$ (unten) gebildet. Bei einem geforderten Grad von $\gamma = 1{,}0$ dagegen würden die beiden Ketten nicht miteinander verschmolzen; im Ergebnis würden die Nominalphrasen np_3 und np_6 beispielsweise nur zu einem Grad von 0,3 koreferieren. Nach der unten beschriebenen Defuzzifizierung — mit einem entsprechend hohen Schwellwert — wäre diese Koreferenz nicht sicher genug und fiele daher aus dem scharfen Ergebnis heraus.

Wie oben schon ausgeführt, erlaubt es das Fuzzy-Verfahren damit, direkten Einfluß auf die Eigenschaften des Ergebnisses auszuüben und dabei Rücksicht auf die Anforderungen nachfolgender Bearbeitungsschritte zu nehmen: werden möglichst lange Ketten gefordert und dafür auch möglicherweise einzelne falsche Zuordnungen in Kauf genommen, läßt sich dies durch einen geringen Konsistenzgrad bei der Kettenverschmelzung erreichen. Sollen die Ergebnisketten dagegen nur möglichst sichere Koreferenzen enthalten, wird man den geforderten Konsistenzgrad höher ansetzen. Diese Steuerbarkeit ist ein direkter Vorteil gegenüber dem klassischen Verfahren, bei dem eine solche Anpassungsfähigkeit nicht gegeben ist.

18.3.2.4 Ketten-Defuzzifizierung

Werden für einen nachgelagerten Verarbeitungsschritt klassische, scharfe Referenzketten benötigt, müssen die Fuzzy-Ketten defuzzifiziert werden.

Hierfür wird eine leicht modifizierte Variante der Maximum-Methode (vergleiche Definition 6.3.1 auf Seite 58) verwendet: Eine scharfe Referenzkette enthält genau diejenigen Nominalphrasen, die in der Fuzzy-Kette mit einem Zugehörigkeitsgrad von mindestens γ vertreten sind.

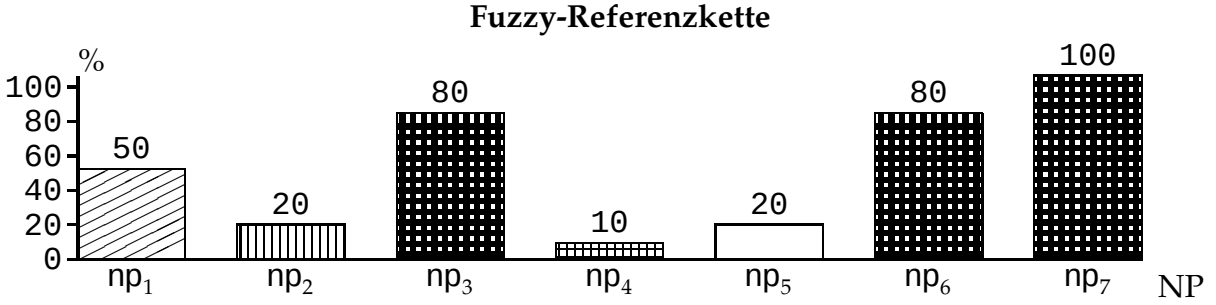


Abbildung 18.9: Fuzzy-Referenzkette zur Defuzzifizierung

Beispiel (Ketten-Defuzzifizierung) Ein Beispiel für eine Fuzzy-Referenzkette zeigt Abbildung 18.9. Defuzzifizieren wir diese mit einem Grad von $\gamma = 0{,}8$, erhalten wir die scharfe Ergebnismenge $\{\mathrm{np}_3, \mathrm{np}_6, \mathrm{np}_7\}$.

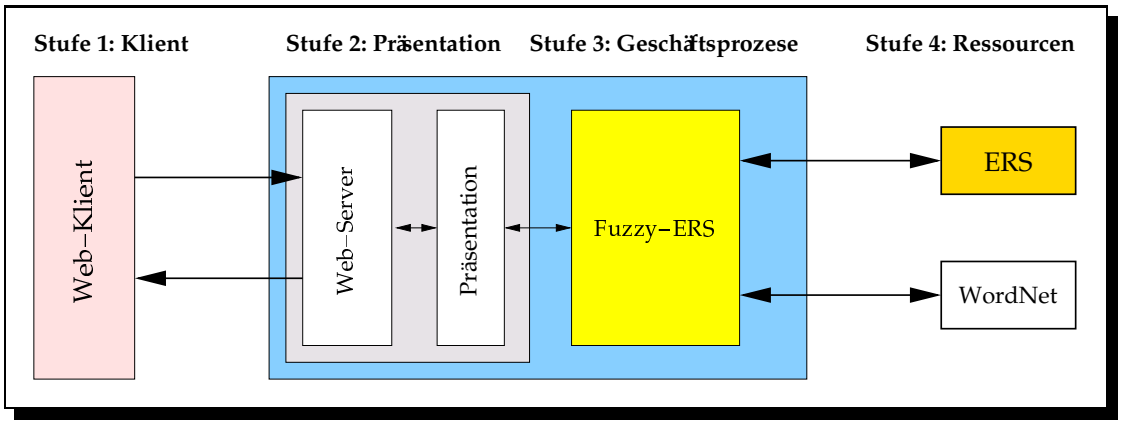


Abbildung 18.10: Die vierstufige Client/Server-Architektur von Fuzzy-ERS

18.4 Realisierung von Fuzzy-ERS

Das hier beschriebene System wurde prototypisch implementiert. Eine weitere Anforderung war dabei, das existierende System in geeigneter Weise in die Architektur einzubinden.

Wir beschreiben zunächst die Architektur von Fuzzy-ERS, bevor wir uns der Implementierung der einzelnen Stufen zuwenden.

18.4.1 Architektur

Wir leiten die für Fuzzy-ERS benötigte Architektur anhand der Auswahltabelle 14.1 auf Seite 217 von unserer Fuzzy-Referenzarchitektur ab.

Für die Steuerung sowie die Ergebnisausgabe soll ein Web-Klient zum Einsatz kommen; ein zusätzlicher autonomer Klient wird nicht gefordert, ebenso keine persistente Speicherung von Fuzzy-Daten. Ein Benutzer verwendet auch keine imperfekten Begriffe zur Interaktion, da Fuzzy-ERS, wie eingangs erwähnt, Fuzzy-Informationen nur zu internen Berechnungen einsetzt. Als Ausgabe werden zur Zeit nur defuzzifizierte, scharfe Ergebnisketten weitergeleitet, ein Austausch imperfekter Daten mit anderen Systemen wird also nicht gefordert. Somit ergibt sich für Fuzzy-ERS eine vierstufige Client/Server-Architektur, die in Abbildung 18.10 dargestellt ist.

Als erste Stufe wird ein Standard-Webbrowser verwendet, mit dem das System gesteuert wird und die Ergebnisse in Form von HTML-Seiten abgerufen werden können. Grundsätzlich sind an dieser Stelle weitere Klienten vorstellbar, zum Beispiel ein automatisches Testsystem.

Die zweite Stufe dient der Aufbereitung der Ergebnisse von Fuzzy-ERS in Form von HTML-Seiten.

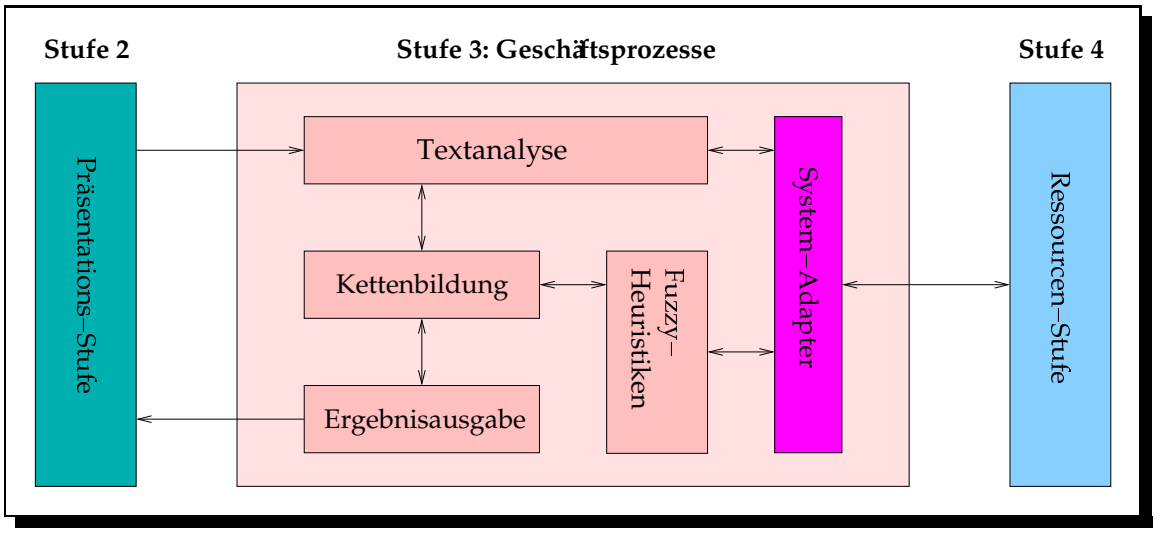


Abbildung 18.11: Aufbau der dritten Stufe von Fuzzy-ERS

Die dritte Stufe bildet den Kern von Fuzzy-ERS. An dieser Stelle sind die oben beschriebenen Algorithmen, insbesondere die Berechnung von Fuzzy-Referenzketten, realisiert. Der Webserver dient zur Steuerung von Fuzzy-ERS mit Hilfe eines WWW-Browsers, sowie der Bereitstellung der Ergebnisse in Form von HTML-Seiten.

Innerhalb der einzelnen Algorithmen werden zusätzliche Ressourcen benötigt, die über die vierte Stufe bereitgestellt werden. Insbesondere wird hier auf Teile des scharfen ERS und auf andere externe Systeme, wie das semantische Wörterbuch WordNet, zurückgegriffen.

Den Aufbau der dritten Stufe im Detail zeigt Abbildung 18.11. Die Textanalyse-Komponente verwaltet zu einem zu analysierenden Text alle zugehörigen Informationen, wie Satzstruktur und Syntaxanalyse, und bedient sich dafür teilweise der Funktionen des scharfen ERS. Im Prototyp steuert sie auch die weiteren Analysen, insbesondere die Koreferenzbestimmung, die in der Kettenkomponente realisiert ist. Diese wiederum verwendet eine Reihe von Fuzzy-Heuristiken, die ihrerseits für benötige Ressourcen auf Systeme der vierten Stufe zugreifen. Die Ergebniskomponente stellt das Modell zur Verfügung, das dann von der Präsentationskomponente in Form von HTML-Seiten dargestellt werden kann.

Die Ressourcen-Stufe schließlich wird von den verwendeten Systemen gebildet, dazu gehört zum einen das existierende ERS, auf das von den Fuzzy-Heuristiken für verschiedene Analysen zurückgegriffen wird, und zum anderen das erwähnte WordNet.

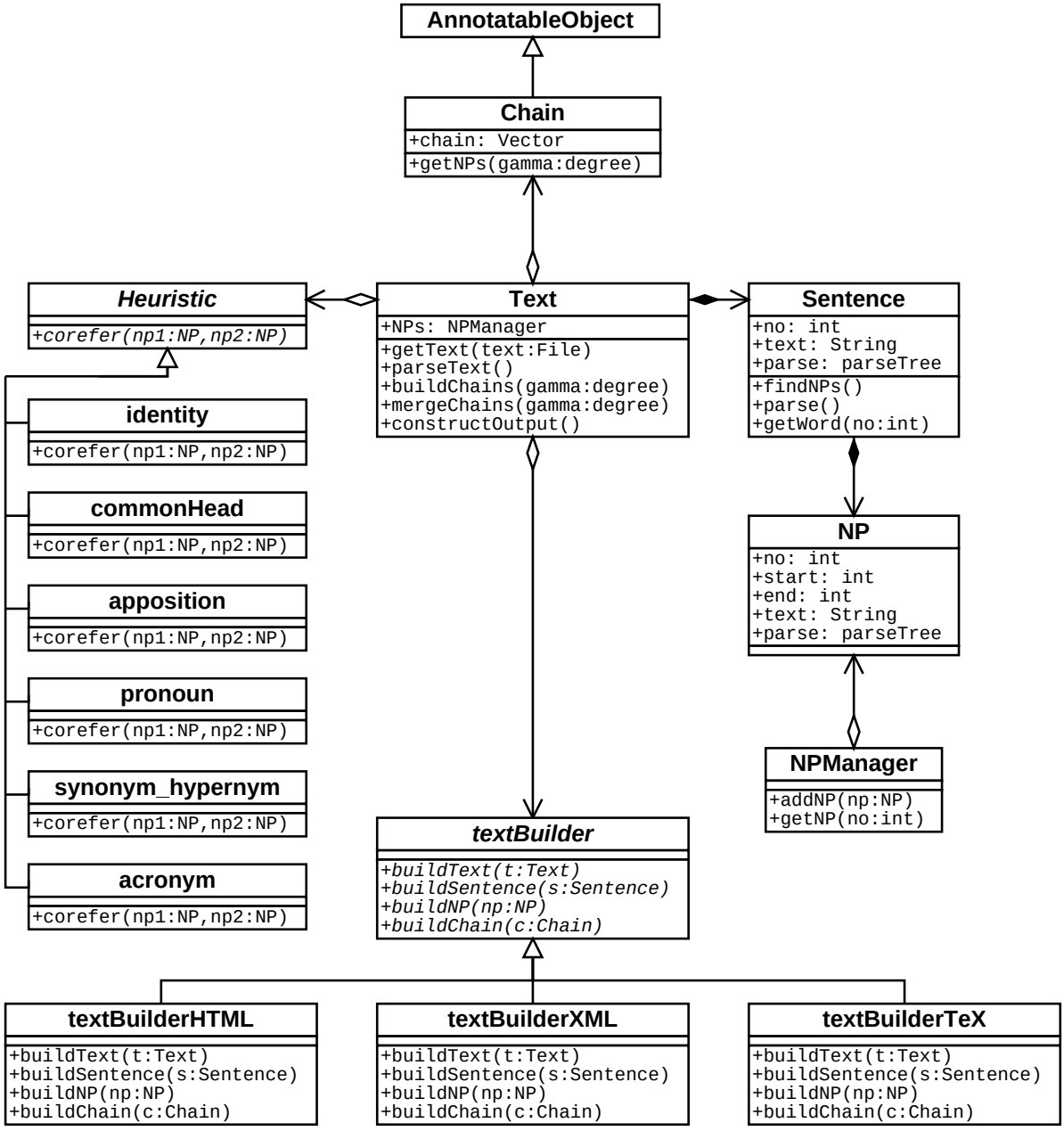


Abbildung 18.12: Klassenhierarchie von Fuzzy-ERS

18.4.2 Implementierung

Die Implementierung des Klienten ist im Prototypen über ein einfaches Web-Formular zum Starten der Analyse gelöst, das mit einem handelsüblichen Browser aufgerufen werden kann. Das Ergebnis erhält man dann in Form miteinander verbundener Webseiten, welche die einzelnen Teilergebnisse der Analyse darstellen.

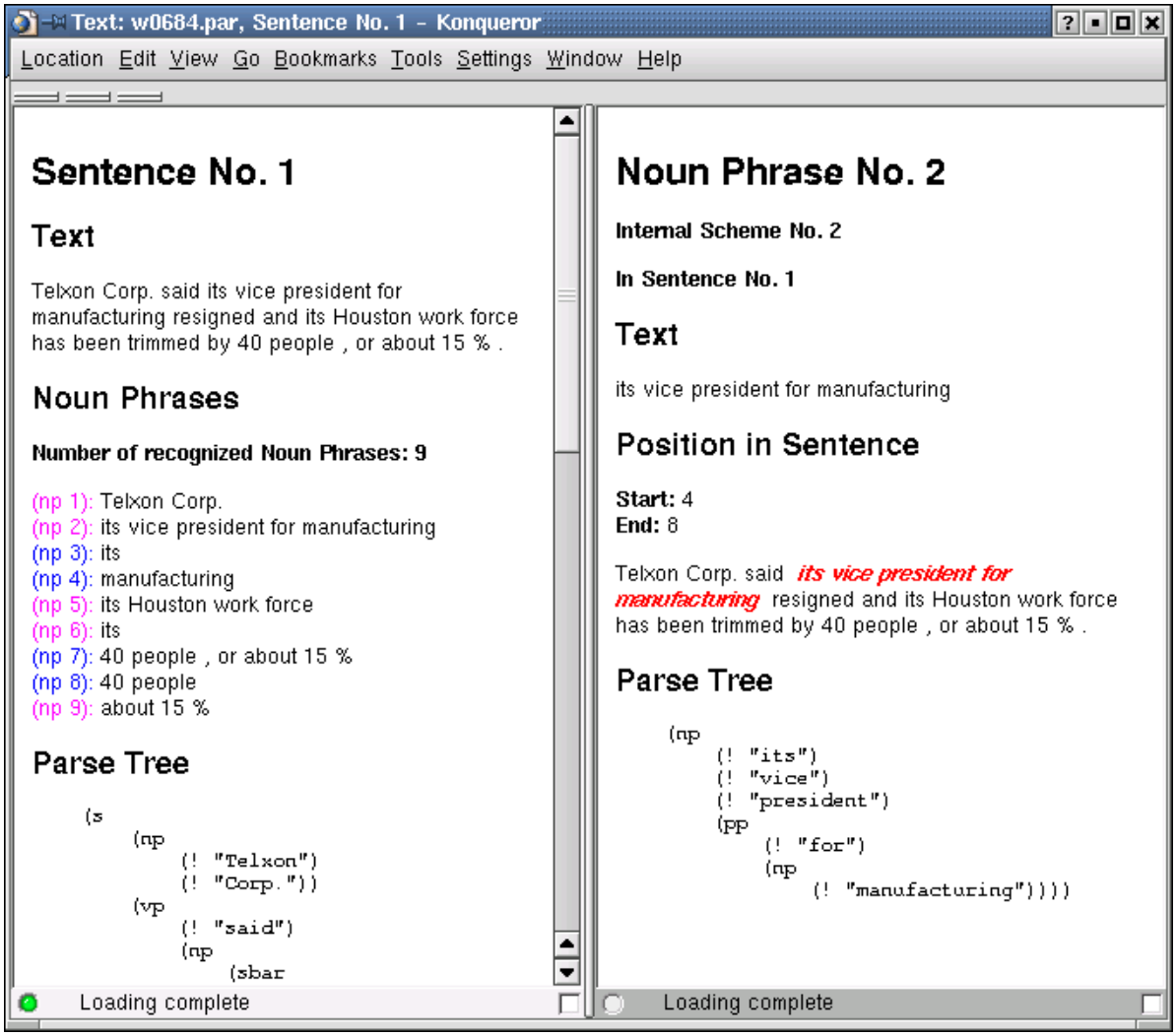


Abbildung 18.13: Beispiel für die HTML-Ausgabe von Satz Nr. 1 mit Nominalphrase Nr. 2

Das Kernsystem ist mittels einer Reihe von Java-Klassen implementiert; die wichtigsten zeigt Abbildung 18.12 auf der vorherigen Seite. Wir beschreiben im folgenden die wesentlichen Funktionen der einzelnen Klassen:

Chain Die Klasse `Chain` enthält alle Informationen zum Aufbau und zur Verwaltung von scharfen sowie von Fuzzy-Referenzketten. Sie benutzt dabei das in Kapitel 12 vorgestellte Annotations-Entwurfsmuster, um jeder scharfen Kette eine Fuzzy-Referenzkette zuzuordnen.

Text Die Klasse `Text` hält die Gesamtinformationen zu einem zu analysierenden Text zusammen. Dazu gehört die Struktur eines Textes in Form einzelner Sätze,

die jeweils von der Klasse `Sentence` verwaltet werden, sowie die ermittelten Referenzketten von Nominalphrasen.

Insbesondere sind hier die Algorithmen zur Erzeugung der einzelnen Ketten (vergleiche Abschnitt 18.3.2 auf Seite 278), der Kettenverschmelzung (vergleiche Abschnitt 18.3.2.3 auf Seite 281), sowie der Ketten-Defuzzifizierung (vergleiche Abschnitt 18.3.2.4 auf Seite 284) realisiert.

Sentence Die Verwaltung der einzelnen Sätze erfolgt in der Klasse `Sentence`. Dazu gehört der eigentliche Text des Satzes in Form einer Zeichenkette, Informationen über die Position des Satzes innerhalb des Gesamttextes, die strukturelle Analyse in Form eines Syntaxbaumes sowie eine Liste der in diesem Satz enthaltenen Nominalphrasen. Die letzten beiden Informationen werden dabei über einen Aufruf von ERS ermittelt.

NP Die Klasse `NP` verwaltet die Informationen zu einer Nominalphrase. Dazu gehören, ähnlich wie bei den Sätzen, der Text der Nominalphrase, ein Syntaxbaum sowie weitere strukturelle Informationen, wie die Position der Nominalphrase innerhalb eines Satzes.

NPManager Der `NPManager` implementiert eine Datenstruktur zum effizienten Zugriff auf alle im System verwalteten Nominalphrasen.

TextBuilder Die `TextBuilder`-Klasse realisiert, wie im gleichnamigen Entwurfsmuster *Erbauer*, den *Direktor* (Director) zur Erzeugung von Ausgabeformaten. Damit wird die Konstruktion eines komplexen Ausgabeobjektes von der internen Repräsentation entkoppelt, so daß von einer internen Struktur problemlos verschiedene Ausgabeformate — etwa HTML und XML — erzeugt werden können.

HTMLBuilder Der `HTMLBuilder` realisiert eine konkrete Unterklasse des *Erbauer*-Entwurfsmusters, nämlich die bereits erwähnte Präsentation in Form von HTML-Seiten. Ein Beispiel zeigt Abbildung 18.13 auf der vorherigen Seite. Weitere vorgesehene Unterklassen erzeugen ein XML-Format zum Datenaustausch mit anderen Systemen, sowie eine TEX-Dokumentation.

Heuristic Die Klasse `Heuristic` ist die abstrakte Oberklasse für die Familie der Fuzzy-Heuristiken. Sie implementiert dabei die Klasse *Strategie* im gleichnamigen Entwurfsmuster.

Identity, commonHead, ... sind die einzelnen Fuzzy-Heuristiken zur Resolution von Nominalkoreferenz, wie sie in Abschnitt 18.3.1 beschrieben sind. Teilweise greifen die Heuristiken dabei auf externe Systeme wie ERS oder WordNet zurück.

Zur Berechnung der Fuzzy-Nominalkoreferenz bedient sich die Klasse `Text` dabei der Fuzzy-Heuristiken, die in Form des Entwurfsmusters *Strategie* angeordnet sind;

die Klasse `Text` übernimmt in diesem Muster die Rolle der *Kontext* (Context)-Klasse. Momentan ist als einzige Strategie der Aufruf aller verfügbaren Heuristiken implementiert; für eine Weiterentwicklung bietet sich hier die Implementierung von Kostenmodellen an, die abhängig vom bereits erreichten Sicherheitsgrad einzelne Heuristiken anhand ihres Ressourcenverbrauchs auswählen.

Besondere Erwähnung verdient noch der in Abbildung 18.11 auf Seite 286 gezeigte Systemadapter. Da das klassische ERS in *Scheme*, einer funktionalen, mit LISP verwandten Sprache implementiert ist, wurde an dieser Stelle eine Java/Scheme-Schnittstelle realisiert, die die Einbettung eines Scheme-Interpreters in ein Java-Objekt ermöglicht und so die Verwendung des existierenden ERS als weitere Ressource erlaubt.

Für den Produktiveinsatz würde man die für einen Text ermittelten Heuristikergebnisse zusätzlich persistent speichern, um bei einer Anfrage daraus die Referenzketten je nach gewünschter Sicherheit zu berechnen. Auf diese Weise muß ein Text nicht mehrfach analysiert werden; es wird zusätzlich möglich, die Koreferenz für eine große Menge von Texten „offline“ im Voraus berechnen zu lassen. Hier zeigt sich wieder eine Stärke unseres Repräsentationsmodells: Ändert sich eine Fuzzy-Heuristik, beispielsweise weil sie verbessert wurde, oder muß eine Heuristik entfernt werden, weil sie fehlerhaft gearbeitet hat, müssen nicht alle Berechnungen wiederholt werden; sondern die gespeicherte Fuzzy-Beschreibung kann einfach syntaktisch modifiziert werden, indem das entsprechende Fuzzy-Literal ausgetauscht oder entfernt wird.

18.4.3 Beispiellauf

Wir illustrieren das Zusammenwirken der einzelnen Bestandteile anhand eines Beispiels. Dazu betrachten wir wieder den Eingangs gezeigten Beispieltext (siehe Abbildung 18.1 auf Seite 263).

Nach der Verarbeitung des Textes wird zunächst der gesamte Text, sowie die einzelnen ermittelten Referenzketten präsentiert, wie schon in Abbildung 18.13 auf Seite 288 gezeigt. Im ersten Ausgabebeispiel wurde, wie in Abbildung 18.14 zu sehen ist, ein Konsistenzgrad von 1,0 für die Kettenverschmelzung gewählt. Als Folge erhält man im Ergebnis nur solche Referenzketten, bei denen die beteiligten Nominalphrasen *sicher* koreferieren. Dabei entstehen jedoch mehrere Einzelketten, die korrekterweise miteinander verschmolzen werden müßten, wie die Referenzen auf das Unternehmen *„Telxon“*. Einem nachfolgenden Verarbeitungsschritt geht so unter anderem die Information verloren, daß das Unternehmen *„Telxon“* ein *„maker of hand-held computers and computer systems“* ist (vergleiche Kette Nr. 1 und Nr. 6), was sich sicherlich nachteilig auswirken dürfte.[5]

[5] Man sieht auch die Auswirkungen eines Fehlers in der verwendeten Fassung der CommonHead-Heuristik, wodurch die falsche Zuordnung von NP0 und NP1 entsteht.

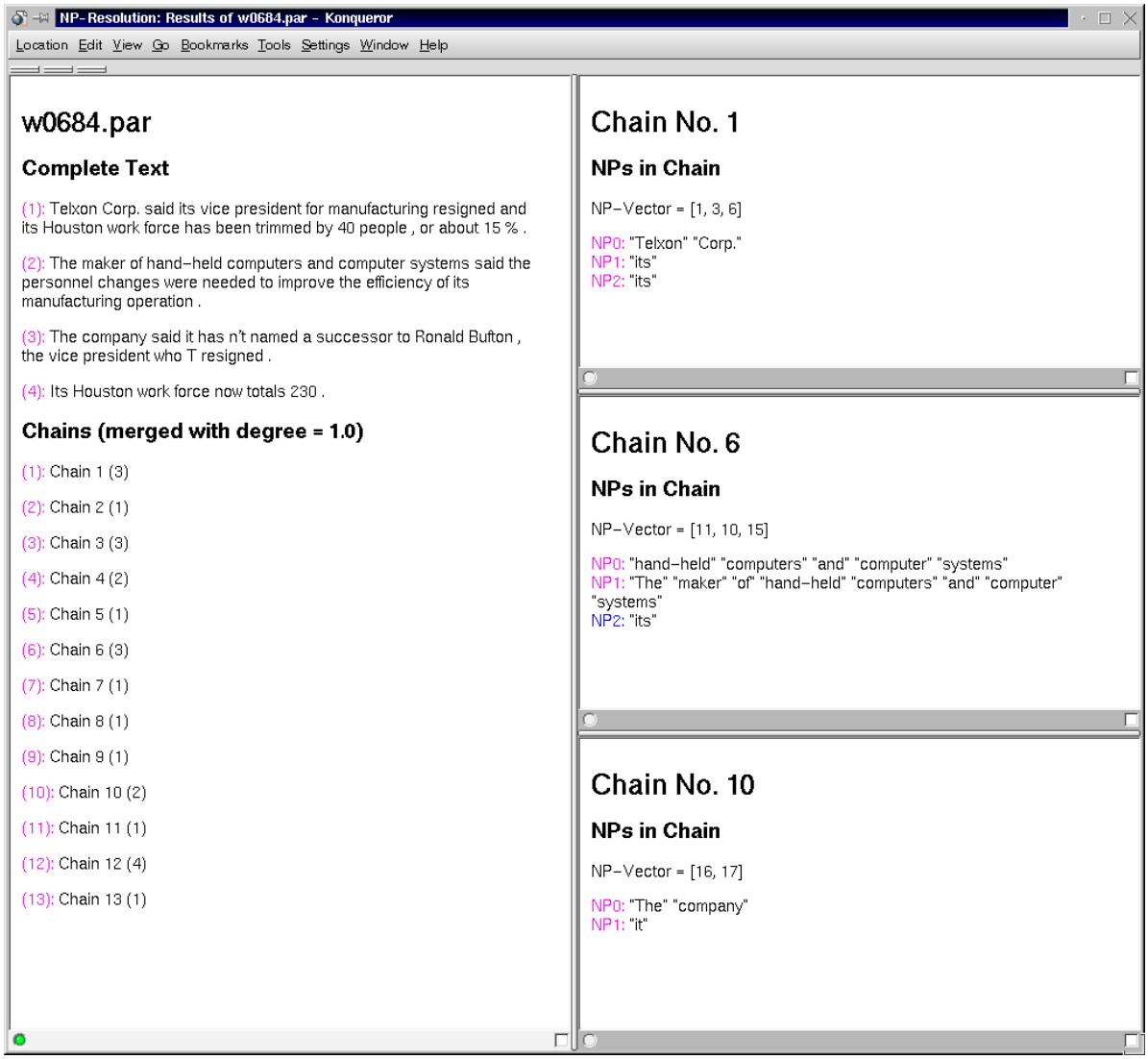


Abbildung 18.14: Beispielausgabe von Fuzzy-ERS mit einer Kettenverschmelzung vom Grad 1,0

Dagegen zeigt das zweite Ausgabebeispiel in Abbildung 18.15 auf der nächsten Seite das Ergebnis bei einem Verschmelzungsgrad von 0,75. Man kann erkennen, das deutlich mehr Einzelketten miteinander verschmolzen wurden, insbesondere wurden alle Einzelreferenzen auf die Firma *„Telxon Corp.“* in einer Kette zusammengefaßt. Käme unser System zum Beispiel wie eingangs illustriert in einer Suchmaschine zum Einsatz, könnten wir so — mit kleinen Einbußen in der Sicherheit — deutlich bessere Ergebnisse liefern, wenn etwa alle Hersteller von *„hand-held computers“* gefunden werden sollen.

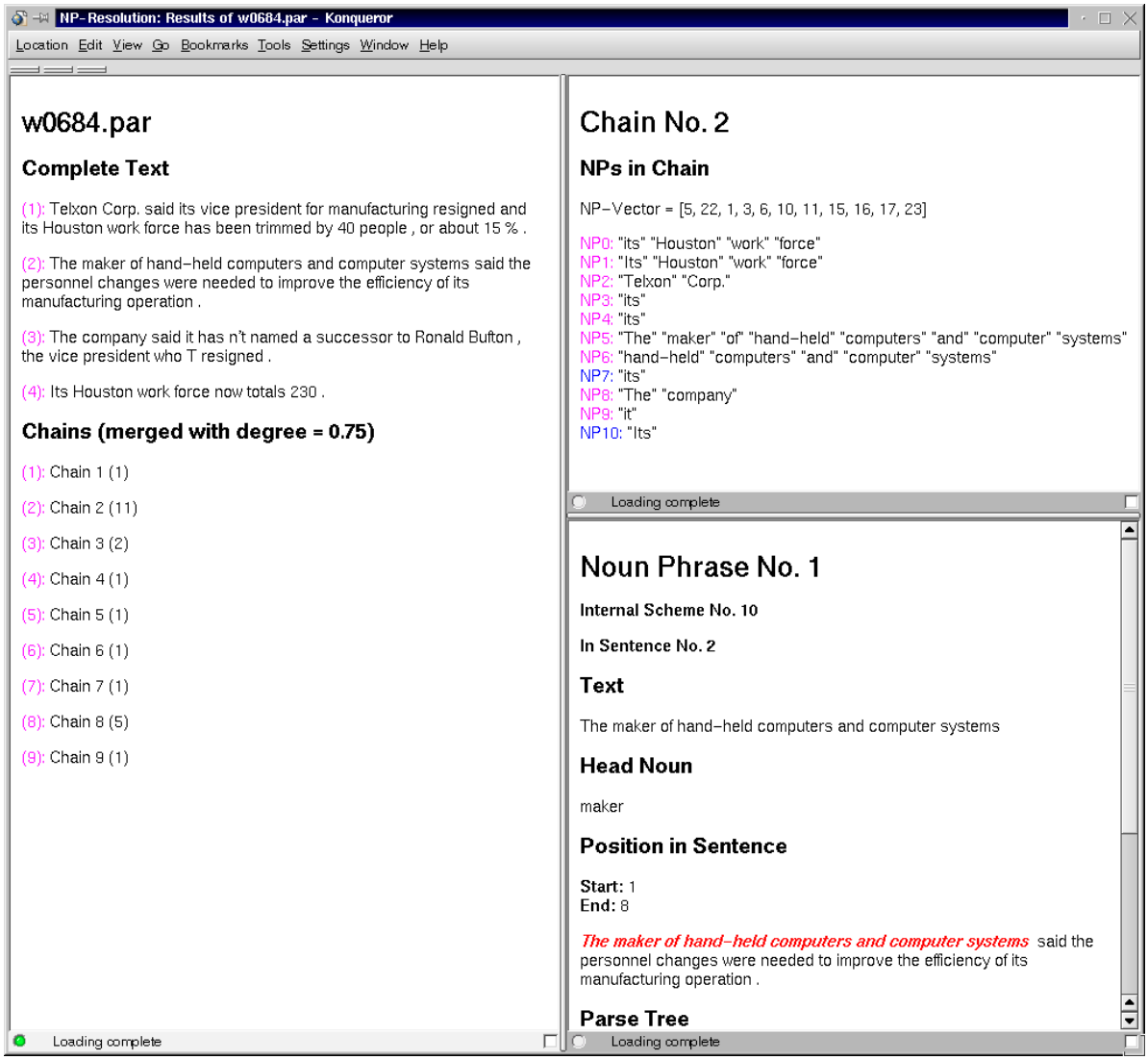


Abbildung 18.15: Beispielausgabe von Fuzzy-ERS mit einer Kettenverschmelzung vom Grad 0,75

18.5 Fazit

Dieses Szenario zeigt, wie unser Fuzzy-Modell bei einem bereits bestehenden und in Einsatz befindlichen System erfolgreich angewendet werden konnte. Wir fassen nun die mit Fuzzy-ERS erreichten Verbesserungen zusammen und gehen dazu die Anforderungen aus Abschnitt 18.2.5 (Seite 267) durch:

Multiple Referenzen: Fuzzy-ERS kann zur Bestimmung der Koreferenz alle verfügbaren Heuristiken anwenden und die Ergebnisse in parallelen Fuzzy-Ketten

vorhalten, die schließlich miteinander verschmolzen werden (Abschnitt 18.3.2.2 auf Seite 280). Damit gehen keine Einzelreferenzen mehr verloren wie im scharfen ERS, und so stehen am Ende der Verarbeitung auch mehr Informationen zur Verfügung um das scharfe Gesamtergebnis zu berechnen.

Unterschiedliche Referenzsicherheiten: Durch die Verwendung von Fuzzy-Heuristiken lassen sich unterschiedlich sichere Referenzen auf natürliche Weise formulieren (siehe Abschnitt 18.3.1 auf Seite 271). Dadurch wird es möglich, auch solche Koreferenzen zu berücksichtigen, die in einem scharfen System ausgeschlossen werden müssen, weil sie unterhalb der noch als sicher empfundenen Grenze liegen.

Beliebige Referenzdistanz: Diese Anforderung ließ sich durch den Einsatz von Fuzzy-Heuristiken problemlos erfüllen. Je nach Art der beteiligten Nominalphrasen kann ihre Distanz einfach in den Sicherheitsgrad ihrer Koreferenz mit einfließen (Abschnitt 18.3.1.3 auf Seite 274).

Stabilere Heuristikausführung: Mit den neu formulierten Algorithmen zur Referenzkettenbildung und -verschmelzung (Abschnitt 18.3.2.2 auf Seite 280 und Abschnitt 18.3.2.3 auf Seite 281) spielen nur noch die Ergebnisse der Heuristiken eine Rolle — und nicht mehr ihre Ausführungsreihenfolge wie im scharfen ERS. Das neue System hat so wesentlich an Robustheit gewonnen.

Steuerbares Ergebnis: Die geforderte Möglichkeit zur Steuerung der Qualität des Ergebnisses ist mit den entwickelten Algorithmen zur Fuzzy-Referenzkettenbildung und deren Defuzzifizierung erreicht worden (Abschnitte 18.3.2.3 und 18.3.2.4 auf den Seiten 281 und 284). Das Beispiel in Abschnitt 18.4.3 auf Seite 290 zeigt anschaulich, wie das Gesamtergebnis durch Einstellen eines Sicherheitsgrades beeinflußt werden kann.

Bei der Modellierung hat sich gezeigt, daß die Vorgaben, die das hier entwickelte Fuzzy-Modell durch seine Repräsentations- und Verarbeitungsverfahren macht, keine Einschränkungen sind, sondern tatsächlich eine Hilfe darstellen. Die semantisch höheren Konzepte wie die Fuzzy Konjunktiven Normalformen entlasten den Entwickler von Details wie dem Umgang mit einzelnen Fuzzy-Mengen und erlauben es stattdessen, sich auf die Modellierung der auftretenden Imperfektionen zu konzentrieren. Gleichermaßen beschleunigen die zur Verfügung gestellten Operationen den Entwurf geeigneter Algorithmen zur Verarbeitung der imperfekten Informationen. Ein positiver Nebeneffekt ist, daß sich am System arbeitende Linguisten nun auf die Formulierung neuer Heuristiken konzentrieren können, da der Aufwand zur Optimierung der Heuristikausführung wegfällt.

Die Realisierbarkeit wurde mit der Implementierung von Fuzzy-ERS nachgewiesen (Abschnitt 18.4 auf Seite 285). Dabei war die entwickelte Referenzarchitektur zusammen mit der Methodik zur Architekturauswahl (Abschnitt 14.3.5 auf Seite 216) ent-

scheidend für die rapide Aufstellung einer modernen, flexiblen, robusten Architektur für Fuzzy-ERS. Die Fuzzy-Bibliothek schließlich ermöglichte eine schnelle Umsetzung verschiedener Entwurfsalternativen und war somit zentraler Baustein für die Implementierung.

18.6 Ausblick

Für dieses erste Fuzzy-Nominalkoreferenzbestimmungssystem wurden die für das scharfe System entwickelten Heuristiken direkt übernommen und dafür fuzzyfiziert. Da die Originalheuristiken jedoch bereits unter der Einschränkung erstellt wurden, daß eine Repräsentation von Unsicherheit nicht möglich ist, können ihre Fuzzy-Gegenstücke nicht an allen Stellen Verbesserungen einbringen. Für eine nachfolgende Version ist daher vor allem interessant, welche neuen Heuristiken unter direkter Berücksichtigung imperfekter Analysen entwickelt werden können. Dies erfordert allerdings erst noch genauere Forschungen in Zusammenarbeit mit Linguisten. Besonders interessant wird dies, wenn Koreferenz über mehr als einen einzelnen Text hinweg untersucht werden soll. Damit öffnet sich eine Reihe neuer Einsatzmöglichkeiten, wie eine neue Art von Internet-Suchmaschinen. In Planung ist überdies ein System, das die in dieser Arbeit gezeigten Ansätze zum Fuzzy-Reengineering mit der Fuzzy-Textanalyse von Programmbeschreibungen (Spezifikation, Dokumentation) verbindet.

Ein weiterer Aspekt ist die Berücksichtigung von Unsicherheiten in vor- und nachgelagerten Systemen. Hier ergibt sich jetzt erstmals die Möglichkeit, die bei der syntaktischen Analyse typischerweise auftretenden Unsicherheiten nicht mehr unter den Tisch fallen zu lassen, sondern sie explizit an die Fuzzy-Nominalkoreferenzresolution weiterzugeben, um sie dann innerhalb der Fuzzy-Heuristiken auszunutzen. Dabei ist unter anderem die Frage interessant, in welcher Form die Ergebnisse eines beispielsweise auf statistischer Basis arbeitenden Parsers in einem Fuzzy-Modell verwendet werden können. Auch die Systeme, die auf den ermittelten Referenzketten aufsetzen, können von einem Fuzzy-Ansatz profitieren, da der mit der ansonst notwendigen Defuzzifizierung einhergehende Informationsverlust nicht mehr auftritt.

Kapitel 19

Zusammenfassung und Ausblick

Dodge this!
Trinity, "The Matrix"

In diesem Kapitel bieten wir einen Rückblick auf die geleistete Arbeit und weisen nach, daß die gesetzten Anforderungen erreicht wurden. Mit einem Ausblick auf weitere offene Fragestellungen schließt die Arbeit ab.

19.1 Resümee

Kehren wir zurück an den Anfang: der Forderung nach der Möglichkeit zum Umgang mit Imperfektion in Datenbanken und Informationssystemen [MS97]:

> *New applications of information systems require stronger capabilities in the area of uncertainty management. Our hope is that lasting interaction between these two areas would facilitate a new generation of information systems that will be capable of servicing these applications.*

Die Vision, die hieraus entsteht, ist die von einer neuen Art von Informationssystem, das besser mit der Begriffswelt seiner Anwender verbunden ist, robuster auf die Imperfektionen und Inkonsistenzen reagiert, mit denen die reale Welt nun einmal durchzogen ist.

Sollen solche Informationssysteme Verbreitung finden, muß eine Basistechnologie zur Verfügung stehen, die als Realisierungsgrundlage dienen kann. Denn sonst müßten diese Grundlagen für jede neue Anwendung wieder und wieder entwickelt werden, was eine weite Verbreitung der Fuzzy-Idee aufgrund des damit verbundenen immensen Aufwandes unmöglich macht [MS97]:

> *Without general systems that possess capabilities for handling imperfect information, such applications must be dealt with in an ad-hoc manner; this usually means that specific algorithms must be designed and specific systems must be built for each application that cannot be satisfied by the present generation of information systems.*

Betrachtete man jedoch den Stand der Forschung vor dieser Arbeit, mußte man feststellen, daß die beiden Welten der Unsicherheitsforscher und der Informationssystempraktiker zu weit voneinander entfernt waren, um sich gegenseitig zu befruchten. Noch einmal [MS97]:

> *Clearly, there is need for cooperation between the scientific communities of information systems and uncertainty. An outright rejection of the problem of uncertainty by the information systems community is unrealistic (even the "simple" problem of null values has yet to receive satisfactory treatment). On the other hand, highly sophisticated solutions might be unacceptable when implemented. As indicated in the introductory remark, truth lies halfway between the attitudes of the information systems radical and the uncertainty zealot.*

Eine der Stärken des in der vorliegenden Arbeit vorgestellten Ansatzes ist, existierende theoretische Arbeiten weiterzuführen und an praktische Anforderungen und Gegebenheiten anzupassen. Die entwickelten Lösungen stehen in der Tradition der Datenbankforschung, auf praktische Anforderungen zu reagieren und dafür effiziente Verfahren zu entwickeln. Viele bekannte Techniken, wie die Separation von Anwendung und Modell und die orthogonale Erweiterung bestehender Konzepte, wurden in dieser Arbeit eingesetzt, um eine Synergie aus bestehenden theoretischen Konzepten und praktischen Technologien zu schaffen. Die Funktion bestimmt die Form — *form follows function* (Frank Lloyd Wright) — um bei der Analogie zur Architektur zu bleiben. Dabei erfüllt das entwickelte Modell nicht nur die gestellten Anforderungen, wie wir unten noch genauer nachweisen werden, sondern bietet vor allen eine effiziente, adäquate Lösung, die für einen Praktiker nachvollziehbar und einsetzbar ist.

19.2 Zusammenfassung der geleisteten Arbeit

Tabelle 19.1 auf der nächsten Seite faßt die Anforderungen zusammen und zeigt, in welchen Abschnitten die einzelnen Anforderungen bearbeitet wurden (eine Klammer deutet dabei eine untergeordnete Berücksichtigung an). Im folgenden gehen wir die Anforderungen in einzelnen durch und diskutieren, wie sie befriedigt werden konnten.

Anforderung \ Kapitel (Nr.: Name)	8: Modellierung	9: Verarbeitung	10: Konsistenzerhaltung	11: Fuzzy OO-Modell	12: Annotationen	13: Fuzzy-Bibliothek	14: Architektur	15: Komponenten	16: Ressourcen
MODERNE ARCHITEKTUR				(✓)	(✓)	✓	✓	✓	✓
DURCHGÄNGIGES VERFAHREN	✓	✓	✓	✓	✓	✓	✓	✓	✓
UNSCHÄRFE, EFFIZIENZ & EFFEKTIVITÄT	✓	✓	✓	✓	✓	✓	✓	✓	✓
PARTIELLE UNSCHÄRFE				✓	✓	✓	✓	(✓)	✓
OBJEKTORIENTIERTES SYSTEM				✓	✓	✓	✓	✓	✓
VERARBEITUNG UNSCHARFER DATEN	(✓)	✓	✓	✓	✓	✓	(✓)	(✓)	(✓)
INTEROPERABILITÄT				✓	✓	✓	✓	✓	✓
KONSISTENZERHALTUNG	✓	✓	✓						

Tabelle 19.1: Bearbeitung der Anforderungen in den Themengebieten

19.2.1 MODERNE ARCHITEKTUR

Die Anforderung nach einer modernen Architektur (siehe Seite 27) verlangt, die für ein Fuzzy-System verwendeten Technologien präzise zu definieren. Überdies sollen als Basis relevante, von der angepeilten Zielgruppe aktuell verwendete Technologien zum Einsatz kommen. Die Kernidee hierbei ist, eine praktisch anwendbare Technologie zu schaffen, die von einem interessierten Systementwickler direkt eingesetzt werden kann.

Konsequenterweise spiegelt sich diese Anforderung in der gesamten Diskussion wider: Bei jeder Entwurfsentscheidung steht die praktische Einsatzfähigkeit im Vordergrund.

Im Detail findet sich in Teil V der Kern zur Erfüllung dieser Anforderung, das in Kapitel 14 (Seite 197) vorgestellte Architekturmodell. Es bringt die einzelnen Fuzzy-Erweiterungen zusammen und zeigt, wie ein herkömmliches Modell gezielt um den Einsatz von Fuzzy-Technologien erweitert werden kann. Die weiteren Kapitel dieses Teils widmen sich der praktischen Umsetzung des Modells, sie bauen dazu auf den technischen Grundlagen auf, die in Teil IV gelegt werden.

Zentraler Neuigkeitswert ist dabei die Betrachtung der *Architektur* eines Fuzzy-Informationssystems. Es wird demonstriert, daß existierende Architekturrahmen inkrementell um Fuzzy-Technologien erweitert werden können und daß eine Kooperation mit vorhandenen klassischen Systembestandteilen möglich ist. Aus dieser Sichtweise heraus konnten neue Ansätze für bekannte Fragestellungen entwickelt werden, wie

die Erweiterung relationaler Datenbanken um Fuzzy-Aspekte, die zusammen mit den weiteren Komponenten in Teil V vorgestellt sind.

19.2.2 DURCHGÄNGIGES VERFAHREN

Zentraler Punkt dieser Anforderung (siehe Seite 28) ist, Imperfektion nicht an nur einer Stelle in ein System oder Modell einzubringen, sondern dabei immer die Auswirkungen einer solchen Änderung, durchgängig über Theorie, Modellierung, Implementierung, Architektur und Einsatz hinweg, zu betrachten.

Diese Vorgehensweise zieht sich durch die gesamte Arbeit. Das theoretische Modell in Teil III enthält nur Eigenschaften, die sich aus den aufgestellten Anforderungen ergeben. Alle Teile dieses Modells wurden dann in Teil IV in ein formales objektorientiertes Datenmodell eingebettet. Es wurde gezeigt, wie dieses mit Hilfe objektorientierter Programmiersprachen implementiert werden kann. Aufbauend auf dem Stand der Technik wurden anschließend in Teil V die Erweiterungen eines Architekturmodells für Fuzzy-Informationssysteme analysiert und umgesetzt.

Wie wir bei der Diskussion dieser Anforderung gezeigt haben, ist gerade die fehlende Durchgängigkeit existierender Ansätze der Grund für die mangelnde Verbreitung von Fuzzy-Technologien in Informationssystemen (siehe auch Abschnitt 2.1 auf Seite 11). Diese Arbeit geht mit dem verfolgten Ansatz daher einen großen Schritt in Richtung praktisch einsetzbarer Fuzzy-Informationssysteme.

19.2.3 UNSCHÄRFE, EFFIZIENZ & EFFEKTIVITÄT

In dieser Anforderung (vergleiche Seite 29) gehen wir direkt auf den oben genannten Aspekt ein, daß ein Verfahren zum Umgang mit Imperfektion *angemessen* zu sein hat: Unsicherheit soll nur dort in ein System Einzug finden, wo dies Vorteile gegenüber einer klassischen Systemlösung bietet. Solche Vorteile können etwa in einer kürzeren Entwicklungszeit, einer einfacheren Modellierung der Anwendungsdomäne oder einem robusteren und vielseitigeren Endsystem liegen. Alle diese Vorteile lassen sich letztlich in der einen oder anderen Form auf die Entwicklungskosten zurückführen.

Der verfolgte Lösungsansatz war daher, bei allen vorgenommenen Fuzzy-Erweiterungen existierender Modelle und Technologien den damit verbundenen Zusatzaufwand so gering wie möglich zu halten. Da Imperfektion für uns nur ein Aspekt unter vielen innerhalb eines Informationssystems ist, darf seine Behandlung auch keinen überproportional hohen Stellenwert einnehmen. Ansonsten würde sein Einsatz aus praktischer Sicht mehr Aufwand verursachen, als die Behandlung der Imperfektion an Vorteilen einbringt.

Damit verbunden ist der Entwurf des formalen Repräsentationsmodells, bei dem vor allem auf Einfachheit und Effizienz geachtet wurde. Es wurden nur solche Ei-

genschaften in das Modell aufgenommen, die aufgrund der Anforderungen benötigt werden. Zwar wurde die Verwendung einfacher Fuzzy-Mengen als nicht ausreichend bewertet, aber darüber hinaus gehende Konzepte, wie ein Inferenzmechanismus, wurden wegen dieser Überlegungen nicht eingeführt. Im Vergleich zu anderen formalen Modellen, die solche Mechanismen anbieten, mag unser Modell daher einfach anmuten — die Mächtigkeit allein kann aber kein Kriterium für einen erfolgreichen Einsatz sein; denn sonst müßten alle existierenden Informationssysteme, deren einzige Möglichkeit zur Repräsentation von Unsicherheit ein Nullwert ist, zum Scheitern verurteilt sein. Dies ist offensichtlich nicht der Fall.

Diese Überlegung, formale Modelle direkt auf ihren Einsatzzweck zugeschnitten zu entwickeln, ist für den Bereich der Fuzzy-Informationssysteme offenbar neu, denn bisher versuchen alle vorgeschlagenen Ansätze lediglich, sich in ihrer Ausdrucksmächtigkeit gegenseitig zu überbieten.

19.2.4 PARTIELLE UNSCHÄRFE

In dieser Anforderung (Seite 30) wurde eine Abkehr von dem bisher vorherrschenden Paradigma der Fuzzy-Informationssysteme formuliert: Imperfektion ist *nicht* die zentrale Systemeigenschaft eines typischen Informationssystems, sondern nur eine weitere, orthogonale Eigenschaft, gleichberechtigt neben vielen anderen wie Persistenz, asynchrone Kommunikation, Transaktionsschutz oder Sicherheit. Entsprechend muß ein Verfahren zum Umgang mit Imperfektion die Möglichkeit bieten, existierende Modelle und Technologien systematisch und orthogonal um neue Konzepte zu erweitern.

Diese Idee der Orthogonalität stand bei allen durchgeführten Erweiterungen bestehender Modelle, Technologien und Systeme im Vordergrund. Die Arbeit konnte bei allen betrachteten Gebieten nachweisen, daß eine solche Erweiterung mit nur minimalen Eingriffen an bestehenden Systemen möglich ist. Damit erfordert der Einsatz von Imperfektion nicht mehr eine komplette Umstellung eines Gesamtsystems. Zur Realisierung wurde die Idee des Frame-Ansatzes in einer Weise mit dem objektorientierten Datenmodell verknüpft, die es erlaubt, Imperfektion präzise dosiert in ein System einzubringen — im Extremfall ist damit nur ein einzelnes Attribut eines einzelnen Objektes unscharf, was sich mit der vorgestellten Lösung elegant umsetzen läßt. Dies macht den Einsatz der Fuzzy-Technologie auch für solche Systeme interessant, bei denen imperfektes Wissen nur vereinzelt und isoliert auftritt.

Insbesondere spiegelt sich diese Anforderung in der Formulierung des objektorientierten Datenmodells und seiner Implementierung mit Hilfe des Annotations-Entwurfsmusters wider, die in Teil IV beschrieben sind. Der nachfolgende Teil setzt dann diesen Gedanken auf der Ebene der Anwendungsarchitektur fort.

Der hier verfolgte Ansatz steht damit in starkem Kontrast zu der Idee existerender Arbeiten, die *fuzzyness* als das dominate Entwurfsprinzip sehen. Kommt man aus der

Welt der Fuzzy-Regelungssysteme oder der künstlichen Intelligenz, ist diese Sichtweise zwar durchaus nachvollziehbar; für die Welt der Informationssysteme, wie sie hier betrachtet werden, führt sie aber nicht zu brauchbaren Entwürfen, wie in dieser Arbeit mehrfach nachgewiesen wurde.

19.2.5 OBJEKTORIENTIERTES SYSTEM

Diese Anforderung (Seite 32) verlangt, daß die entwickelten Konzepte die Einbettung und den Einsatz von Imperfektion in objektorientierten Systemen ermöglichen.

Obwohl erst seit neuerem zentraler Forschungsgegenstand, wurden bereits viele Vorschläge zur Erweiterung objektorientierter Datenmodelle um Fuzzy-Techniken gemacht, die wir in Teil I diskutiert haben. Auch hier stellt die verwendete Vorgehensweise eine Abkehr von bekannten Ansätzen dar: Bisherige Vorschläge betrachten mögliche Erweiterungen zwar theoretisch, vernachlässigen aber den zentralen Aspekt des praktischen Einsatz. Dies ist nicht akzeptabel, da der einzige Sinn und Zweck eines Datenmodells in seiner Anwendung zur Modellierung einer Anwendungsdomäne liegt, die typischerweise in einer praktischen Implementierung mündet. Folglich muß vor der Erweiterung des objektorientierten Datenmodells die Frage stehen, was denn die Anforderungen und Einschränkungen konkreter objektorientierter Fuzzy-Informationssysteme sind.

Als Ergebnis bietet unser formales objektorientiertes Fuzzy-Datenmodell, wie es in Kapitel 11 vorgestellt wurde, eine neue Herangehensweise bei der Einbettung von Fuzzy-Informationen: eine strikte Trennung der klassischen Bestandteile von den notwendigen Fuzzy-Erweiterungen. Diese Erweiterungen sind durchweg orthogonal ausgeführt, so daß sie sich problemlos mit Hilfe existierender Technologien umsetzen lassen.

Wichtig ist dabei, daß die vorliegende Arbeit nicht einfach ein einzelnes Datenmodell oder eine einzelne Programmiersprache betrachtet, sondern vielmehr aufzeigt, wie ein gegebenes Modell, eine gegebene Sprache systematisch um die entwickelten Fuzzy-Konzepte erweitert werden können.

Die Arbeit geht damit deutlich über den Stand der Forschung hinaus, da sie sich nicht damit begnügt, nur die prinzipielle Einbettung von Fuzzy-Konzepten in ein objektorientiertes Datenmodell aufzuzeigen, sondern zum einen explizit die Konsequenzen für eine praktische Umsetzung und deren Implementierung betrachtet und zum anderen direkt auf die Entwicklung von Anwendungen mittels des erweiterten Datenmodells eingeht. Zusätzlich wird der Verhaltensanteil betrachtet, worauf wir in Zusammenhang mit der nächsten Anforderung eingehen.

19.2.6 VERARBEITUNG UNSCHARFER DATEN

Die Anforderung VERARBEITUNG UNSCHARFER DATEN (Seite 33) resultiert aus der wenig aufsehenerregenden Beobachtung, daß das objektorientierte Datenmodell aus zwei Teilen besteht: der Struktur und dem Verhalten. Es ist folglich nicht ausreichend, nur die Struktur von Objekten um Fuzzy-Aspekte zu erweitern, ohne deren Verarbeitung mit Hilfe von Objekt-Methoden zu berücksichtigen.

Auch wenn diese Beobachtung selbst nicht schwierig ist, wird die Verarbeitung der imperfekten Daten von anderen Ansätzen nicht explizit behandelt, sondern mit den immer anwendbaren Datenbankoperationen *Insert, Update* und *Delete* als abgedeckt betrachtet. Dies ist zwar grundsätzlich richtig, übersieht aber eine Besonderheit der Anwendung von Fuzzy-Informationssystemen. Denn hier geht es nicht nur um die Abbildung des sich ändernden Zustandes einer realen Welt, bei der Datenbankoperationen korrespondierende Änderungen der Miniwelt durchführen — dafür sind einfache *Updates* (im Sinne von Katsuno und Mendelzon [KM91]) tatsächlich ausreichend. Vielmehr ist bei Fuzzy-Informationssystemen, wie sie in dieser Arbeit betrachtet werden, die Sammlung von imperfekten Informationen zu einer statischen Welt wichtig; dies kann eine Welt sein, wie sie einmal existiert hat (Beispiel der Verwaltung von Zeugenaussagen), wie sie gerade existiert (Beispiel des Reverse Engineering von Altsystemen) oder wie sie einmal existieren soll (Beispiel der Fahrzeugauswahl in einem Entscheidungshilfesystem). Hierfür sind Operationen notwendig, die eine *Revision* des angesammelten Wissens ermöglichen. Auf diese gehen wir weiter unten in Zusammenhang mit der Anforderung KONSISTENZERHALTUNG noch genauer ein.

Die Definition der Operatoren selbst ist in Teil III zu finden, während auf ihre Implementierung in Teil IV eingegangen wird. Neben den oben erwähnten Revisionsoperatoren sind hier auch weitere elementare Verarbeitungsoperatoren vorgestellt.

Die vorliegende Arbeit zeigt damit, daß es nicht reicht, imperfektes Wissen nur strukturell zu behandeln. Ohne eine geeignete, an die Anforderungen von Fuzzy-Informationssystemen angepaßte Verarbeitungssemantik kann ein Modell keinen praktischen Erfolg erzielen. Einem Anwendungsentwickler muß es möglich sein, Algorithmen auf semantisch höherer Ebene als der einzelner Fuzzy-Mengen zu entwickeln. Genau dies ist mit den entwickelten Operatoren möglich, wie unter anderem in den Anwendungsbeispielen gezeigt wird.

19.2.7 INTEROPERABILITÄT

Die Forderung nach der Interoperabilität (Seite 34) von Fuzzy-Systemen und -Systembestandteilen mit ihren klassischen Gegenstücken geht wiederum auf den Kostenaspekt zurück: Realistisch betrachtet wird die Fuzzy-Technologie nicht genug Vorteile bieten, um einen kompletten Neuanfang zu rechtfertigen. Vielmehr wird sie — wie alle anderen Technologien — einen speziellen Aufgabenbereich in einem System abdecken und zusammen mit anderen Technologien eingesetzt werden.

Der ingenieursmäßigen Entwicklung von Verfahren, die eine Koexistenz und Kooperation von klassischer und Fuzzy-Welt ermöglichen, muß daher ein eigener Wert zugesprochen werden; sie als „Notbehelf“ anzusehen wird ihrer Bedeutung nicht gerecht. Auch hier spielt das bereits erwähnte Prinzip der Orthogonalität eine tragende Rolle: denn dadurch wird es erst möglich, auf existierenden Technologien inkrementell aufzubauen, anstelle komplett neue entwickeln zu müssen.

Zusätzlich zu den Erweiterungen auf Technologieebene, die in Teil IV durchgeführt werden, schafft Teil V eine Reihe von Brücken zwischen den neuen Konzepten der Fuzzy-Informationssysteme und den klassischen Systembestandteilen. Anstatt beispielsweise grundlegende Änderungen an Datenbankmanagementsystemen zu fordern, wird demonstriert, wie ein beliebiges relationales Schema in orthogonaler Weise um Fuzzy-Inhalte erweitert werden kann. Gleichermaßen fordern wir nicht die sofortige Änderung sämtlicher kooperierender Systeme, nur weil an einer Stelle Fuzzy-Techniken eingesetzt werden, sondern zeigen mit Technologien wie dem Fuzzy-XML-Format einen sanften Migrationsweg auf.

19.2.8 KONSISTENZERHALTUNG

Die Bewahrung der Konsistenz (Seite 35) ist ein zentraler Aspekt in der Datenbank- und Informationssystementwicklung.

Das in Teil III entwickelte Modellierungs- und Verarbeitungsmodell hat daher einen genau definierten Konsistenzbegriff als Kern, der auf der Semantik einer komplexen Fuzzy-Beschreibung definiert ist. Wie oben ausgeführt, benötigen Fuzzy-Informationssysteme Revisionsoperatoren, die eine nicht-monotone Änderung der Wissensmenge ermöglichen. Entsprechend wurden die bekannten Operatoren Expansion, Revision und Kontraktion auf das entwickelte Modell übertragen. In dieser Übersetzung garantieren sie die Erhaltung eines geforderten Konsistenzgrades und ermöglichen so erst die immer wieder von Fuzzy-Informationssystemen erwartete Eigenschaft, robust mit inkonsistenten Informationen umzugehen.

Die Arbeit bietet damit eine neue Möglichkeit, Syntax und Semantik in einem Fuzzy-Modell zu vereinen und so die Vorteile der ausdrucksstärkeren Syntax mit der vielseitig verwendbaren Semantik von Fuzzy-Mengen zu vereinen.

19.3 Ausblick

Wie die vorliegende Arbeit demonstriert, ist die Einbringung eines neuen Konzeptes in eine so entwickelte und etablierte Welt wie die der Informationssysteme mit einiger Anstrengung verbunden. Dennoch muß die hier verfolgte Vorgehensweise, auf existierenden Konzepten und Technologien aufzubauen, beibehalten werden, sollen

Fuzzy-Informationssysteme jemals Erfolg haben. Die Erfahrung zeigt, daß kein Mangel an Einsatzgebieten für Imperfektion in Informationssystemen besteht. Grund für ihre mangelnde Verbreitung waren daher die fehlenden Verfahren und Technologien, die einen Einsatz in adäquater und effizienter Weise ermöglichen.

Die hier verfolgten Ansätze lassen sich auf benachbarte Gebiete fortschreiben: die Erweiterung von Entwicklungsprozessen und Entwicklungswerkzeugen, die Sammlung von Fuzzy-Algorithmen, die von der Notwendigkeit der Modifikation einzelner Fuzzy-Mengen abstrahieren, die Dokumentation von praktischen Erfahrungen in Form neuer Entwurfsmuster.

Im Vordergrund muß dabei die praktische Entwicklung von Fuzzy-Informationssystemen stehen: nur so läßt sich der Vorteil dieses Ansatzes *en masse* demonstrieren und die für eine Weiterentwicklung notwendige breite Unterstützung erzeugen. Denn die grundsätzliche Möglichkeit, Imperfektion auf irgendeine Weise in Informationssysteme einzubringen, steht inzwischen außer Frage.

Interessant ist auch ein weiterer, über diese technischen Fragen hinausgehender Aspekt: Bei der Zusammenarbeit mit Spezialisten aus anderen Fachdomänen, wie im Fall des Reengineering oder der Computerlinguistik, hat sich jedesmal gezeigt, daß eine gewisse Übergangs- und Gewöhnungszeit notwendig ist, bevor die Fuzzy-Technologie gewinnbringend eingesetzt werden kann. Die traditionelle Schule lehrt, Imperfektion nicht zuzulassen und verweist sie daher in scharfe Grenzen, für die man aber mit einem oft unverhältnismäßig hohen Aufwand bezahlen muß.

Selbst ein geschlossener, leicht anwendbarer Ansatz wie das in dieser Arbeit entwickelte Modell setzt daher ein geändertes Grundverständnis beim Anwender voraus: das Streben nach Präzision im Detail muß durch das Verständnis um die einer Domäne innewohnende Imperfektion und deren vorteilhafte Ausnutzung durch explizite Modellierung ersetzt werden.

Teil VII

Anhang

Anhang A

Wissensrevision in der Aussagenlogik

It'll take more than the likes of you to capture The Crimson Binome!
Capt. Capacitor, ReBoot v3.3.1: "Return of The Crimson Binome"

In diesem Anhang werden kurz die Postulate der Wissensrevision aus der Aussagenlogik dargestellt [Gär88], [Neb92].

A.1 Definitionen

Zuerst werden die Voraussetzungen definiert, die in den nachfolgenden Abschnitten verwendet werden.

Definition A.1.1 (Aussagenlogische Sprache) *Gegeben sei eine aussagenlogische Sprache $\mathcal{L}$ mit:*

Atomen $(a, b, c, \ldots \in \Sigma)$,

Wahr $\top$,

Falsch $\bot$,

Konnektoren $(\wedge, \vee, \neg, \rightarrow, \leftrightarrow)$,

aussagenlogischen Formeln $(x, y, \ldots \in \mathcal{L})$,

Folgerbarkeit $\vdash$,

Abschluß *Cn (Cons):* $Cn(K) \stackrel{\text{def}}{=} \{x | K \vdash x\}$,

Theorien (deduktiv abgeschlossene Satzmengen): $(F, G, \ldots) : Cn(F) = F$,

Basen (arbiträre Satzmengen): $(X, Y, \ldots)$.

Auf dieser Sprache gibt es drei verschiedene Operationen zur Änderung des Zustandes: Expansion ($+$), Kontraktion ($\overset{\circ}{-}$) und Revision ($\overset{\circ}{+}$).

A.2 Die AGM-Postulate für Expansion

Die von Alchourrón, Makinson und Gärdenfors aufgestellten Postulate für die Expansion sind:

GE1 $F + x$ ist wieder eine Theorie.

GE2 Die Expansion ist erfolgreich: $x \in F + x$.

GE3 Zunahme des Wissens: $F \subseteq F + x$.

GE4 Wenn die Information schon bekannt ist, ändert sich die Formel nicht:
Wenn $x \in F$, **dann** $F + x = F$.

GE5 Monotonie: **Wenn** $F \subseteq F'$, **dann** $F + x \subseteq F' + x$.

GE6 Für alle Formeln F und alle Propositionen x ist $F + x$ die kleinste Formel die GE1–GE5 erfüllt.

A.3 Die AGM-Postulate für Revision

Die folgenden sind die sechs Basispostulate, die jede Revisionsoperation erfüllen sollte:

GR1 Nach einer Revisionsoperation $\overset{\circ}{+}$ soll wieder ein epistemischer Zustand vorhanden sein: $F \overset{\circ}{+} x = Cn(F \overset{\circ}{+} x)$.

GR2 Jede Revision soll erfolgreich sein: $x \in F \overset{\circ}{+} x$.

GR3 Das Ergebnis der Revision soll nur Formeln enthalten, die aus F oder x oder beidem folgen: $F \overset{\circ}{+} x \subseteq F + x$.

GR4 Falls die neue Aussage mit dem alten epistemischen Zustand verträglich ist, soll x und F enthalten sein:
Wenn $\neg x \notin F$, **dann** $F + x \subseteq F \overset{\circ}{+} x$.

GR5 Inkonsistenzen sollen vermieden werden:
$F \overset{\circ}{+} x = Cn(\bot)$ **nur falls** $\vdash \neg x$.

GR5 Die syntaktische Form der neuen Aussage spielt keine Rolle:
Wenn $\vdash x \leftrightarrow y$, **dann** $F \overset{\circ}{+} x = F \overset{\circ}{+} y$.

Aus GR3 und GR4 folgt: Wenn $\neg x \notin F$, dann $F + x = F \overset{\circ}{+} x$.

Nun kommen noch zwei ergänzende Postulate:

GR7 Eine Revision durch eine Konjunktion soll enthalten sein in der Revision durch einen der Konjunktoren expandiert um den anderen Konjunktor:
$F \overset{\circ}{+} (x \wedge y) \subseteq (F \overset{\circ}{+} x) + y$.

GR8 Der Gegenfall zu GR7: **Wenn** $\neg y \notin F \overset{\circ}{+} x$, **dann** $(F \overset{\circ}{+} x) + y \subseteq F \overset{\circ}{+} (x \wedge y)$.

Aus GR7 und GR8 folgt: **Wenn** $\neg y \notin F \overset{\circ}{+} x$, **dann** $(F \overset{\circ}{+} x) + y = F \overset{\circ}{+} (x \wedge y)$.

A.4 Die AGM-Postulate für Kontraktion

Die folgenden sind die sechs Basispostulate, die jede Kontraktionsoperation erfüllen sollte:

GK1 Nach einer Kontraktionsoperation $F \overset{\circ}{-} x$ soll wieder ein epistemischer Zustand vorhanden sein.

GK2 Eine Kontraktion soll keine neuen Aussagen einbringen: $F \overset{\circ}{-} x \subseteq F$.

GK3 Wenn x nicht in F enthalten ist, soll auch nichts geändert werden:
Wenn $x \notin F$, **dann** $F \overset{\circ}{-} x = F$.

GK4 Wenn die zu entfernende Aussage nicht gültig ist, wird sie durch die Kontraktion entfernt:
Wenn $\nvdash x$, **dann** $x \notin F \overset{\circ}{-} x$.

GK5 Eine Kontraktion sollte sich rückgängig machen lassen:
Wenn $x \in F$, **dann** $F \subseteq (F \overset{\circ}{-} x) + x$.

GK6 Die syntaktische Form der zu entfernenden Aussage spielt keine Rolle:
Wenn $\vdash x \leftrightarrow y$, **dann** $F \overset{\circ}{-} x = F \overset{\circ}{-} y$.

Dies sind die Basis-Postulate der Kontraktion. Wie bei der Revision gibt es auch hier zwei Zusatz-Postulate:

GK7 Wenn eine Konjunktion entfernt wird, sollte weniger Information entfernt werden, als wenn parallel jeder der beiden Konjunktoren einzeln entfernt wird: $(F \overset{\circ}{-} x) \cap (F \overset{\circ}{-} y) \subseteq F \overset{\circ}{-} (x \wedge y)$

GK8 Der Gegenfall zu GK7: **Wenn** $x \notin F \overset{\circ}{-} (x \wedge y)$, **dann** $F \overset{\circ}{-} (x \wedge y) \subseteq F \overset{\circ}{-} x$.

A.5 Epistemische Verschanzung

Gärdenfors hat eine Reihe von Postulaten aufgestellt, die eine epistemische Verschanzung erfüllen muß.

GEV1 Transitivität: $\forall x, y, z : x \preceq_\epsilon y \wedge y \preceq_\epsilon z \Rightarrow x \preceq_\epsilon z$.

GEV2 Dominanz: $\forall x, y : x \vdash y \Rightarrow x \preceq_\epsilon y$.

GEV3 Konjunktivität: $\forall x, y \in F : (x \preceq_\epsilon x \wedge y) \vee (y \preceq_\epsilon x \wedge y)$.

GEV4 Minimum: **Wenn** $F \neq F_\perp$, **dann** $x \notin F$ genau dann wenn $x \preceq_\epsilon y$ für alle y.

GEV5 Maximum: **Wenn** $y \preceq_\gamma x$ für alle y, **dann** $\vdash x$.

A.6 Die Harper/Levi-Identitäten

Die Revision läßt sich anhand der Levi-Identität definieren:

$$F \overset{\circ}{+} x = Cn((F \overset{\circ}{-} \neg x) \cup \{x\})$$

Ebenso läßt sich die Kontraktion anhand der Harper-Identität definieren:

$$F \overset{\circ}{-} x = Cn((F \overset{\circ}{+} \neg x) \cap F)$$

Literaturverzeichnis

[AGM85] ALCHOURRÓN, C.E., P. GÄRDENFORS und D. MAKINSON: *On the Logic of Theory Change: Partial Meet Contraction and Revision Functions*. Journal of Symbolic Logic, 2(50):510–530, Juni 1985.

[AHV95] ABITEBOUL, SERGE, RICHARD HULL und VICTOR VIANU: *Foundations of Databases*. Addison-Wesley, 1995.

[AIS+77] ALEXANDER, CHRISTOPHER, SARA ISHIKAWA, MURRAY SILVERSTEIN, MAX JACOBSON, INGRID FIKSDAHL-KING und SHLOMO ANGEL: *A Pattern Language*. Oxford University Press, New York, 1977.

[All00] ALLAMARAJU, SUBRAHMANYAM ET. AL.: *Professional Java Server Programming*. Wrox, 2000.

[BCK98] BASS, LEN, PAUL CLEMENTS und RICK KAZMAN: *Software Architecture in Practice*. SEI Series in Software Engineering. Addison Wesley Longman, 1998.

[Bec00] BECK, KENT: *Extreme Programming explained*. Addison-Wesley, 2000.

[Ber95] BERGLER, SABINE: *From Lexical Semantics to Text Analysis*. In: SAINT-DIZIER, P. und E. VIEGAS (Herausgeber): *Computational Lexical Semantics*. Cambridge University Press, Cambridge, UK, 1995.

[Ber97] BERGLER, SABINE: *Towards Reliable Partial Anaphora Resolution*. In: *Operational Factors in Practical, Robust Anaphora Resolution for Unrestricted Texts*, Madrid, Spain, 1997. Proceedings of a workshop sponsored by the Association for Computational Linguistics.

[Bie97] BIEWER, BENNO: *Fuzzy-Methoden*. Springer-Verlag, 1997.

[BK95] BOSC, P. und J. KACPRZYK (Herausgeber): *Fuzziness in Database Management Systems*, Band 5 der Reihe *Studies in Fuzziness*. Physica-Verlag, 1995.

[BK96] BERGLER, SABINE und SONJA KNOLL: *Coreference Patterns in the Wall Street Journal*. In: PERCY, CAROL et al. [PML96].

[BLP94a] BORDOGNA, GLORIA, DARIO LUCARELLA und GABRIELLA PASI: *A Fuzzy Object Oriented Data Model*. In: *Proceedings of the Third IEEE Conference on Computational Intelligence*, Band 1 der Reihe *Fuzzy Systems*, Seiten 313–318. IEEE, Juni 1994.

[BLP94b] BORDOGNA, GLORIA, DARIO LUCARELLA und GABRIELLA PASI: *A Fuzzy Object Oriented Data Model*. In: 3^{rd} *International Conference on Fuzzy Systems*, Band 1, Seiten 313–318. IEEE, Orlando, Florida, 26.–29. Juni 1994.

[Bol79] BOLLOBÁS, BÉLA: *Graph Theory*. Springer-Verlag, 1979.

[Bos94] BOSS, BIRGIT: *An index based on superimposed coding for a fuzzy object oriented database system*. In: *Proceedings of the First International Joint Conference of the North American Fuzzy Information Processing Society Biannual Conference. Industrial Fuzzy Control and Intelligent Systems Conference, and the NASA Joint Technology Workshop on Neural Networks and Fuzzy Logic, NAFIPS/IFIS/NASA*, Seiten 289–290, 1994.

[Bos96] BOSS, BIRGIT: *Fuzzy-Techniken in objektorientierten Datenbanksystemen zur Unterstützung von Entwurfsprozessen*. Nummer 4 in *Dissertationen zu Datenbanken und Informationssystemen*. Infix, 1996.

[BP93a] BOSC, PATRICK und HENRI PRADE: *An introduction to fuzzy set and possibility theory-based approaches to the treatment of uncertainty and imprecision in data base management systems*. In: 2^{nd} *UMIS Workshop (Uncertainty Management in Information Systems: from Needs to Solutions)*. Catalina, California, 2.–5. April 1993.

[BP93b] BOSC, PATRICK und HENRI PRADE: *An Introduction to Fuzzy Set and Possibility Theory-Based Approaches to the Treatment of Uncertainty and Imprecision in Data Base Management Systems*. Prepared for the Second UMIS Workshop, 1993.

[Buc95] BUCKLEY, JAMES P.: *A Hierarchical Clustering Strategy for Very Large Fuzzy Databases*. In: *IEEE International Conference on Systems, Man and Cybernetics*, Band 4, Seiten 3573–3578. IEEE, 1995.

[Bud97] BUDD, TIMOTHY: *An Introduction To Object-Oriented Programming*. Addison Wesley Longman, zweite Auflage, 1997.

[Cal97] CALUWE, RITA DE (Herausgeber): *Fuzzy and Uncertain Object-Oriented Databases*, Band 13 der Reihe *Advances in Fuzzy Systems — Applications and Theory*. World Scientific, 1997.

[CCV97] CROSS, VALERIE, RITA DE CALUWE und NANCY VANGYSEGHEM: *A Perspective from the Fuzzy Object Data Management Group (FODMG)*. In: *Proceedings of the Sixth IEEE International Conference on Fuzzy Systems*, Band 2, Seiten 721–728. IEEE, 1997.

[Che98] CHEN, GUOQING: *Fuzzy Logic in Data Modelling*. Kluwer Academic Publishers, 1998.

[Chr97] CHRISTOPH, ALEXANDER: *Realisierung einer Multi-Client Datenbankanbindung mit Objektbustechnologie und Java*. Studienarbeit, Institut für Programmstrukturen und Datenorganisation (IPD), Fakultät für Informatik, Universität Karlsruhe, Juni 1997.

[CMR99] CHAUDHRY, NAUMAN, JAMES MOYNE und ELKE A. RUNDENSTEINER: *An extended database design methodology for uncertain data management*. Information Sciences, (121):83–112, 1999.

[Cou90] COURCELLE, BRUNO: *Graph Rewriting: An Algebraic and Logic Approach*. In: LEEUWEN, JAN VAN [vL90], Kapitel 5, Seiten 193–242.

[Cox99] COX, EARL: *The Fuzzy Systems Handbook*. AP Professional, 2. Auflage, 1999.

[Cro95] CROSS, VALERIE: *Fuzzy extensions to the object model*. In: *IEEE International Conference on Systems, Man and Cybernetics 1995. Intelligent Systems for the 21st Century*, Band 4, Seiten 3630–3635. IEEE, 1995.

[Cro96] CROSS, VALERIE: *Towards a unifying framework for a fuzzy object model*. In: *Proceedings of the Fifth IEEE International Conference on Fuzzy Systems*, Band 1, Seiten 85–92. IEEE, 1996.

[Doy91] DOYLE, JON: *Rational Belief Revision (Preliminary Report)*. In: ALLEN, J.A., R. FIKES und E. SANDEWALL (Herausgeber): *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR '91)*. Morgan Kaufmann, April 1991.

[Doy92a] DOYLE, JON: *Reason Maintenance and Belief Revision: Foundations vs. Coherence Theories*, Seiten 29–51. Cambridge University Press, Cambridge, UK, 1992.

[Doy92b] DOYLE, JON: *Reason Maintenance and Belief Revision: Foundations vs. Coherence Theories*. In: GÄRDENFORS, P. (Herausgeber): *Belief Revision*, Seiten 29–51. Cambridge University Press, Cambridge, UK, 1992.

[DP91] DUBOIS, DIDIER und HENRI PRADE: *Epistemic Entrenchment and Possibilistic Logic*. Artificial Intelligence, (50):223–239, 1991.

[DP92] DUBOIS, DIDIER und HENRI PRADE: *Fuzzy Sets and Possibility Theorie: Some Applications to Interference and Decision Processes*. In: NENSCH, B. (Herausgeber): *Fuzzy Logic — Theorie und Praxis*, Seiten 67–83. 1992. Dortmunder Fuzzy Tage, 10. Juni 1992, Dortmund.

[DP93] DUBOIS, DIDIER und HENRI PRADE: *Belief revision and updates in numerical formalisms*. In: *13th International Joint Conference on Artificial Intelligence (IJCAI '93)*, Band 1, Seiten 620–625. IJCAI, Chambéry, France, 28. August–3. September 1993.

[DP94] DUBOIS, DIDIER und HENRI PRADE: *A Survey of Belief Revision and Updating Rules in Various Uncertainty Models*. International Journal of Intelligent Systems, 9(1):61–100, January 1994.

[DvdR92] DIGNUM, F. und R.P. VAN DE RIET: *Addition and removal of information for a knowledge base with incomplete information*. Data&Knowledge Engineering, (8):293–307, 1992.

[Edw99] EDWARDS, JERI: *3-Tier Client/Server At Work*. John Wiley & Sons, 1999. Revised Edition.

[ERZD95] EVERSHEIM, WALTER, AXEL ROGGATZ, HANS-JÜRGEN ZIMMERMANN und THOMAS DERICHS: *Kurze Produktentwicklungszeiten durch Nutzung unsicherer Informationen*. it+ti Informationstechnik und Technische Informatik, 37(5), 1995.

[Fai99] FAIRLIE, BORIS: *Zur Verarbeitung von Berechnungserfahrung und Informationsunschärfe bei der rechnerunterstützten Produktauswahl*. Nummer 311 in *Fortschritt-Berichte VDI*. VDI-Verlag, Düsseldorf, 1999.

[Fel98] FELLBAUM, CHRISTIANE (Herausgeber): *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[FFCM99] FLANAGAN, DAVID, JIM FARLEY, WILLIAM CRAWFORD und KRIS MAGNUSSON: *Java Enterprise in a Nutshell*. O'Reilly, 1999.

[FS00] FOWLER, MARTIN und KENDALL SCOTT: *UML Distilled*. Object Technology Series. Addison-Wesley, 2. Auflage, 2000.

[Gär88] GÄRDENFORS, PETER: *Knowledge in Flux*. MIT Press, 1988.

[Gär92] GÄRDENFORS, PETER: *Belief Revision: An Introduction*, Seiten 1–28. Cambridge University Press, Cambridge, UK, 1992.

[GCV93] GYSEGHEM, NANCY VAN, RITA DE CALUWE und RIA VANDENBERGHE: *UFO: Uncertainty and Fuzziness in an Object-oriented model*. In: *Second IEEE International Conference on Fuzzy Systems*, Band 2, Seiten 773–778. IEEE, 1993.

[GHJV95] GAMMA, ERICH, RICHARD HELM, RALPH JOHNSON und JOHN VLISSIDES: *Design Patterns*. Addison-Wesley, 1995.

[GRS00] GÖRZ, G., C.-R. ROLLINGER und J. SCHNEEBERGER (Herausgeber): *Handbuch der Künstlichen Intelligenz*. Oldenbourg Verlag, 3. Auflage, 2000.

[Hau98] HAUKE, WOLFGANG: *Fuzzy-Modelle in der Unternehmensplanung*, Band 68 der Reihe *Physica-Schriften zur Betriebswirtschaft*. Physica-Verlag, 1998.

[Hel95] HELMER, SVEN: *Entwicklung von Kostenmodellen für eine auf Superimposed Coding basierende Zugriffsstruktur für Fuzzy-Datenbanken*. Diplomarbeit, Forschungszentrum Informatik (FZI), Karlsruhe, Februar 1995.

[HNS00] HOFMEISTER, CHRISTINE, ROBERT NORD und DILIP SONI: *Applied Software Architecture*. The Addison-Wesley object technology series. Addison-Wesley, 2000.

[Ill00] ILLIAD: *Evil Geniuses In A Nutshell*. O'Reilly, 2000. A User Friendly Guide To World Domination.

[JW00] JAHNKE, JENS H. und ANDREW WALENSTEIN: *Reverse Engineering Tools as Media for Imperfect Knowledge*. In: *Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE'00)*, Seiten 22–31. IEEE, 2000.

[KF88] KLIR, GEORGE J. und TINA A. FOLGER: *Fuzzy Sets, Uncertainty, and Information*. Prentice-Hall, 1988.

[KGK95] KRUSE, RUDOLF, JÖRG GEBHARDT und FRANK KLAWONN: *Fuzzy-Systeme*. Leitfäden der Informatik. B.G. Teubner, Stuttgart, 2. Auflage, 1995.

[KGP94] KRUSE, RUDOLF, JÖRG GEBHARDT und RAINER PALM (Herausgeber): *Fuzzy Systems in Computer Science*. Verlag Vieweg, 1994.

[Köl00] KÖLSCH, ULRIKE: *Methodische Integration und Migration von Informationssystemen in objekt-orientierte Umgebungen*. Nummer 68 in *Dissertationen zu Datenbanken und Informationssystemen (DISDBIS)*. infix, 2000.

[KLM+97] KICZALES, GREGOR, JOHN LAMPING, ANURAG MENDHEKAR, CHRIS MAEDA, CRISTINA VIDEIRA LOPES, JEAN-MARC LOINGTIER und JOHN IRWIN: *Aspect-Oriented Programming*. In: *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, LNCS 1241, Finnland, Juni 1997. Springer-Verlag.

[KM91] KATSUNO, HIROFUMI und ALBERTO O. MENDELZON: *On the Difference between Updating a Knowledge Base and Revising it*. In: *KR*, Seiten 387–394. Cambridge, Massachusetts, 22.–25. April 1991.

[Kos93] KOSKO, BART: *Fuzzy Thinking*. Hyperion, 1993.

[KST02] KAZAKOS, WASSILIOS, ANDREAS SCHMIDT und PETER TOMCZYK: *Datenbanken und XML*. Springer-Verlag, 2002.

[KW98] KLIR, GEORGE J. und MARK J. WIERMAN: *Uncertainty-Based Information*, Band 15 der Reihe *Studies in Fuzziness and Soft Computing*. Physica-Verlag, 1998.

[LA94] LAW, WILLIAM S. und ERIK K. ANTONSSON: *Implementing the Method of Imprecision: An Engineering Design Example*. In: *3rd International Conference on Fuzzy Systems*, Band 1, Seiten 358–363. IEEE, Orlando, Florida, 26.–29. Juni 1994.

[Lan97] LANG, HANS-PETER: *Entwicklung einer Fuzzy-Komponente zur Verwaltung und Verarbeitung unscharfer Informationen in objektorientierten Systemen*. Studienarbeit, Institut für Programmstrukturen und Datenorganisation (IPD), Fakultät für Informatik, Universität Karlsruhe, 1997.

[LGL98] LASCIO, LUIGI DI, ANTONIO GISOLFI und VINCENZO LOIA: *Uncertainty processing in user-modeling activity*. Information Sciences, (106):25–47, 1998.

[LKK93] LOCKEMANN, PETER C., GERHARD KRÜGER und HEIKO KRUMM: *Telekommunikation und Datenhaltung*. Addison Wesley Longman, 1993.

[LL95] LANG, S.M. und PETER C. LOCKEMANN: *Datenbankeinsatz*. Springer-Verlag, 1995.

[LL00] LOCKEMANN, PETER C. und GERGELY LUKÁCS: *Imperfection and the Human Component: Adding Robustness to Global Information Systems*. In: BRINKKEMPER, SJAAK, EVA LINDENCRONA und ARNE SOLVBERG (Herausgeber): *Information Systems Engineering: State of the Art and Research Themes*, Seiten 3–14. Springer Verlag, 2000.

[Luk99] LUKACS, GERGELY: *Imperfect Data in Information Systems: Overview and Case Study*. Universität Karlsruhe, IPD, März 1999.

[Luk00] LUKACS, GERGELY: *Towards an Interactive Query Environment for a Multi-Attribute Decision Model with Imprecise Data and Vague Query Conditions*. In: *Proceedings of 11th International Workshop on Database and Expert Systems Applications (DEXA 2000)*, Seiten 708–712, Greenwich, London, UK, 4.–8. September 2000. IEEE Computer Society Press.

[LX98] LEE, JONATHAN und NIEN-LIN XUE: *FOOM: A Fuzzy Object-Oriented Modeling for Imprecise Requirements*. In: *1998 Conference of the North American Fuzzy Information Processing Society (NAFIPS)*, Seiten 345–349, August 1998.

[LXHY99] LEE, JONATHAN, NIEN-LIN XUE, KUO-HSUN HSU und STEPHEN J. YANG: *Modeling imprecise requirements with fuzzy objects*. Information Sciences, (118):101–119, 1999.

[Men00] MENZEL, W.: *Unscharfe Mengen*. Universität Karlsruhe, Fakultät für Informatik, Sommer 2000. Vorlesungsskript.

[MH00] MONSON-HAEFEL, RICHARD: *Enterprise JavaBeans*. O'Reilly, 2. Auflage, 2000.

[Min75] MINSKY, MARVIN: *A Framework for Representing Knowledge*. In: WINSTON, P. (Herausgeber): *The Psychology of Computer Vision*, Seiten 211–277. MacGraw-Hill, New York, 1975.

[MS97] MOTRO, AMIHAI und PHILIPPE SMETS (Herausgeber): *Uncertainty Management In Information Systems*. Kluwer Academic Publishers, 1997.

[MT94] MÜLLER, K. und M. THÄRINGEN: *Applications of Fuzzy Hierarchies and Fuzzy MADM Methods to Innovative System Design*. In: *3rd International Conference on Fuzzy Systems*, Band 1, Seiten 364–367. IEEE, Orlando, Florida, 26.–29. Juni 1994.

[Neb89] NEBEL, BERNHARD: *A Knowledge Level Analysis of Belief Revision*. In: BRACHMANN, R., H.J. LEVESQUE und R. REITER (Herausgeber): *Principles of Knowledge Representation and Reasoning: Proceedings of the 1st International Conference*, Seiten 301–311. Morgan Kaufmann, Mai 1989.

[Neb90] NEBEL, BERNHARD: *Reasoning and Revision in Hybrid Representation Systems*. Lecture Notes in Artificial Intelligence. Springer-Verlag, 1990.

[Neb91] NEBEL, BERNHARD: *Belief Revision and Default Resoning: Syntax-Based Approaches*. In: ALLEN, J.A., R. FIKES und E. SANDEWALL (Herausgeber): *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR '91)*, Seiten 417–428. Morgan Kaufmann, April 1991.

[Neb92] NEBEL, BERNHARD: *Syntax-Based Approaches to Belief Revision*. In: GÄRDENFORS, P. (Herausgeber): *Belief Revision*, Seiten 52–88. Cambridge University Press, Cambridge, UK, 1992.

[Neb93] NEBEL, BERNHARD: *Wissensrevision und Nichtmonotone Logiken*. Vortrags-Folien, 27. Februar–6. März 1993. KI Frühjahrsschule (KIFS '93), Günne/Möhnesee.

[OA94] OTTO, KEVIN N. und ERIK K. ANTONSSON: *Modeling Imprecision in Product Design*. In: 3rd *International Conference on Fuzzy Systems*, Band 1, Seiten 346–351. IEEE, Orlando, Florida, 26.–29. Juni 1994.

[OHE99] ORFALI, ROBERT, DAN HARKEY und JERI EDWARDS: *Client/Server Survival Guide*. John Wiley & Sons, 3. Auflage, 1999.

[Ped96] PEDRYCZ, WITOLD (Herausgeber): *Fuzzy Modelling Paradigms and Practice*. International Series in Intelligent Technologies. Kluwer Academic Publishers, 1996.

[Pet96] PETRY, FREDERICK E.: *Fuzzy Databases Principles and Applications*. International Series in Intelligent Technologies. Kluwer Academic Publishers, 1996.

[PML96] PERCY, CAROL, CHARLES MEYER und IAN LANCASHIRE (Herausgeber): *Synchronic corpus linguistics. Papers from the Sixteenth International Conference on English Language Research on Computerized Corpora*, Language and Computers series, Amsterdam, 1996. Rodopi Verlag.

[RDK98] REZNIK, LEONID, VLADIMIR DIMITROV und JANUSZ KACPRZYK (Herausgeber): *Fuzzy Systems Design: Social and Engineering Applications*, Band 17 der Reihe *Studies in Fuzziness and Soft Computing*. Physica-Verlag, 1998.

[RMB01] RUH, WILLIAM A., FRANCIS X. MAGINNIS und WILLIAM J. BROWN: *Enterprise Application Integration*. Wiley Computer Publishing, 2001.

[Rot92] ROTT, HANS: *On The Logic of Theory Change: More Maps Between Different Kinds of Contraction Function*. In: GÄRDENFORS, P. (Herausgeber): *Belief Revision*, Seiten 122–141. Cambridge University Press, Cambridge, UK, 1992.

[Rot94] ROTT, HANS: *Coherent choice and epistemic entrenchment (Preliminary report)*. In: NEBEL, BERNHARD und LEONIE DRESCHLER-FISCHER (Herausgeber): *KI-94: Advances in Artificial Intelligence*, Nummer 861 in *Lecture Notes in Artificial Intelligence*, Seiten 284–295. Springer-Verlag, 1994.

[Sch87] SCHÖNING, UWE: *Logik für Informatiker*. BI Wissenschaftsverlag, 1987.

[Sch95] SCHULZ, BENEDIKT: *Eine graphische Benutzeroberfläche für ein fuzzy Entwurfsdatenbanksystem*. Studienarbeit, Forschungszentrum Informatik (FZI), Karlsruhe., 1995.

[SSJt02] SINGH, INDERJEET, BETH STEARNS, MARK JOHNSON und THE ENTERPRISE TEAM: *Designing Enterprise Applications with the J2EE Platform, Second Edition*. Addison-Wesley, 2002. `http://java.sun.com/blueprints/guidelines/designing_enterprise_application_2e/index.html`.

[SUY99] SAYGIN, YÜCEL, ÖZGÜR ULUSOY und ADNAN YAZICI: *Dealing with fuzziness in active mobile database systems*. Information Sciences, (120):23–44, 1999.

[TCY93] TSENG, FRANK S.C., ARBEE L.P. CHEN und WEI-PANG YANG: *Refining imprecise data by integrity contraints*. Data&Knowledge Engineering, (11):299–316, 1993.

[TKS91] TANAKA, KATSUMI, SUSUMU KOBAYASHI und TOMOMI SAKANOUE: *Uncertainty Management in Object-Oriented Database Systems*. In: *Dexa*, Seiten 251–256. Berlin, 1991.

[UIHT98] UMANO, MOTOHIDE, TAKAFUMI IMADA, ITSUO HATONO und HIROYUKI TAMURA: *Fuzzy Object-Oriented Databases and Implementation of Its SQL-type Data Manipulation Language*. In: *The 1998 IEEE International Conference on Fuzzy Systems Proceedings*, Band 2, Seiten 1344–1349. IEEE, 1998. IEEE World Congress on Computational Intelligence.

[vL90] LEEUWEN, JAN VAN (Herausgeber): *Handbook of Theoretical Computer Science*, Band B: Formal Models and Semantics. The MIT Press/Elsevier, 1990.

[Wei96] WEISBROD, JOACHIM: *Unscharfes Schließen*. Nummer 117 in *Dissertationen zur künstlichen Intelligenz*. Infix, Sankt Augustin, 1996.

[Wei98] WEISBROD, JOACHIM: *A new approach to fuzzy reasoning*. Soft Computing, 2, Juni 1998.

[Wit95] WITTE, RENÉ: *Wissensrevision in Fuzzy-Entwurfsdatenbanken — Entwurf und Realisierung —*. Diplomarbeit, Forschungszentrum Informatik (FZI), Karlsruhe, 1995.

[Wit02] WITTE, RENÉ: *Fuzzy Belief Revision*. In: *9th Intl. Workshop on Non-Monotonic Reasoning (NMR'02)*, Seiten 311–320, Toulouse, France, 19.-21. April 2002.

[YC99] YAZICI, ADNAN und DOGAN CIBICELI: *An access structure for similarity-based fuzzy databases*. Information Sciences, (115):137–163, 1999.

[YG99] YAZICI, ADNAN und ROY GEORGE: *Fuzzy Database Modeling*, Band 26 der Reihe *Studies in Fuzziness and Soft Computing*. Physica-Verlag, 1999.

[YGA98] YAZICI, ADNAN, ROY GEORGE und DEMET AKSOY: *Design and implementation issues in the fuzzy object-oriented data model*. Information Sciences, (108):241–260, 1998.

[Yi98] YAZICI, ADNAN und ALI ÇINAR: *Conceptual Design of Fuzzy Object-Oriented Databases*. In: *Proceedings of the Second International Conference on Knowledge-Based Intelligent Electronic Systems*, Band 2, Seiten 229–305. IEEE, April 1998.

[YOTN87] YAGER, R.R., S. OVCHINNIKOV, R.M. TONG und H.T. NGUYEN (Herausgeber): *Fuzzy Sets and Applications: Selected Papers by L.A. Zadeh*. Wiley&Sons, 1987.

[Zad65] ZADEH, L.A.: *Fuzzy Sets*. In: YAGER, R.R. et al. [YOTN87], Seiten 29–44. Ursprünglich veröffentlicht in *Information and Control*, Vol. 8, New York: Academic Press, 1965, Seiten 338–353.

[Zad78] ZADEH, L.A.: PRUF — *A Meaning Representation Language for Natural Languages*. In: YAGER, R.R. et al. [YOTN87], Seiten 499–568. Ursprünglich veröffentlicht in *International Journal for Man-Machine Studies*, Vol. 10, London: Academic Press, 1978, Seiten 395–460.

[ZCF+97] ZANIOLO, CARLO, STEFANO CERI, CHRISTOS FALOUTSOS, RICHARD T. SNODGRASS, V. S. SUBRAHMANIAN und ROBERTO ZICARI: *Advanced Database Systems*. Morgan Kaufmann Publishers, Inc., 1997.

[ZS94] ZIMMERMANN, H.-J. und H.-J. SEBASTIAN: *Fuzzy Design — Integration of Fuzzy Theory with Knowledge-Based System-Design*. In: *3rd International Conference on Fuzzy Systems*, Band 1, Seiten 352–357. IEEE, Orlando, Florida, 26.–29. Juni 1994.

Stichwortverzeichnis