Westfälische Wilhelms-Universität Münster

Institut für Wirtschaftsinformatik

Lehrstuhl für Wirtschaftsinformatik und Informationsmanagement

Diplomhausarbeit

im Fachgebiet

Wirtschaftsinformatik

Entwurf und Realisierung einer auf Fuzzy-Logik basierenden Entscheidungskomponente zur Integration in eine Workflow-Entwicklungsumgebung

Themensteller:	**Prof. Dr. Jörg Becker**
Betreuer:	**Dipl.-Wi.-Ing. Markus Rehfeldt** **Dipl.-Wi.-Ing. Klaus Turowski**
vorgelegt dem:	**Prüfungsamt für wirtschaftswissenschaftliche Prüfungen der Westfälischen Wilhelms-Universität**
Ausgabetermin:	**26. Juni 1996**
Abgabetermin:	**18. September 1996**
vorgelegt von:	**Marc Schönefeld** **Elisabethstr. 39** **59320 Ennigerloh**

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

μ_A, μ_B, μ_C	Zugehörigkeitsfunktionen $\mu_A(x)$, $\mu_B(x)$, $\mu_C(x)$
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CAM	Computer Aided Manufacturing
CAP	Computer Aided Planning
CAQ	Computer Aided Quality Ensurance
CAx	Alle „Computer Aided“ - Technologien
CIM	Computer Integrated Manufacturing
DLZ	Durchlaufzeit
DREM	Development and Runtime Environment Münster
EWP	Erweiterungsprinzip
FES	Fuzzy-Experten-System
FLE	Fuzzy-Logik-Entscheidungskomponente
FUREGA	Fuzzy Rules Extraction by a Genetic Algorithm
KKV	Komparativer Konkurrenz-Vorteil
KMU	Kleine und Mittelständische Unternehmen
MFC	Microsoft Foundation Classes
OMP	Orthogonal Matching Pursuit
PPS	Produktionsplanung und -steuerung
ROI	Return On Investment
WFMS	Workflow-Management-System
eEPK	erweiterte ereignisgesteuerte Prozeßkette

Definitionsverzeichnis

Symbolverzeichnis

$\Re$	Menge der reellen Zahlen
$\Re_0^+$	Menge der positiven reellen Zahlen (incl. 0)
$\boldsymbol{I}_A(x)$	Indikator-Funktion
$\varnothing$	leere Menge
$\langle a;b\rangle$	offenes Intervall
$[a;b\rangle$	nach oben offenes Intervall
$\langle a;b]$	nach unten offenes Intervall
$[a;b]$	geschlossenes Intervall
$\forall$	für alle (Allquantor)
$\exists$	es exisitiert (Existenzquantor)

1. Einleitung

1.1 Motivation, Problemstellung

Die Internationalisierung des Wettbewerbs führt zu Veränderungen in der Produkt- und der Kundenstruktur vieler Industrieunternehmen. Daher suchen diese nach Möglichkeiten, den Wirkungsgrad ihrer internen Ablauforganisation[1] diesem Strukturwandel gemäß zu optimieren. Die Beibehaltung der Wettbewerbsfähigkeit geht mit höherer Kundennähe und einer flexiblen, d. h. der Nachfragesituation entsprechend reagierenden, Auftragsabwicklung einher. Es hat sich insbesondere im Fertigungsbereich ein gleitender Wandel[2] in der Bewertung der beteiligten Haupt- und Ersatzziele der Industrieunternehmen vollzogen. Der dispositive Faktor eines Unternehmens orientiert seine Strategien und Handlungen primär am vorhandenen Zielsystem, welches sich aus dem Leitbild, dem Umfeld und den Unternehmenspotentialen[3] ableitet. Die Vorteilhaftigkeit von Alternativen läßt sich aufgrund der Wechselwirkungen mit Markt- und Kundenverhalten jedoch nicht allein durch Analyse des Erreichungsgrades von Unternehmenszielen beurteilen, sondern orientiert sich an internen, technischen Ersatzzielen, die durch die Unternehmenspotentiale[4] bestimmt sind. Als bestimmende Faktoren haben kundenorientierte Aspekte wie Termintreue, Flexibilität und Materialflußorientierung die Orientierung an der maximalen Kapazitätsauslastung abgelöst. Ebenso erfolgte, induziert durch die Fortschritte in der Soft- und Hardwaretechnologie, ein Wandel von der periodenorientierten[5] zur ereignisorientierten Planung. Als Hauptvorteil ist hier die verbesserte Reagibilität gegenüber unvorhergesehenen Ereignissen, beispielsweise bei Maschinenausfällen, zu nennen. Die einem Aufgabenträger bei der konventionellen Auftragsbearbeitung zur Verfügung stehenden Methoden sind verursachen durch die Replikation von Daten und durch die manuelle Vorgangsbearbeitung inkonsistenzbedingter Bearbeitungsfehler. Zeitpotentiale, die durch rechnergestützte Parallelisierung[6] von Vorgängen entstehen, bleiben wegen der Orientierung am sequentiellen Prinzip der Arbeitsteilung, oft ungenutzt. Die Effizienz einer zeitgemäßen und zielkonformen Auftragsabwicklung mißt sich nun - verglichen mit der konventionellen Vorgehensweise - an Anforderungen, wie beispielsweise der Synchronisation der vorher disjunkten Tätigkeitsbereiche und der Dauer sowie der Transparenz des Auftragsdurchlaufs. Aus vielen betrieblichen Prozessen[7] lassen sich durch geeignete Modellierung Potentiale für

1 Vgl. Rautentrauch (1993), S. 41.
2 Vgl. Rembold et al. (1994), S. 21ff.
3 Vgl. Adam (1993), S.118.
4 Vgl. Adam (1993), S. 118f.
5 Vgl. Kurbel (1993), S. 119.
6 Vgl. Rembold, et al. (1994), S. 21.
7 Vgl. Turowski (1994), S. 114.

die parallele Bearbeitung ermitteln. Zur Unterstützung dieser Ziele setzen viele zeitgemäße Unternehmen auf die Integration von Instrumenten aus der Informationstechnologie. Herkömmlich wird die industrielle Auftragsabwicklung durch den Einsatz von Produktionsplanungs- und -steuerungssystemen (PPS-Systemen) unterstützt. Der Nutzen dieser Systeme muß im Rahmen des Business Process Reengineering neu hinterfragt werden. Ein Hilfsmittel zur Aufdeckung von Optimierungspotentialen im PPS-Bereich sind Entwicklungswerkzeuge[8] für das Workflow-Management[9].

Die parallele Ausführung von Vorgängen kann durch die gezielte Anwendung von Replikation sowie durch die Berücksichtigung von Segmentierungspotentialen oder durch die frühzeitige Auswertung von vagen und unscharfen Informationen vorgenommen werden.[10] Als Information soll im folgenden entscheidungsorientiertes Wissen[11] verstanden werden.

Der Geschäftsprozeß[12] zwischen Vertrieb, Planung, Fertigung und Versand wird in der Industrie als Auftragsabwicklung bezeichnet. Die in einen solchen Prozeß einfließenden Informationen haben oft vagen Charakter[13]. Sie können aber als solche in ein konventionelles auftragsbegleitendes System nicht berücksichtigt werden, sondern erst, wenn sie sich bereits konkretisiert haben und als sicher gelten. Somit wird ein potentieller Zeitvorsprung zur Erringung eines komparativen Konkurrenz-Vorteils[14] (KKV) eingebüßt. Unscharfe Daten werden oft künstlich zu scharfen Werten verdichtet, womit als Konsequenz wichtige Meta-Informationen wie der Unschärfegrad oder die Spannweite der Information verlorengehen. Der Nutzen von auftragsbegleitenden PPS-Systemen als operative Planungs- und Steuerungsinstrumente bleibt somit gering[15].

Als Workflow wird das ausführbare Abbild[16] eines Geschäftsprozesses bezeichnet. Es liegt der Gedanke nah, Informationen im auftragsbegleitenden Workflow-Management-System[17] bereits dann zu erfassen und zu verarbeiten, wenn diese nur vage vorliegen, da nachgeschaltete Bereiche bereits Planungen starten können[18]. Nimmt ein Informationssystem[19] derartig unscharfe Daten an, so sollte gewährleistet sein, daß der Grad der Unschärfe der Informationen sich auch in den Ergebnissen, den von diesem System

8 Vgl. Rehfeldt, Turowski (Workflow) (1995), S. 367.
9 Vgl. Jablonski, S. 13.
10 Vgl. Becker et al. (Planungsobjekte) (1996), S. 11.
11 Vgl. Mag (1977), S. 5.
12 Vgl. Scheer (1994), S. 10.
13 Vgl. Turowski (1994), S. 114.
14 Vgl. Adam (1993), S. 120.
15 Vgl. Rehfeldt, Turowski (Fuzzy-Ansätze) (1994), S. 3.
16 Vgl. Jablonski (1995), S. 16.
17 Vgl. Scheer (1994), S. 712f., auch Jablonski (1995), S. 14.
18 Vgl. Eversheim et al. (1995), S. 49.
19 Vgl. Mechler (1993), S. 15.

generierten Entscheidungsvorschlägen, wiederfindet. Eine Planung, die mit einem solchen explizit propagierten Unschärfegrad ausgestattet ist, bietet somit die Chance einer - durch erhöhte Transparenz und gesteigerte Sicherheit - verbesserten Qualität, wodurch der Erfüllungsgrad der Unternehmensziele erhöht wird.

1.2 Umfang und Thema dieser Arbeit

Diese Arbeit gliedert sich in einen theoretischen und einen praktischen Teil, wobei innerhalb des ersten Teils zunächst die generelle Bedeutung von Unschärfe fokussiert wird, um anschließend die Modellierung von Unschärfe durch Fuzzy-Sets[20] (d. h. unscharfe Mengen) und den Einsatz von Methoden aus dem Bereich der Fuzzy-Set-Theorie, zu motivieren. Es werden die Auswirkungen und Nutzenpotentiale des Einsatzes von Fuzzy-Logik-basierten Datentypen bei der Modellierung von Geschäftsprozessen herausgearbeitet. Ein gesonderter Abschnitt wendet sich den sprachlichen Gestaltungsfreiräumen zur Formulierung von akquiriertem Expertenwissen zu.

Der praktische Teil dieser Arbeit findet sich in den, im Anhang dieser Arbeit aufgeführten, Quelltexten wieder. Das Ergebnis ist die Erweiterung einer Workflow-Entwicklungsumgebung um eine auf Fuzzy-Logik basierenden Entscheidungskomponente, welche in der Programmiersprache C++ implementiert[21] wurde. Die Entwicklung der Entscheidungskomponente unterteilt sich in die Bereiche Analyse, Design und Implementation. Der Bereich der Analyse findet sich bereits im theoretischen Grundlagenteil zur Fuzzy-Set-Theorie wieder, die Beschreibung des Designs erfolgt durch die Vorstellung der als Objekttypen identifizierten Klassen. In der Beschreibung der Implementation finden sich Details zur softwaretechnischen Realisierung der auf unscharfen Mengen definierten Operatoren und Operationen.

Die betriebswirtschaftliche Relevanz der Ergebnisse dieser Arbeit wird anhand des Anwendungsbeispiels aus der industriellen Auftragsabwicklung illustriert. Hierbei werden insbesondere die für die Praxis relevanten Nutzenpotentiale, die sich durch die Fuzzifikation[22] des Verhaltens von Planungsobjekten[23] und der Struktur von Informationsobjekten bieten, aufgezeigt.

---

20 Vgl. Zadeh (1965), S. 339.
21 Vgl. Stroustrup (1992), S. 7.
22 Vgl. Schmidt et al. (1993), S. 82
23 Vgl. Becker et al. (1995), S. 33.

2. Entscheidungen bei Unschärfe

In diesem Kapitel wird zunächst der Begriff der Unschärfe erläutert und die bei unscharfen Entscheidungssituationen auftretenden Problemstellungen[24] beleuchtet. Nach der Vorstellung grundlegender Definitionen der Fuzzy-Set-Theorie, der Theorie der unscharfen Mengen, erfolgt die Darstellung von Methoden zur Verknüpfung von unscharfen Informationen, insbesondere linguistisch beschriebener Regeln und Fakten.

2.1 Struktur eines Entscheidungsprozesses

Zur Sicherung einer erfolgsorientierten Unternehmensführung ist das Fällen rationaler Entscheidungen, und somit eine Planung der Entscheidungsfindung, notwendig. Um eine komplexe Entscheidungssituation in eine für den Aufgabenträger transparente Form zu überführen, wird ein Modell (Bild 2.1) erstellt.

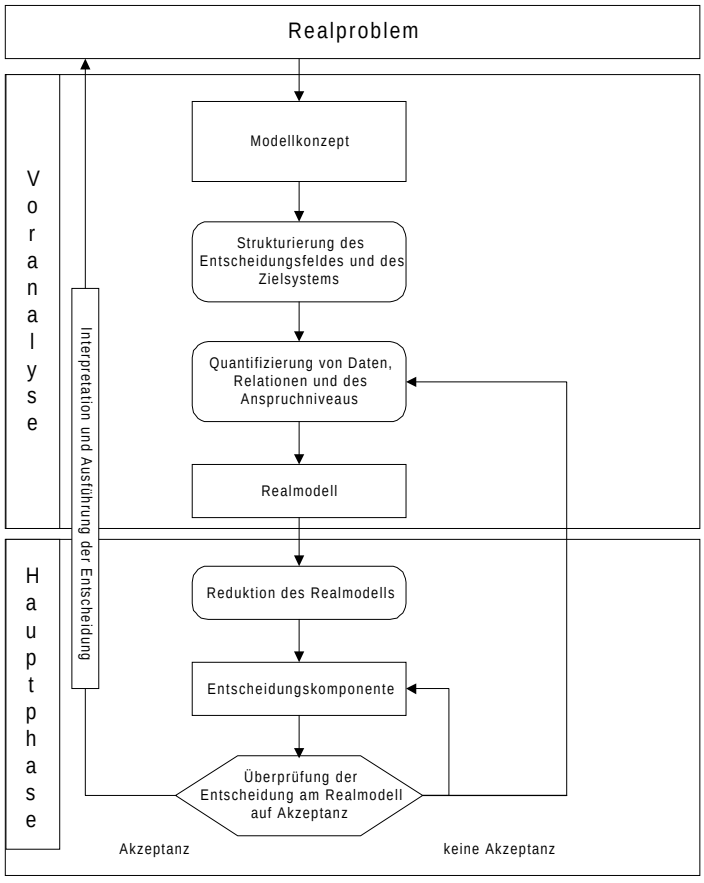


Bild 2.1: Struktur des Entscheidungsprozesses[25]

Als Modell wird eine vereinfachende Darstellung eines Sachverhaltes aus der Realwelt verstanden. Die Modellierung erfolgt - um das Modell softwaretechnisch auswerten zu

[24] Vgl. Mechler (1994), S. 32f.
[25] Nach Rommelfanger (1994), S. 3.

können - durch eine mathematische Beschreibung. Das Realmodell soll eine hohe Strukturähnlichkeit mit dem Realproblem aufweisen, um die Ergebnisse der Modellanalyse auf das Realproblem transformieren zu können. Bei der Formulierung eines Entscheidungsmodells offenbart sich oft ein Zielkonflikt[26], zwischen möglichst geringer Modellkomplexität und hoher Abbildungstreue, bei der Darstellung aller Zusammenhänge und Wirkungen.

Die Vereinfachung auf die relevanten Daten und Zusammenhänge ist notwendig, um das Realproblem auf eine für den Entscheider gedanklich erfaßbare Form zu reduzieren. Folgende Hindernisse erschweren eine präzise mathematische Beschreibung von Realweltzusammenhängen:

- Ist der *Faktor Mensch*[27] in ein Modell integriert, ist es sinnvoll, das einem Menschen eigene Verhalten und Problemverständnis in den Entscheidungsfindungsprozeß zu integrieren[28].

- Realweltsituationen lassen sich oft nicht scharf definieren, d. h., zeigen kein streng deterministisches Verhalten. Sie können somit nicht durch präzise Ursache-Wirkung-Beziehungen beschrieben werden[29].

- Die komplette Beschreibung eines realen Systems würde ein weit höheres Datenvolumen beanspruchen, als ein Mensch jemals gleichzeitig wahrnehmen, verarbeiten und verstehen kann.

Die Menge der, empirisch belegten, Hypothesen, welche das Realmodell bildet, gliedert sich in ein Entscheidungsfeld und ein Zielsystem. Das Entscheidungsfeld beinhaltet die Aussagen, welche die Umgebung des Aufgabenträgers betreffen, und das Zielsystem beschreibt seine angestrebten Ziele[30].

Das Entscheidungsfeld setzt sich aus verschiedenen Komponenten zusammen, dies sind im einzelnen[31]:

- Der Alternativenraum, in welchem sich die möglichen Aktionen des Entscheidungsträgers bewegen. Die Menge A der Alternativen kann durch ein Restriktionensystem oder im überschaubaren Fall durch Aufzählung beschrieben werden.

---

26 Vgl. Adam (1994), S. 4.
27 Vgl. Suh, Suh (1993), S. 153.
28 Vgl. Turksen et al. (1992), S. 350.
29 Vgl. Zimmermann (1991), S. 3.
30 Vgl. Mag (1977), S.23.
31 Vgl. Rommelfanger (1944), S. 1.

- Der Zustandsraum S; jeder Zustand wird durch eine Kombination der relevanten Umweltdaten[32] beschrieben.

- Die Ergebnisfunktion $g: A \times S \rightarrow E$, die jedem Tupel $(a;s) \in A \times S$ eine Folgerung $g(a;s) \in E$ zuordnet.

- Ein Informationssystem, das über die Verfügbarkeit, die Kosten und Auswirkungen zusätzlicher Informationen Auskunft gibt.

Im Zielsystem befinden sich neben den die Zielgrößen quantitativ beschreibenden Zielfunktionen die Präferenzen des Entscheidungsträgers, welche durch Nutzenfunktionen dargestellt werden. Die Teilnutzenfunktionen werden häufig zu einer Gesamtnutzenfunktion aggregiert.

Vor Aufstellung eines Realmodells wird häufig ein Modellkonzept erstellt, in welchem die relevanten Hypothesen - qualitativer oder quantitativer Art - erfaßt werden. Die Hypothesen können neben Ursache-Wirkung-Beziehungen auch Vermutungen enthalten. Zur Modellierung eines Zielsystems und eines Entscheidungsfeldes werden

- die vagen Hypothesen konsolidiert,

- die qualitativen und verbalen Aussagen quantifiziert und

- die Anspruchsniveaus gesetzt.

Das hieraus abgeleitete Realmodell dient als Ausgangspunkt für ein operationales Modell, an welchem weitere Vereinfachungen vorgenommen werden müssen, um recheneffiziente Verfahren in Form einer softwarebasierten Entscheidungskomponente einsetzen zu können.

Es erfolgt eine Validierung des generierten Entscheidungsvorschlages an den Restriktionen und Anspruchsniveaus des Realmodells. Die Analyse des Ergebnisses und zusätzliche Informationen können als Ausgangspunkt zur Änderung des Realmodells oder der verwendeten Entscheidungskomponente dienen.

Bei der Beschreibung der Struktur des Modellkonzepts offenbart sich für den Aufgabenträger oft die Schwierigkeit der adäquaten sprachlichen Darstellung. Man stellt oft fest, daß die Umgangssprache nicht ausreichend ist, um Ziele und Modellzusammenhänge aus dem Bereich des Realproblems genau zu formulieren. Beim Übergang vom Modell-

[32] Vgl. Mag (1977), S.44.

konzept zum Realmodell stellt die *wortarme*[33] Sprache der Mathematik eine weitere Hürde zur realweltnahen Darstellung des Modells dar.

2.2 Anwendbarkeit von Entscheidungsmodellen

Die Darstellung von Entscheidungsmodellen zur Gewinnung qualitativer Urteile erfolgt oft durch eine auf klassischen mathematischen Methoden basierende Modellierung. Die adäquate Darstellung scheitert trotz der so gewonnenen Einblicke in Systemzusammenhänge und Wirkungspfade, da Entscheidungsträger in ihrem Bestreben, möglichst frühzeitig qualitative Entscheidungen treffen, nicht unterstützt werden.

TURKSEN[34] nennt dafür mehrere Gründe.

Große Einschränkungen

Die Prämissen des Modells sind zu restriktiv, daher kann eine Aufweichung der Restriktionen notwendig sein, um Ziele unscharf zu formulieren. Oft entscheidet die ungefähre Größenordnung des erreichbaren Deckungsbeitrages über die Annahme eines Auftrages. Die Festlegung einer scharfen Deckungsbeitragsgrenze als Prämisse würde in diesem Fall zu einer unplausiblen Handlungsempfehlung führen, da bereits eine marginale Unterschreitung dieser Grenzvorgabe zu einer Auftragsablehnung führt.

Hohe Komplexität

Das Modell weist aufgrund seiner mathematischen Formulierung einen hohen Komplexitätsgrad und somit eine geringe Flexibilität auf. Die Akzeptanz[35] des Modells für den Aufgabenträger ist stark von dessen Fähigkeit abhängig, die mathematische Struktur des Modells nachzuvollziehen oder es anpassen zu können. Kann die Komplexität, die sich auch in der Zahl der verwendeten Regeln widerspiegelt, reduziert werden und/oder eine natürlichsprachliche Möglichkeit der Regelrepräsentation zur Verfügung gestellt werden, so ist das System für den Aufgabenträger leichter handhabbar und veränderbar.

---

33 Vgl. Rommelfanger (1994), S. 2.
34 Vgl. Turksen (1992), S. 2.
35 Vgl. Kemper (1987), S.14f.

Verfügbarkeit der Informationen

Die Anwendbarkeit eines Modells ist stark von den Informationskosten[36] und dem Grad der Beschaffbarkeit der Eingangsgrößen abhängig. Um in einer Konkurrenzsituation eine vorteilhafte Entscheidung treffen zu können, ist ein komparativer Konkurrenz-Vorteil (KKV) in Bezug auf den Faktor Zeit oft ausschlaggebend. In vielen Situationen ist aufgrund der geringen Informationsreife[37] oft nur eine qualitative Einschätzung der Parameter möglich.

Problemstellungen aus dem steuernden Teilbereich der Auftragsabwicklung[38], der Produktionssteuerung, sind oft von stochastischem Charakter. Störungen in Materialfluß[39] und Maschinenverfügbarkeit können weitgehend nicht exakt vorausgesehen werden, weil die Einflußgrößen derartiger Störungen[40] unter vertretbarem Aufwand nicht meßbar und somit nicht modellierbar sind. Anstatt Staus und Engpässe zu modellieren, bietet es sich an, diese Störungen durch die Anwendung von Regeln zu beheben. Regeln können im Rahmen der Wissensakquisition durch die Befragung von Experten gewonnen werden. Zur Abbildung unscharf formulierter Aussagen bietet sich die Anwendung von Methoden aus dem Bereich der Fuzzy-Set-Theorie[41] an.

36 Vgl. Adam (1993), S. 254.
37 Vgl. Eversheim et al. (1995), S. 48.
38 Vgl. Becker et al. (Auftragsabwicklung) (1996), S.9.
39 Zur Bedeutung des Materialflusses für die Qualität der betrieblichen Leistungserstellung vgl. Helfrich (1993), S. 14.
40 Vgl. Milberg et al. (1991), S.8.
41 Kurz: Fuzzy-Methoden, vgl. Abschnitt 4.3.

2.3 Fuzzy-Set-Theorie

Vage, unsichere und unscharfe[42] Zusammenhänge im Modellkonzept, die einen stochastischen Charakter haben, können durch statistische Methoden beschrieben werden. Allerdings führen diese Ansätze durch die oft praktizierte Reduktion von unsicheren oder qualitativ beschriebenen Daten auf Mittelwerte zu einer künstlichen Verfälschung der Modellzusammenhänge. Die von ZADEH begründete Theorie der unscharfen Mengen (im folgenden *Fuzzy-Set-Theorie*) ermöglicht es, die Modellierung von Zusammenhängen mit nicht-stochastisch induzierter Unschärfe auf eine mathematisch genaue Grundlage zu stellen und sie somit in eine softwaretechnisch verwertbare Form zu transformieren. Er begründet dies wie folgt[43]:

[Essentially, such a framework [fuzzy set theory] provides a natural way of dealing with problems in which the source of imprecision is the absence of sharply defined criteria of class membership rather than the presence of random variables.]

2.3.1 Arten von Unschärfe

In der Literatur finden sich die folgenden Arten von Unschärfe.

Stochastische Unschärfe[44]

Die stochastische Unschärfe stammt aus zu Mittelwerten verdichteten Aussagen, wie beispielsweise: „Ein ROI von 3 Jahren erlaubt -statistisch betrachtet- ein Investitionsvolumen von 400.000DM“. Diese Art der Unschärfe wird im weiteren Verlauf dieser Arbeit nicht behandelt. Eine Abgrenzung der Fuzzy-Set-Theorie zur Wahrscheinlichkeitstheorie beschreiben u. a. DUBOIS/PRADE[45].

Intrinsische Unschärfe[46]

Die intrinsische Unschärfe kommt in der Beschreibung menschlicher Eindrücke und Einschätzungen zum Ausdruck. Die Aussagen „hohe Kosten“ oder „schlechte Auftragslage“ beschreiben die vorherrschende Situation nicht exakt, da keine betragsmäßige Festlegung von Kosten, die „hoch“ sind, erfolgt. Eine exakte Festlegung der unteren Grenze für „hohe Kosten“ scheitert semantisch an der, in der logischen Konsequenz folgenden, anderen Deutung eines um einen Pfennig kleineren Kostenvolumens.

---

42 Im folgenden subsumiert *Unschärfe* die Begriffe Vagheit, Unsicherheit und Unschärfe.
43 Vgl. Zadeh (1965), S. 339.
44 Vgl. Zimmermann (1991), S. 3.
45 Vgl. Dubois, Prade (1991), S. 1059f.
46 Vgl. Rommelfanger (1994), S. 4.

Informelle Unschärfe[47]

Die informelle Unschärfe ist dadurch gekennzeichnet, daß Aussagen zwar exakt definiert sind, jedoch die praktische Durchführung der Informationsgewinnung Schwierigkeiten bereitet oder nur a posteriori möglich ist. Diese Art der Unschärfe liegt auch bei der Unmöglichkeit einer genauen Messung vor.

Unscharfe Relationen

Unscharfe Relationen als Beziehungen zwischen einzelnen Größen sind nicht dichotom[48]. Beispiele sind die Relationen „ist nicht viel kleiner als“ oder „ist erheblich kostengünstiger als“. In der Kombination mit der intrinsischen Unschärfe sind Aussagen wie „Wenn die Auftragslage schlecht ist, dann ist die Kapazitätsauslastung zu gering“ ebenfalls unscharfe Relationen. Unscharfe Beziehungen finden sich in Aussagen wie „Eine neue Verpackungsmaschine sollte nicht viel mehr als 25.000 DM kosten“. Für die Anwendung von quantitativen Methoden zur Lösung derartiger unscharfer Entscheidungsprobleme bietet die Anwendung der Fuzzy-Set-Theorie einen vereinfachenden und intuitiven Ansatz. Die graphische Darstellung einer unscharfen Relation findet sich in Bild 2.2.

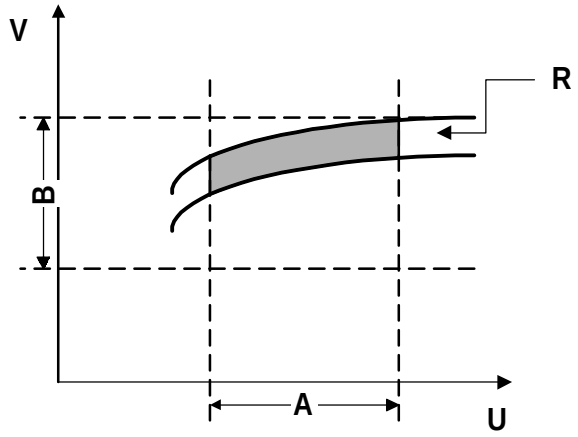


Bild 2.2: Unscharfe Relation[49]

2.3.2 Scharfe Mengen

Bevor auf die unscharfen Mengen eingegangen wird, wird zur Abgrenzung und Sensibilisierung für deren besonderen Merkmale, im folgenden der Begriff der Menge im klassischen Sinne beschrieben.

---

47 Vgl. Rommelfanger (1994), S. 4.
48 Vgl. Zimmermann (1991), S. 284.
49 Nach Aliev, Bonfig, Aliew (1994), S. 19.

Definition 2.1: Die klassische Menge

Der Grundansatz zur geordneten Darstellung von Informationen ist die Verwendung der klassischen Menge[50]

$$M = \{x \mid \mu_M(x)\},\ \mu_M(x) : U \rightarrow \{0;1\}.$$

In M befinden sich alle Elemente x aus der Grundmenge U, für welche das einstellige, zweiwertige Prädikat $\mu_M(x)$ erfüllt ist. Man bezeichnet $\mu_M(x)$ als diskrete *Indikatorfunktion* oder diskrete *Zugehörigkeitsfunktion;* denn der Wert von $\mu_M(x)$ entscheidet eindeutig über die Zugehörigkeit jedes Elementes $x \in U$ zur Menge M. Der Wertebereich von $\mu_M(x)$ ist auf die beiden Werte 0 und 1 begrenzt. Aufgrund der Eigenschaft der *scharfen* Zugehörigkeitsabgrenzung bezeichnet man die klassische Menge auch als *scharfe* Menge[51], denn es gilt:

$x \in M$, falls $\mu_M(x) = 1$ und

$x \notin M$, falls $\mu_M(x) = 0$.

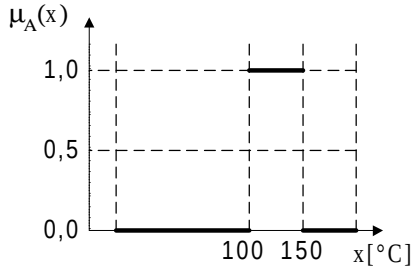


Bild 2.3: Scharfe Menge „Normale Betriebstemperatur“

Wird die technische Spezifikation eines Fertigungsaggregates durch die normale Betriebstemperatur mit dem Intervall I=[100,150] beschrieben, so gilt in diesem Bereich für den Wert der Zugehörigkeitsfunktion $\mu_M(x) = 1\ \forall\ x \in I$. An allen anderen Stellen ist $\mu_M(x) = 0$. Eine graphische Darstellung dieses Sachverhaltes gibt Bild 2.3.

2.3.2.1 Beispiel[52]

Ein mittelständisches Unternehmen der Möbelindustrie produziere Wohnzimmerschränke. Das Unternehmen kann täglich mindestens 5 und höchstens 9 Stück herstellen. Jede andere Stückzahl sei nicht produzierbar. Die Menge der möglichen täglichen Stückzahlen werde durch S = {5, 6, 7, 8, 9} abgebildet, wobei die möglichen Anzahlen

50 Vgl. Böhme (1993), S. 5.
51 Synonym: Crisp-Set.
52 Vgl. Rommelfanger (1994), S. 7.

gleichermaßen produziert werden können. Wird mit diesen möglichen Produktionszahlen die Höhe der Stückkosten kombiniert, ergibt sich die Menge der Tupel (Stückzahl; Stückkosten) als

T = {(5;18.000 DM), (6;24.000 DM), (7;27.000 DM), (8;28.000 DM), (9;35.000 DM)}.

Bei der Zielsetzung, zu möglichst geringen Stückkosten zu produzieren, wird das Unternehmen sich für die Produktion von 8 Schränken entscheiden. Die Ergebnismenge ist also $\{8\} \subset S$.

Ist die Zielsetzung des Unternehmens, zu „vertretbar geringen Stückkosten“ zu produzieren, so ist die Ergebnismenge nicht eindeutig bestimmt. Neben den Stückkosten ist gegebenenfalls aus anderen Rahmenkriterien oder Umweltumständen - wie beispielsweise der Höhe des Verkaufspreises oder der Marktlage - die jeweilige Einschätzung von „vertretbaren“ Stückkosten zu ermitteln.

Bewertung

Die Darstellung mit einer scharfen Menge kann nicht zur Modellierung der unscharfen Bewertung - der Stückpreis sei vertretbar - dienen. Dieses Bewertungsproblem kann durch die Einführung gradueller Zugehörigkeitsgrade gelöst werden, denn neben der strengen Zugehörigkeit und Nichtzugehörigkeit ist auch jeder reeller Zwischenwert im Intervall [0;1] zugelassen. Es eröffnet sich somit die Möglichkeit, Sachverhalte mit intrinsischer oder linguistischer Unschärfe - beispielsweise Ausdrücke wie „Umrüstkosten sind hoch“ oder „die Schlupfzeit des Auftrags B ist kurz“[53] - durch eine allgemeine Zugehörigkeitsfunktion $\mu : U \rightarrow [0;1]$ zu beschreiben.

2.3.3 Unscharfe Mengen

Bei der Bildung von Modellen wird die Realität herkömmlicherweise mit Hilfe mathematischer Ausdrucksmittel beschrieben. Oft ergibt sich hierbei ein Strukturbruch. Das Modell wird an die mögliche Darstellungsform angepaßt. Durch die Anwendung von Fuzzy-Methoden, basierend auf unscharfen Mengen, ist es jedoch möglich, das Instrumentarium an die Problemstellung anzupassen.

Es können drei Stufen[54] unterschieden werden.

1. Wissensakquisition: Festlegung des zur Problemlösung benötigten Expertenwissens.

[53] Vgl. Turksen et al. (1992), S. 351.
[54] Vgl. Bothe (1995), S. 25.

2. Modellierung: Erstellung von Zugehörigkeitsfunktionen.

3. Inferenz: Bildung einer Schlußfolgerung durch die Verknüpfung von Einzelaussagen.

Im folgenden werden die Grundlagen der Fuzzy-Set-Theorie vorgestellt.

Definition 2.2: Unscharfe Menge

Wenn die in einer Trägermenge[55] S vorhandenen Elemente bezüglich einer unscharfen Aussage eingeschätzt werden, dann wird

$\mathbf{A} = \{(x,\mu_A(x))\},\ x \in S$

eine unscharfe Menge [56] auf S genannt.

Definition 2.3: Zugehörigkeitsfunktion

Die Funktion

$\mu_A(x):S \rightarrow \Re$

gibt die Kompatibilität der Elemente von S gegenüber dem beschreibenden Prädikat **A** an. $\mu_A(x)$ wird Zugehörigkeitsfunktion genannt.

Die Zugehörigkeitsgrade sind der Ausdruck einer *subjektiven* Bewertung durch ein Individuum. Die unscharfe Menge ergibt sich somit direkt aus ihrer Zugehörigkeitsfunktion.

Unscharfe Mengen sind eine Verallgemeinerung des klassischen Mengenbegriffes. Wird der Wertebereich der Zugehörigkeitsfunktion auf die Extremwerte 0 und 1 beschränkt, so ergibt sich die Grundmenge S, wenn $\forall\ x \in S$ gilt: $\mu_A(x) = 1{,}0$. Die leere unscharfe Menge $\varnothing$ ergibt sich, wenn $\forall\ x \in S$ gilt: $\mu_A(x) = 0{,}0$.

2.3.4 Eigenschaften unscharfer Mengen

Im folgenden werden die im weiteren Verlauf dieser Arbeit benötigten Zusammenhänge zwischen scharfen und unscharfen Mengen dargestellt. Die unscharfe Menge **A** mit der Zugehörigkeitsfunktion $\mu_A(x)$ beinhalte beliebige Elemente x aus der Trägermenge S. Die reelle Zahl α sei konstant in [0;1].

---

55 Synonym: Support (engl. für Trägermenge).

56 Synonyme: Fuzzy-Set, Fuzzy-Menge.

Definition 2.4: Die Höhe einer unscharfen Menge

Die Höhe einer unscharfen Menge **A** über S wird durch

$$hgt(\mathbf{A}) = Sup\{\mu_A(x) \mid x \in S\}$$

angegeben.

Die Höhe entspricht im Fall einer abzählbaren Trägermenge dem höchsten Zugehörigkeitsgrad der einzelnen Elemente. Allgemein kann die Höhe durch das Supremum der Zugehörigkeitsfunktion ermittelt werden. Wird - wie im folgenden Verlauf dieser Arbeit - davon ausgegangen, daß der Wertebereich dieser Funktion auf das Einheitsintervall [0;1] beschränkt ist, so gilt $hgt(\mathbf{A}) \leq 1{,}0$. Ist $hgt(\mathbf{A}) = 1{,}0$ spricht man von einer *normalisierten* Bewertungs- oder Zugehörigkeitsfunktion. Jede nicht normalisierte (*subnormale*[57]) unscharfe Menge kann mittels einer Division der Zugehörigkeitsgrade der einzelnen Elemente durch die Höhe normalisiert werden.

Definition 2.5: Niveau-Menge einer unscharfen Menge

Ist **A** eine unscharfe Menge mit der Zugehörigkeitsfunktion $\mu_A(x)$, so heißt die scharfe Menge

$$N_\alpha(\mathbf{A}) = \{x \mid x \in S \wedge \mu_A(x) \geq \alpha\}$$

Niveau-Menge vom Grad α oder α-Schnitt von **A**. N_α ist eine scharfe Menge. Die Menge

$N'_\alpha(\mathbf{A}) = \{x \mid x \in S \wedge \mu_A(x) > \alpha\}$ nennt man strengen α-Schnitt von **A**.

Die alternative horizontale Sicht[58] auf eine unscharfe Menge wird durch eine Darstellung in Form ihrer Niveau-Mengen beschrieben.

Definition 2.6: Träger einer unscharfen Menge

Ist **A** eine unscharfe Menge mit der normalisierten Zugehörigkeitsfunktion $\mu_A(x)$, so heißt die scharfe Menge $T(\mathbf{A})$ Träger von **A** mit

$$T(\mathbf{A}) = N_{0,0}(\mathbf{A}) = \{x \mid x \in S \wedge \mu_A(x) \geq 0\}.$$

Der Träger einer unscharfen Menge beinhaltet diejenigen Elemente, welche dem Prädikat der Zugehörigkeitsfunktion mit einem positiven Zugehörigkeitsgrad entsprechen.

---

57 Vgl. Aliev, Bonfig, Aliew (1994), S. 13.
58 Vgl. Kruse et al. (1995), S. 20.

Definition 2.7: Kern einer unscharfen Menge

Ist **A** eine unscharfe Menge mit der normalisierten Zugehörigkeitsfunktion $\mu_A(x)$, so heißt die scharfe Menge K

$$K(\mathbf{A}) = N_{1,0}(\mathbf{A}) = \{x \mid x \in U \wedge \mu_A(x) = 1{,}0\}$$

Kern von **A**.

Der Kern einer unscharfen Menge beinhaltet diejenigen Elemente, welche dem Prädikat der Zugehörigkeitsfunktion mit absoluter Sicherheit[59] entsprechen.

Der Zusammenhang zwischen der α-Niveau-Menge, dem Träger und dem Kern einer unscharfen Menge **A** wird in Bild 2.4 graphisch dargestellt:

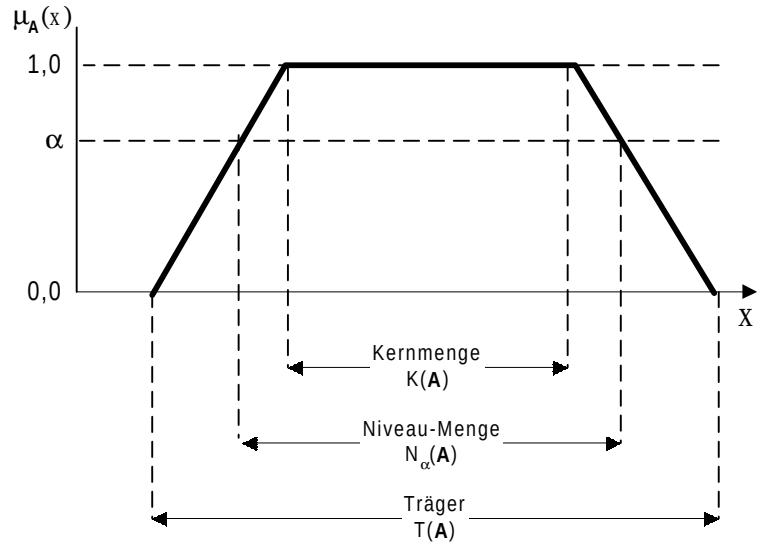


Bild 2.4: Kern, Niveaumenge und Träger einer unscharfen Menge

Definition 2.8: Konvexität einer unscharfen Menge

Die unscharfe Menge $\mathbf{A} = \{(x,\mu_A(x)) \mid x \in S\}$ ist dann konvex, wenn ihre Trägermenge S konvex ist und wenn gilt:

$$\mu_A(\lambda x_1+(1-\lambda)x_2) \geq \text{Min}(\mu_A(x_1), \mu_A(x_2)) \;\forall\; x_1,x_2 \in S, \forall\; \lambda \in [0;1]$$[60].

Der Zusammenhang[61] zwischen konvexen unscharfen Mengen und α-Schnitten besteht darin, daß eine unscharfe Menge **A** über S auf $\Re$ konvex ist und nur dann konvex ist,

59 Die „absolute Sicherheit“ wird durch den Zugehörigkeitsgrad 1,0 beschrieben.
60 Vgl. Zadeh (1965), S. 347.
61 Vgl. Böhme (1993), S. 122.

wenn ihre sämtlichen α-Schnitte zueinander konvex sind. Es ist jedoch nicht notwendig, daß der unscharfen Menge eine konvexe Zugehörigkeitsfunktion zugrundeliegt.

Definition 2.9: Mächtigkeit einer unscharfen Menge

Die Trägermenge S der unscharfen Menge **A** sei endlich. Die absolute Mächtigkeit von **A** ergibt sich als :

$$|\mathbf{A}| = \sum_{x \in S} \mu_A(x).$$

Die relative Mächtigkeit einer unscharfen Menge berechnet sich aus:

$$||\mathbf{A}|| = \frac{|\mathbf{A}|}{|S|}.$$

Ist die Trägermenge S einer unscharfen Menge **A** unendlich, so errechnet sich die absolute Mächtigkeit von **A** im Intervall $[x_{min}; x_{max}]$ aus:

$$|\mathbf{A}| = \int_{x_{min}}^{x_{max}} \mu_A(x) dx .$$

Die relative Mächtigkeit berechnet sich somit im Falle einer stetigen Funktion μ(x) aus

$$||\mathbf{A}|| = \frac{|\mathbf{A}|}{x_{max} - x_{min}}.$$

Die Mächtigkeit der scharfen Trägermenge S entspricht der Anzahl der Elemente der unscharfen Menge **A** mit einem positiven Zugehörigkeitsgrad. Aufgrund der Abhängigkeit von der Mächtigkeit der zugrundeliegen Trägermenge ist beim Vergleich der relativen Mächtigkeiten von unscharfen Mengen darauf zu achten, daß diese dieselbe Trägermenge haben.

2.3.5 Fuzzy-Logik

Die einer unscharfen Menge zugrundeliegende Semantik reicht gewöhnlich nicht aus, um Entscheidungen zu fällen. Oft ist es notwendig, mehrere unscharfe Mengen aussagenlogisch miteinander zu verknüpfen. An Operatoren für unscharfe Mengen ist dabei die Bedingung zu knüpfen, daß sie für die Randwerte {0;1} im Verhalten mit dem klassischen einstelligen Operator *nicht* und den zweistelligen Operatoren *und* und *oder* der Booleschen Algebra übereinstimmen. Es existieren mehrere Lösungen zur

Modellierung dieser Operatoren, wovon zunächst die verbreitetste Lösung vorgestellt wird, um sie im weiteren Verlauf in einem Gesamtzusammenhang zu systematisieren. Zunächst soll jedoch nur auf die einstellige, auf eine unscharfe Menge bezogene Operation der Komplementbildung eingegangen werden:

Definition 2.10: Komplement einer unscharfen Menge

Ist **A** eine unscharfe Menge mit der normalisierten Zugehörigkeitsfunktion $\mu_A(x)$, so wird die unscharfe Komplementbildung meist durch die Beziehung

$$\mu_{CA}(x) = 1 - \mu_A(x)$$

modelliert. Diese Komplementbildung C**A** stellt eine Transformation der klassischen mengentheoretischen Negation auf unscharfe Mengen dar.

Definition 2.11: Erweitertes Komplement einer unscharfen Menge

BOTHE[62] definiert eine parametrisiertes Komplement für unscharfe Mengen:

$$\mu_{CA}(x) = (1-\mu_A(x))^p, \text{ mit } p > 0.$$

Definition 2.12: Kartesisches Produkt zweier unscharfer Mengen

Seien **A** und **B** unscharfe Mengen mit den Trägern X und Y und den Zugehörigkeitsfunktionen $\mu_A(x)$ und $\mu_B(y)$. Es gilt $x \in X$ sowie $y \in Y$. Die Zugehörigkeitsfunktion $\mu_{A\times B}(x,y)$ des kartesischen Produktes **A**×**B** ermittelt sich nach:

$$\mu_{A\times B}(x,y) = \min(\mu_A(x),\mu_B(y)).$$

2.3.5.1 Durchschnitts- und Vereinigungsbildung von unscharfen Mengen

ZADEH[63] definiert die Operatoren *min* und *max* für die Verknüpfungen zur Durchschnitts- und Vereinigungsbildung von unscharfen Mengen. Das Verknüpfungsergebnis **C** ist ebenfalls eine unscharfe Menge. Die Zugehörigkeitsgrade der Elemente ergeben sich aus einer punktweisen Verknüpfung der Zugehörigkeitsgrade der Elemente von **A** und **B**.

---

62 Vgl. Bothe (1995), S. 37.
63 Vgl. Zadeh (1965), S. 340.

Definition 2.13: Durchschnitt unscharfer Mengen

Der Durchschnitt **C** zweier unscharfer Mengen **A** und **B** ist punktweise definiert durch :

$\mathbf{C} = \mathbf{A} \cap \mathbf{B} := \{(x ; \mu_C(x)) \mid x \in X, \mu_C(x) = \min(\mu_A(x), \mu_B(x))\}$.

Bild 2.5 illustriert die Durchschnittsbildung zweier unscharfer Mengen.

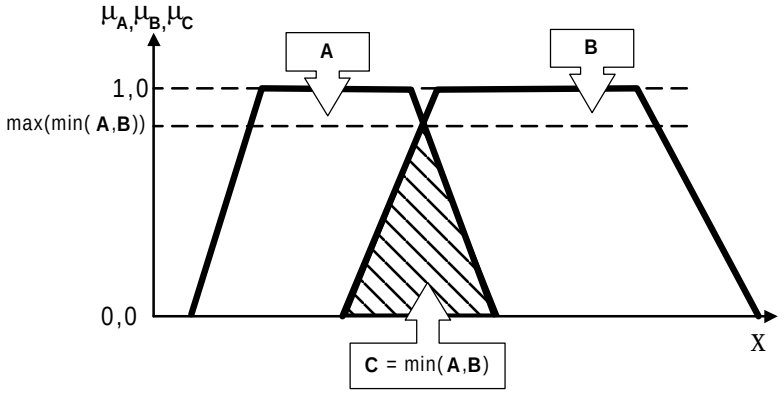


Bild 2.5: Durchschnitt unscharfer Mengen

Wie im weiteren Verlauf dieser Arbeit ersichtlich wird, ist für die Inferenzbildung die maximale Höhe der Schnittmenge von unscharfen Mengen nützlich. Diese Höhe beschreibt den *Kompatibilitätsgrad* zwischen zwei unscharfen Mengen.

Definition 2.14: Vereinigung unscharfer Mengen

Die Vereinigung **C** zweier unscharfer Mengen **A** und **B** ist punktweise definiert durch:

$\mathbf{C} = \mathbf{A} \cup \mathbf{B} := \{(x ; \mu_C(x)) \mid x \in X, \mu_C(x) = \max(\mu_A(x), \mu_B(x))\}$.

Bild 2.6 veranschaulicht die Vereinigung unscharfer Mengen.

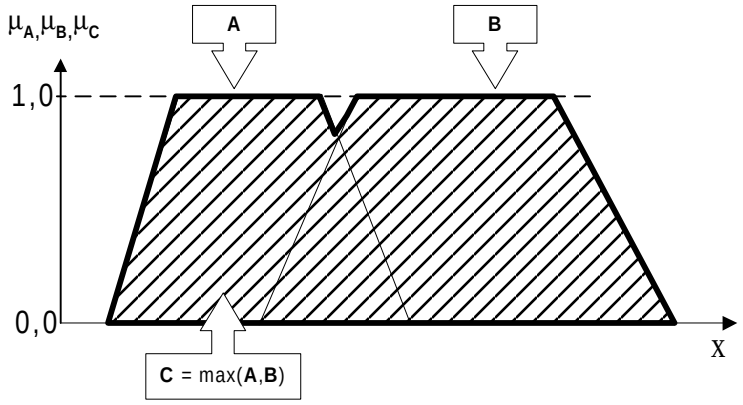


Bild 2.6: Vereinigung unscharfer Mengen

Die Operatoren *min* bzw. *max* zur Darstellung von Durchschnitt bzw. Vereinigung unscharfer Mengen ermöglichen durch ihre einfache Berechenbarkeit und durch ihren relativ einfachen Aufbau eine schnelle maschinelle Verarbeitung. Diese Operatoren weisen eine hohen Übereinstimmungsgrad in ihren logischen[64] Eigenschaften mit den Operatoren der klassischen Aussagenlogik auf. Eine Ausnahme bildet allerdings das Gesetz der ausgeschlossenen Mitte, G sei die Gesamtmenge:

$\mathbf{A} \cap C\mathbf{A} \neq \varnothing$ und $\mathbf{A} \cup C\mathbf{A} \neq G$.

ROMMELFANGER[65] bezeichnet aufgrund dieser Eigenschaften den *max*-Operator als *logisches oder* und den *min*-Operator als *logisches und* zur Verknüpfung unscharfer Mengen.

Ein weiterer Aspekt zur Beurteilung unscharfer Operatoren sind Eigenschaften wie Anpassungsfähigkeit und die Notwendigkeit der Modellierung von menschlichen Einschätzungen, deren Vieldeutigkeit beispielsweise in der Unschärfe der Bedeutung von *und*, *oder* und *nicht* im alltäglichen Sprachgebrauch deutlich wird. Nur in einigen Fällen entspricht dieses der mathematisch-logischen Semantik von *und*, *oder* und *nicht*. Das Operatorenpaar *min* und *max* entspricht als einziges den geforderten Eigenschaften des Axiomensystems nach BELLMANN und GIERTZ[66].

ROMMELFANGER[67] fordert als wichtigste Eigenschaften unscharfer Operatoren:

- Distributivität.
- Vereinigung und Durchschnitt dürfen nicht zu Werten führen, die zwischen den Ausgangswerten liegen.

64 Vgl. Rommelfanger (1994), S. 21, Darstellung im Anhang.
65 Vgl. Rommelfanger (1994), S. 21.
66 Vgl. Rommelfanger (1994), S. 21.
67 Vgl. Rommelfanger (1994), S. 22.

- Ausschluß eines kompensatorischen Ausgleiches zwischen hohen und niedrigen Werten..

Verzichtet man auf die Eigenschaft der Distributivität, so ist es möglich, weitere Operatoren zur Vereinigungs- und zur Durchschnittsbildung auf unscharfen Mengen zu definieren, die den verbleibenden Kriterien genügen. Diese Operatoren werden im folgenden vorgestellt. In 2.3.9 werden weitere Kriterien zur Bewertung unscharfer Operatoren vorgestellt.

Im folgenden werden einige häufig verwendete Operatoren zur Verknüpfung unscharfer Mengen vorgestellt. Diese werden im folgenden Kapitel in einem Gesamtzusammenhang systematisiert.

Definition 2.15: Algebraisches Produkt

Das *algebraische Produkt* **C** zweier unscharfer Mengen **A** und **B** ist durch die Zugehörigkeitsfunktion $\mu_C = alg_t(\mu_A, \mu_B)$ definiert, wobei:

$alg_t(x,y) = xy$ mit $x,y \in [0;1]$

Definition 2.16: Algebraische Summe

Die *algebraische Summe* **C** zweier unscharfer Mengen **A** und **B** ist durch die Zugehörigkeitsfunktion $\mu_C = alg_s(\mu_A, \mu_B)$ definiert, wobei:

$alg_s(x,y) = x+y-xy$ mit $x,y \in [0;1]$.

Das algebraische Produkt und die algebraische Summe werden zur Modellierung des natürlich-sprachlichen *und* bzw. *oder* eingesetzt, wenn *min* zu große bzw. *max* zu kleine Ergebnisse liefert. Es ist allerdings zu beachten, daß das *min*/*max*-Operatorenpaar bereits bei einer ordinalskalierten Datensituation zu Ergebnissen führt. Das algebraische Produkt und die algebraische Summe erfordern hingegen metrisch-skalierte Daten.

Definition 2.17: Beschränkte Differenz

Die *beschränkte Differenz* **C** zweier unscharfer Mengen **A** und **B** ist durch die Zugehörigkeitsfunktion $\mu_C = bes_t(\mu_A, \mu_B)$ definiert, wobei:

$bes_t(x,y) = \max(0,x+y-1)$ mit $x,y \in [0;1]$.

Definition 2.18: Beschränkte Summe

Die *beschränkte Summe* **C** zweier unscharfer Mengen **A** und **B** ist durch die Zugehörigkeitsfunktion $\mu_C = bes_s(\mu_A, \mu_B)$ definiert, wobei:

$$bes_s(x,y) = \min(1, x+y) \qquad \text{mit } x,y \in [0;1].$$

Definition 2.19: Drastisches Produkt

Das *drastische Produkt* **C** zweier unscharfer Mengen **A** und **B** ist durch die Zugehörigkeitsfunktion $\mu_C = dra_t(\mu_A, \mu_B)$ definiert, wobei:

$$dra_t(x,y) = \begin{cases} x & \text{für } y=1 \\ y & \text{für } x=1 \\ 0 & \text{für } x,y < 1 \end{cases} \qquad \text{mit } x,y \in [0;1].$$

Definition 2.20: Drastische Summe

Die *drastische Summe* **C** zweier unscharfer Mengen **A** und **B** ist durch die Zugehörigkeitsfunktion $\mu_C = dra_s(\mu_A, \mu_B)$ definiert, wobei:

$$dra_s(x,y) = \begin{cases} x & \text{für } y=0 \\ y & \text{für } x=0 \\ 1 & \text{für } x,y > 0 \end{cases} \qquad \text{mit } x,y \in [0;1].$$

Definition 2.21: Algebraischer t-Quotient

Der *algebraische t-Quotient* **C** zweier unscharfer Mengen **A** und **B** ist durch die Zugehörigkeitsfunktion $\mu_C = quo_t(\mu_A, \mu_B)$ definiert, wobei:

$$quo_t(x,y) = \frac{alg_t(x,y)}{alg_s(x,y)} = \frac{xy}{x+y-xy} \qquad \text{mit } x,y \in [0;1].$$

Definition 2.22: Algebraischer s-Quotient

Der *algebraische s-Quotient* **C** zweier unscharfer Mengen **A** und **B** ist durch die Zugehörigkeitsfunktion $\mu_C = quo_s(\mu_A, \mu_B)$ definiert, wobei:

$$quo_s(x,y) = \frac{alg_s(x,y)-alg_t(x,y)}{1-alg_t(x,y)} = \frac{x+y-2xy}{1-xy} \qquad \text{mit } x,y \in [0;1].$$

Der algebraische t-Quotient und der algebraischem s-Quotient setzt sich in Zähler und Nenner aus Kombinationen von algebraischem Produkt und Summe zusammen.

2.3.6 t-Normen

Die Verknüpfungen auf unscharfen Mengen mit den Zadeh-Operatoren *min* und *max* gibt oft nicht den Sachverhalt wieder, der durch den Gebrauch von *und* und *oder* aus dem natürlich-sprachlichen Sprachgebrauch beschrieben wird. Ein Ansatz, die Freiheitsgrade dieses sprachlichen Spielraums zu nutzen, bietet ein allgemeineres Instrument zur Darstellung von Konjunktionen: das Gerüst der t-Normen für Funktionen $t: [0;1]\times[0;1]\rightarrow[0;1]$.

Definition 2.23: t-Norm

Ein binärer Operator t(x,y) wird t-Norm[68] genannt, wenn die folgenden Eigenschaften (T1)-(T5) gelten (wobei $x,x',y,y',z \in [0;1]$):

(T1) $t(x,0)=0$; d. h. 0 ist das Nullelement

(T2) $t(x,1)=t(1,x)=x$, d. h. 1 ist das Neutralelement.

(T3) $t(x,y) \leq t(x',y')$, wenn $x<x' \wedge y<y'$, d. h. t ist monoton wachsend.

(T4) $t(x,y) = t(y,x)$, d. h. t ist kommutativ.

(T5) $t(t(x,y),z) = t(x,t(y,z))$, d. h. t ist assoziativ.

2.3.6.1 Beispiele für t-Normen

Von den folgenden, bereits vorgestellten Operatoren werden die Kriterien für t-Normen erfüllt:

- Minimum-Operator,
- Algebraisches Produkt,
- Beschränkte Differenz,
- Algebraischer t-Quotient,
- Drastisches Produkt

[68] Kurz : triangular-Norm.

2.3.7 s-Normen

Zu jeder t-Norm t existiert eine s-Norm[69] s: $[0;1]\times[0;1]\rightarrow[0;1]$, welche den zum Durchschnittsoperator t korrespondierenden Vereinigungsoperator darstellt. Zwischen t und s bestehen aufgrund der de-Morgan-Gesetze[70] die Dualitätsbeziehungen:

$t(x,y) = 1-s(1-x,1-y)$ und $s(x,y) = 1-t(1-x,1-y)$.

Definition 2.24: s-Norm

Ein binärer Operator s(x,y) wird s-Norm genannt, wenn die folgenden Eigenschaften (S1)-(S5) gelten (wobei $x,x',y,y',z \in [0;1]$):

(S1) $s(x,1) = 1$, d. h. 1 ist das Einselement

(S2) $s(x,0) = s(0,x) = x$, d. h. 0 ist das Neutralelement.

(S3) $s(x,y) \leq s(x',y')$, wenn $x<x' \wedge y<y'$, d. h. s ist monoton wachsend.

(S4) $s(x,y) = s(y,x)$, d. h. s ist kommutativ.

(S5) $s(s(x,y),z) = s(x,s(y,z))$, d. h. s ist assoziativ.

2.3.7.1 Beispiele für s-Normen

Von den folgenden, bereits vorgestellten Operatoren werden die Kriterien für s-Normen erfüllt:

- Maximum-Operator,
- Algebraische Summe,
- Beschränkte Summe,
- Algebraischer s-Quotient,
- Drastische Summe

---

69 Auch : t-Conorm.

70 Vgl. Jaanineh, Maijohann (1996), S. 91.

2.3.8 Ordnung der Normen-Kombinationen

Die Ordnung des Verhaltens der beschriebenen t-Normen läßt sich in folgender Ordnungskette beschreiben: Sie werden nach oben vom numerisch stärksten Operator *min* und nach unten vom numerisch schwächsten Operator dra_t eingeschlossen. Diese Ordnung macht man sich beispielsweise im folgenden Fall zunutze: Liefert die Disjunktion mit dem min-Operator zu große Zugehörigkeitsgrade, kann das algebraische Produkt oder das drastische Produkt als t-Norm gewählt werden, da diese Operatoren kleinere aber höchstens gleiche Werte wie der min-Operator liefern.

Die s-Normen verhalten sich diametral zu den zu den t-Normen. Die Anordnungsketten der t-Normen und der s-Normen sind in Bild 2.7 dargestellt.

$$dra_t(a,b) \le ... \le bes_t(a,b) \le ... \le alg_t(a,b) \le ... \le min(a,b)$$

$$dra_s(a,b) \ge ... \ge bes_s(a,b) \ge ... \ge alg_s(a,b) \ge ... \ge max(a,b)$$

Bild 2.7: Anordnungsketten der t-Normen und der s-Normen

Nur die t-Norm *min* (und somit auch die s-Norm *max*) erfüllt die Forderung nach Idempotenz, die garantiert, daß bei einer reflexiven Verknüpfung einer unscharfen Aussage mittels *und* bzw. *oder* der Wahrheitswert der Aussage erhalten bleibt; denn es gilt

$$\min(a,a)=a \Rightarrow \mathbf{A} \cap \mathbf{A} = \mathbf{A} \quad \text{sowie} \quad \max(a,a)=a \Rightarrow \mathbf{A} \cup \mathbf{A} = \mathbf{A}.$$

Das *min*/*max*-Operatorenpaar wird - aufgrund der auf die Werte $\{\mu_A(x),\mu_B(x)\}$ beschränkten Ergebnismenge - auch *nicht-interaktive*[71] Verknüpfung genannt. Operatorenpaare, deren Ergebnisse in der Regel von den Eingangswerten abweichen, werden *interaktive* Verknüpfungen genannt.

2.3.9 Kriterien[72] zur Auswahl von Operatoren auf unscharfen Mengen

Die Wahl von Verknüpfungsoperatoren wird durch ihre situationsabhängige Leistungsfähigkeit bestimmt, die von ihren jeweiligen Eigenschaften abhängt. Je nach Art der konkreten Anwendung werden sich die Gewichte der Auswahlkriterien verschieben. Die Eigenschaften von Operatoren lassen sich in Rationalitätsanforderungen und pragmatische Aspekte unterteilen.

---

71 Vgl.Rommelfanger (1994), S. 24

72 Vgl. Jaanineh, Maijohann (1996), S. 107f.

2.3.9.1 Rationalitätsanforderungen

Diese Gruppe von Anforderungen behandelt die mathematischen Kriterien, die bei der Auswahl eines geeigneten Operators entscheidend sind.

Kommutativität

Die Kommutativität einer Operation ist gegeben, falls die Anordnung der Argumente oder die Verknüpfungsreihenfolge beliebig ist. Nach (T2) und (S2) besitzen alle t-Normen und s-Normen diese Eigenschaft.

Assoziativität

Die Assoziativität einer Operation ist die Eigenschaft der beliebigen gruppenweisen Zusammenfassung von Teilergebnissen. Dieses Kriterium wird nur dann ausschlaggebend sein, wenn im Vorfeld nicht bekannt ist, wie viele unscharfe Mengen miteinander verknüpft werden sollen.

Stetigkeit

Ändert sich ein Argument der Operation nur geringfügig, so wird sich bei einem stetigen Operator das Ergebnis auch nur geringfügig ändern. Die Stetigkeit eines Operators verhindert große Sprünge im Ergebnisbereich bei geringen Änderungen der Operanden. Je nach Beschaffenheit der Anwendung kann diese Eigenschaft wichtig sein.

Monotonie

Steigt der Zugehörigkeitsgrad des Ergebnisses einer Operation bei Erhöhung eines Operanden oder bleibt das Ergebnis zumindest gleich, so spricht man von einem monotonen Operator.

Stabilität

Das Ergebnis einer Verknüpfung mit einem stabilen Operator bewegt sich zwischen Minimum und Maximum der Operanden. Ist dieses Verhalten gefordert, so wird man auf Durchschnittsoperatoren zurückgreifen.

Idempotenz

Werden identische Operanden mit einem Operator verknüpft, der den Wert der Eingangsgrößen als Ergebnis liefert, so spricht man von einem idempotenten Operator.

2.3.9.2 Pragmatische Aspekte

Diese Gruppe von Anforderungen behandelt die technischen und pragmatischen Kriterien zur Auswahl eines geeigneten Operators.

Empirische Eignung[73]

Die Eignung eines Operators sollte nicht nur nach seinen formalen mathematischen Qualitäten (wie bspw. Erfüllung des Assoziativgesetzes) bewertet werden, sondern sollte den zu modellierenden Realweltzusammenhang adäquat abbilden. Im Normalfall wird diese Eignung nur durch empirische Versuche nachzuweisen sein.

Adaptionsfähigkeit

Soll ein Operator an verschiedene Gegebenheiten anpaßbar sein, so wird man einen parametrisierten Operator wählen, der eine Feinabstimmung des Verhaltens des Operators innerhalb bestimmter Grenzen erlaubt. Aufgrund des komplexen Aufbaus der meisten parametrisierten Operatoren steigt mit der Höhe der Adaptionsfähigkeit meist der numerische Aufwand.

Numerischer Aufwand

Spielt neben dem semantischen Verhalten eines Operator auch die Verarbeitungsdauer eine Rolle, wird oft die Auswahl der Operatoren auf diejenigen beschränkt sein, die nur einen niedrigen Verarbeitungsaufwand erfordern.

Aggregationsverhalten

Das Aggregationsverhalten beschreibt, wie sich das Ergebnis bei einer erhöhten Anzahl von zu verknüpfenden Zugehörigkeitsfunktionen verändert. So verringert sich bei Anwendung des algebraischen Produktes mit erhöhter Zahl von unscharfen Mengen tendenziell der maximale Zugehörigkeitsgrad des Ergebnisses.

---

[73] Vgl. Zimmermann (1991) , S. 42.

Skalenverhalten

Ein Operator ist um so universeller anwendbar, je geringer seine Anforderungen an das vorhandene Skalenniveau der Zugehörigkeitsfunktionen des Datenmaterials sind. Das min/max-Operatorenpaar ist bereits ab einem Ordinalskalenniveau einsetzbar, während die Verwendung von algebraischem Produkt/Summe intervall-skalierte Zugehörigkeits-funktionen voraussetzt.

Definitionsgemäß erfüllen sowohl die nicht-parametrischen als auch die parametrischen t-Normen und s-Normen die Eigenschaften der Assoziativität, der Kommutativität, der Monotonie und der De-Morgan-Gesetze.

2.3.10 Beispiel zu den Eigenschaften der Operatoren

Die, bereits beschriebene[74], scharfe Menge T dient als Grundlage zur Ermittlung der unscharfen Menge

A = {(5;0,3), (6;0,4), (7;0,6), (8;0,9), (9;0,6)},

die die Aussage „Wochenproduktion von Wohnzimmerschränken zu vertretbaren Stückkosten“ beschreibt. Aufgrund von Marktbeobachtungen wurde eine zweite unscharfe Menge

B = {(5;1,0), (6;1,0), (7;1,0), (8;0,8), (9;0,3)}

ermittelt, die die „Menge der pro Woche absetzbaren Wohnzimmerschränke“ angibt. Mit Hilfe der vorgestellten Operatoren soll nun diejenige Produktionsmenge ermittelt werden, welche diese Aussagen zum höchsten Grade erfüllt. Zur Ermittlung des Ergebnisses in Form der unscharfen Menge **C** werden die beiden Ergebnisse mit den vorgestellten t-Norm-Operatoren verknüpft.

Menge x	$\mu_A(x)$	$\mu_B(x)$	min(...)	$alg_t(...)$	$bes_t(...)$	$quo_t(...)$	$dra_t(...)$
5,00	0,30	0,80	0,30	0,24	0,10	0,28	0,00
6,00	0,40	0,20	0,20	0,08	0,00	0,15	0,00
7,00	0,70	1,00	0,70	0,70	**0,70**	0,70	**0,70**
8,00	0,90	0,80	**0,80**	**0,72**	**0,70**	**0,73**	0,00
9,00	0,60	0,30	0,30	0,18	0,00	0,25	0,00

Tabelle 1: Vergleich der Operatoren

Aus Tabelle 1 geht hervor, daß die Verknüpfung mit dem min-Operator eine Wochenproduktion von 8 Schränken stark präferiert. Bei Anwendung von algebraischem Produkt und algebraischem t-Quotient werden die Produktionsmengen 7 und 8 Stück

[74] Vgl Abschnitt 2.3.2.1.

ungefähr gleich stark präferiert. Bei Anwendung der beschränkten Differenz erreichen die Produktionsanzahlen von 7 oder 8 Stück die gleiche Bewertung. Das drastische Produkt liefert eine eindeutige Entscheidung für die Produktion von 7 Schränken, die Produktion von 8 Schränken wird „drastisch“ abgelehnt. Aufgrund der auf den jeweiligen Einsatzzweck bezogenen Eignung von Fuzzy-Operatoren, ergibt sich die Vielfalt an möglichen Lösungen, deren Aussagekraft und Relevanz empirisch überprüft werden muß.

2.3.11 Parametrisierte t- und s-Normen

Die folgenden parametrisierten Operatorentypen erlauben durch die Angabe eines festen reellwertigen Parameters γ die problemorientierte Anpassung der t-Norm und der dazugehörigen s-Norm. Das einzelne Operatorenpaar läßt sich stufenlos durch Variation des reellwertigen Parameters aus der Operatorenfamilie der in Bild 2.7 angegebenen, Ordnung der t-Normen und s-Normen auswählen.

Hamacher-Operatorenfamilie

Die Hamacher-Operatorenfamilie erlaubt die stufenweise Bewegung im t-Normen-Bereich zwischen dem algebraischen t-Quotient und dem drastischen Produkt. Im s-Normen-Bereich bewegt sich die Hamacher-Operatorenfamilie zwischen der drastischen Summe und dem algebraischen s-Quotient.

Definition 2.25: Hamacher-Operatorenfamilie

Mit Hilfe der Hamacher-Operatorenfamilie[75] und Variation von $\gamma \in [0;\infty\rangle$ läßt sich durch

$$H_t(x,y,\gamma) = \frac{xy}{\gamma+(1-\gamma)(x+y-xy)} \qquad \text{mit } x,y\in[0;1]$$

eine t-Norm, und durch

$$H_s(x,y,\gamma) = \frac{x+y-(2-\gamma)xy}{1-(1-\gamma)xy} \qquad \text{mit } x,y\in[0;1]$$

eine s-Norm modellieren.

Der Laufbereich der Hamacher-Operatorenfamilie im Bereich der t- und s-Normen wird in Bild 2.8 dargestellt[76].

[75] Vgl. Bothe (1993), S. 55.

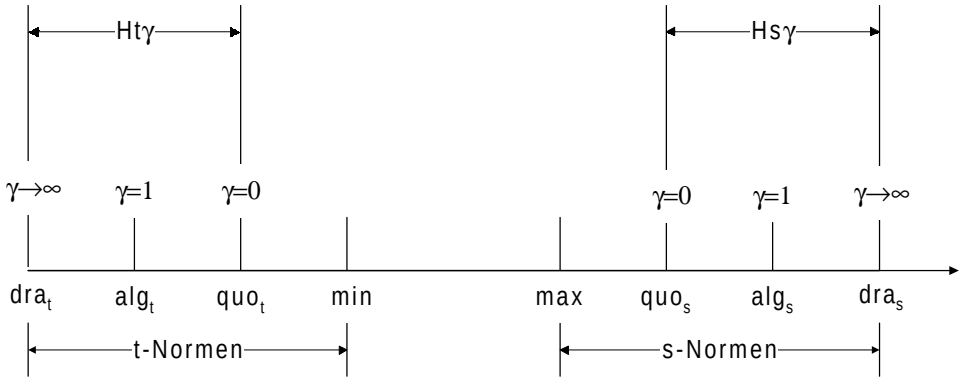


Bild 2.8: Laufbereich der Hamacher-Operatorenfamilie

Yager-Operatorenfamilie

Eine Alternative zur Hamacher-Operatorenfamilie ist die Yager-Operatorenfamilie, deren Bandbreite sämtliche t-Normen und s-Normen abdeckt.

Definition 2.26: Yager-Operatorenfamilie

Die Yager-Operatorenfamilie[77] modelliert durch Variation von $\gamma \in \langle 0;\infty \rangle$ sowohl eine t-Norm

$$Y_t(x,y,\gamma) = 1-\min(1, \sqrt[\gamma]{(1-x)^\gamma+(1-y)^\gamma}) \qquad \text{mit } x,y \in [0;1]$$

als auch eine s-Norm

$$Y_s(x,y,\gamma) = \min(1, \sqrt[\gamma]{x^\gamma+y^\gamma}) \qquad \text{mit } x,y \in [0;1].$$

Der Laufbereich der YAGER-Operatorenfamilie bewegt sich bei Variation des festen Parameters γ vom drastischen Produkt(Summe) zum Minimum(Maximum)-Operator. Dieser Zusammenhang ist in Bild 2.9 dargestellt[78].

[76] Nach Bothe (1993), S. 56.
[77] Vgl. Bothe (1993), S. 56.
[78] Nach Bothe (1993), S. 57.

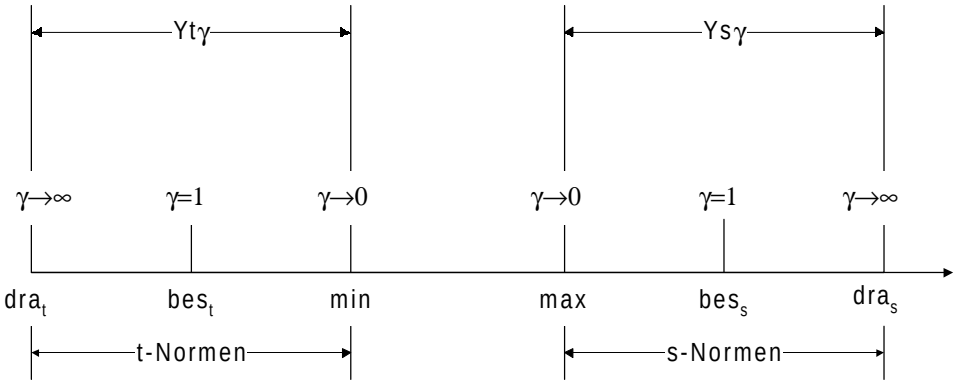


Bild 2.9: Laufbereich der Yager-Operatorenfamilie

Eine Übersicht über weitere parametrisierte Operatorenfamilien findet sich bei MEYER ET AL..[79].

2.3.11.1 Bewertung der parametrisierten Operatorenfamilien

Parametrisierte Operatorenfamilien erlauben eine genaue graduelle Justierung des verwendeten Durchschnittes und der Vereinigung von unscharfen Mengen. Mit dem Zugewinn an Flexibilität stellt sich jedoch das Problem der für die Anwendung adäquaten Parameterwahl. Nach ROMMELFANGER[80] sprechen empirische Untersuchungen gegen die Verwendung von Operatoren, die zu größeren Werten als der Maximum-Operator, bzw. zu kleineren Werten als der Minimum-Operator führen.

2.3.12 Durchschnittsoperatoren

Aufgrund der von der Anordnungskette der t-Normen und der s-Normen vorgegebenen Reihenfolge wird das Abbildungsverhalten des Maximum-Operators als pessimistisch (und somit das Abbildungsverhalten des Minimum-Operators als optimistisch) beschrieben[81]. Zur Abbildung von menschlichen Entscheidungsregeln werden Operatoren benötigt, welche eine kompensatorische Verknüpfung zur Verbindung der Bedingungen ermöglichen. Derartige Operatoren tragen der Erkenntnis Rechnung, daß im alltäglichen Sprachgebrauch die Bedeutungen von *und* bzw. *und* stufenlos variieren können. Die Darstellung von Operatoren, mit denen sich konkrete Realweltsituationen modellieren ließen, steht mehr im Blickpunkt als die Orientierung am Rahmenwerk der t-Normen. Je nach Situation haben die Konnektive[82] *und* und *oder* im Alltagsgebrauch kompensatorische Eigenschaften, die sich oft in einer geeignet gewichteten Kombination

79 Vgl. Mayer et al. (1993), S. 38f.
80 Vgl. Rommelfanger (1994), S. 25.
81 Vgl. Rommelfanger (1994), S. 27.
82 Vgl. Zimmermann, Zysno (1980), S. 37.

von Disjunktion und Konjunktion darstellen lassen. Die Klasse der Durchschnittsoperatoren - die ein sprachliches *sowohl-als-auch* ausdrücken - bewegt sich zwischen den numerisch kleineren t-Normen und den numerisch größeren s-Normen. Diese Einordnung ist in Bild 2.10 dargestellt.

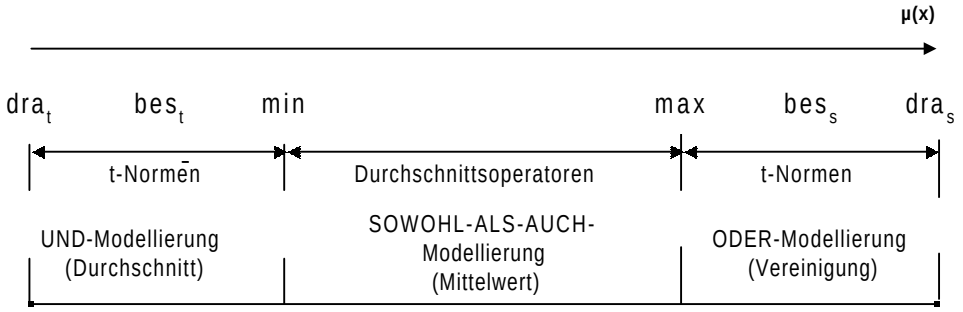


Bild 2.10: Anordnung der Durchschnittsoperatoren

2.3.12.1 Nicht-parametrisierte Durchschnittsoperatoren

In Tabelle 2 werden die wichtigsten nicht-parametrisierten Durchschnittsoperatoren vorgestellt.

Bezeichnung des Operators	Rechenvorschrift ($x,y \in [0;1]$)
Arithmetisches Mittel	$d_{am}(x,y) = \frac{x+y}{2}$
Geometrisches Mittel	$d_{gm}(x,y) = \sqrt[2]{xy}$
Harmonisches Mittel	$d_{hm}(x,y) = \frac{2xy}{x+y}$
Doppelt geometrischer Mittelwert	$d_{dgm}(x,y) = 1-\sqrt{(1-x)(1-y)}$
Doppelt harmonischer Mittelwert	$d_{dhm}(x,y) = \frac{x+y-2xy}{2-x-y}$

Tabelle 2: Nicht parametrisierte Durchschnittsoperatoren

2.3.12.2 Parametrisierte Durchschnittsoperatoren

Die wichtigsten parametrisierten Durchschnittsoperatoren auf zwei unscharfen Mengen, die zur gewichteten Abbildung (durch Variation des Kompensationsgrades $\gamma \in [0;1]$) des sprachlichen *sowohl-als-auch* dienen können, sind in Tabelle 3 dargestellt.

Bezeichnung des Operators	Rechenvorschrift ($x,y,\gamma \in [0;1]$)
Algebraische Kompensation	$A(x,y,\gamma) = alg_t(x,y)^{1-\gamma} * alg_s(x,y)^{\gamma}$
Min-Max-Kompensation	$K(x,y,\gamma) = \min(x,y)^{1-\gamma} * \max(x,y)^{\gamma}$
Konvexe Min-Max-Kompensation	$kK(x,y,\gamma) = (1-\gamma)\min(x,y) + \gamma\max(x,y)$

Tabelle 3: Parametrisierte Durchschnittsoperatoren

Definition 2.27: Kompensatorisches Und

ZIMMERMANN[83] stellt mit dem *kompensatorischen und*

$$\mu_{Komp}(x) = \left(\prod_{i=1}^{m} \mu_i(x)\right)^{(1-\gamma)} \left(1 - \prod_{i=1}^{m} (1-\mu_i(x))\right)^{\gamma}$$

eine Verallgemeinerung der algebraischen Kompensation für eine beliebige Anzahl von Parametern zur Verfügung, die nicht nur die Extremwerte der Operatoren berücksichtigt.

2.3.12.3 Anwendung von parametrisierten Durchschnittsoperatoren

In der folgenden Situation bietet sich die Anwendung eines parametrisierten Durchschnittsoperators an: Ein Unternehmen nimmt dann einen zusätzlichen Auftrag an, wenn dieser einen hohen Deckungsbeitrag verspricht *und* ausreichend Maschinenkapazität verfügbar ist. Diese beiden Bedingungen sind unscharf und lassen sich durch unscharfe Mengen ausdrücken. Das Anspruchsniveau liegt bei 0,75. Verknüpft man beide Aussagen mit *min*, so würden bei Beurteilung der Verfügbarkeit von Maschinenkapazität mit 0,3 sehr lukrative Zusatzaufträge mit einem Deckungsbeitrag, welcher mit einer Bewertung von 0,9 als hoch angesehen wird, abgelehnt. Diese nicht-kompensatorische Verknüpfung *min* liefert ein Ergebnis von 0,3. Eine Verknüpfung mittels des arithmetischen Mittelwertes würde in einer Gesamtbewertung von 0,6 resultieren. Die algebraische Kompensation (nach Tabelle 3) mit einem Parameter von 0,9 liefert den Wert von ca. 0,82. Die eindeutige Dominanz des Konjunktionscharakters gegenüber dem Disjunktionscharakter läßt sich an der Nähe zum Ergebnis einer Verknüpfung mit der algebraischen Summe (0,93) erkennen. Diese Bewertung würde zu einer Überschreitung des Anspruchsniveaus und zu einer Annahme des Auftrags führen.

[83] Vgl. Zimmermann (1991), S. 37.

2.3.12.4 Wahl des Kompensationsgrades

Die Wahl des Kompensationsgrades γ wird von der jeweiligen Problemstellung abhängen, ZIMMERMANN und ZYSNO[84] beschreiben dazu einen praktischen Ansatz.

2.3.13 Unscharfe Zahlen

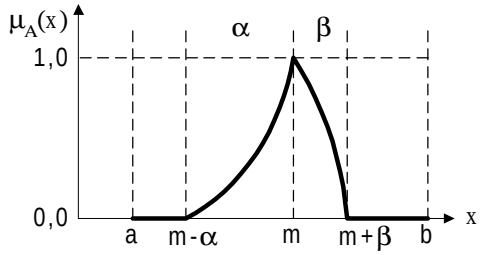


Bild 2.11: Unscharfe Zahl

Unscharfe Zahlen bilden einen Sonderfall der unscharfen Mengen. Sie erlauben die Darstellung von unscharfen reellwertigen Größen, die z. B. durch die Aussage „Für diesen Zusatzauftrag werden ungefähr 500 Mannstunden benötigt“ beschrieben werden. Nur genau an der Stelle m (=500 in der Beispielaussage) nimmt die normalisierte Zugehörigkeitsfunktion ihren Maximalwert 1,0 an. Der Grad der Unschärfe wird durch die Spannweiten α und β wiedergegeben. Die Zugehörigkeitsfunktion steigt von m-α bis zu m monoton an und fällt von m bis m+β monoton. Die einer unscharfen Zahl zugrundeliegende unscharfe Menge ist konvex. Der Verlauf der Zugehörigkeitsfunktion einer unscharfen Zahl ist in Bild 2.11 abgebildet.

[84] Vgl. Zimmermann, Zysno (1980), S. 46.

Definition 2.28: Unscharfe Zahl

Eine unscharfe Menge **A** auf der Menge $\Re$ wird eine unscharfe Zahl[85] genannt, wenn

- **A** konvex ist und
- genau eine reelle Zahl x_0 existiert, für die $\mu_A(x) = 1{,}0$ gilt und
- $\mu_A(x)$ abschnittsweise stetig ist.

$\mu_A(x_0)$ wird auch Gipfelpunkt von **A** genannt. Eine unscharfe Zahl **A** ist positiv, wenn gilt: $\mu_A(x) = 0 \; \forall \; x \leq 0$. Sie ist negativ, wenn gilt: $\mu_A(x) = 0 \; \forall \; x \geq 0$.

2.3.13.1 Beispiel

Für einen Produktionsprozeß erstellt der Fertigungssteuerer eine Prognose über die Wartungszeiten eines Betriebsmittels in Stunden pro Monat. Er modelliert den Sachverhalt „Ausfallzeit für Maschine X“ aufgrund fehlender Vergangenheitsdaten mit einer unscharfen Zahl. Die mathematische Darstellung ist wie folgt:

$$\mu_A(x) = \begin{cases} \frac{1}{2}x-1 & 2 \leq x \leq 4 \\ \frac{-1}{4}x+2 & 4 \leq x < 8 \\ 0 & \text{sonst} \end{cases} .$$

Der Verlauf von „ungefähr 4“ findet sich in Bild 2.12 wieder.

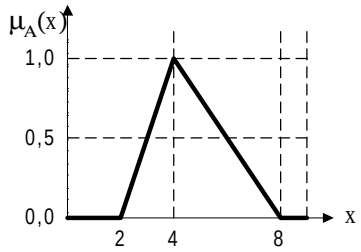


Bild 2.12: Unscharfe Zahl „ungefähr 4“

[85] Synonyme: Fuzzy-Zahl, Fuzzy-Number.

2.3.14 Diskrete unscharfe Zahlen

Bei der Modellierung von Sachverhalten aus der Praxis liegen oft nur vereinzelte Eckdaten auf einer Grundmenge vor, so daß die Stetigkeitsvoraussetzung laut obiger Definition nicht erfüllt sind. Dies führt zum Begriff der diskreten unscharfen Zahl:

Eine unscharfe Menge **A'** auf einer diskreten Grundmenge $X \subset \Re$ nennt man diskrete unscharfe Zahl, wenn eine unscharfe Zahl **A** auf $\Re$ existiert, so daß gilt:

$\mu_{A'}(x) = \mu_A(x) \; \forall \; x \in X.$

Die zu einer diskreten unscharfen Zahl **A'** gehörige unscharfe Zahl **A** auf $\Re$ erhält man durch Verbindung der Punkte mittels eines Polygonzuges.

2.3.15 Unscharfe Intervalle[86]

Neben unscharfen Zahlen sind auch unscharfe Intervalle bei der Modellierung von Unschärfe hilfreich. Sie können zur Modellierung von Zugehörigkeitsfunktionen, die durch das Verhalten von Meßreihen[87] vorgegeben sind, oder deren Kern ein Intervall ist, dienen. Eine konvexe normalisierte unscharfe Menge **I** wird als unscharfes Intervall bezeichnet, wenn:

- ein scharfes Intervall $M = [m_1,m_2]$ existiert, für das gilt: $\mu_I(x) = 1,.0 \; \forall \; x \in M$ und
- $\mu_I(x)$ abschnittsweise stetig ist.

Der 1-Schnitt eines unscharfen Intervalls ergibt wieder das scharfe Intervall M. ALIEV, BONFIG und ALIEW[88] bezeichnen unscharfe Intervalle als flache oder glatte unscharfe Zahlen. Die Darstellung in Bild 2.13 zeigt ein unscharfes Intervall.

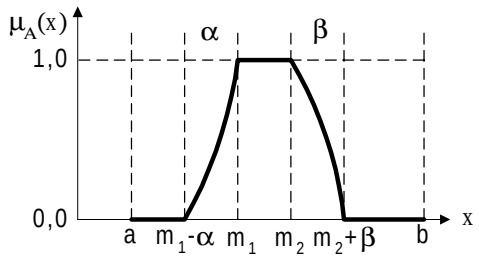


Bild 2.13: Unscharfes Intervall

86 Vgl. Bothe (1995), S. 60.
87 Vgl. Bothe (1995), S. 85.
88 Vgl. Aliev et al. (1995), S. 89.

2.4 Fuzzy-Arithmetik

Die Betrachtung einer unscharfen Menge reicht i. d. R. nicht aus, um komplexe Sachverhalte zu modellieren. Insbesondere benötigt man zur Aggregation von unscharfen Mengen Operationen, die ein Schema für die elementaren arithmetischen Verknüpfungen von unscharfen Zahlen und Intervallen zur Verfügung stellen.

2.4.1 Das Erweiterungsprinzip

Ein grundlegendes Konzept zur Behandlung unscharfer Zahlen ist das Erweiterungsprinzip (EWP), welches die Fuzzifikation der klassischen Strukturen und Verfahren der reellwertigen Algebra gestattet, d. h. klassische arithmetische Verfahren können universell auf unscharfe Zahlen[89] übertragen werden. Das Grundprinzip des EWP ist, die Zugehörigkeitsfunktion einer unscharfen Menge **A** auf dem Träger X durch eine funktionale Abbildung in die Zugehörigkeitsfunktion der unscharfen Menge **B** auf dem Träger Y zu übertragen.

$$\mathbf{A} = \{(x;\mu_A(x))\}$$

$$\mathbf{B} = f(\mathbf{A}) = \{(f(x),\mu_A(x))\}$$

Führen mehrere Elemente aus X zu einem Element aus Y, dann ist derjenige Funktionswert mit dem maximalen Zugehörigkeitsgrad so zu wählen, daß folgendes gilt:

$\mu_B(y) = \sup_x(\mu_A(x))$**,** bzw. $\mu_B(y) = \mu_A(x)$ im bijektiven[90] Falle.

Definition 2.29: f-Erweiterung

Seien **A**, **B** und **C** unscharfe Mengen auf den Trägermengen X,Y bzw. Z mit den Zugehörigkeitsfunktionen $\mu_A(x)$ und $\mu_B(y)$. Es gilt $x \in X$, $y \in Y$ sowie $z \in Z$ und $x,y,z \in [0;1]$. Es sei

$f: X \times Y \rightarrow Z$ gemäß $f(x,y) = z$.

Als f-Erweiterung des kartesischen Produktes $\mathbf{A} \times \mathbf{B}$ sei die unscharfe Menge $\mathbf{C} = f(\mathbf{A},\mathbf{B})$ erklärt. Es gilt $\mu_C(z) = \mu_{f(A,B)}(z) = \max(\min(\mu_A(x),\mu_B(y)) \ \forall \ x,y,z$ für die $f(x,y) = z$ gilt. Eine Darstellung der f-Erweiterung des kartesischen Produktes zwischen zwei unscharfen Mengen findet sich in Tabelle 4.

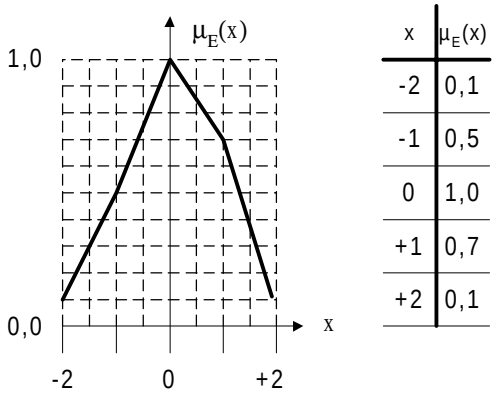
---

89 Vgl. Böhme (1993), S. 141f.
90 Vgl. Opitz (1991), S. 126.

2.4.2 Beispiel zum Erweiterungsprinzip

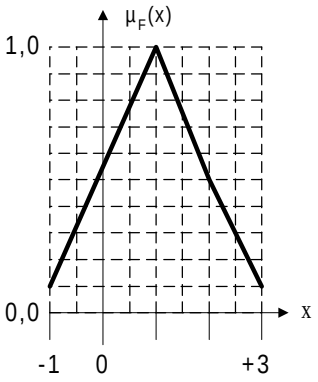
Die unscharfen Mengen **E** und **F** repräsentieren zwei relative Bereitstellungszeitpunkte für zwei Komponenten E und F einer Baugruppe G. Zur Ermittlung des frühesten Montagezeitpunktes **G** für die Baugruppe G werden diese beiden unscharfen Werte nach dem Erweiterungsprinzip addiert. Dazu wird eine zweistellige klassische Additionsfunktion f(x,y) = x+y angewandt.

Es sei die unscharfe Zahl **E** über der Trägermenge X = {-2,-1,0,+1,+2} und die unscharfe Zahl **F** über der Trägermenge Y = {+2+,+3+,+4,+5,+6} definiert. Eine graphische Darstellung der unscharfen Zahl **E** findet sich in Bild 2.14, die unscharfe Zahl **F** ist in Bild 2.15 dargestellt.



x	$\mu_E(x)$
-2	0,1
-1	0,5
0	1,0
+1	0,7
+2	0,1

Bild 2.14: Unscharfe Zahl **E**



x	$\mu_F(x)$
-1	0,1
0	0,7
+1	1,0
+2	0,5
+3	0,1

Bild 2.15: Unscharfe Zahl **F**

Es wird das unscharfe kartesische Produkt **K** der unscharfen Mengen **E** und **F** ermittelt. Für die einzelnen Elemente g = e+f des kartesischen Produktes wird je ein potentieller Zugehörigkeitsgrad $\min(\mu_E(e), \mu_F(f))$ ermittelt.

		-2	-1	0	1	2
		(0,10)	(0,50)	(1,00)	(0,70)	(0,10)
-1	(0,30)	(0,10)	(0,30)	(0,30)	(0,30)	(0,10)
0	(0,70)	(0,10)	(0,50)	(0,70)	(0,70)	(0,10)
1	(1,00)	(0,10)	(0,50)	(1,00)	(0,70)	(0,10)
2	(0,50)	(0,10)	(0,50)	(0,50)	(0,50)	(0,10)
3	(0,10)	(0,10)	(0,10)	(0,10)	(0,10)	(0,10)

Tabelle 4: F-Erweiterung

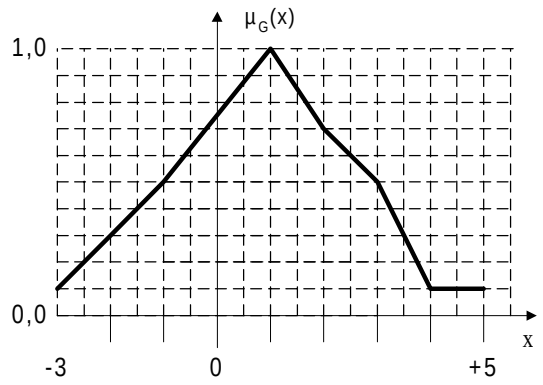
Die Trägermenge Z der resultierenden unscharfen Menge **G** setzt sich aus den verschiedenen Resultaten der f-Erweiterung des kartesischen Produktes zusammen. Als Zugehörigkeitsgrad wird für jedes Element aus Z der maximale Zugehörigkeitsgrad ermittelt:

$$\mu_G(z) = \mu_{E+F}(z) = \sup_{z=x+y} \min(\mu_E(x), \mu_F(y)) .$$

Tabelle 5 zeigt die notwendigen Verknüpfungen. Die resultierende unscharfe Zahl **G** ist in Bild 2.16 graphisch dargestellt.

x	μ(x)		
-3	μ(((-2)+(-1))	= max(0,10)	= 0,10
-2	μ(((-2)+(-0) oder ((-1) + 1))	= max(0,30;0,10)	= 0,30
-1	μ(((-2)+1) oder ((-1)+0) oder (0+(-1)))	= max(0,10;0,50;0,30)	= 0,50
0	μ(((-2)+2) oder ((-1)+1) oder (0+0) oder (1+(-1)))	= max(0,10;0,50;0,70;0,30)	= 0,70
1	μ(((-2)+3) oder ((-1)+2) oder (0+1) oder (1+0) oder (2+(-1))	= max(0,10;0,50;1,00;0,70;0,10)	= 1,00
2	μ((-1+3) oder (0+2) oder (1+1) oder (2+0))	= max(0,10;0,50;0,70;0,10)	= 0,70
3	μ((0+3) oder (1+2) oder (2+1) oder (3+0))	= max(0,10;0,50;0,10)	= 0,50
4	μ((1+3) oder (2+2))	= max(0,10;0,10)	= 0,10
5	μ(2+3)	= max(0,10)	= 0,10

Tabelle 5: Ermittlung der Zugehörigkeitsgrade der unscharfen Zahl G



x	$\mu_G(x)$
-3	0,1
-2	0,3
-1	0,5
0	0,7
1	1,0
2	0,7
3	0,5
4	0,1
5	0,1

Bild 2.16: Unscharfe Zahl **G**

Bei der Betrachtung der unscharfen Zahl **G** stellt man fest, daß die x-Koordinate des Gipfelpunktes ebenfalls beim Wert 1.0 liegt, welches die Summe aus den Gipfelpunkten von **E** und **F** ist. Die Unsicherheit von **G**, die sich in der, gegenüber **E** und **F,** verbreiterten Spannweite bemerkbar macht, beinhaltet die akkumulierte Unsicherheit der unscharfen Zahlen **E** und **F**.

Zusammenfassend lassen sich die arithmetischen Operationen auf unscharfen Zahlen nach dem Erweiterungsprinzip wie in Tabelle 6 aufgeführt darstellen.

Operation	Rechenvorschrift
Addition	$\mu_{A+B}(z)=\sup_{z=x+y}\min(\mu_A(x),\mu_B(y) = \sup \min(\mu_A(x),\mu_B(z-x))$
Subtraktion	$\mu_{A-B}(z)=\sup_{z=x-y}\min(\mu_A(x),\mu_B(y) = \sup \min(\mu_A(x),\mu_B(x-z))$
Multiplikation	$\mu_{A*B}(z)= \sup_{z=x*y}\min(\mu_A(x),\mu_B(y)) = \sup \min(\mu_A(x),\mu_B(z/x)$
Division	$\mu_{A/B}(z)= \sup_{z=x/y}\min(\mu_A(x),\mu_B(y)) = \sup \min(\mu_A(x),\mu_B(x/z)$

Tabelle 6: Arithmetische Operationen nach dem Erweiterungsprinzip[91]

2.4.3 LR-Zahlen und ihre arithmetische Behandlung

Das Rechnen mit unscharfen Zahlen nach dem Erweiterungsprinzip gestaltet sich wegen der Mengenproduktbildung numerisch sehr aufwendig[92]. Werden beispielsweise vier unscharfe Zahlen mit je 10 Stützpunkten nach dem Erweiterungsprinzip verknüpft, so benötigt man bereits allein 10^4=10.000 Rechenschritte zur Ermittlung des kartesischen Produkts, wobei zusätzlich der rechnerische Aufwand zur nachfolgenden Supremum- und Minimumbildung sowie die Komplexität der Funktion f(u,v,w,x) mitberücksichtigt werden muß. Um arithmetische Operationen auf unscharfen Zahlen effizient behandeln zu können, ist es notwendig, bestimmte Restriktionen bezüglich der Zugehörigkeitsfunktionen einzuführen. Eine Vereinfachung ergibt sich, wenn die beiden Teiläste der unscharfen Zahl einem bestimmten Verlaufstyp entsprechen. Eine unscharfe Zahl kann somit durch einen bestimmten Verlaufstyp charakterisiert werden. Werden zwei unscharfe Zahlen arithmetisch miteinander verknüpft, so entsteht - u. U. auch erst nach einer Approximation - als Ergebnis wieder eine unscharfe Zahl dieses Verlaufstyps. Die Verlaufsform der linken und rechten Teiläste einer unscharfen Zahl wird nach DUBOIS/PRADE[93] durch die Referenzfunktionen L(x) und R(x) angegeben. Die durch sie dargestellten unscharfen Zahlen werden als LR-Zahlen bezeichnet. Handelt es sich bei diesen Referenzfunktionen um lineare Funktionen, so erhält man LR-Zahlen in Dreiecksform. L(x) und R(x) können voneinander verschieden sein. In Definition 2.30 erfolgt die formale Darstellung des oben genannten Sachverhaltes.

91 Vgl. Demandt (1993), S. 36.
92 Vgl. Mayer et al. (1993), S. 54.
93 Vgl. Dubois, Prade (Fuzzy Numbers) (1993), S. 128.

Definition 2.30: Referenzfunktion

Eine Funktion R: $\Re_0^+ \rightarrow [0;1]$ wird Referenzfunktion genannt, wenn

- R(0) = 1,
- R(1) = 0 und
- R im Bereich $\Re_0^+$ monoton fallend ist.

Es gibt verschiedene Familien von Referenzfunktionen. Es stellt sich allerdings das Problem, eine dem Anwendungsfall entsprechende Referenzfunktion auszuwählen. Für die meisten Fälle bietet sich aufgrund der numerischen Einfachheit die Referenzfunktion R(x) = max(0,1-x) an. Sind L(x) und R(x) lineare Funktionen, so spricht man von trapezoidalen Fuzzy-Zahlen.

Definition 2.31: LR-Zahl

Kann eine unscharfe Zahl **M** durch eine linke und eine rechte Referenzfunktion L(x) bzw. R(x) und die reellwertigen Schwankungsbreiten α und β und *m* beschrieben werden, so wird sie LR-Zahl um den Gipfelpunkt *m* genannt. Ihre Zugehörigkeitsfunktion ist wie folgt definiert:

$$\mu_M(x)=\begin{cases} L\left(\frac{m-x}{\alpha}\right) & x \le m \\ R\left(\frac{x-m}{\beta}\right) & x > m \end{cases} .$$

Eine LR-Zahl kann durch das Tripel

$\mathbf{M}=(m,\alpha,\beta)_{LR}$

dargestellt werden.

Je größer ihre Schwankungsbreiten sind, desto größer ist die Unschärfe, die der LR-Zahl innewohnt. Sind diese null, liegt der Sonderfall der scharfen Zahl *m* vor. Unscharfe Zahlen können auch als gewichtetes Intervall um ihren Häufungspunkt interpretiert werden; lineare Operationen auf ihnen lassen sich auf Intervalloperationen abbilden. LR-Zahlen bieten den Vorteil der Einfachheit und der numerischen Effizienz[94]. Hinsichtlich der Addition und Multiplikation bilden sie jedoch keinen Gruppenverband. Die LR-Zahlen sind nicht abgeschlossen in Bezug auf die Multiplikation; denn die resultierenden Referenzfunktionen sind nicht mehr linear. Das Produkt zweier LR-Zahlen kann nur

[94] Vgl. Mayer et al. (1993), S. 59f.

approximativ in eine LR-Darstellung überführt werden kann. Bild 2.17 gibt eine graphische Beschreibung einer LR-Zahl.

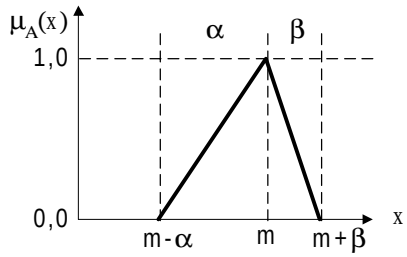


Bild 2.17: LR-Zahl

2.4.3.1 Arithmetische Verknüpfung von LR-Zahlen

Die von Dubois/Prade[95] beschriebenen Verfahren zur arithmetischen Verknüpfung von LR-Zahlen liefern ein numerisch effizientes Formelwerk, mit dem sich der Berechungsaufwand gegenüber der Anwendung des Erweiterungsprinzips stark verringern läßt.

Definition 2.32: Addition von LR-Zahlen

Die Addition zweier LR-Zahlen $O = (o,\alpha,\beta)_{LR}$ und $P = (p,\gamma,\delta)_{LR}$ wird nach folgender Vorschrift vorgenommen:

$$\mathbf{O} + \mathbf{P} = (o,\alpha,\beta)_{LR}+(p,\gamma,\delta)_{LR} = (o+p,\alpha+\gamma,\beta+\delta)_{LR}.$$

Die Addition von LR-Zahlen mit unterschiedlichen Referenzfunktionen beschreibt BÖHME[96].

Definition 2.33: Negation von LR-Zahlen

Die Negation einer LR-Zahl $O = (o,\alpha,\beta)_{LR}$ wird nach folgender Vorschrift vorgenommen:

$$-\mathbf{O} = -(o,\alpha,\beta)_{LR} = (-o, \beta,\alpha)_{RL}.$$

Definition 2.34: Subtraktion von LR-Zahlen

Analog zur reellwertigen Subtraktion zweier Zahlen a-b = a+(-b) kann nun die Definition der Subtraktion von LR-Zahlen abgeleitet werden:

---

95 Vgl. Dubois,D., Prade,H. (Fuzzy Numbers) (1993): S. 112 - 148.
96 Vgl. Böhme (1993), S. 127.

$$\mathbf{O}\text{-}\mathbf{P} = (o,\alpha,\beta)_{LR}\text{-}(p,\gamma,\delta)_{LR} = (o,\alpha,\beta)_{LR}+(-p,\delta,\gamma)_{RL} = (o\text{-}p,\ \alpha+\delta,\beta+\alpha)_{LR}.$$

Definition 2.35: Skalar-Multiplikation einer LR-Zahl

Durch Anwendung der unscharfen Skalar-Multiplikation ist es möglich, eine reellwertige scharfe Zahl λ mit einer unscharfen Zahl **O** zu multiplizieren. Die Berechnungsvorschriften hängen dabei vom Vorzeichen der Zahl λ ab, so daß eine Fallunterscheidung vorgenommen werden muß:

$$\lambda\mathbf{O} = \begin{cases} (\lambda o,\lambda\alpha,\lambda\beta)_{LR} & \text{für } \lambda \geq 0 \\ (\lambda o,-\lambda\beta,\lambda\alpha)_{RL} & \text{sonst} \end{cases}.$$

Definition 2.36: Multiplikation von LR-Zahlen

Werden zwei unscharfe Zahlen multiplikativ verknüpft, so wird man das Ergebnis nur approximativ ermitteln, da die Ermittlung des gesamten Verlaufs der Zugehörigkeitsfunktion numerisch enorm aufwendig ist. Böhme zeigt, daß die unscharfe Produktbildung von LR-Zahlen nicht abgeschlossen ist. Das Produkt zweier LR-Zahlen läßt sich nicht in LR-Form darstellen. Man weicht deshalb auf eine Alternative in Form einer Approximation aus, in der wieder nach den Vorzeichen der Faktoren unterschieden wird:

Wenn $\mathbf{O} > 0$, $\mathbf{P} > 0$, dann gilt[97]:

$$\mathbf{O}*\mathbf{P} = (o,\alpha,\beta)_{LR*}(p,\gamma,\delta)_{LR} \approx (op,o\gamma+p\alpha-\alpha\gamma,o\delta+p\beta+\beta\delta)_{LR}$$

Haben beide Zahlen relativ kleine Spannweiten, so läßt sich die Formel wie folgt vereinfachen[98]:

$$\mathbf{O}*\mathbf{P} = (o,\alpha,\beta)_{LR*}(p,\gamma,\delta)_{LR} \approx (op,o\gamma+p\alpha,o\delta+p\beta)_{LR}$$

Der erheblich geringere Berechnungsaufwand, gegenüber der Ermittlung mittels des Erweiterungsprinzips, wiegt, laut MAYER[99], das durch Approximation gewachsene Maß an Unschärfe auf. Die Güte der Approximation nimmt mit zunehmendem Abstand vom Gipfelwert ab.

---

97 Zur Berechnung bei anderen Vorzeichenkonstellationen, vgl. Mayer et al. (1993), S. 60.
98 Vgl. Böhme (1993), S. 136.
99 Vgl. Mayer et al.(1993), S. 60.

Definition 2.37: Kehrwert und Division von LR-Zahlen

Der reziproke Wert einer unscharfen Zahl wird benötigt, um eine Division zweier unscharfer Zahlen vornehmen zu können. Analog zur Division bei scharfen Zahlen kann bei unscharfen Zahlen auch mit dem Kehrwert des Divisors unscharf multipliziert werden. Der Kehrwert einer positiven unscharfen Zahl $\mathbf{A} = (m,\alpha,\beta)$ läßt sich wie folgt darstellen:

$$\mathbf{A^{-1}} = (m,\alpha,\beta)^{-1}_{LR} = \left(\frac{1}{m}, \frac{\beta}{m^2}, \frac{\alpha}{m^2}\right)_{RL}.$$

Der Quotient zweier positiver LR-Zahlen $\mathbf{O} = (o,\alpha,\beta)$ und $\mathbf{P} = (p,\gamma,\delta)$ läßt sich näherungsweise durch die LR-Zahl

$$\frac{\mathbf{O}}{\mathbf{P}} = \mathbf{O*P^{-1}} \approx \left(\frac{o}{p}, \frac{o\delta+p\alpha}{p^2}, \frac{o\gamma+p\beta}{p^2}\right)_{LR}$$

darstellen.

2.4.3.2 LR-Darstellung unscharfer Intervalle

Ähnlich der Darstellung unscharfer Zahlen können auch unscharfe Intervalle mit Hilfe von Referenzfunktionen abgebildet werden. BÖHME[100] beschreibt die unscharfen Intervalle auch als Fuzzy-Plateauzahlen.

Definition 2.38: Zugehörigkeitsfunktion eines unscharfen LR-Intervalls

Kann ein unscharfes Intervall $\mathbf{M}$ durch eine linke und eine rechte Referenzfunktion L(x) bzw. R(x) und die reellwertigen Schwankungsbreiten α und β sowie durch das Intervall $I = [m_1, m_2]$ beschrieben werden, so wird es LR-Intervall um das Gipfelplateau I genannt. Die entsprechende Zugehörigkeitsfunktion ist wie folgt definiert:

$$\mu_M(x) = \begin{cases} L\left(\frac{m_1-x}{\alpha}\right) & x \le m_1 \\ 1 & m_1 < x \le m_2 \\ R\left(\frac{x-m_2}{\beta}\right) & x > m_2 \end{cases}$$

Folgendes Quadrupel stellt ein LR-Intervall dar

$$\mathbf{M} = (m_1, m_1, \alpha, \beta)_{LR}.$$

[100] Vgl. Böhme (1993), S. 127.

LR-Intervalle ähneln den LR-Zahlen, gilt $m_1=m_2$ ergeben sich die unscharfen Zahlen als Spezialfall. L(x) und R(x) seien Referenzfunktionen, mit der Größe $c=m_2-m_1$ wird der Toleranzbereich des LR-Intervalls bezeichnet. Sind L(x) und R(x) lineare Funktionen, so spricht man von trapezoidalen Fuzzy-Mengen. Im folgenden werden die häufig benötigten Operatoren der Addition und der Multiplikation von LR-Intervallen definiert.

Definition 2.39: Addition von LR-Intervallen

Zwei positive LR-Intervalle $\mathbf{O} = (o_1,o_2,\alpha,\beta)_{LR}$ und $\mathbf{P} = (p_1,p_2,\chi,\delta)_{LR}$ werden analog zur unscharfen Addition von LR-Zahlen addiert

$$\mathbf{O}(o_1,o_2,\alpha,\beta)_{LR} + \mathbf{P}(p_1,p_2,\gamma,\delta)_{LR} = (o_1+p_1,o_2+p_2,\alpha+\gamma,\beta+\delta)_{LR}.$$

Definition 2.40: Multiplikation von LR- Intervallen

Zwei positive LR-Intervalle $\mathbf{O} = (o_1,o_2,\alpha,\beta)_{LR}$ und $\mathbf{P} = (p_1,p_2,\chi,\delta)_{LR}$ werden analog zur Produktbildung von LR-Zahlen multipliziert

$$\mathbf{O}(o_1,o_2,\alpha,\beta)_{LR} * \mathbf{P}(p_1,p_2,\gamma,\delta)_{LR} = (o_1p_1,o_2p_2,o_1\gamma p_1\alpha,\gamma,o_2\delta p_2\beta)_{LR}.$$

2.5 Plausibles Schließen

Zur Behandlung linguistisch beschriebener Regeln und Fakten ist die Nachbildung von Schlußfolgerungstechniken, die ein plausibles - der Art des menschlichen Schlußfolgerns nach *Faustregeln* ähnliches - Schließen erlauben, notwendig. Faustregeln sind heuristisches Wissen, welches eine gewisse Robustheit gegenüber kleinen Abweichungen der Beobachtung von der Prämisse einer Regel aufweist . Zur Sicherstellung einer solchen Robustheit finden bei der Anwendung des plausiblen Schließens Implikations-operatoren Verwendung. Das folgende Kapitel beschreibt die Bedeutung von Fuzzy-Implikationsoperatoren, bevor auf die Schlußfolgerungsmethoden nach dem generalisierten *Modus ponens* eingegangen wird.

2.5.1 Fuzzy-Implikationen

Von herausragender Bedeutung ist die Definition der Implikation. In der klassischen Logik wird oft mit der materiellen Implikation[101] gearbeitet. Man geht hierbei von der Vorstellung aus, daß die Implikation nur genau dann wahr ist, wenn auch ihre Prämisse

[101] Vgl. Zimmermann (Technologien) (1995), S. 6.

erfüllt ist. Schlußfolgerungsmethoden in einer unscharfen Umgebung führen speziell zu zwei Arten von Problematiken.

- Jede Art von Unsicherheit in den Vorbedingungen muß adäquat auf die Schlußfolgerungen übertragen werden.
- Wenn verschiedene Arten von Unsicherheit entstehen, so sollte es möglich sein, ein Gesamtmaß der Unsicherheit zu ermitteln.

Die Anwendung einer Wenn-Dann-Regel entspricht einer aussagenlogischen Implikation. Wird die verbale Darstellung einer Regel formal dargestellt, so ergibt sich:

$$a \rightarrow b \Leftrightarrow \bar{a} \vee b$$

Eine Umformung[102] der obigen Formel führt aufgrund der Gesetze der Aussagenlogik zu einer Disjunktion:

$$a \rightarrow b \Leftrightarrow (a \wedge b) \vee (\bar{a} \wedge 1)$$

Die Elemente dieser Formel werden - wie in Tabelle 7 angegeben - fuzzifiziert:

a	$\Rightarrow$	$\mathbf{A}$	$=$	$\{x, \mu_A(x) \mid x \in X\}$
b	$\Rightarrow$	$\mathbf{B}$	$=$	$\{y, \mu_B(y) \mid y \in Y\}$
$\bar{a}$	$\Rightarrow$	C$\mathbf{A}$	$=$	$\{x, 1-\mu_A(x) \mid x \in X\}$
$\bar{b}$	$\Rightarrow$	C$\mathbf{B}$	$=$	$\{y, 1-\mu_B(y) \mid y \in Y\}$
1	$\Rightarrow$	E	$=$	$\{x, 1 \mid x \in X\}$
0	$\Rightarrow$	$\varnothing$	$=$	$\{x, 0 \mid x \in X\}$
$\wedge \backslash \vee$	$\Rightarrow$	beliebige t-Norm/s-Norm-Kombination		

Tabelle 7: Fuzzifikation der klassischen Implikation

Wird für die logische Disjunktion und für die Konjunktion die Entsprechung des *min*/*max*-Operatorenpaares eingesetzt, so ergibt sich folgende Fuzzifikation:

WENN a DANN b $\Rightarrow$ WENN $\mathbf{A}$ DANN $\mathbf{B}$

$$a \rightarrow b \quad \Rightarrow \quad \mathrm{imp}(\mu_A, \mu_B)_{ZA}$$

$$(a \wedge b) \vee (\bar{a} \wedge 1) \quad \Rightarrow \quad \max(\min(\mu_A, \mu_B), \min(1-\mu_A, 1))$$

[102] Vgl. Jaanineh, Maijohann (1996), S. 181.

Aus dem klassischen Implikationsoperator konnte somit ein unscharfer Implikationsoperator abgeleitet werden.

Definition 2.41: Zadeh-Implikationsoperator

Nach Anwendung der Äquivalenzbeziehung $\min(1-\mu_A,1)=1-\mu_A$ kann der Zadeh-Implikationsoperator

$$imp_{ZA}(\mu_A,\mu_B) = \max(\min(\mu_A,\mu_B),1-\mu_A)$$

abgeleitet werden.

Neben dem Zadeh-Implikationsoperator können durch Ausnutzung der bei der Termumformung zur Verfügung stehenden Freiheitsgrade andere Implikationsoperatoren abgeleitet werden. Die in der Literatur beschriebenen Implikationsoperatoren[103] beschreibt Tabelle 8.

Name	Rechenvorschrift
Zadeh	$imp_{ZA}(\mu_A, \mu_B) = \max(\min(\mu_A+\mu_B), 1-\mu_A)$
Mamdani	$imp_{MA}(\mu_A, \mu_B) = \min(\mu_A,\mu_B)$
Kleene-Dienes	$imp_{KD}(\mu_A, \mu_B) = \max(1-\mu_A, \mu_B)$
Reichenbach	$imp_{RB}(\mu_A, \mu_B) = 1-\mu_A+ \mu_A\mu_B$
Lukasiewicz	$imp_{LU}(\mu_A, \mu_B) = \min(1, 1-\mu_A+\mu_B)$

Tabelle 8: Implikationsoperatoren

Bis auf den Implikationsoperator nach Mamdani beruhen die hier aufgeführten Operatoren auf der klassischen Implikation. Der Mamdani-Implikationsoperator beruht auf einer Näherungslösung. Er stellt die Grundlage für das approximative Schließen dar. Eine empirische Studie des Verhaltens von Implikationsoperatoren bei der Wahl verschiedener t-Normen und s-Normen findet sich bei TILLI[104]. Um mit dem generalisierten *Modus ponens* vereinbar zu sein, haben die hier vorgestellten Operatoren mit den klassischen Implikationsoperatoren gemeinsam, daß auch für sie der Fall eines falschen Schlusses aus einer wahren Prämisse ausgeschlossen werden muß[105].

[103] Vgl. Bothe (1995), S. 126f.; Jaanineh, Maijohann (1996), S.184.
[104] Vgl. Tilli (1992), S.180f.
[105] Vgl. Jaanineh, Maijohann (1996), S. 30.

2.5.2 Generalisierter Modus ponens

Ein wichtiger Anwendungsbereich der Fuzzy-Logik ist die Bildung unscharfer Schlußfolgerungen. Aufgrund von verbal beschriebenen Fakten und Regeln werden intuitive Schlußfolgerungen gewonnen. Ein Beispiel gibt die Schlußfigur[106] in Tabelle 9 wieder.

Implikation: Prämisse:	Wenn die Schlupfzeit *niedrig* ist, dann ist die Auftragspriorität *hoch*. Schlupfzeit ist *mittel*
Schluß:	Auftragspriorität ist *normal*

Tabelle 9: Schlußfigur

Die Mechanismen für das *plausible Schließen* lassen sich durch Verallgemeinerung der - aus der klassischen Aussagenlogik stammenden - Schlußfigur *Modus ponens* herleiten.

Die Darstellung des klassischen *Modus ponens* unter Gültigkeit der scharfen Prämisse p und der scharfen Folgerung q findet sich in Tabelle 10.

Implikation:	$p \rightarrow q$	WENN p DANN q
Prämisse:	p	Es gilt p
Schluß:	q	Es folgt q

Tabelle 10: Modus ponens

Bei Anwendung des generalisierten *Modus ponens* (Tabelle 11) wird die scharfe Prämisse durch eine unscharfe Prämise p' - die nicht notwendigerweise mit der ebenfalls unscharfen Regelbedingung p übereinstimmen muß - ersetzt. Bei einer unscharfen Schlußfolgerung erfährt nun q' einen Wert der von q abweicht, wenn p' von p abweicht.

Implikation:	$p \rightarrow q$	WENN p DANN q
Prämisse:	p'	Es gilt p'
Schluß:	q'	Es folgt q'

Tabelle 11: Generalisierter Modus ponens

Bezüglich des Verhaltens der Konsequenz bei der Anwendung des generalisierten *Modus ponens* werden die Kriterien der Tabelle 12 festgelegt.

[106] Vgl. Jaanineh, Maijohann (1996), S. 28.

1.	WENN p = **A**	DANN q = **B**
2.	WENN p = sehr **A**	DANN q = sehr **B**
3.	WENN p = sehr **A**	DANN q = **B**
4.	WENN p = ungefähr **A**	DANN q = ungefähr **B**
5.	WENN p = ungefähr **A**	DANN q = **B**
6.	WENN p = nicht **A**	DANN q = nicht **B**
7.	WENN p = nicht **A**	DANN q = unbestimmt

Tabelle 12: Kriterien für das plausible Schließen

Durch das Zutreffen oder Nicht-Zutreffen dieser Kriterien wird das Verhalten eines plausiblen Schlußverfahrens charakterisiert. Sie geben darüber Auskunft wie sich ein Abweichen der Prämisse p' von p auf die Folgerung q' auswirkt.

2.6 Zusammenfassung

In diesem Kapitel wurden die grundlegenden Elemente der Fuzzy-Set-Theorie vorgestellt. Aufbauend auf den Grundlagen der Logik unscharfer Aussagen wurde das Konzept der t-Normen beschrieben. Zur arithmetischen Behandlung unscharfer Sachverhalte wurden die unscharfen Zahlen vorgestellt. Das Erweiterungsprinzip erlaubt die generelle Übertragung von arithmetischen Verknüpfungen aus der Welt der reellen Zahlen auf unscharfe Zahlen. Es zeigt sich, daß die vereinfachte Darstellung unscharfer Zahlen als LR-Zahlen Vorteile in Form einer Verringerung des Komplexitätsgrades bei arithmetischen Operationen gegenüber der Verknüpfung mit dem Erweiterungsprinzip mit sich bringt. Zur Verarbeitung unscharfer Ursache-Wirkung-Zusammenhänge wurde durch die Herleitung des Zadeh-Implikationsoperators ein Werkzeug zur unscharfen Entscheidungsfindung vorgestellt.

3. Unscharfe Wissensrepräsentation und -verarbeitung

Zur rechnergestützten Handhabung von Problemenstellungen, wie sie beispielsweise in den planenden Teilsystemen der Produktionsplanung und -steuerung auftreten, bieten sich mehrere Ansätze an. Der erste Ansatz besteht darin, das zu lösende Problem als Ganzes abzubilden. Eine Darstellung als Totalmodell sprengt jedoch aufgrund ihres Komplexitätsgrades entweder die Kapazitäten des darstellenden Systems oder überfordert durch die beschreibende Datenmenge die Aufnahmefähigkeit des Benutzers[107]. Ein weiterer Ansatz ergibt sich durch die Nutzung von Expertenwissen zur Steuerung des Systems. Oft erfolgt diese Art der Entscheidungsunterstützung durch Expertensysteme(XPS).

Zur Darstellung des heuristischen Erfahrungswissens von Experten bietet sich die Formulierung des operationalen Entscheidungsmodells in WENN-DANN-Regeln an. Diese Regeln werden in einer Wissensbasis gesammelt und durch Anwendung von Inferenzmechanismen zur Generierung eines Entscheidungsvorschlages genutzt. Ein Expertensystem kann dazu dienen, den Aufgabenträger, i. d. R. ein Nicht-Experte, bei der Entscheidungsfindung durch Generierung von Vorschlägen zu unterstützen. Die Vorschläge sollten in ähnlicher Art und Weise zur Problemlösung beitragen wie die von einem menschlichen Experten gegebenen Ratschläge. Dieses Kapitel stellt dar, wie durch die Integration von Fuzzy-Ansätzen die in Informationen enthaltene Unsicherheit und Vagheit in einen Entscheidungsprozeß integriert und genutzt werden kann.

3.1 Regelbasierte Expertensysteme

Expertensysteme ermöglichen es, dem Benutzer das Problemlösungswissen eines Experten, der in der Lage ist, dieses Problem zu lösen, zur Verfügung zu stellen.

KURBEL[108] definiert den Begriff *Expertensystem* wie folgt:

Definition 3.1: Expertensystem

> Ein Expertensystem ist ein Programm, das in einem eng abgegrenzten Anwendungsbereich die spezifischen Problemlösungsfähigkeiten eines menschlichen Experten zumindest annähernd erreicht oder übertrifft.

---

107 Vgl. Zimmermann (1995), S. 8.
108 Vgl. Kurbel (1992), S. 22ff.

3.1.1 Anforderungen

BOTHE[109] nennt folgende Anforderungen an regelbasierte Expertensysteme.

- Die Kommunikation zwischen dem System und den Benutzern soll in einer standardisierten Fachsprache erfolgen, um eine natürliche Sprachumgebung zu schaffen.
- Die Art der Entscheidungsfindung des regelbasierten Expertensystems soll in ähnlicher Weise erfolgen wie diejenige eines menschlichen Experten. Dies erleichtert die Nachvollziehbarkeit und somit die Akzeptanz[110].
- Die Robustheit des Systems muß, auch bei einer datenmäßig unzureichenden Entscheidungssituation, gewährleistet sein, d. h. das System soll den Unsicherheitsgrad seiner Entscheidung explizieren.
- Die Wissensbasis muß erweiterbar sein.
- Die Wissensbasis und die Problemlösungskomponente sollen voneinander unabhängig und logisch getrennt sein.

3.1.2 Struktur

Die Struktur eines regelbasierten Expertensystems läßt sich nach ZIMMERMANN[111] entsprechend Bild 3.1 darstellen.

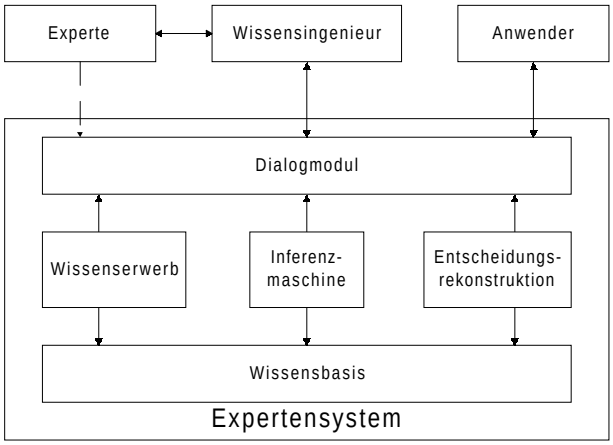


Bild 3.1: Regelbasiertes Expertensystem

[109] Vgl Bothe (1995), S. 113.
[110] Vgl. Rautenstrauch (1993), S. 54.
[111] Nach Zimmermann (1987), S. 262.

3.1.2.1 Wissenserwerbskomponente

Durch Anwendung der Wissenserwerbskomponente kann das Wissen der Wissensbasis gepflegt werden. Neben der Aufnahme von neuem Wissen kann veraltetes Wissen auch gelöscht oder Wissen, das sich im Zeitverlauf verändertet hat, revidiert werden. Der Aufbau der Wissenserwerbskomponente ist stark von der Darstellungsform des Wissens, welches beispielsweise in Form von Produktionsregeln vorliegt, abhängig.

3.1.2.2 Dialogmodul

Der Benutzer benutzt diese Komponente zur Wissensabfrage. Der Experte gibt sein Wissen entweder direkt in das System ein, oder er wird im Regelfall von einem Wissensingenieur interviewt, welcher dann dieses Wissen systemgerecht aufbereitet und erfaßt.

3.1.2.3 Wissensbasis

In der Wissensbasis befindet sich das Fachwissen des Experten bezüglich des Anwendungsgebietes des Expertensystems. Ist dieses Wissen in WENN-DANN-Regeln beschrieben, spricht man von einer Regelbasis. Ebenfalls in der Wissensbasis enthalten ist eine Beschreibung derjenigen Realweltobjekte, auf die sich die Regeln beziehen. Im Falle eines Fuzzy-Expertensystems[112] werden die Ausprägungen dieser Objekte mit Hilfe linguistischer Variablen beschrieben.

3.1.2.4 Problemlösungskomponente

Die Problemlösungskomponente besteht im wesentlichen aus einer Inferenzmaschine, die es erlaubt, mit Hilfe des in der Wissensbasis enthaltenen Wissens eine Problemlösung zu finden. Die in der Wissensbasis enthaltenen Regeln werden daraufhin untersucht, ob sie feuern, d. h. ob sie zur Lösung des Problems anwendbar sind. In herkömmlichen Expertensystemen feuert eine Regel entweder vollständig oder nicht. In Fuzzy-Expertensystemen feuert eine Regel entsprechend ihrem Kompatibilitätsgrad zur Faktensituation. Jede einzelne Regel wird durch die Inferenzmaschine auf die die Problemstellung beschreibenden Fakten angewendet, und es wird jeweils eine Schlußfolgerung ermittelt. Abschließend erfolgt eine Verknüpfung der Teilschlußfolgerungen zu einem Gesamtschluß.

---

[112] Vgl. Abschnitt 3.3

3.1.2.5 Erklärungskomponente

Die Erklärungskomponente informiert den Benutzer über die Arbeitsweise des Expertensystems. Der Benutzer kann die einzelnen Schritte der Problemlösung nachvollziehen und anhand der Zwischenergebnisse erkennen, inwieweit die Ausprägungen der Elemente der Faktenbasis zur Bildung der Schlußfolgerung beigetragen haben.

3.1.3 Wissensrepräsentation

Die Darstellung von Expertenwissen kann auf verschiedene Weisen erfolgen, die sich in Kategorien der deklarativen[113] und prozeduralen Darstellung einteilen lassen. Man klassifiziert ein Paradigma der Wissensrepräsentation als deklarativ, wenn eine pure Darstellung von Fakten erfolgt, wobei die problembezogene Anwendung der Sachzusammenhänge nicht beschrieben wird. Die prozedurale Repräsentation betont die aktive Nutzung von Wissen. Einsatzzweck und Verarbeitungsformalismen finden sich bereits in der Darstellung wieder. Heuristisches Wissen und Expertenwissen lassen sich aufgrund der intuiven Form oft besser in einer prozeduralen als in einer deklarativen Form darstellen.

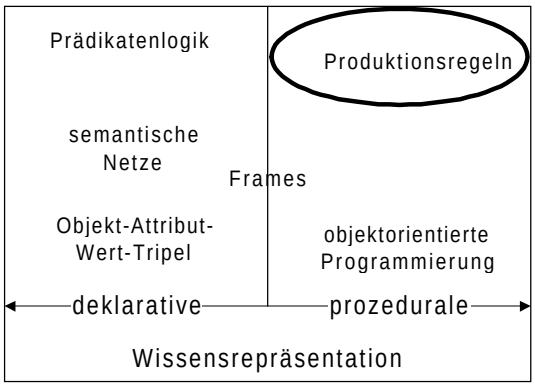


Bild 3.2: Arten der Wissensrepräsentation[114]

Unter den Methoden zur prozeduralen Wissensrepräsentation besitzen die Produktionsregeln aufgrund ihrer Ähnlichkeit zu bestehenden Ausdrucksweisen, die sich auch in Gesetzestexten und in heuristischem Wissen in Form von Daumenregeln als WENN-DANN-Ausdrücke wiederfinden, eine Vorrangstellung. Für eine regelbasierte Darstellung von Wissen spricht die einfache Konfigurierbarkeit und die intuitive Semantik. Ein Nachteil dieser Darstellungsart ist die potentielle Möglichkeit des Widerspruches zwischen den einzelnen Regeln einer Regelbasis.

---

113 Vgl. Kurbel (1992), S. 36ff.
114 Nach Kurbel (1993), S.37.

3.2 Grenzen von Expertensystemen der ersten Generation[115]

Bereits in der ersten Generation der Expertensysteme besteht der Bedarf zur Behandlung und Darstellung natürlich-sprachlicher Problemstellungen. Linguistisch formulierte Zusammenhänge führen zur Darstellung von Regeln. Hierarchische Abhängigkeiten und Strukturen können durch Meta-Regeln ebenfalls abgebildet werden. Die Regeln und Schlußfolgerungsmethoden basieren auf Darstellungsformen und Methoden der klassischen Aussagenlogik. Eine klassische Regelbasis hat den folgenden Aufbau:

WENN $X = A_1$ DANN $Y = B_1$

WENN $X = A_2$ DANN $Y = B_2$

...

WENN $X = A_n$ DANN $Y = B_n$

Liegt eine Beobachtung der Art „X ist A_K“ ($k \in [1;n]$) vor, so kann die Schlußfolgerung „Y ist B_K“ getroffen werden.

Die Regeln und Schlußfolgerungen basieren auf präzisen numerischen Werten oder auf präzisen linguistischen Termen einer linguistischen Variable. Die verwendeten linguistischen Terme weisen jedoch keine Toleranzbereiche zur graduellen Behandlung von Grenzfällen, d. h. linke und rechte Spannweiten, auf. Die zugrundeliegenden Zugehörigkeitsfunktionen der Terme können aufgrund der scharfen Modellierung nur die Werte 0 und 1 annehmen. Als Bedingung zur Gewinnung einer Schlußfolgerung „Y ist B_k“ muß eine Beobachtung präzise den linken Seiten einer Regel A_k, $k \in [1,n]$ entsprechen. Um garantieren zu können, zu jeder Faktensituation eine anwendbare Regel zur Verfügung stellen zu können, ist es notwendig in der Regelbasis eines Expertensystems eine große Zahl von Regeln, die sich nur marginal in den Werten unterscheiden, bereitzuhalten. Aufgrund dieser Einschränkung in der Wissensrepräsentation und die sich daraus ableitende Einengung der Schlußfolgerungsfunktionalität stoßen die Expertensysteme der ersten Generation an Anwendungs- und Akzeptanzgrenzen.

3.3 Expertensysteme der zweiten Generation (Fuzzy-Expertensysteme)

TURKSEN bezeichnet Fuzzy-Expertensysteme auch als Expertensysteme der zweiten Generation. Er rechtfertigt dies aus zwei Perspektiven:

[115] Vgl. Turksen (1992), S.2.

- Aus der Sicht der Expertensysteme ist - durch die Abkehr von der zweiwertigen Logik, hin zu Fuzzy-Methoden - ein Generationswechsel eingetreten.

- Aus der Sicht der Wissenschaften, welche Expertensysteme anwenden, wie beispielsweise des Ingenieurwesens, der Unternehmensforschung und der Betriebswirtschaft. Die in diesen Fachrichtungen erstellten Modelle wurden durch die Abbildungsmöglichkeiten in klassischen Expertensystemen auf zweiwertige Darstellungen reduziert. Es eröffnet sich die Möglichkeit der *Fuzzifikation* vieler klassischer Modelle in diesen Wissenschaften.

Der Generationssprung läßt sich besonders in zwei Merkmalen erkennen:

- in der unscharfen Wissensrepräsentation und

- in den unscharfen Inferenzmechanismen.

3.3.1 Unscharfe Wissensrepräsentation

Durch die Nutzung der erweiterten Möglichkeiten zur Modellierung von Wissen, d. h. die Verwendung von unscharfen Datentypen wie linguistischen Termen und Variablen, bleibt der Unschärfegrad und die Begrifflichkeit von Expertenaussagen erhalten. Durch die Verwendung von Fuzzy-Methoden ist bereits die Auswertung von ausschließlich ordinal-skalierten Beziehungen und Aussagen möglich. Die linguistische Darstellung erlaubt die Abbildung von unscharfen Aussagen und Wahrheitswerten sowie qualititativen Einschätzungen ohne eine Vortäuschung von Präzision.

Liegt einer mit scharfen Daten arbeitenden Entscheidungskomponente eine Situation zur Verarbeitung vor, für die sich in der Regelbasis keine Entsprechung findet, d. h. die von den scharfen Regeln nicht abgebildet wurde, so kann keine Konsequenz gezogen werden. Wird hingegen mit linguistischen Fakten und Regeln operiert, so liegt ein Entscheidungsmuster vor, es kann aus dem Grad der Entsprechung von Fakt und Regel die Gewichtung der entsprechenden Teilkonsequenz gewonnen werden. Das Charakteristische der unscharfen Wissensverarbeitung basiert daher auf der Fähigkeit, die vorhandenen Vorbedingungen aufgrund ihrer Kompatibilität zur aktuellen Umweltsituation zu gewichten und aus der Gesamtheit der gewichteten Vorbedingungen eine Gesamtkonsequenz zu kreieren.

3.3.2 Unscharfe Inferenzmechanismen

Durch die Verwendung von Fuzzy-Inferenzmechanismen wird eine flexible, mathematisch fundierte, Berechnungsgrundlage zur Bildung von Schlußfolgerungen in regelbasierten Systemen zur Verfügung gestellt. Durch die Fuzzy-Logik ist es möglich, die menschlichen Methoden zur Bildung von Schlußfolgerungen nachzuvollziehen. Dies gilt insbesondere für die Anwendung von Daumenregeln, d. h. die Bildung von Analogien. In einem Fuzzy-Expertensystem lassen sich die in der Regelbasis vorhandenen Regeln wie folgt darstellen:

$$\text{WENN } X = A_k \text{ DANN } Y = B_k, \; k \in [1,n]$$

In A_K und B_K finden sich im Gegensatz zu Expertensystemen der ersten Generation linguistische Terme, d. h. die Ausprägungen von linguistischen Variablen, die durch beliebige Zugehörigkeitsfunktionen modelliert werden können. Stellt man die Beobachtung „X ist A'" fest, d. h. A' entspricht nicht präzise einem der in der Domäne der linguistischen Variable enthaltenen Terme, so kann unbeachtet des Fehlens einer genau entsprechenden Regel eine Schlußfolgerung „Y ist B'" nach dem generalisierten *Modus ponens*[116] getroffen werden. Aufgrund dieser Toleranz gegenüber Abweichungen wird die Zahl der in der Regelbasis notwendigen Regeln stark reduziert und eine Regelexplosion wie in klassischen Expertensystemen vermieden. Das System bleibt übersichtlich und leichter wartbar.

3.3.3 Linguistische Ausdrücke

Die Art und Weise, in der ein Mensch die auf ihn einströmenden Informationen verarbeitet, ist individuell und durch die Persönlichkeit[117] geprägt. Beurteilungen und Zusammenhänge werden oft subjektiv beschrieben. Eine typische unscharfe Beurteilung ist beispielsweise die Aussage „Kunde B ist zuverlässig". Der Mensch ist in der Lage, auf der Grundlage von unscharfen Informationen, Sachverhalte zu verstehen, zu verarbeiten und Schlußfolgerungen zu ziehen. Zur rechnergestützten Interpretation unscharfer Information[118] liefert die Fuzzy-Set-Theorie das Instrumentarium der linguistischen Ausdrücke. Dieses setzt sich aus linguistischen Variablen, linguistischen Termen und Modifikatoren zusammen.

---

116 Vgl. Abschnitt 2.5.2.
117 Vgl. Mechler (1994), S. 29.
118 Zu den Ebenen der Information, vgl. Mag (1977), S. 5.

3.3.3.1 Linguistische Variablen

Eine linguistische Variable[119](LV) *V* stellt eine Variable dar, deren Wertebereich aus linguistischen Ausdrücken besteht. *V* kann alle Werte aus der Domäne *K* annehmen. *K* beinhaltet bspw. die atomaren linguistischen Terme *gering*, *mittel* und *hoch* für eine linguistische Variable „Stückzahl“ (Bild 3.3). Durch die Anwendung von Operatoren wie *nicht*, *oder* und *und* sind auch logische Kombinationen aus diesen Basistermen verfügbar. Die Anwendung von Modifikatoren wie *sehr* oder *mehr oder weniger* vergrößert den Ausdrucksraum einer linguistischen Variable. Eine linguistische Variable, die eine Entsprechung in der Realwelt besitzt, basiert auf einer meßbaren Basisgröße (in diesem Beispiel: Stückzahl x) und übernimmt deren Wertebereich und natürliche Maßeinheit. Linguistische Variablen, die keine natürliche Entsprechung haben, z. B. „Kundenzufriedenheit“, werden üblicherweise auf dem Einheitsintervall [0;1] abgebildet.

Definition 3.2: Linguistischer Term

Ein linguistischer Term setzt sich zusammen aus

- dem Namen des Terms und
- der unscharfen Menge, die den Term mathematisch beschreibt.

Definition 3.3: Linguistische Variable

Eine linguistische Variable setzt sich zusammen aus

- dem Namen der linguistischen Variablen,
- der Menge der Basisterme der linguistischen Variablen und
- den Angaben zum Wertebereich.

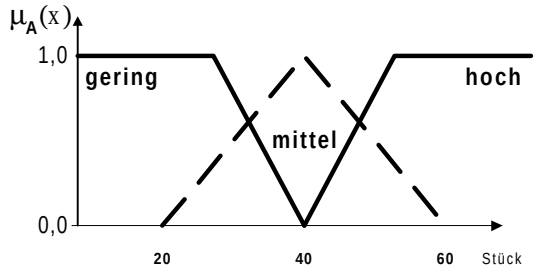


Bild 3.3: Linguistische Variable „Stückzahl“

[119] Vgl. Jaanineh, Maijohann (1996), S. 169.

3.3.3.2 Linguistische Modifikatoren

Aus Gründen der Übersichtlichkeit wird der Anwender bemüht sein, die Anzahl der Basisterme (Terme im Grundbereich[120]) einer linguistischen Variable begrenzt zu halten. Oft wird jedoch kein Basisterm (bspw. „hoch“) einer linguistischen Variablen (bspw. „Deckungsbeitrag“) benötigt, sondern ein *sehr* oder *etwas* („sehr hoch“ bzw. „etwas hoch“) des entsprechenden Terms. Durch Anwendung linguistischer Modifikatoren[121] kann die Verstärkung und Verringerung des Schärfegrades eines linguistischen Terms modelliert werden. Ein Modifikator bewirkt eine numerische Abschwächung oder eine numerische Verstärkung der Zugehörigkeitsfunktion. Modifikatoren sind unäre Operatoren. Sie manipulieren genau eine unscharfe Menge. Die Gipfelwerte und Spannweiten einer unscharfen Menge bleiben durch die Anwendung von linguistischen Modifikatoren unberührt.

Man unterscheidet mehrere Arten von linguistischen Modifikatoren:

- Konzentrationsoperatoren,
- Dilatationsoperatoren,
- Kontrastintensivierungsoperatoren,
- Negationsoperatoren.

Definition 3.4: Konzentrationsoperator

> Sei $\mathbf{A}$ eine unscharfe Menge über dem Träger S, so erhält man durch Potenzieren der Zugehörigkeitsfunktion den Konzentrationsoperator
>
> $kon_n(\mathbf{A}) = \{(x; \mu_A(x)^n) \mid x \in S\}$.

Ein Konzentrationsoperator bewirkt durch eine numerische Abschwächung der Zugehörigkeitsfunktion eine linguistische Verstärkung, die auch durch das Wort *sehr* zum Ausdruck kommt. Zur numerischen Abschwächung von normalisierten Zugehörigkeitsfunktionen im Bereich [0;1] wird eine beliebige Potenzfunktion mit einem Exponenten größer als eins verwendet.

---

120 Synonym: primary space.

121 Synonyme: linguistic modifier, linguistic hedge.

Definition 3.5: Dilatationsoperator

Sei **A** eine unscharfe Menge über dem Träger S, so erhält man durch Radizieren der Zugehörigkeitsfunktion den Dilatationsoperator:

$$dil_n(\mathbf{A}) = \{(x;\sqrt[n]{\mu_A(x)}) \mid x \in S\} .$$

Ein Dilatationsoperator bewirkt durch eine numerische Verstärkung der Zugehörigkeitsfunktion eine linguistische Abschwächung, entsprechend dem natürlich-sprachlichen *mehr oder weniger*. Zur numerischen Verstärkung von normalisierten Zugehörigkeitsfunktionen im Bereich [0;1] wird eine beliebige Wurzelfunktion mit einem Wurzelexponenten größer als eins verwendet.

Definition 3.6: Kontrastintensivierungsoperator

Sei **A** eine unscharfe Menge über dem Träger S, so erhält man durch Anwendung des Kontrastintensivierungsoperators:

$$int_n(\mathbf{A}) = \begin{cases} \{x\ ;\ 2^{n-1}\mu_A(x)^n\} & \text{für } \mu_A(x) \in [0{,}0;0{,}5] \wedge x \in S \\ \{x\ ;\ 1-2^{n-1}(1-\mu_A(x))^n\} & \text{für } \mu_A(x) \in [0{,}5;1{,}0] \wedge x \in S \end{cases}$$

Ein Kontrastintensivierungsoperator ist eine Kombination aus Konzentrationsoperator und Dilatationsoperator. Er arbeitet für Zugehörigkeitswerte unter 0,5 wie ein Konzentrationsoperator, für Zugehörigkeitswerte über 0,5 wirkt er wie ein Dilatationsoperator, weshalb sichergestellt sein muß, daß eine Kontrastintensivierung eines Zugehörigkeitsgrades von 0,5 wieder einen Zugehörigkeitsgrad von 0,5 ergeben muß.

Negationsoperatoren

Die bereits behandelten Negationsoperatoren (siehe Definition 2.10) dienen zur Abbildung des linguistischen Modifikators *nicht*.

Bild 3.4 gibt die Auswirkungen der linguistischen Modifikatoren auf den mittleren Term der oben beschriebenen linguistischen Variablen „Stückzahl“ wieder.

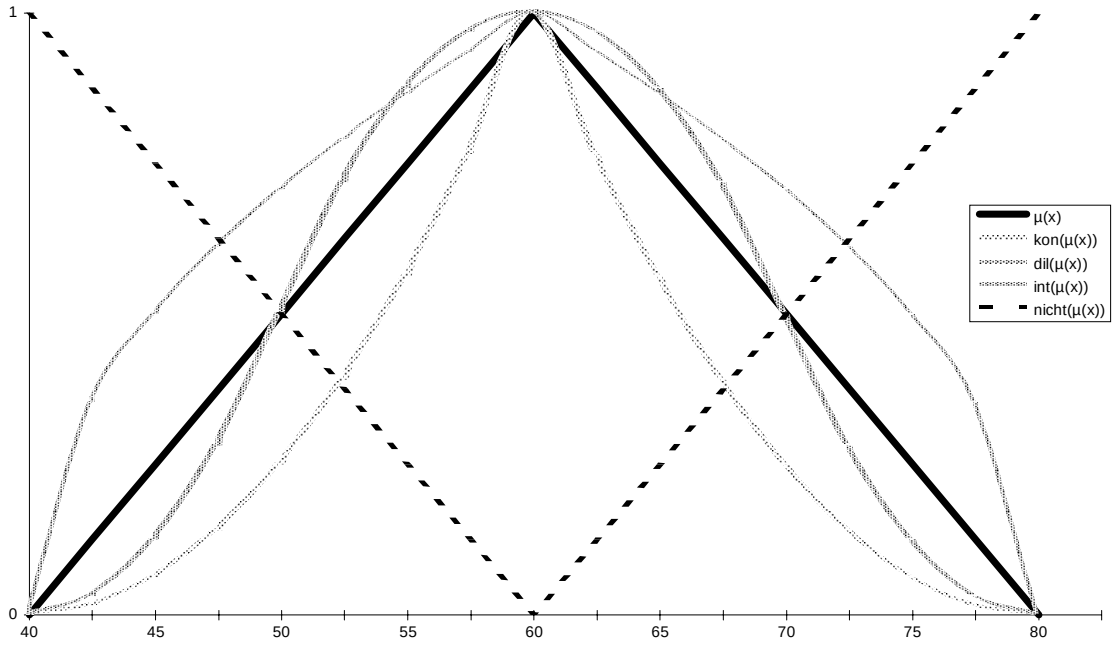


Bild 3.4: Linguistische Modifikatoren

3.3.3.3 Linguistische Approximation

Die bisherigen Ausführungen beschäftigten sich damit, die zu einem gegebenen natürlichsprachlichen Ausdruck entsprechende unscharfe Menge zu modellieren. Der umgekehrte Fall ist ebenso denkbar. Zur Beschreibung von unscharfen Mengen, werden geeignete Verfahren benötigt. Linguistische Modifikatoren dienen dazu, die Basisterme einer linguistischen Variablen geeignet zu verändern und den jeweiligen Abstand zu der Zugehörigkeitsfunktion der zu untersuchenden unscharfen Menge zu ermitteln. Zur Ermittlung des Abstandes zwischen zwei unscharfen Mengen bedient man sich häufig des *euklidischen Abstandes*[122] ihrer Zugehörigkeitsfunktionen:

$$ED(\mathbf{A},\mathbf{B}) = \sqrt{\sum_{x \in S}(\mu_A(x)-\mu_B(x))^2}\ .$$

Das Ziel der linguistischen Approximation nach dem *Best-Fit*[123]-Verfahren ist es, zu einer unscharfen Menge **A** denjenigen Term **B** - evtl. ein durch Anwendung von linguistischen Modifikatoren gewonnener Term - einer linguistischen Variablen zu finden, dessen

122 Vgl. Mayer et al. (1993), S. 67.
123 Vgl. Mayer et al. (1993), S. 69.

euklidischer Abstand zu **A** unter allen betrachteten Alternativen minimal ist. In der Literatur[124] finden sich weitere Ansätze.

3.4 Akquisition, Formulierung und Validierung der Regelbasis

Es bestehen mehrere Möglichkeiten Wissen für ein regelbasiertes Fuzzy-System zu gewinnen. Zum einen können aus der Untersuchung einer Datensituation die entsprechenden WENN-DANN-Regeln gewonnen werden. , In diesem Fall liegt kein explizites Expertenwissen vor. Mit genetischen Algorithmen[125] oder durch Zuhilfenahme künstlicher neuronaler Netze, also mit Hilfe von Methoden des maschinellen Lernens, können Regeln gewonnen werden. Unter dem Begriff Data-In-Rules-Out, kurz DIRO[126], werden die Methoden zur datengesteuerten Regelgewinnung subsumiert. Eine andere Methode besteht in der Akquisition des Wissens von Experten, die für beispielhafte Zustände einer Problemstellung Handlungsempfehlungen geben und bewerten können.

3.4.1 Wissensakquisition

Der Begriff der Wissensakquisition bezeichnet die Gewinnung von Wissen für ein wissensbasiertes System[127]. Die folgenden drei Grundprinzipien[128] der Wissensakquisition können unterschieden werden.

3.4.1.1 Indirekte Wissensakquisition

Bei der indirekten Wissensakquisition wird eine Zwischenperson, ein sog. Knowledge Engineer, eingeschaltet, der das aus dem Interview mit dem Experten gewonnene Wissen in eine für das regelbasierte Fuzzy-System verständliche Form bringt. Nicht nur die verwendete Interviewtechnik, sondern auch die dem eigentlichen Interview vorausgehende Bildung einer gemeinsamen Begrifflichkeit durch Experten und Knowledge Engineer, macht diese Vorgehensweise erheblich zeitaufwendig.

3.4.1.2 Direkte Wissensakquisition

Im Gegensatz zur indirekten Wissensakquisition kommuniziert bei der direkten Ausprägung der Experte direkt mit dem System. Hierbei muß der Experte über die

[124] Vgl. Zimmermann (1991), S. 143f.
[125] Vgl. Nelles (1996), S. 489.
[126] Vgl. Kosko (1995), S. 203f.
[127] Vgl. Kurbel (1992), S. 17f.
[128] Vgl. Kurbel (1992), S. 68f.

Fähigkeit verfügen, sein Wissen systemkonform zu formulieren. Diese Benutzerschnittstelle muß in der Lage sein, daß vom Experten formulierte Wissen direkt im System darzustellen. Voraussetzung ist, daß der Experte über entsprechende Fähigkeiten verfügt, sein Wissen in der Art und Weise zu formulieren, daß es vom System dargestellt werden kann. Aufgrund des vorausgesetzten Know-hows über das System eignet sich die direkte Wissensakquisition im allgemeinen eher für die Wartung und Erweiterung einer Wissensbasis als für deren erstmalige Erstellung.

3.4.1.3 Automatische Wissensakquisition

Bei der automatischen Wissensakquisition können mit Hilfe von Ansätzen aus dem Bereich der genetischen Algorithmen [129] nach der FUREGA-Methode Fuzzy-Regeln aus einer Menge von Daten erstellt und bewertet werden. Bei einer großen Menge von Meßdaten können die von den genetischen Algorithmen generierten Regeln eine Approximation des Zusammenhangs zwischen den zu identifizierenden Wirkungsgrößen geben. Bei der Akquisition von Expertenwissen ist diese Methode aber bei der erstmaligen Gewinnung einer Regelbasis nicht anwendbar. WEBER und ZIMMERMANN beschreiben mit Fuzzy-ID3[130] eine Erweiterung der klassischen ID-3-Methode, welche speziell bei der Akquisition von Wissen für Fuzzy-Expertensysteme Einsatz findet.

3.4.2 Formulierung

Zur Darstellung des zu verarbeitenden Expertenwissens ist zuerst eine geeignete Form der verschiedenen Regeln, in ihrer Gesamtheit Regelbasis genannt, zu wählen. Neben der hier verwendeten Darstellung in Form von Produktionsregeln(Wenn-Dann-Form) können auch Methoden wie die Prädikatenlogik oder semantische Netze Verwendung finden. Produktionsregeln haben den Vorteil der einfachen Formulier- und Lesbarkeit. Da die Regeln das Wissen eines menschlichen Experten zur Lösung des behandelten Problems wiedergeben, sind diese i. d. R. unscharf formuliert.

Die Prämissen von Produktionsregeln enthalten unscharfe Ausdrücke der Art

„WENN Auftragspriorität = *hoch* UND Kundenbewertung = *wichtig*“,

die den Zustand wiedergeben, der in der Folgerung „dann plane Auftrag *früh* ein“ eine Handlungsempfehlung generiert. Regeln können durch die oben beschriebenen Vorgehensweisen zur Wissensakquisition gewonnen werden.

---

129 Vgl. Nelles (1996), S. 491.
130 Vgl. Weber, Zimmermann (1991), S. 24f.

Die Prämissen und Folgerungen sind durch die Verwendung von Termen der linguistischen Variablen intuitiv verständlich und leicht wartbar. Bei komplexen Problemen können Regelbasen einen großen Umfang annehmen. Aus diesem Grund ist es empfehlenswert, Regelbasen aufgrund der Übersichtlichkeit, Wartbarkeit und des Nachverfolgens von Kenn- und Zwischenwerten in mehrere hierarchisch verbundene[131] überschaubare Teilregelbasen aufzuteilen, die die verschiedenen Stufen hierarchischer Entscheidungsfindung widerspiegeln. Ein Beispiel zur Anwendung hierarchischer Regelbasen findet sich in Kapitel 6.

3.4.3 Validierung einer Regelbasis

Die in einer Regelbasis enthaltenen Regeln können auch widersprüchlich und falsch sein. Zur Beseitigung von strukturellen Anomalien innerhalb einer Regelbasis müssen verschiedene Fälle beachtet werden, beispielhaft werden die folgenden Möglichkeiten[132] der Inkonsistenz einer Regelbasis aufgeführt.

- Es fehlen Regeln, d. .h. ein Teil des Zustandsraums kann durch die vorhandenen Regeln nicht erreicht werden.
- Es bestehen schleifenartige Strukturen in der Regelbasis, z. B. „WENN A DANN B“ und „WENN B DANN A“. Es ist oft schwer zu erkennen, welche Regel aus der Regelbasis eliminiert werden muß.
- Redundante Regeln führen zu den gleichen Ergebnissen, da bspw. die Bedingung einer Regel die Teilmenge einer anderen Regelbedingung ist und beide Regeln zu einer gleichen Schlußfolgerung führen.
- Durch logische Widersprüchlichkeit und die Verletzung semantischer Rahmenbedingungen können falsche Fakten erzeugt werden, die aus der Anwendung falscher Regeln entstehen.

3.5 Fuzzy-Inferenz

Der Begriff des approximativen Schließens[133] beschreibt die Vorgehensweise aus unscharfen Prämissen und Fakten unscharfe Folgerungen abzuleiten. Das Schlußverfahren selber ist mathematisch exakt formuliert und orientiert sich an einer Verallgemeinerung der klassischen Implikationsregeln. Beim approximativen Schließen

131 Vgl. v.Altrock (1994), S. 376.
132 Vgl. Zlatareva (1994), S. 715f.
133 Vgl. Böhme (1993) , S. 231.

spielen linguistische Variablen eine zentrale Rolle. Diese Veränderlichen können Werte annehmen, die wiederum durch Modifikatoren (Dilatation und Konzentration s.o.) sprachlich verändert werden, um die Unschärfe der menschlichen Sprache, die sich in Ausdrücken wie „ziemlich hoch“ wiederfindet, abbilden zu können.

Bei der Anwendung des klassischen *Modus ponens* der Aussagenlogik wird von folgenden Prämissen ausgegangen[134]:

- Es stehen nur die Wahrheitswerte *wahr* und *falsch* zur Modellierung der Bedingungen und der Folgerung zur Verfügung.
- Die Bedingung und Folgerung einer Regel müssen scharf definierte, deterministische Aussagen sein.
- Die Beobachtung muß voll und ganz der Bedingung der Regel entsprechen.
- Die Regeln gehen nicht über die Anwendung von Allquantoren (∀**)** und Existenzquantoren (∃) hinaus.

Diese Einengung auf die Abbildungsmöglichkeiten der dualen Logik erschwert die Verarbeitung und Benutzung von menschlichem Expertenwissen. Dieses kann meist nicht adäquat in Form von scharfen dualen Unterscheidungen (ja/nein) dargestellt werden, sondern verlangt eine Wissensdarstellung in Form verschiedener Abstufungen von Wahrheit und Sicherheit. Die Brücke, die die Grundlagen der dualen Logik aufgreift und um die Eigenschaften des menschlichen Schlußfolgerungsverhaltens erweitert, findet sich in den Formen der Inferenzmechanismen der Fuzzy Logik.

3.5.1 Grundschema der Fuzzy-Inferenz

Regelbasen werden aus Expertenwissen, das softwaretechnisch konserviert werden soll, gewonnen. Dies ist beispielsweise beim Ausscheiden von sehr kompetenten Mitarbeitern aus einer Unternehmung notwendig.. Regelbasen beschreiben ein Entscheidungsmuster zur Lösung eines Problems und basieren auf unscharfen Zusammenhängen und Beobachtungen bezüglich der Einflußparameter. Das Entscheidungsmuster der Regelbasis wird auf eine bestimmte Entscheidungssituation in Form einer Faktenbasis (bestehend aus aktuellen Beobachtungen) nach dem Vorbild eines Fuzzy-Controllers (Bild 3.5) angewandt. Das Ergebnis dieser Verarbeitung dieses unscharfen Schließens ist eine Schlußfolgerung. Diese wird dem Benutzer in Form eines unscharfen Ergebnisses

[134] Vgl. Zimmermann (1995) S. 5

zurückgeliefert, welches in unscharfer Form weiterverwendet werden kann. Folgende Fälle sind denkbar:

- Es erfolgt die natürlichsprachliche Verwertung des Ergebnisses, - in Form einer Handlungsempfehlung oder einer Bewertung - durch eine linguistische Approximation.
- Es erfolgt eine Defuzzifikation, d. h. die Umsetzung in einen scharfen Stellwert zur Anbindung an nicht-fuzzifizierte Systeme.
- Die Schlußfolgerung kann in eine neue Entscheidungssituation einfließen, so daß eine Verkettung von Inferenzen vorliegt.

Es ergeben sich folgende Stufen bei der Abarbeitung des unscharfen Schlußfolgerns.

- Linguistische Zuordnung der Beobachtungsgrößen (Fuzzifikation)
- Bildung einer Schlußfolgerung (Inferenz)
- Aufbereitung der Schlußfolgerung für das Zielsystem (Defuzzifikation)

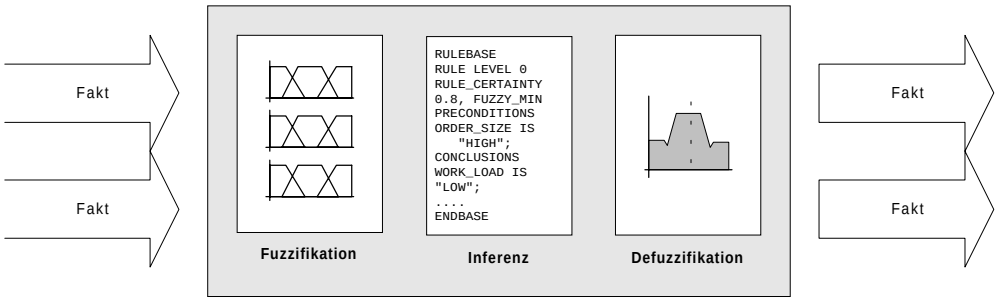


Bild 3.5: Ablauf des approximativen Schließens

3.5.2 Fuzzifikation

Als Fuzzifikation wird die Transformation quantitativer Größen in qualitative Terme einer linguistischen Variablen bezeichnet. Zum Zeitpunkt der Erkennung der Eingangsgrößen können die Einflußgrößen als Fakten in verschiedenen Zuständen vorliegen. Im Falle einer Messung können diese Werte in Form von reellen Werten vorliegen. Aufgrund der Zugehörigkeitsgrade des Eingangswertes zu den Termen der linguistischen Variablen werden Kompatibilitätsgrade[135] des Faktes zu jedem Term der linguistischen Variablen ermittelt.

[135] Vgl. Rommelfanger (1994) S. 163, Synonyme: Degree of Fulfillment, Erfüllungsgrad.

3.5.3 Inferenz

Implikation: Wenn p = A Dann q = B

Prämisse: p = A'

Folgerung q = B'

Die unscharfe Inferenz besteht aus drei Stufen:

1. Aggregation
2. Implikation
3. Akkumulation

3.5.3.1 Die Aggregation

Um den Erfüllungsgrad von Regeln zu ermitteln, werden die Erfüllungsgrade der einzelnen Prämissen mit einem unscharfen Operator[136] kombiniert. Die Wahl von unscharfen Operatoren ist von der Problemstellung abhängig. Es ist zu beachten, daß der gewählte Operator den zu modellierenden natürlich-sprachlichen Zusammenhang adäquat abbildet. Wenn Prämissen in einem substituierenden Verhältnis zueinander stehen, spricht dies für die Verwendung von Durchschnittsoperatoren.

3.5.3.2 Die Implikation

In der Phase der Implikation wird der Erfüllungsgrad der Regelkonklusion ermittelt. Zuerst wird der aggregierte Erfüllungsgrad der Prämissen mittels einer t-Norm mit dem Sicherheitsfaktor der Regel verknüpft. Meist wird hier das algebraische Produkt gewählt. Nun wird die Konklusion aufgrund der Abweichung des Faktes von der Prämisse entsprechend des verwendeten Implikationsoperators modifiziert.

3.5.3.3 Die Akkumulation

Es ist möglich, daß die resultierenden Regelkonklusionen verschiedene Ausprägungen einer linguistischen Variablen annehmen. Diese Einzelergebnisse werden abschließend für jede linguistische Variable mit einer s-Norm-Verknüpfung, meistens mit dem *max*-Operator, akkumuliert.

---

[136] Vgl. Abschnitt 2.3.5.

3.5.3.4 Max-Min-Inferenz und Max-Prod-Inferenz

Wird als Implikationsoperator der Minimum-Operator benutzt (Mamdani-Implikation), so spricht man von einer Max-Min-Inferenz, bei einer Max-Prod-Inferenz wird zur Verknüpfung auf das algebraische Produkt zurückgegriffen.

3.5.4 Aufbereitung der Schlußfolgerung für das Zielsystem

Neben der bereits beschriebenen Möglichkeit zur linearen Approximation eines Inferenzergebnisses, besteht oft die Notwendigkeit, das vorliegende unscharfe Ergebnis in einen skalaren Wert zu transformieren. Man spricht hierbei von einer Defuzzifikation.

3.5.4.1 Defuzzifikation[137]

Das Ergebnis der Defuzzifikation stellt eine quantitative Repräsentation der unscharfen Ergebnismenge dar. Mit einer Defuzzifikation ist immer ein qualitativer Informationsverlust verbunden, da das Defuzzifikationsresultat keine Aussagen über den Grad der Unschärfe im Ergebnis zuläßt.

Da das nach einer Fuzzy-Inferenz vorliegende Ergebnis als unscharfe Menge vorliegt, ist es zur Gewinnung eines scharfen Stellwertes notwendig, diese auf eine geeignete Art und Weise in eine Realzahl umzuwandeln. Im folgenden werden die verschiedenen Methoden zur Defuzzifikation beschrieben.

3.5.4.1.1 Maximum-Methode

Bei dieser Methode (Bild 3.6) wird ein Wert aus dem Definitionsbereich der unscharfen Menge gewählt, dessen Zugehörigkeitsgrad maximal ist. Jeder reelle Wert zwischen 7,0 und 8,0 kann als Ergebnis gewählt werden. Liegen mehrere Maxima oder ein maximales Plateau[138] vor, kann diese Methode zu einem nicht-stabilen oder oszillierendem Verhalten des Ergebnisses führen. Die nachfolgenden Modifikation dieser Methode löst dieses Problem.

3.5.4.1.2 Randmaximum-Methode

Beim Vorhandensein von Plateaus orientiert man sich an dessen Eckpunkten. Bei der Wahl dieser Methode wird der kleinste bzw. der größte x-Wert ausgewählt, dessen

---

137 Vgl. Bothe (1995), S. 146f.
138 Vgl. Abschnitt 2.3.15.

Zugehörigkeitsgrad maximal ist. Im Beispiel (Bild 3.7) werden die Werte 7 (Linkes-Maximum) und 8 (Rechtes-Maximum) ermittelt.

3.5.4.1.3 Mitte-der-Maxima-Methode

Der arithmetische Mittelwert der auftretenden Maxima wird als Stellwert ausgewählt. Allerdings muß vor Auswahl dieser Methode beachtet werden, daß der resultierende Wert selber kein Maximum sein muß, sondern suboptimal oder sogar minimal sein kann. Im Beispiel (Bild 3.8) beträgt der nach der Mittelpunktsmethode defuzzifizierte Wert 7,5.

3.5.4.1.4 Schwerpunkt[139]-Methode

Der Schwerpunkt der Zugehörigkeitsfunktion der unscharfen Menge wird als Stellwert übernommen. Der Schwerpunkt x_0 der Zugehörigkeitsfunktion $\mu(x)$, $x \in X$ ergibt sich dabei durch Integration über X nach

$$x_0 = \frac{\int x\mu(x)dx}{\int \mu(x)dx}$$ [140].

Im Vergleich zu den bisher genannten Methoden kann der Rechenaufwand bei zeitkritischen Operation zu Problemen führen. Werden wie im DREM[141]-System stückweise lineare Polygonzüge verwandt, so vereinfacht sich die Berechnung des Schwerpunktes. Im Beispiel (Bild 3.9) ist der defuzzifizierte Wert ca. 5,76. Wie beim Mitte-der-Maxima-Verfahren ist auch bei dieser Methode nicht garantiert, daß an der ermittelten Stelle ein Maximum vorliegt.

---

139 Synonyme: Centroid-Methode, Center-Of-Gravity.

140 Zur numerischen Optimierung der Schwerpunkts-Defuzzifikation existieren Verfahren, die die Integration umgehen, vgl. Mayer et al.(1993), S. 82.

141 Vgl. Abschnitt 5.2.

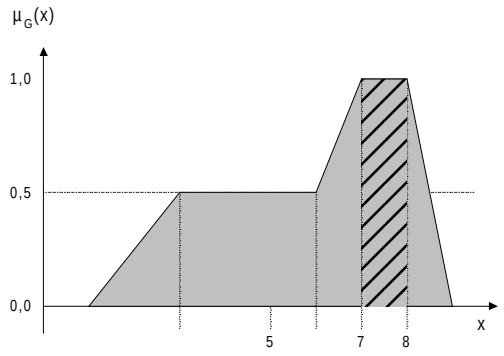


Bild 3.6: Maximum-Defuzzifikation

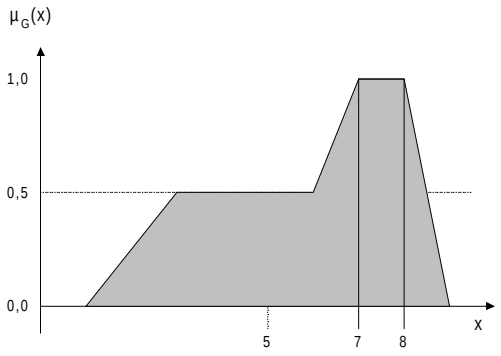


Bild 3.7: Randmaximum-Defuzzifikation

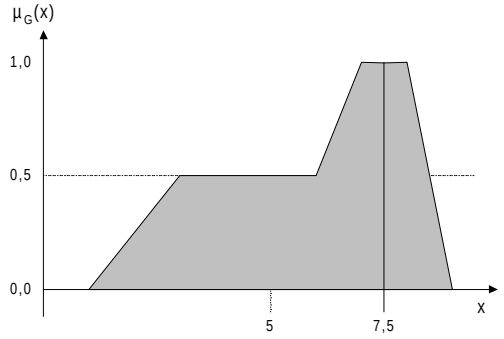


Bild 3.8: Mitte-der-Maxima-Defuzzifikation

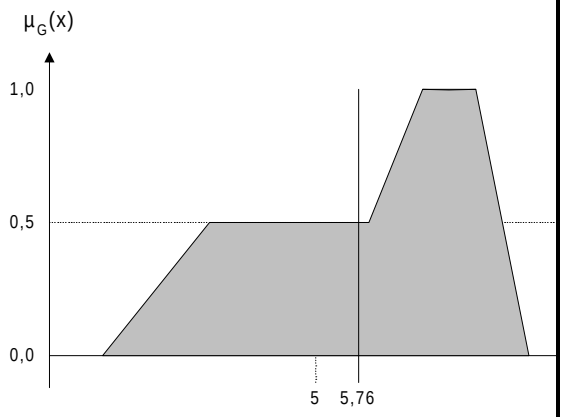


Bild 3.9: Schwerpunkts-Defuzzifikation

3.5.5 Beispiel zur Fuzzy-Inferenz[142]

In einer Werkshalle hat sich vor einer Maschine eine Warteschlange mit noch zu bearbeitenden Werkstücken aufgebaut. Mit Hilfe linguistischer Variablen werden die Schlupfzeit[143] (Bild 3.10) und die Anzahl der Vorgänger einer wartenden Verrichtung (Bild 3.11) durch linguistische Symbole abgebildet. In einer Regelbasis (Tabelle 13) sind die Auswirkungen dieser Parameter auf die Notwendigkeit bzw. Dringlichkeit einer Neuplanung angegeben. Die Angabe eines Ergebnisses erfolgt durch die linguistische Variable Wechselempfehlung (Bild 3.12).

142 Nach Schmidt (1993), S. 82f.

143 Schlupfzeit = Fertigungsendtermin-aktuelles Datum. Eine alternative Darstellung einer linguistischen Variablen „Schlupfzeit“ findet sich bei Hopf (1995), S. 104.

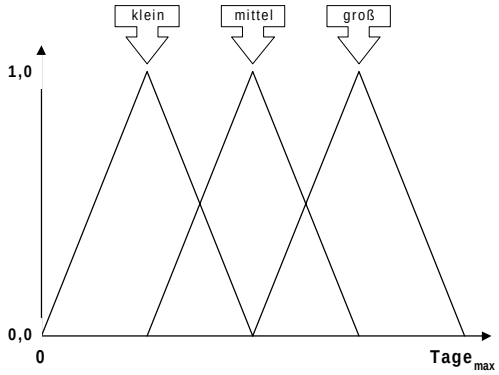


Bild 3.10: Linguistische Variable „Schlupfzeit“

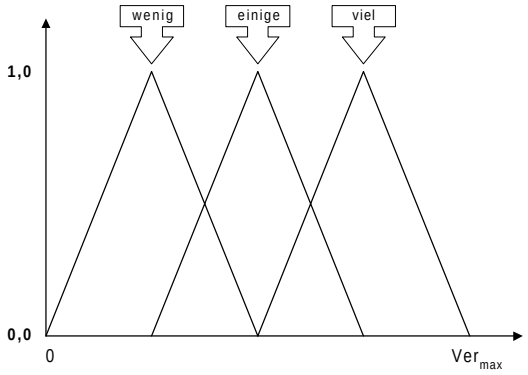


Bild 3.11: Linguistische Variable „Verrichtungen“

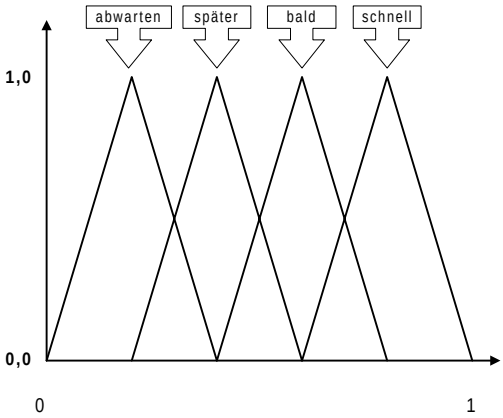


Bild 3.12: Linguistische Variable „Wechselempfehlung“

Die Regelbasis zur Ermittlung der Wechseldringlichkeit läßt sich, wie in Tabelle 13 dargestellt, angeben.

UND		Schlupfzeit		
		klein	mittel	groß
Anzahl Verrich-tungen	wenig	bald	später	abwarten
	einige	schnell	später	abwarten
	viele	schnell	bald	später

Tabelle 13: Regelbasis zur Wechselempfehlung

Die Anzahl der maximalen Verrichtungen betrage 16 Stück und die maximale Schlupfzeit sei auf 32 Tage festgelegt.

Für eine konkrete Situation, bei einer Anzahl von 11 wartenden Verrichtungen und einer Schlupfzeit von 18 Tagen, soll eine Wechselempfehlung ermittelt werden. Tabelle 14 zeigt die, mit diesen Fakten ermittelte, Faktenbasis.

Fakt A	Schlupfzeit	=	18
Fakt B	Anzahl Verrichtungen	=	11

Tabelle 14: Faktenbasis zur Ermittlung der „Wechselempfehlung“

3.5.5.1 Fuzzifikation

Für die beiden Eingangsvariablen „Schlupfzeit“ und „Anzahl der Verrichtungen“ wird nun ein Erfüllungsgrad festgelegt. Aus einer min-Kombination der Prämissen einer Regel wird der Gesamterfüllungsgrad einer Regel ermittelt.

In der Phase der Fuzzifikation ermitteln sich die Erfüllungsgrade durch Ermittlung der Schnittpunkte der Fakten mit den linguistischen Termen der beiden linguistischen Variablen „Schlupfzeit“ und „Anzahl der Verrichtungen“. Dies geschieht gemäß den Abbildungen von Bild 3.13 und Bild 3.14.

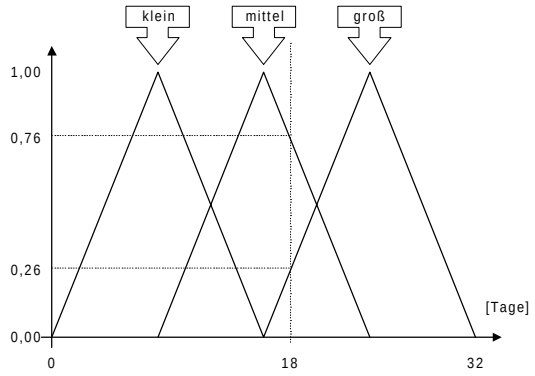


Bild 3.13: Erfüllungsgrade der Schlupfzeit

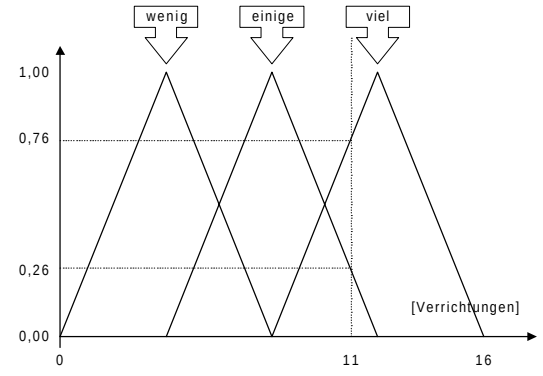


Bild 3.14: Erfüllungsgrade der Verrichtungsanzahl

Wird die Regelbasis in Textform abgebildet, so entstehen 9 Regeln. Für jede dieser Regeln wird eine Gesamtkompatibilität durch Minimumbildung der Einzelerfüllungsgrade ermittelt. Tabelle 15 zeigt diesen Zusammenhang.

Regel	WENN Schlupfzeit =	UND WENN Verrichtungsanzahl =	DANN Wechsel-empfehlung	Erfüllungs-grad (Schlupfzeit)	Erfüllungs-grad (Verrichtung)	Kompatibilität der Gesamtregel
1	klein	wenig	bald	0,00	0,00	0,00
2	klein	einige	schnell	0,00	0,26	0,00
3	klein	viele	schnell	0,00	0,76	0,00
4	mittel	wenig	später	0,76	0,00	0,00
5	**mittel**	**einige**	**später**	**0,76**	**0,26**	**0,26**
6	**mittel**	**viele**	**bald**	**0,76**	**0,76**	**0,76**
7	groß	wenig	abwarten	0,26	0,00	0,00
8	**groß**	**einige**	**abwarten**	**0,26**	**0,26**	**0,26**
9	**groß**	**viele**	**abwarten**	**0,26**	**0,76**	**0,26**

Tabelle 15: Ermittlung der Kompatibilitätsgrade

3.5.5.2 Inferenz

Die Regelkonsequenzen werden nun mit dem Erfüllungsgrad ihrer auslösenden Regel bewertet, wobei sich, in Abhängigkeit von der zur Inferenzbildung verwendeten t-Norm, verschiedene unscharfe Mengen als Ergebnis für die Wechselempfehlung präsentieren.

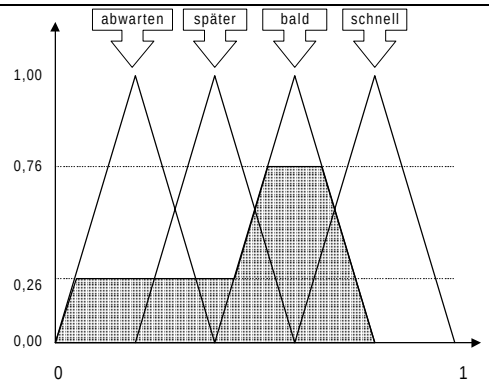


Bild 3.15: Max-Min-Inferenz

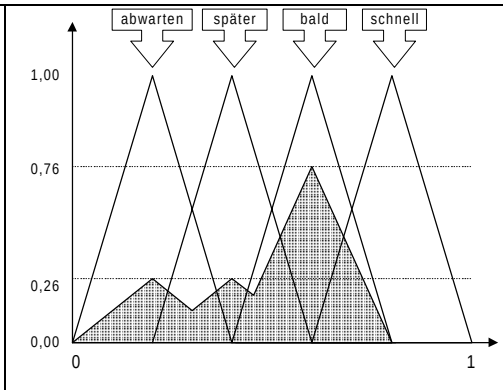


Bild 3.16: Max-Prod-Inferenz

Die Abbildungen in Bild 3.15 und Bild 3.16 geben die Inferenzbildung nach den Verfahren *max-min* und *max-prod* wieder.

3.5.5.3 Defuzzifikation

Eine Defuzzifikation bietet sich im *max-min*-Fall nach der Mitte-der-Maxima-Methode an, während sich bei dem *max-prod*-Fall eine Defuzzifierung nach der Maximum-Methode anbietet. Da jedoch der Definitionsbereich der Wechselempfehlung auf dem Einheitsintervall liegt, ist die Aussagekraft der defuzzifizierten Größe begrenzt. Aus diesem Grunde wird eine linguistische Approximation durchgeführt. Die größte Entsprechung des Ergebnisses bei der Max-Min-Inferenz als auch in der Max-Prod-Inferenz ist mit dem Term *bald* gegeben. Die Anwendung der Max-Min-Inferenz führt

aufgrund des Abschneidens von Fläche mit dem *min*-Operator zur Plateaubildung und damit zur Überbewertung[144] der Terme mit geringer Kompatibilität. Es wird nur der kleinere Teil ihrer Fläche abgeschnitten und somit tragen sie überproportional zum aggregierten Ergebnis bei.

3.6 Verkettung

Wird eine hierarchische Entscheidungsfindung abgebildet oder liegt eine große Zahl von Regeln vor, ist es notwendig, die Ausführung von Fuzzy-Inferenzen zu verketten. Es werden zunächst die Verkettungsmethoden der klassischen Expertensysteme vorgestellt, darauffolgend wird die Verkettung von Fuzzy-Inferenzen beschrieben. Zur Darstellung des Weges durch die Zustände einer komplexen Entscheidungssituation benötigt man einen Zustandsgraphen. Jeder Knoten symbolisiert eine Entscheidungssitutation, in der Regeln (Kanten) ausgewertet werden. Neben der Rückwärtsverkettung (R-Pfeile) und der Vorwärtsverkettung (V-Pfeile) wird auch die Fuzzy-Verkettung (F-Pfeile) dargestellt. Durch Auswertung einer Regel gelangt man von einer Faktensituation zu einer neuen. Der Zustandsgraph, der einen verketteten Durchlauf durch eine Regelbasis zeigt, findet sich in Bild 3.17.

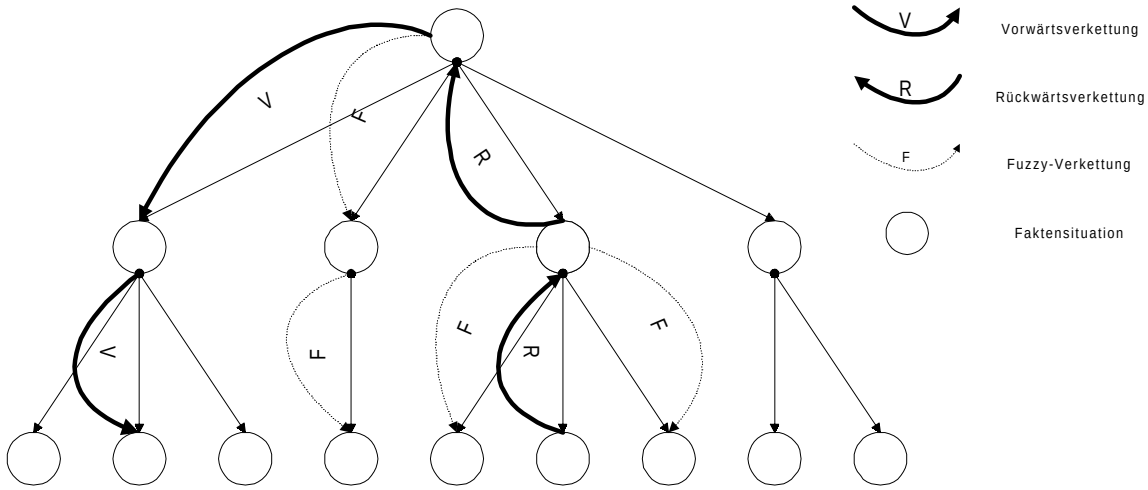


Bild 3.17: Verkettung im Zustandsraum[145]

144 Vgl. Rommelfanger (1996), S. 26.
145 Nach Rembold et al. (1994), S. 178.

3.6.1 Vorwärtsverkettung[146]

Die Vorwärtsverkettung durchsucht einen Zustandsraum von oben nach unten. Zu Beginn der Suche stehen die konkreten Beobachtungen, die problembeschreibenden Daten und die problemlösenden Regeln fest, deshalb bezeichnet man die Vorwärtsverkettung auch als datengesteuert. Bei der Anwendung von Regeln entstehen neue, derivative Fakten. Die Suche wird beendet, wenn die festgelegte Zielbedingung durch ein Fakt erreicht wird. In der industriellen Praxis wird die Vorwärtsverkettung zur Lösung von Planungsproblemen genutzt. Die Vorwärtsverkettung findet dann Anwendung, wenn

- sich ein Problem durch ein vorgegebenes Ziel formulieren läßt, oder
- die Anzahl der anwendbaren Regeln durch eine Einschränkung des Suchraums (Pruning) reduziert werden soll oder
- zur Lösung des Problems neue Daten beschafft werden müssen.

3.6.2 Rückwärtsverkettung[147]

Bei der Rückwärtsverkettung beginnt die Suche am Ziel und sucht diejenigen Regeln, die zu diesem Ziel führen. Die Suche setzt an dem so gefundenen Subziel an und setzt sich aufwärts im Zustandsraum fort bis eine endgültige Faktensituation erreicht ist. Aufgrund dieser Orientierung an Zielgrößen bezeichnet man die Rückwärtsverkettung auch als zielgesteuert. Oft wird die Rückwärtsverkettung zur Planung von Aktionen genutzt. Diese Methode findet dann Anwendung, wenn

- im Zustandsraum viele Ziele vorhanden sind oder
- eine vollständige Beschreibung der problembezogenen Daten im System vorliegt.

3.6.3 Fuzzy-Verkettung

Zur Modellierung von Expertenwissen werden häufig Regeln in der Form

„WENN Kunde = wichtig, DANN Priorität = hoch“

zur Modellierung von Expertenwissen angewandt. Ob eine Regel zur Entscheidungsfindung beitragen kann, hängt von ihrer Kompatibilität zur Faktensituation ab. Ist eine

146 Vgl. Rembold et al. (1994), S. 179; vgl. Mechler (1993), S. 38.
147 Vgl. Rembold et al. (1994), S. 179 ; vgl. Mechler (1993), S. 39.

Regel kompatibel, so spricht man davon, daß sie feuert. Der Erfüllungsgrad[148] einer Regel ergibt sich aus der Kompatibilität zwischen Prämisse und vorliegendem Fakt. Er gibt das Gewicht an, mit dem eine Regel *feuert*, d. h. zum aggregierten Gesamtergebnis beiträgt. Zur Reduzierung des Rechenaufwandes kann ein Schwellenwert, der festlegt ab welchem Erfüllungsgrad eine Regel zum Gesamtergebnis beiträgt, vorgegeben werden. Wird mit unscharfen Werten operiert, können mehrere Regeln entsprechend ihres Erfüllungsgrades feuern, im scharfen Fall feuert eine Regel entweder voll oder gar nicht.

Innerhalb eines Expertensystems wird durch die in der Regelbasis enthaltenen Regeln ein Entscheidungsmuster zur Problemlösung beschrieben. Zur Bildung einer Schlußfolgerung ist von einer softwarebasierten Entscheidungskomponente - aufgrund der gegebenen originären Faktensituation - derjenige Pfad durch die Menge der Regeln zu finden, der eine Lösung des Problems liefert. Anfangs ist zu prüfen, welche Regeln in der Faktensituation anwendbar sind. Nach einer Auswertung stellen diese Regeln neue derivative Fakten zur Verfügung. Existiert in der Regelbasis nur eine Hierarchiestufe, liefern diese neuen Daten bereits eine Lösung des Problems. Im Normalfall werden ausgehend von dieser, um derivate Fakten erweiterte Situation, diejenigen Regeln angewandt, die durch die derivativen Fakten gefeuert werden. Bei dieser Art der Verkettung hängt die Performanz der Entscheidungsfindung von der Anzahl der zur Verfügung stehenden Regeln ab. Ein OMP-Verfahren zur optimalen Regelselektion wird von SHMILOVICI[149] und MAIMON beschrieben. Bei der Verkettung von Fuzzy-Inferenzen darf der pro Inferenz potentiell steigende Unsicherheitsgrad der Schlußfolgerung laut DRIANKOV und HELLENDOORN[150] nicht vernachlässigt werden.

3.7 Zusammenfassung

In diesem Kapitel wurden die Grundlagen von Fuzzy-Expertensystemen beschrieben und insbesondere wurde das Wesen und die Nutzenpotentiale der linguistischen Wissensrepräsentation dargestellt, abschließend wurden die Grundlagen der Fuzzy-Inferenz zur Unterstützung der Entscheidungsfindung unter Unschärfe dargestellt.

148 Synomym: Degree of Fulfillment, Kompatibilitätsgrad.
149 Vgl. Shmilovici, Maimon (1996), S.592.
150 Vgl. Driankov, Hellendoorn (1995), S.103f.

4. Einsatz unscharfer Informationen im betrieblichen Umfeld

In betrieblichen Abläufen begegnet man oft Informationen vagen Charakters. Trotz der geringeren Informationsreife ist es für Entscheidungsträger wichtig, vorläufige qualitative Informationen[151] zur frühen Entscheidungsfindung zu nutzen. Es ist in der Regel nicht möglich, in Informationssystemen unscharfe Daten abzubilden oder gar für die Entscheidungsfindung aufzubereiten. Gerade zur Sicherung von Wettbewerbsvorteilen ist es jedoch wichtig, Informationen möglichst früh zu nutzen und über betriebliche Informationssysteme auch für jeden potentiellen Entscheidungsträger und jede betroffene betriebliche Einheit, nutzbar zu machen. In die klassischen, die Auftragsabwicklung unterstützenden, Produktionsplanungs- und -steuerungssysteme (PPS-Systeme) gehen Informationen erst zu einem Zeitpunkt ein, in dem diese zwar einen höhere Reife besitzen, aber ihre strategische Bedeutung bereits verloren haben (können). Unter diesem Aspekt ist es wichtig, Informationen bereits im Stadium geringer Schärfe in ein Informationssystem einfließen zu lassen, um den aufgrund dieser Information bereits vorbestimmten Entscheidungsraum optimal nutzen zu können. Überlegungen zur Integration von vagen Informationen in PPS-Systeme finden sich bei REHFELDT und TUROWSKI[152]. Im folgenden wird die PPS als zentrales Instrument der Auftragsabwicklung herausgestellt und die in sie einfließenden und aus ihr abgeleiteten unscharfen Daten analysiert. Trotz ihrer Unschärfe kann der Inhalt unscharfer Information oft zu einer frühzeitigen Koordination der nachgeschalteten betrieblichen Aufgaben genutzt werden. Dies kann zur Erreichung

- einer Verkürzung der Durchlaufzeiten (DLZ) und
- einer Erhöhung der Planungssicherheit

genutzt werden.

4.1 Verkürzung von Durchlaufzeiten

Wird erkannt, daß zwischen Arbeitsgängen keine Ressourcenabhängigkeit besteht, so können diese Arbeitsgänge unabhängig voneinander und parallel bearbeitet werden, denn es besteht keine Notwendigkeit zur sequentiellen Anordnung. Eine Verkürzung von Auftragsdurchlaufzeiten wäre die Folge, die neben potentiell verringerter Kapitalbindung auch positive Auswirkungen auf den gesamten Fertigungsablauf besitzt.

---

151 Vgl. Nietsch et al. (1994), S. 13.
152 Vgl. Rehfeldt, Turowski (Technology) (1994), S. 1642.

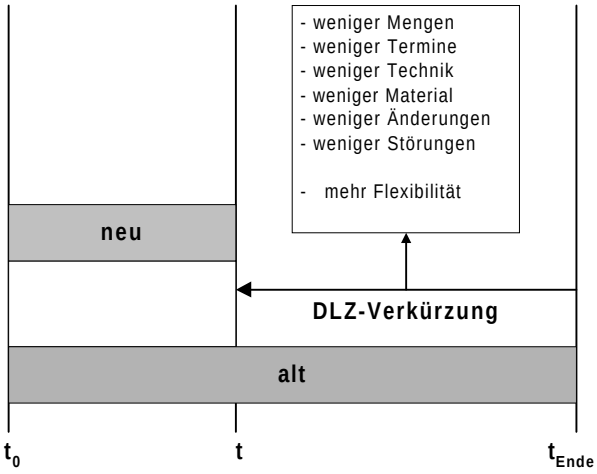


Bild 4.1: Verkürzung der Durchlaufzeit[153]

Bild 4.1 zeigt die positiven Auswirkungen von Durchlaufzeitverkürzungen auf

- die Reduktion von Lagerbeständen[154],
- die Erhöhung der Prognosegenauigkeit,
- die Verringerung von Steuerungsproblemen und -aufwand und
- der verringerte Änderungsaufwand im PPS-System.

4.1.1 Verbesserung der Planungssicherheit

Der Vertrieb eines Auftragsfertigers[155] kann bereits vor Bekanntwerden der exakten Bestellmenge eines Auftrags die grobe Größenordnung, in Form einer qualitativen Kategorie wie *klein, mittel* und *groß*, an die Vorfertigung weitergeben. Diese kann so bereits mit der groben Terminierung dieses Auftrags beginnen, obwohl der Vertrieb noch nicht in der Lage ist, die Größe eines Auftrags in genauen Stückzahlen zu beziffern. Das vorläufige Planungsergebnis der Vorfertigung kann nach der Grobplanung an den Vertrieb zurückfließen und so bereits dem Kunden mitgeteilt werden. Weil auch auf dessen Seite mit vorläufigen Lieferdaten bereits Planungen verbunden werden können, kann ein KKV bzgl. der Zufriedenheit des Kunden erzielt werden. Auf der Beschaffungsseite kann durch frühzeitige Lieferantenauswahl die Lieferbereitschaft von benötigten Teilen sichergestellt und Angebote frühzeitig selektiert werden. Eine Übersicht über diesen Sachverhalt gibt Bild 4.2.

153 Nach Helfrich (1993), S. 175.
154 Vgl. Helfrich (1993), S. 175.
155 Vgl. Kurbel (1993), S. 195f.

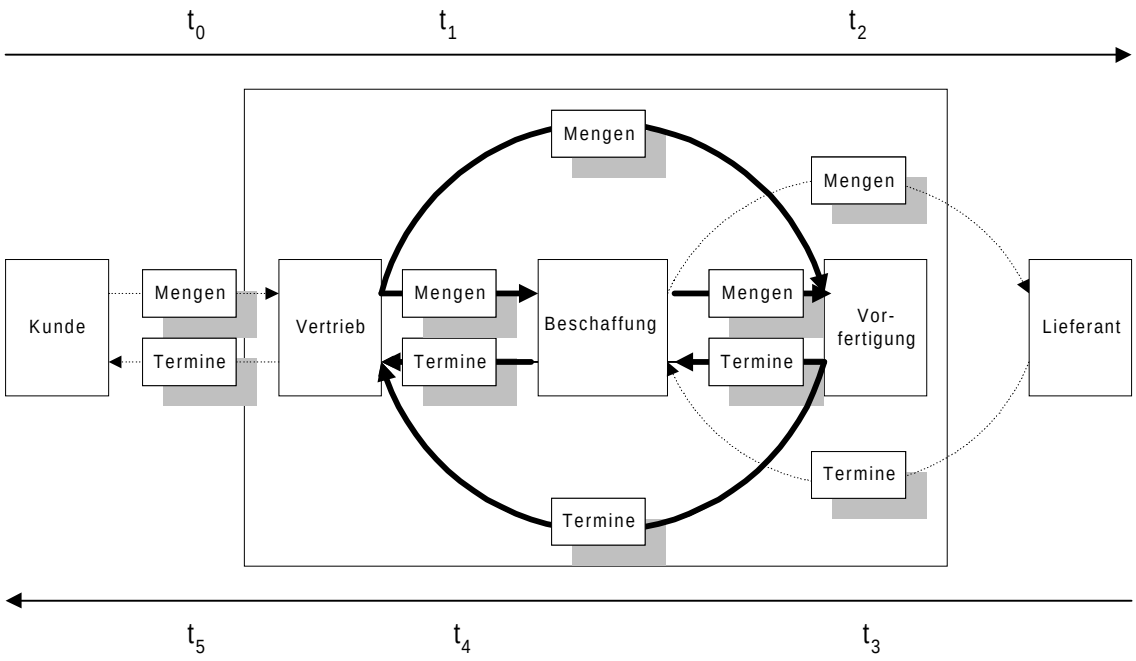


Bild 4.2: Erhöhte Planungssicherheit

4.2 Unschärfe in der industriellen Leistungserstellung

Viele im Rahmen der betrieblichen Leistungserstellung benötigten Daten sind zunächst unscharf. In einem zeitgemäßen Industrieunternehmen sollten diese Daten nicht nur in die informellen Wege zur betrieblichen Leistungserstellung einfließen, sondern sollten in den verschiedenen softwaregestützten Teilsystemen zur Planung und Steuerung der betrieblichen Abläufe verarbeitet werden. Gewöhnlich werden diese Systeme innerhalb eines PPS-Systems integriert. Auch auf der primär technischen Seite (CAx[156]) gehen unscharfe Daten aufgrund ihrer Herkunft aus Arbeitsplänen, Stücklisten und Betriebsmittelspezifikationen in die Produktentwicklung und Arbeitsvorbereitung ein. Die integrative Gesamtsicht auf die PPS- und CAx-Zusammenhänge wird als Computer Integrated Manufacturing[157] (CIM) bezeichnet. Die Zusammenführung von CIM und unscharfen Daten wird in Bild 4.3 dargestellt.

156 CAx = {CAE, CAD, CAP, CAM , CAQ, CAI, ...}, vgl. Becker, Rosemann (1993), S. 12.
157 Vgl. Becker, Rosemann (1993), S.14. ; vgl. Becker (1991) S.166 - 191.

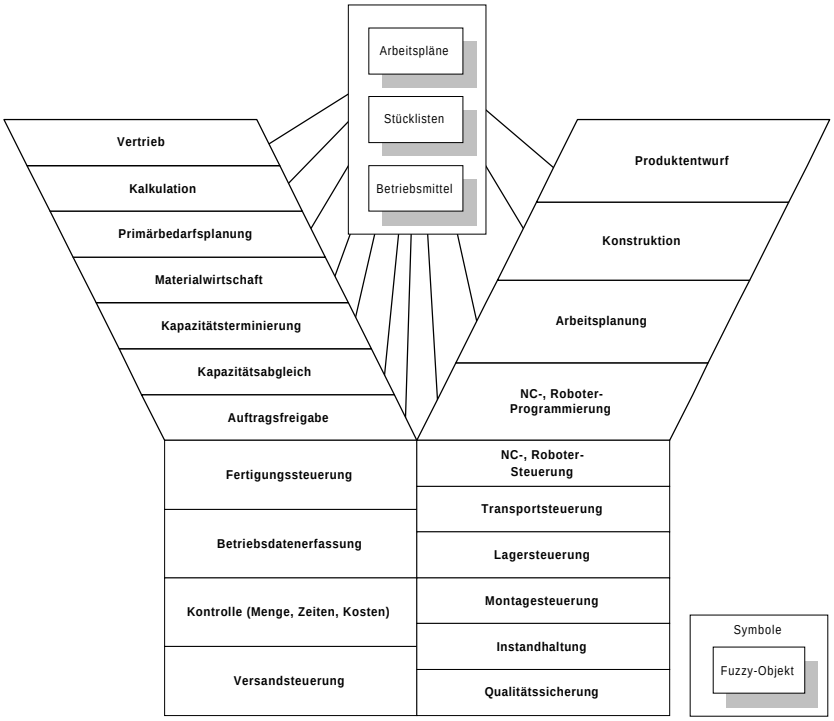


Bild 4.3: Das Y-CIM-Modell mit unscharfen Daten[158]

In herkömmlichen PPS-Systemen finden ausschließlich scharfe Informationsflüsse statt. Jedoch sind die in einem derartigen System kursierenden Daten von ihrem Wesen her unscharf. Damit diese Informationen in einem PPS-System verarbeitet werden können, werden sie häufig künstlich zu einem scharfen Datum verdichtet, aber fließen auf informellen Wege[159] weiterhin unscharf - am System vorbei - weiter. Diese Diskrepanz zwischen der Abbildung im PPS-System und der Realität führt potentiell zu Fehlentscheidungen. Wird mit der Erfassung von Daten in ein Informationssystem gewartet, bis diese sicher sind, wird durch den Verzicht auf Parallelisierungspotentiale, auf die Nutzung von wertvollen Zeitvorteilen[160] und somit auf die Sicherheit der Planung verzichtet. Das Konzept einer auf Planungsobjekte verteilten PPS verspricht eine erhöhte Flexibilität und Sicherheit in der Planung[161]. Die Explikation von Unschärfe in Planungsobjekten führt zum integrativen Konzept der Fuzzy-Objekte[162]. Ein Fuzzy-Informationsobjekt kapselt die unscharfen Daten, und ein Fuzzy-Planungsobjekt beinhaltet die Methoden in Form von Regelbasen zur Steuerung betrieblicher Aufgaben, bspw. der Materialwirtschaft[163] oder der Fertigungssteuerung. Die sich innerhalb der technischen CIM-Subsysteme eingeführten Simultaneous-Engineering[164]-Ansätze verfolgen -

158 Nach Scheer (1994), S.93.
159 Vgl. Turowski (1995), S 216.
160 Vgl. Rehfeldt, Turowski (Coordination) (1995), S. 1778.
161 Vgl. Becker et al. (1995), S. 37.
162 Vgl. Rehfeldt, Turowski (1996), S. 1983f.
163 Vgl. Nietsch et al. (1993), S. 24.
164 Vgl. Womack, Jones, Roos (1992), S. 121-123; Becker, Rosemann (1993), S. 43f.

im Rahmen der zeitgleich zur Auftragsentwicklung ablaufenden Produktentwicklung[165] - eine ähnliche Zielsetzung zur Parallelisierung von unabhängigen Vorgängen.

Die kundenorientierte Auftragsabwicklung umfaßt die betrieblichen Abläufe vom Eingang eines Kundenauftrags in den Vertrieb und die Disposition der benötigten Bestandteile des Auftrages. Auf der Fertigungsseite wird die Auftragsabwicklung komplettiert durch die Einplanung und Durchführung der Fertigungsaufträge sowie dem Versand der Auftragsposten. Die Qualität einer solchen Auftragsabwicklung eines Unternehmens kann durch eine Erhöhung des Informiertheitsgrades der involvierten Aufgabenträger potentiell erhöht werden. Freiheitsgrade in Bezug auf Mengen und Endtermine können - nur sofern sie bekannt sind - vorteilhaft genutzt werden. Das Grobschema der Auftragsabwicklung in Bild 4.4 zeigt eine in Fuzzy-Objekte zerlegte PPS.

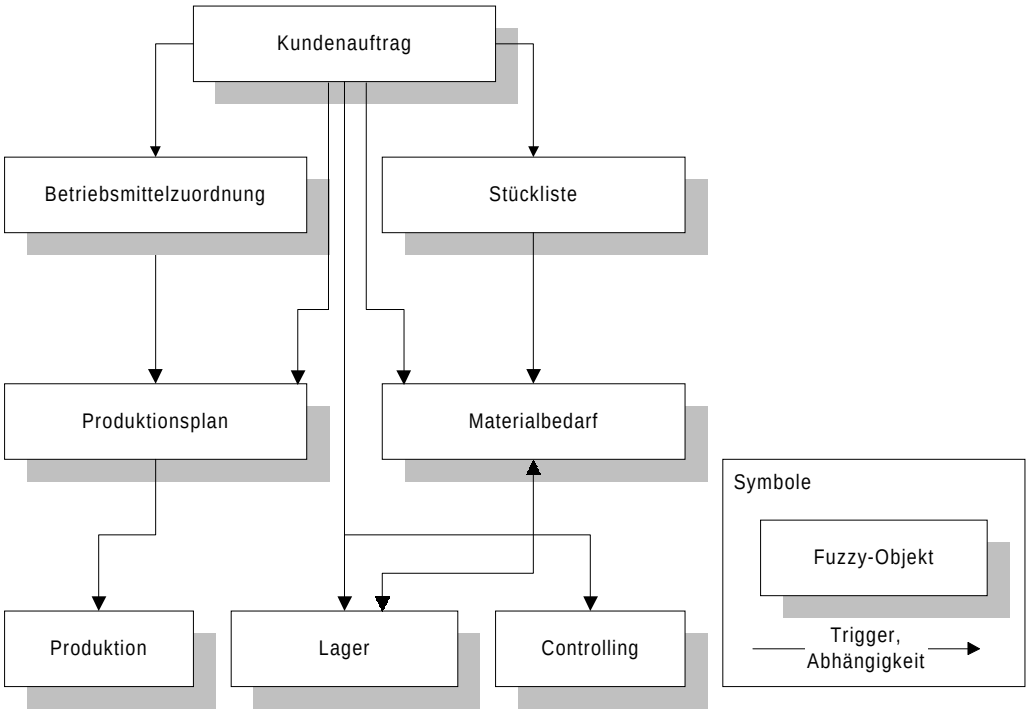


Bild 4.4: Fuzzy-Objekt-basierte PPS[166]

[165] Zu den Zielen des Simultaneous Engineering vgl. Eversheim, Laufenberg (1995), S. 33.
[166] Vgl. Rehfeldt, Turowski (1996), S. 1986.

Die für die Materialwirtschaft zuständigen Beschaffungsabteilungen, Disposition und Einkauf arbeiten häufig auf der Basis vager und somit unsicherer Daten. Der Vertrieb kann diese Informationen frühzeitig zur Verfügung stellen. Die Vorfertigung kann aufgrund vorläufiger Vertriebsinformationen frühzeitig Kapazitäten für diese Aufträge reservieren und einen groben Fertigungsplan, in Form von unscharfen Fälligkeitsterminen und Mengen[167], erstellen[168]. Der oben beschriebene Ablauf des planerischen Teils der industriellen Auftragsabwicklung mit unscharfen Daten ist in Form einer ereignisgesteuerten Prozeßkette (eEPK)[169] in Bild 4.5 dargestellt.

[167] Vgl. Bild 4.2.
[168] Vgl. Turksen (1992), S. 351.
[169] Vgl. Scheer (1994), S. 50f.

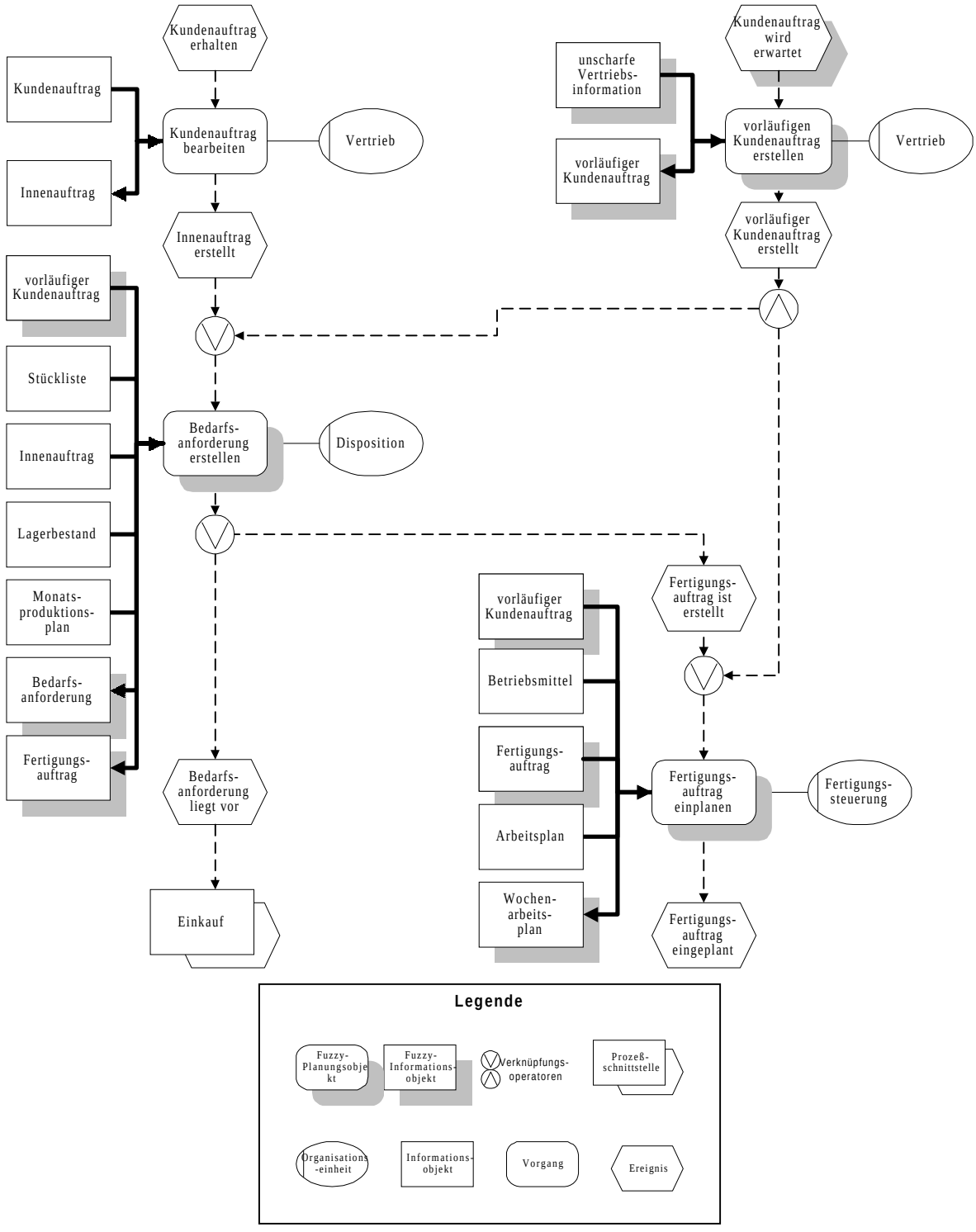


Bild 4.5: Auftragsabwicklung mit unscharfen Daten[170]

[170] Vgl. Becker et al. (Planungsobjekte) (1996), S.12.

4.3 Fuzzy-Methoden in der PPS

Die explizite Integration von unscharfen Daten in die PPS[171] führt zu der Erkenntnis, daß auch Methoden bereitgestellt werden müssen, um diese Daten in eine für die Entscheidungsfindung verwertbare Form zu transformieren. Die folgende Tabelle gibt eine Übersicht über die mittels Fuzzy-Methoden zu lösenden Teilprobleme der PPS. Fuzzy-Methoden werden in einem Fuzzy-Planungsobjekt abgelegt.

PPS-Funktion	Beispiele für anwendbare Fuzzy-Methoden
Vertrieb	Fuzzy-Clustering, Trendbeurteilung, Neueinplanungen, Nachfrageprognosen
Kalkulation	Kalkulation unter Zuhilfenahme unscharfer Zahlen zur Risikoberücksichtigung, Plankostenrechnung, Personalplanung
Primärbedarfsplanung	Bedarfsprognose, Bewertung von Kundenaufträgen
Materialwirtschaft	Verfügbarkeit, Lieferantenauswahl, Lieferbereitschaft, Teilestammdaten: Ausschußquote, Wiederbeschaffungszeit Stücklisten: Teilelebensdauer, Recyclingeigenschaften
Kapazitätsterminierung	Betriebsmittel: Kapazitätsangebot, Zugehörigkeiten, Wartungsbedarf, erforderliche Qualifikation
Kapazitätsabgleich	Modellierung von Kapazitätsangebot und -nachfrage durch unscharfe Zahlen Fuzzifikation von Kapazitätsangebot und -nachfrage, Einführung von unscharfen Einlastungsschranken.
Auftragsfreigabe	Frühzeitige Prioritätsbestimmung, Betriebsmittelbelegung
Fertigungssteuerung	Disposition, Störungsbehandlung, Auswahl von Alternativarbeitsplänen, Arbeitspläne: Modellierung von Zeitgrößen durch unscharfe Zahlen
Betriebsdatenerfassung	Frühzeitige Bereitstellung und Aggregation von qualitativen Rückmeldungen
Kontrolle von Mengen, Zeiten und Kosten	Frühzeitige Bereitstellung und Aggregation von qualitativen Rückmeldungen
Versandsteuerung	Routenplanung, Packprobleme, Kommissionierungsoptimierung

Tabelle 16: Fuzzy-Methoden in der PPS[172]

[171] Vgl. Becker (1991), S.166-191.
[172] Vgl. Rehfeldt, Turowski (Integration) (1994), S. 3.

4.4 Fuzzy-Objekte in der PPS

In der PPS lassen sich im Ablauf der Auftragsabwicklung die in Tabelle 17 aufgeführten Informationsobjekte, die unscharfe Attribute enthalten, identifizieren. Diese enthaltenen Attribute dienen als Eingangsgrößen für Fuzzy-Methoden. Die aus Fuzzy-Methoden resultierenden unscharfen Größen können wiederum als Attribute in Fuzzy-Objekte[173] einfließen.

Fuzzy-(Informations-)-Objekt	unscharfe Attribute	Menge	Zeit
(vage) Vertriebsinformationen	Anfragen	+	+
	saisonale Abrufe	+	+
(vorläufiger) Kundenauftrag	Liefertermin		+
	Liefermenge	+	
Innenauftrag	Sonderaufträge	+	+
	Auftragsposition	+	+
Stückliste	Alternativstruktur	+	
	Toleranzen	+	
Lagerbestand[174]	Reservierungen	+	+
Quartalsplan	Personalverfügbarkeit	+	+
	Zukaufteile	+	+
Bedarfsanforderung	Liefertermin		+
	Liefergröße	+	
Betriebsmittel	Kapazität	+	+
	Störzeiten		+
Arbeitsplan	Fertigungszeiten		+
	Alternativen, Toleranzen	+	
Wochenproduktionsplan	Verfügbarkeit von Ressourcen	+	+
	Sonderaufträge	+	+
	Lieferterminprobleme	+	+
	Losgrößen	+	

Tabelle 17: Fuzzy-(Informations-)Objekte mit unscharfen Attributen[175]

4.5 Zusammenfassung

In diesem Kapitel wurden die der PPS inhärenten Unschärfepotentiale aufgegriffen und durch den Fuzzy-Objekte-Ansatz zur expliziten Integration von Unschärfe - innerhalb des Gesamtzusammenhangs der industriellen Auftragsabwicklung - systematisiert.

173 Vgl. Rehfeldt, Turowski (1996), S. 1986.
174 Vgl. Hönerloh et al. (1996), S. 119.
175 Vgl. Becker et al.(Auftragsabwicklung) (1996), S.55.

5. Integration der Fuzzy-Logik-Entscheidungskomponente in DREM

Die Entwicklungs- und Laufzeit-Umgebung DREM[176] kann als Basis für viele betriebliche Anwendungen dienen. Als wichtigste Einsatzbereiche seien das Workflow-Managementanwendungen und PPS-Anwendungen genannt. DREM stellt einen Prototypen dar, der, auf einer verteilten Datenbankschicht basierend, die Darstellung unscharfer Daten und die parallele Ausführung von Methoden der instantiierten Objekte[177] erlaubt. Eine typische, auf DREM[178] aufbauende Anwendung ist die Modellierung und Simulation von Geschäftsprozessen, bspw. die Abbildung des Informationsflusses bei der Abwicklung eines Auftrags entlang der Wertschöpfungskette innerhalb eines Unternehmens, wie sie in Kapitel 4 thematisiert wurde.

Dieses Kapitel beschreibt zunächst die grundlegenden Merkmale des DREM-Basissystems. Darauf aufbauend erfolgt die Beschreibung der praktischen Umsetzung der in den ersten Kapiteln beschriebenen Grundlagen der Fuzzy-Set-Theorie. Das Ergebnis dieser Umsetzung ist die Implementation einer Fuzzy-Logik-Entscheidungskomponente zur Integration in die Workflow-Entwicklungsumgebung DREM. Es werden die Aufgaben der Entscheidungskomponente beschrieben und es wird auf die Bedeutung der objektorientierten Programmiersprache C++ für die Implementierung eingegangen. Anschließend wird die implementierte Klassenhierarchie[179] beschrieben und Sprachbeispiele innerhalb der DREM-Programmiersprache vorgestellt.

5.1 Workflow-Entwicklungsumgebung

Als Workflow bezeichnet man einen eingegrenzten Prozeß, der zur Produktion oder zum Verbrauch einer betrieblichen Leistung führt. Bei der Analyse eines Workflows - er ist unterteilt in verschiedene Ereignisse und Schritte - liegt der Schwerpunkt bei der Untersuchung seines dynamischen Verhaltens. Ein Workflow-Management-System[180] wie auch die Entwicklungsumgebung DREM dient zur Unterstützung der Organisation und Koordination des Datenflusses zwischen den Benutzern und den involvierten Softwarekomponenten innerhalb des Unternehmens. Im Rahmen eines Workflow-Projekts[181] werden mehrere Phasen durchlaufen, die bei der Planung anfangen und bei der Implementierung einer Workflow-Anwendung[182] aufhören. Im Normalfall liegt der

176 DREM = Development and Runtime-Environment Münster.
177 Vgl. Meyer, S. 72.
178 Vgl. Rehfeldt, Turowski (Workflow) (1995), S. 366f.
179 Vgl. Meyer (1990), S. 67.
180 Vgl. Jablonski (1995), S. 13.
181 Vgl. Hagemeyer et al. (1996), S. 23.
182 Vgl. Leymann (1996), S.7.

Hauptanwendungsaspekt von Workflow-Anwendungen in der softwaregestützten Steuerung von Arbeitsschritten durch Sicherstellung des Informationsflusses zwischen den Teilbereichen eines Unternehmens.

Eine auf DREM aufbauende Workflow-Anwendung zur Auftragsabwicklung kann durch die bereits im DREM-Basissystem vorhandenen Basistypen, wie die linguistischen Variablen und die unscharfer Mengen, die Unschärfe dieses Prozesses berücksichtigen. Entscheidungsmuster werden in Regelbasen gespeichert und können im Bedarfsfall ereignisgesteuert zur dezentralen Entscheidungsfindung herangezogen werden. Benötigt ein Aufgabenträger eine Entscheidungsunterstützung durch das DREM-System, so kann er ein solches Entscheidungsmuster auf die Datensituation anwenden. Es ist auch möglich, daß ein aktiver Geschäftsprozeß (in Form eines DREM-Planungsobjektes) den Aufruf einer Fuzzy-Methode zur Entscheidungsfindung erfordert.

5.2 Die Struktur von DREM

Der zentrale Begriff innerhalb des DREM - Systems ist das Objekt, d. h. die Abbildung einer Entität aus der Realwelt. Technisch ist ein solches Objekt als Einheit aus Struktur, bestehend aus Datenelementen und Verhalten, zu beschreiben. Durch das Versenden von Nachrichten (Message-Passing) können Objekte asynchron, auch über mehrere Stationen verteilt, miteinander kommunizieren.

5.2.1 Objektorientierung

DREM stellt als Basissystem bereits viele der grundlegenden Funktionalitäten[183] eines objektorientierten Datenbanksystems zur Verfügung. Diese werden im folgenden beschrieben:

5.2.1.1 Modellierungs-Eigenschaften

Neben der Darstellung von komplexen Objekten und Objektidentität können benutzerdefinierte Typen und Klassen modelliert werden. Ein Vererben von Struktur und Verhalten von Objekten ist möglich.

[183] Vgl. Vossen (1994), S. 347f und S. 577f.

5.2.1.2 Spracheigenschaften

Eine speziell auf die Einsatzpotentiale von DREM zugeschnittene Programmiersprache erlaubt die Kapselung von Daten, die Objektinitialisierung mit Konstruktoren und ad-hoc-Abfragen. Daneben erlaubt sie die Darstellung von Daten vagen Charakters und die asynchrone Kommunikation von Objekten. Zum Zeitpunkt der Objekttypdefinition wird aus Performanzgründen ein Kompilat erzeugt, welches bei Objektaktivität einer Ausführungseinheit übergeben und von dieser ausgeführt wird.

5.2.1.3 Systemeigenschaften

Neben der Sicherstellung der Persistenz werden Sperrmechanismen und Methoden zum optimierten Sekundärspeicherzugriff eingesetzt.

5.2.2 Parallelität

Aufgrund der Architektur der DREM-Kommunikationskomponente und der Verfügbarkeit von geeigneten Sprachkonstrukten können Methoden von verschiedenen Objekten innerhalb von DREM parallel, bzw. bei Ressourcenabhängigkeit nebenläufig ausgeführt werden, wodurch die Abbildung von unabhängigen Ereignissen erlaubt wird.

5.2.3 Verteilung

Beim Zugriff auf Objekte ist es für den Benutzer transparent, an welchem Ort sich ein von ihm gewünschtes Objekt befindet, ob es lokal auf seiner eigenen oder auf einer entfernt liegenden Station persistent ist. Im Bedarfsfall kann ein Objekt auf eine dedizierte Station migriert werden, so daß es nicht redundant gehalten und eine Objektreplizikation vermieden wird.

5.2.4 Unschärfe

Durch die Abbildungsmöglichkeit von Daten und Operatoren aus dem Bereich der Fuzzy-Logik kann DREM linguistisch beschriebene Zusammenhänge darstellen und berechnen. Unscharfe Zahlen und Mengen werden technisch auf Polygonzügen abgebildet, für unscharfe Zahlen sind die Grundrechenarten implementiert. Neben der Darstellung von Mengen mit scharfer Zugehörigkeitsbestimmung können innerhalb von DREM auch diskrete unscharfe Mengen abgebildet werden, welche die Darstellung gradueller Zugehörigkeiten von Elementen erlauben. Linguistische Variablen sind

benutzerdefinierte Typen, die den Wertebereich für auszuwertende Realweltbeobachtungen bilden. Ein Fakt bezieht sich immer auf eine linguistischen Variable und kann einerseits klassisch scharf, also durch eine Ganzzahl oder Realzahl dargestellt sein. Andererseits kann der Fakt im unscharfen Fall entweder einen Term einer linguistischen Variablen oder eine beliebige unscharfe Zahl innerhalb ihres Wertebereichs annehmen. Neben der Benutzung von Sprachkonstrukten besteht die Möglichkeit, Werte vagen Charakters durch eine graphische Benutzerschnittstelle anzulegen. Werden unscharfe Daten unter Verwendung von Sprachkonstrukten deklariert, so sind sie transient, wird auf Datenelemente von Objektinstanzen zugegriffen, so werden diese Änderungen persistent.

5.2.5 Entscheidungskomponente

WENN-DANN-Regeln, die auf linguistischen Variablen aufbauen, dienen der Beschreibung einer natürlichsprachlichen Regelbasis, auf deren Grundlage - ausgehend von einer konkreten Beobachtung - Entscheidungen generiert werden. Regeln können je nach ihrem Beitrag zur Entscheidungsfindung und/oder ihres Sicherheitsgrades gewichtet werden. In der hier zu beschreibenden Entscheidungskomponente können die Regeln hierarchisch angeordnet sein, so daß eine Verkettung von Schlußfolgerungen erfolgen kann. Als Schlußfolgerungsmethode kommt neben dem approximativen Schließen auch das numerisch aufwendigere plausible Schließen zum Einsatz.

5.3 Fuzzy-Logik-basierte Entscheidungskomponente

Die Workflow-Entwicklungsumgebung DREM besteht aus mehreren Modulen, einen Teilbereich davon zeigt Bild 5.1. Die Fuzzy-Logik-Entscheidungskomponente (FLE) gehört zum Bereich der Ausführungseinheit für Methoden (im folgenden Interpreter genannt), da ein Großteil der in der FLE angesiedelten Methoden von Routinen des Interpreters aufgerufen wird.

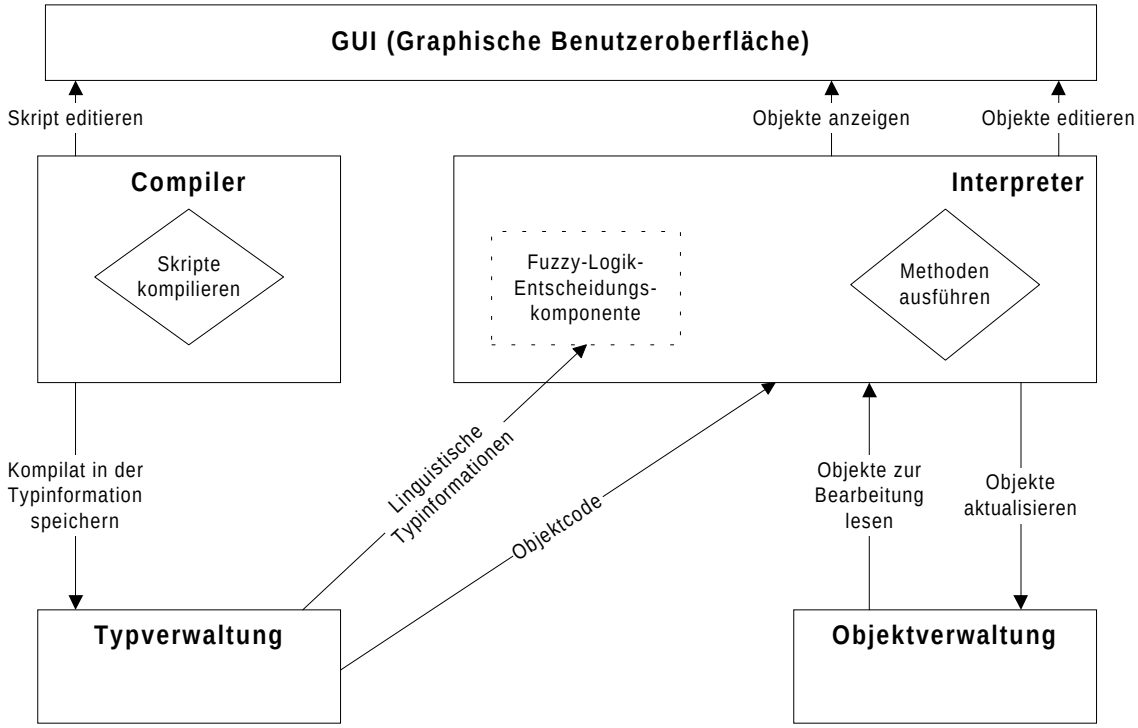


Bild 5.1: Ausschnitt aus der DREM-Systemarchitektur

Der Interpreter verarbeitet schrittweise die Methoden, die er in Form von Objektcode erhält. Es gibt eine Reihe von verschiedenen Opcodes[184], die mit den Befehlen der DREM--Programmiersprache korrespondieren. Die hier betrachteten Befehle zur Unterstützung von unscharfen Datentypen können in drei Hauptbestandteile aufgeteilt werden.

1. Befehle zur Definition und Manipulation von unscharfen Mengen.

2. Befehle zur arithmetischen Verknüpfung von unscharfen Zahlen.

3. Befehle zur Behandlung von Regeln, Fakten und zur Durchführung von Inferenzen.

5.4 Grundkonzeption der Fuzzy-Logik-Entscheidungskomponente

Da Regelbasen hierarchisch organisiert sein können, erlaubt die FLE dem Benutzer des DREM-Systems neben der Durchführung einfacher Fuzzy-Inferenzen noch die Verkettung mehrerer Inferenzen. Die Entscheidungskomponente ist in der Lage - aufgrund der bereits beschriebenen Überlegungen zur Unschärfepropagation - neben der einfachen Form des approximativen Schließens, die Funktionalität des plausiblen Schließens zur Verfügung zu stellen. Als Grundlage hierfür stehen verschiedene Implikationsoperatoren zur Verfügung. Als Schnittstellen zu skalaren Daten der Ausprägung Ganzzahl oder

[184] Vgl. Blieberger et al. (1992) S. 171.

Realzahl sind verschiedene Fuzzifikations- und Defuzzifikationsverfahren implementiert. Diese Verdichtungen von unscharfen Mengen können der Kommunikation mit nicht-fuzzifizierten Komponenten der betrieblichen Informationssystem-Architektur dienen. Der Datentyp der linguistischen Variablen eignet sich zur Modellierung unscharfer Fakten, er stellt natürlichsprachliche Sachverhalte wie „Deckungsbeitrag ist *hoch*“ dar. Die Domäne einer linguistischen Variablen ist in einer linguistischen Typinformation hinterlegt. Durch eine linguistische Approximation kann einer unscharfen Information, innerhalb des Kontextes einer linguistischen Variable, ein Term dieser Variablen zugeordnet werden. Fakten sind Ausprägungen eines bestimmten Typs der linguistischen Variablen und können zur Darstellung einer Realweltsituation in einem Faktenbasis-Objekt zusammengefaßt werden. Regeln in Form von WENN-DANN-Ausdrücken können in einem Regel-Objekt dargestellt werden. Mehrere Regeln werden in einem Regelbasis-Objekt zusammengefaßt. Bei arithmetischen Operationen werden die beteiligten unscharfen Operanden dahingehend untersucht, ob sie LR-Kriterien entsprechen, d. h. ob sie eine LR-Zahl oder ein LR-Intervall darstellen. Ist dies der Fall, so werden spezielle arithmetische Verfahren angewandt, die den Rechenaufwand zur arithmetischen Behandlung der Basistypen von unscharfen Zahlen und Mengen stark reduzieren. Um neben einfachen LR-Fuzzy-Datentypen auch komplexere unscharfe Mengen, bspw. Inferenzergebnisse, abbilden zu können, ist die interne Darstellung eines Fuzzy-Sets als Polygonzug implementiert.

5.5 Leistungsmerkmale der Fuzzy-Logik-Entscheidungskomponente

Es folgt eine Beschreibung der Leistungsmerkmale der FLE.

5.5.1 Unterstützte t- und s-Normen

Neben den bereits beschriebenen t-und s-Normen soll die Modellierung von unscharfen Beziehungen auch mit parametrisierten Operatoren möglich sein. Im einzelnen unterstützt die Fuzzy-Entscheidungskomponente die Verwendung der in Tabelle 18 angegebenen t-Normen und s-Normen.

klassische Operatoren	parametrisierte Operatorenfamilien
Minimum/Maximum	Hamacher-Operatoren
Algebraische(s) Produkt/Summe	Yager-Operatoren
Beschränkte Differenz/Summe	Weber-Operatoren
Drastische Differenz/Summe	Dubois/Prade-Operatoren
Algebraischer t-Quotient/s-Quotient	

Tabelle 18: Unterstützte t- und s-Normen

5.5.2 Unterstützte Durchschnittsoperatoren

Innerhalb der Fuzzy-Entscheidungskomponente stehen die in Tabelle 19 aufgeführten Durchschnittsoperatoren zur Verfügung. Zur flexiblen Anpassung der Kompensation zwischen den Operatoren können auch parametrisierte Durchschnittsoperatoren zur Modellierung verwendet werden.

klassische Operatoren	parametrisierte Operatoren
arithmetisches Mittel	min/max-Kompensator
harmonisches Mittel	prod/sum-Kompensator
geometrisches Mittel	konvexer min/max-Kompensator

Tabelle 19: Unterstützte Durchschnittsoperatoren

5.5.3 Inferenzmethoden

DREM stellt je nach Bedarf den Mechanismus des approximativen Schließens oder die numerisch aufwendigere Methode des plausiblen Schließens zur Verfügung. Für viele Regelungszwecke reicht das approximative Schließen aus, jedoch kann bei der Untersuchung auf Wirkungszusammenhänge bei der Weitergabe von Unsicherheiten die Anwendung des plausiblen Schließens angebrachter sein.

5.5.3.1 Approximatives Schließen

Beim approximativen Schließen wird durch die Bildung von Erfüllungsgraden zwischen Regeln und vorliegenden Fakten eine Kenngröße zur Gewichtung der Teilschlußfolgerung gewonnen. Als Verknüpfungsoperator kann eine beliebige t-Norm verwendet werden. Liegen mehrere Schlußfolgerungen zu derselben linguistischen Variablen vor, so werden diese mit einem s-Norm-Operator verknüpft.

5.5.3.2 Plausibles Schließen nach dem generalisierten Modus Ponens

Wie bereits ausgeführt, erlaubt das plausible Schließen als Erweiterung zum approximativen Schließen die explizite Weitergabe des Sicherheitsniveaus und des Wahrheitswertes von Prämissen an Schlußfolgerungen. Zur Durchführung des plausiblen Schließens werden deshalb Implikationsoperatoren benötigt. Die folgenden Implikationsoperatoren werden von der FLE zur Verfügung gestellt.

- Zadeh-Implikation,
- Mamdani-Implikation,
- Goedel-Implikation,
- Gaines-Implikation,
- Kleene-Dienes-Implikation,
- Reichenbach-Implikation,
- Larsen-Implikation,
- Goguen-Implikation.

5.5.4 Mehrstufigkeit zur Hierarchieabbildung

Die FLE erlaubt die Verkettung von Regeln innerhalb von Regelbasen. Das verkettete Schließen wird auch als syllogistisches Schließen bezeichnet. Ein Syllogismus liegt beispielsweise in folgender Regelbasis vor:

```
...
(1) Wenn P = A Dann Q = B
(2) Wenn Q = B Dann R = C
...
```

Da die Regel (2) nur in Abhängigkeit der Gültigkeit der Regel (1) aktiviert wird, muß die Inferenz mehrstufig - gemäß einer bestimmten Hierarchie - ablaufen. In der ersten Stufe wird ein Zwischenergebnis für eine linguistische Variable vom Typ Q gebildet, welches beim Abarbeiten der zweiten Regelbasisstufe temporär in die Faktenbasis aufgenommen wird. Um eine derartige Hierarchie innerhalb einer Regelbasis aufzubauen, müssen die Regeln jeweils mittels eines Attributes einer bestimmten Hierarchieebene zugeordnet werden. Die Beispielanwendung in Kapitel 7 greift die Möglichkeit des verketteten Schließens auf.

5.5.5 Input/Output-Verhalten

Bei der Entscheidungsfindung werden aus Eingangsgrößen Schlußfolgerungen gewonnen, hierbei besteht zuweilen die Notwendigkeit, einzelne Bewertungen zu abgeleiteten Bewertungen zu aggregieren. Es sind folgende Fälle denkbar:

5.5.5.1 SISO - Single Input Single Output

Durch Auswertung einer Eingangsgröße (z. B. Auftragsvolumen) wird eine Schlußfolgerung (z. B. Risikostatus des Auftrages) gewonnen.

5.5.5.2 MISO - Multiple Input Single Output

Durch Auswertung mehrerer Eingangsgrößen (z. B. Kundenbewertung und Auftragsvolumen) wird eine Schlußfolgerung (z. B. Auftragspriorität) gewonnen.

5.5.5.3 MIMO - Multiple Input Multiple Output

Durch Auswertung mehrerer Eingangsgrößen (z. B. Kundenbewertung und Auftragsvolumen) werden mehrere Schlußfolgerungen (z. B. Auftragspriorität und Risikostatus) gewonnen.

5.5.6 Arten der Defuzzifikation

Um eine unscharfe Menge, bspw. die Schlußfolgerung einer Fuzzy-Inferenz, in eine reellwertige Größe zu transformieren, führt man eine Defuzzifikation durch. Die FLE unterstützt die folgenden Methoden (vgl. 3.5.4.1) der Defuzzifikation:

- Schwerpunkt-Methode,
- Maximum-Methode,
- Mitte-der-Maxima-Methode,
- Randmaximum-Methode.

5.6 Technische Realisierung

Grundlage für die Implementierung eines Systems in einer objektorientierten Programmiersprache wie C++ ist das vorherige Entwerfen eines Datenmodells, welches die Abhängigkeiten der implementierten Klassen darstellt.

Der Begriff Klasse im Sinne von C++ ist eine Menge gleichartiger Objekte und ist durch folgende Merkmale[185] charakterisiert:

- Sie umfaßt eine (auch leere) Menge von Datenelementen. Die Basisdatentypen der Elemente können verschieden sein.
- Sie enthält eine Schnittstelle in Form von Elementmethoden, die das Verhalten der Objekte und die Zugriffe auf die Datenelemente beschreibt.
- Die Schnittstelle der Klasse ist in mehrere Bereiche, die den Zugriff auf die Datenelemente und Methoden reglementieren[186], eingeteilt.

5.6.1 Zur verwendeten Basisklassenbibliothek Microsoft Foundation Classes

Die FLE stützt sich, als Teil von DREM, auf den vom Microsoft Visual C++-Compiler zur Verfügung gestellten Microsoft Foundation Classes (MFC) ab. Diese Klassenbibliothek stellt Container zur Listen- und Kollektionsbehandlung zur Verfügung.

5.7 Aspekte der Objektorientierung

Die Wahl einer Programmiersprache nach dem Paradigma der Objektorientierung läßt sich speziell bei der Wahl von C++ durch die folgenden Aspekte motivieren[187].

5.7.1 Kopplung von Struktur und Verhalten

Klassen beschreiben in C++ die Struktur und das Verhalten von Objekten. Die Datenelemente einer Klasse bilden die Struktur; als Verhalten bezeichnet man die Gesamtheit der Methoden einer Klasse. Durch die Kopplung von Verhalten und Struktur können zusammengehörige Sachverhalte auch in ihrer internen Abbildung als EDV-technische Einheit verarbeitet werden.

---

185 Vgl. Mayer (1993), S. 99.
186 Vgl. Meyers(1995), S. 130.
187 Vgl. Mayer (1993), S. 99f.

5.7.2 Code-Recycling durch Vererbung

Die von C++ zur Verfügung gestellten Vererbungsmechanismen erlauben es neue Klassen von bereits deklarierten Klassen in ihrer Struktur und in ihrem Verhalten abzuleiten. Dies setzt jedoch ein entsprechendes Design der bereits vorhandenen Klassen voraus. Das geerbte Verhalten der abgeleiteten Klassen kann auf Wunsch durch das Überschreiben der entsprechenden Methode modifiziert werden.

5.7.3 Robustheit durch Kapselung

Durch die Möglichkeit den Zugriff auf bestimmte Strukturelemente wie Daten und Methoden zu kapseln kann das Verhalten eines Objektes vor Seiteneffekten geschützt werden. Der Entwickler einer Klasse steht vor der Wahl, ob er den Zugriff auf Strukturelemente frei zuläßt, ihn verbietet oder durch die Einführung von Zugriffsmethoden reguliert. Dieser Schutz vor unkontrolliertem Zugriff auf den privaten Daten- und Funktionsbereich einer Klasse wird durch den Begriff *information hiding* beschrieben.

5.8 Design der Klassenhierarchie

Der objektorientierte Entwurf (Design) kann in folgende Teilschritte eingeteilt werden.

- Identifikation der als Klassen abzubildenden Realweltausschnitte,
- Evaluierung des Klassenverhaltens,
- Evaluierung der Abhängigkeiten zwischen den Klassen,
- Definition der Schnittstellen zur Intra-Klassen-Kommunikation.

Diese Schritte werden am vorliegenden Beispiel der FLE beschrieben.

5.9 Klassen

Bei der Identifikation der zu implementierenden Klassen ist darauf zu achten, daß ein hinreichender Detaillierungsgrad eingehalten, aber ein hinreichend hohes Abstraktionsniveau gewahrt bleibt. STROUSTRUP beschreibt das Ziel dieser Abwägung als Gleichgewicht zwischen Einfachheit und Allgemeinheit.

Folgende Klassen lassen sich in Teilbereiche aufteilen.

5.9.1 Die Klasse CFuzzySetEntry

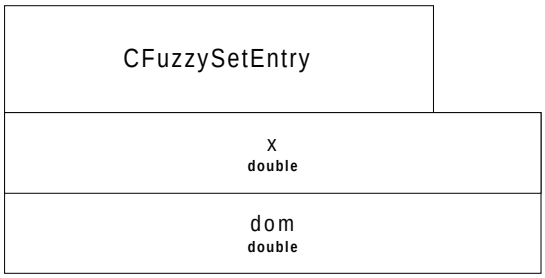


Bild 5.2: Klasse CFuzzySetEntry

Die Klasse CFuzzySetEntry beschreibt einen Punkt innerhalb eines vom CFuzzySet dargestellten Polygonzuges. Die Datenelemente dieser Klasse sind der Wert auf der x-Achse und der entsprechende Zugehörigkeitsgrad. Das Verhalten dieser Klasse wird in der Datei FCSE.H und FCSE.CPP beschrieben. Die wichtigsten Methoden sind das Setzen des x-Wertes und des Zugehörigkeitsgrades.

5.9.2 Die Klasse CInterval

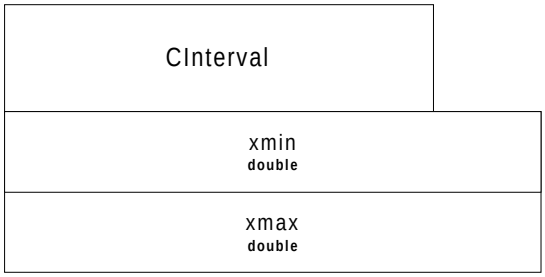


Bild 5.3: Klasse CInterval

Die Klasse CInterval wird zur Beschreibung des Wertebereichs eines FuzzySets und einer linguistischen Variable benötigt. Ein Objekt dieser Klasse hat zwei reellwertige Datenelemente, die die Spannweite [xmin;xmax] eines Intervalls repräsentieren. In den Dateien FCINTRVL.H und FCINTRVL.CPP wird das Verhalten dieser Klasse beschrieben. Die wichtigsten Methoden eines CInterval-Objekts sind das Setzen und Lesen der Randwerte sowie die Addition von Intervallen.

5.9.3 Die Klasse CFuzzySet

Scharfe Mengen können als Klasse identifiziert werden[188]. Ein Objekt der Klasse CFuzzySet liefert zu einem beliebigen reellen Wert x einen Zugehörigkeitswert zu einer unscharfen Menge, die durch eine beliebige abschnittsweise lineare, stetige Zugehörigkeitsfunktion beschrieben ist. Die Vorteile der Implementierung einer abschnittsweise linearen Darstellung von Fuzzy-Sets finden sich bei BAEKELAND und KERRE[189]. Die einzelnen Punkte werden als Instanzen der Klasse CFuzzySetEntry abgebildet und werden in einem CObArray[190], einer von der MFC zur Verfügung gestellten Containerklasse[191], gespeichert.

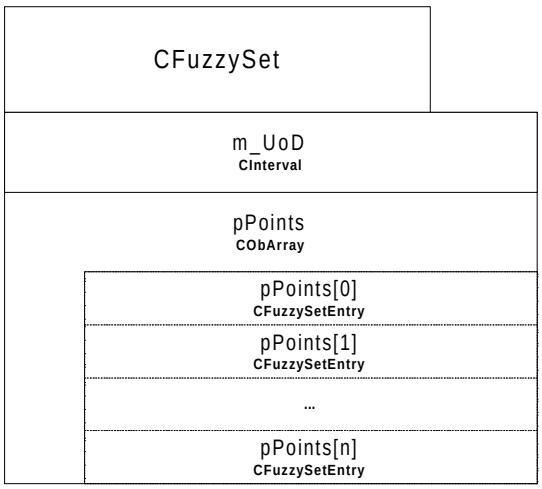


Bild 5.4: Klasse CFuzzySet

Spezielle Konstruktoren[192] erlauben die direkte Definition von Dreiecks-Zahlen und unscharfen Intervallen, deren linke und rechte Referenz-Funktionen allerdings linear sein muß. Die interne Darstellung einer unscharfen Menge als Polygonzug erlaubt die Abbildung von komplexen Fuzzy-Mengen, die als Zwischenergebnisse und Ergebnisse von Inferenzen entstehen können. In den Dateien FCSET.H und FCSET.CPP wird das Verhalten dieser Klasse beschrieben. Die wichtigsten Methoden sind das Einfügen und Löschen von Punkten eines Polygonzuges, das Ermitteln von Defuzzikationsergebnissen nach verschiedenen Methoden und die Verknüpfung durch einen beliebigen Fuzzy-Operator. Die Klasse CFuzzySet überprüft den Polygonzug auf LR-Kriterien und führt, falls diese erfüllt sind, eine Verknüpfung nach den LR-Vorschriften durch. Weitere

188 Vgl. 2.3.3.
189 Vgl. Baekeland, Kerre (1988), S. 120.
190 Vgl. Prosise (1996), S. 212.
191 Vgl. Stroustrup (1992), S. 332.
192 Vgl. Stroustrup (1992), S. 168.

Operationen, wie das Ermitteln einer konvexen Hülle[193], können zur Umwandlung von unscharfen Mengen in unscharfe Zahlen dienen.

5.9.4 Die Klasse CLingTerm

Ein linguistischer Term besteht aus einem CString[194]-Objekt, welches den Bezeichner darstellt, und einem Wert in Form einer unscharfen Menge, d. h. einem CFuzzySet-Objekt.

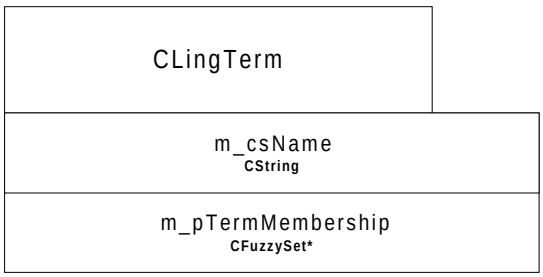


Bild 5.5: Klasse CLingTerm

Die Methoden eines CLingTerm-Objekts werden in den Dateien FCLV.H und FCLV.CPP beschrieben.

5.9.5 Die Klasse CLingVar

Die Klasse CLingVar beschreibt die technische Implementierung einer linguistischen Variablen. Zur Beschreibung einer linguistischen Variablen dienen die Datenelemente Name, Wertebereich und eine Kollektion ihrer möglichen linguistischen Ausprägungen. Innerhalb der DREM-Systemhierarchie wird durch ein CLingVar-Objekt die Domäne einer linguistischen Variablen abgebildet. Eine Übersicht über den Aufbau eines CLingVar-Objekts gibt Bild 5.6.

193 Vgl. Sedgewick (1992), S. 411f und Luther, Ohsmann (1989), S.38f.
194 Vgl. Prosise (1996), S. 24.

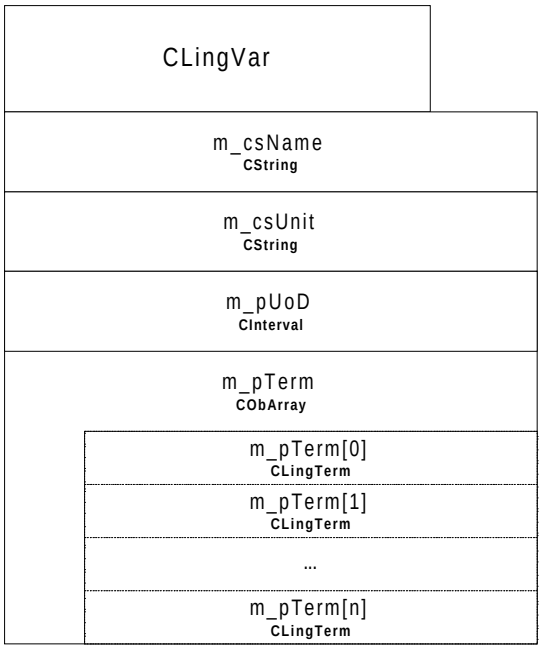


Bild 5.6: Klasse CLingVar

Das Verhalten eines CLingVar-Objekts wird in den Dateien FCLV.H und FCLV.CPP beschrieben.

5.9.6 Die Klasse CFakt

Durch ein Objekt der Klasse CFakt wird eine Instanz einer linguistischen Variablen beschrieben (im folgenden Fakt).

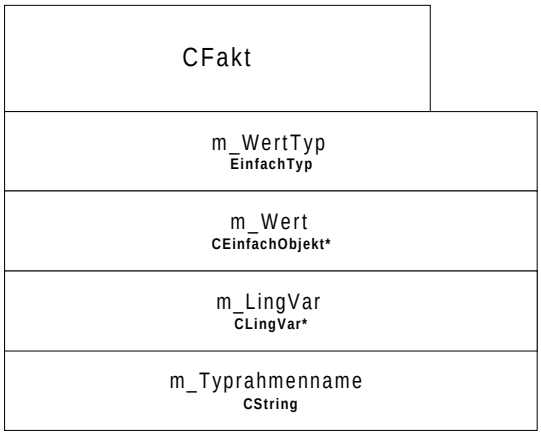


Bild 5.7: Klasse CFakt

Die Ausprägung eines Fakten kann entweder durch eine scharfe oder eine unscharfe Information beschrieben sein. Enthält das CFakt-Objekt in dem Attribut m_Wert vom Typ CEinfachObjekt ein CGanzzahl-Objekt oder ein CRealzahl-Objekt, so wird eine scharfe Information abgebildet. Ist im Attribut m_Wert ein Objekt vom Typ

CZeichenkette oder vom Typ CFuzzySet enthalten, so werden unscharfe Informationen abgebildet. Das Verhalten eines CFakt-Objekts ist in den Dateien TYPFRAME.H und TYPFRAME.CPP ersichtlich.

5.9.7 Die Klassen CRulePart, CAntecedent, CConsequent

Die Klasse CRulePart ist die Basisklasse zur Abbildungen von Regelteilen in Form der Wenn-Bedingungen (CAntecedent) und der Dann-Bedingungen (CConsequent). Aus dem Aufbau von Regeln der Form

WENN LingVarA = TermA DANN LingVarB = TermB

ist ersichtlich, daß der WENN- und der DANN-Teil einer Regel strukturgleich sind. Deshalb beinhaltet ein Objekt der Klasse CRulePart einen Verweis auf ein CLingVar-Objekt und einen Text in Form eines CStrings, der über den verwendeten Term Auskunft gibt(siehe Bild 5.8). Die Klassen CAntecedent und CConsequent werden durch Vererbung[195] von CRulePart abgeleitet.

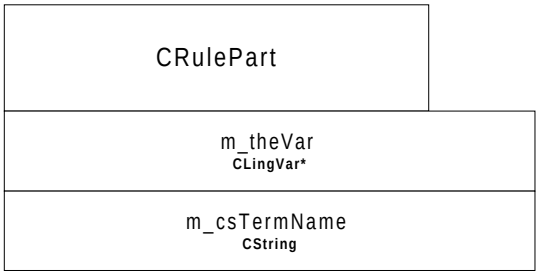


Bild 5.8: Klasse CRulePart

Die Methoden der Klasse CRulePart sind in den Dateien FCRULE.H und FCRULE.CPP aufgeführt.

5.9.8 Die Klasse CAntecedent

Die Klasse CAntecedent entspricht der Wenn-Klausel einer Regel. Diese Klasse ist durch Vererbung von der Klasse CRulepart abgeleitet und entspricht in Aufbau und Verhalten dieser Klasse. Die Methoden der Klasse CAntecedent sind in den Dateien FCRULE.H und FCRULE.CPP aufgeführt.

---

[195] Vgl. Meyer (1990), S. 67.

5.9.9 Die Klasse CConsequent

Die Klasse CConsequent ist die Darstellung einer Dann-Klausel einer Regel. Diese Klasse ist durch Vererbung von der Klasse CRulePart abgeleitet und entspricht in Aufbau und Verhalten dieser Klasse. Die Methoden der Klasse CConsequent sind in den Dateien FCRULE.H und FCRULE.CPP aufgeführt.

5.9.10 Die Klasse CRule

Eine Regel (engl. Rule) kann in einen Wenn- und in einen Dann-Teil aufgeteilt werden. Der Wenn-Teil ist ein Feld von Objekten der Klasse CAntecedent (Bedingung).

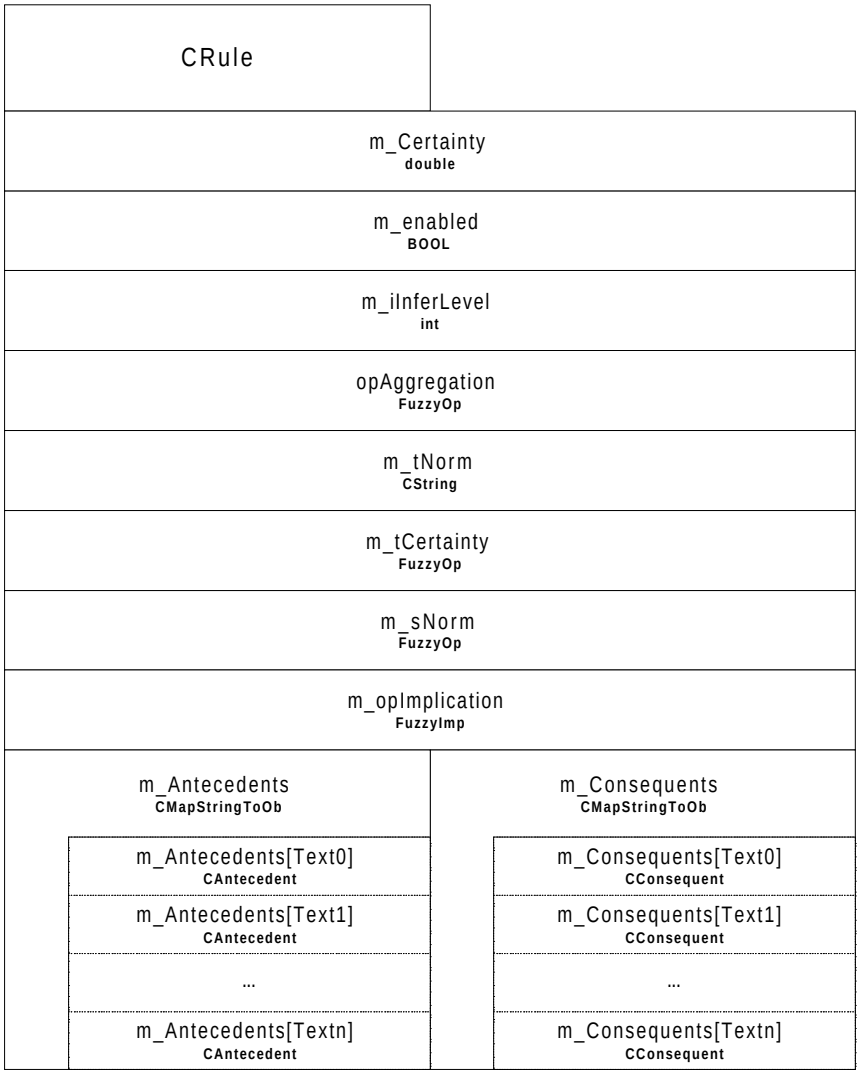


Bild 5.9: Klasse CRule

Diese Objekte werden mit einem Aggregationsoperator, bei UND meistens dem *min*-Operator bei ODER dem *max*-Operator verknüpft, um dann mittels Inferenz Schlußfolgerungen aus den Objekten der CConsequent-Liste durch Inferenz ein Objekt vom

Typ CFakt zu aggregieren. Eine Regel kann mit einem reellwertigen Sicherheitsmaß versehen sein, welches Auskunft darüber gibt, ob die Regel als vollkommen sicher vorausgesetzt wird, bei einem Sicherheitsmaß von 1,0 oder ob sie unsicher sind, d. h. ein Sicherheitsmaß kleiner als 1,0 besitzt. Regeln können vor Ausführung einer Inferenz abgeschaltet werden, wenn der logische Schalter m_enabled auf FALSE gesetzt wird. Die Angabe eines Fuzzy-Implikationsoperators erlaubt flexible Festlegung der Art des Übergabe der Sicherheit von der Wenn- an die Dann-Klausel. Eine Übersicht über den Aufbau eines CRule-Objektes gibt die Darstellung in Bild 5.9. Die Methoden der Klasse CRule sind in den Dateien FCRULE.H und FCRULE.CPP aufgeführt.

5.9.11 Die Klasse CRulebase

Mehrere logisch zusammengehörige Regeln werden in einer Regelbasis (Bild 5.10) zusammengefaßt und gemeinsam gepflegt.

Die Hierarchiestufe gibt an, welche Regeln zuerst abgearbeitet werden müssen, um Schlußfolgerungen für die nächste Ebene zu erhalten. Bei der Inferenzbildung (d. h. der Kombination der Regelbasis mit einer Faktensituation spezifiziert durch eine Faktenbasis) werden zuerst die Regel der Hierarchiestufe 0 evaluiert, dann die Ergebnisse ermittelt und an die nächste Hierarchiestufe 1 weitergereicht. Dies geschieht bis alle Stufen der Regelbasis abgearbeitet sind. Die Methoden der Klasse CRulebase sind in den Dateien FCRULE.H und FCRULE.CPP aufgeführt.

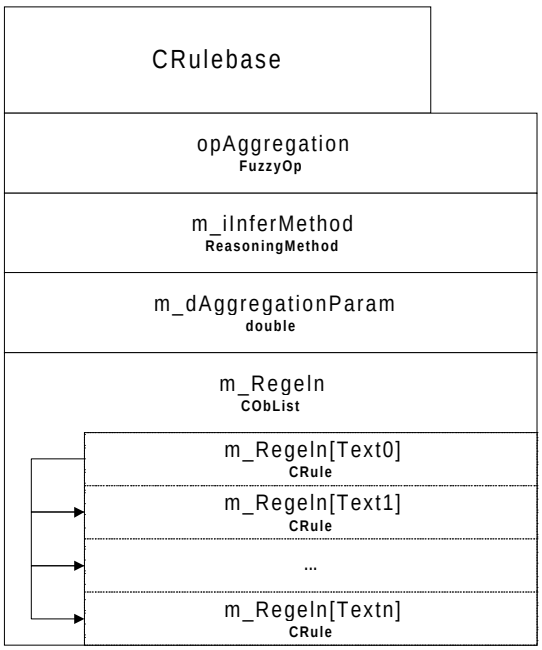


Bild 5.10: Klasse CRulebase

5.10 Einbettung der Fuzzy-Logik-Entscheidungskomponente in DREM

Um die Funktionen der FLE dem Benutzer des DREM-Systems zur Verfügung zu stellen, wurden die Sprachstruktur[196] und die interne Funktionalität des Interpretermoduls um folgende Elemente erweitert.

5.10.1 Datentypen

Auf die von der FLE bereitgestellten Datentypen kann auch in der DREM-Programmiersprache zurückgegriffen werden. Da die Funktionalität keine andere ist als bei den zugrundeliegenden C++-Klassen, werden die davon abgeleiteten DREM-Datentypen nur kurz beschrieben. Es folgen für die einzelnen Datentypen jeweils eine kurze Erläuterung und ein kurzes Sprachbeispiel[197].

5.10.1.1 Unscharfe Mengen

Durch den Datentyp FUZZYSET können stetige unscharfe Mengen abgebildet werden. Die Zugehörigkeitsfunktion der unscharfen Menge wird durch den Verlauf des zugrundeliegenden Polygonzugs bestimmt. Unscharfe Mengen können mittels Fuzzy-Operatoren miteinander kombiniert oder als unscharfe Werte Fakten zugewiesen werden. Zur Transformation in einen skalaren Wert ist es möglich, eine Defuzzifikation durchzuführen. Das folgende Beispiel zur DREM-Programmiersprache zeigt, wie eine unscharfe Menge angelegt und in einer Maske angezeigt wird.

```
DECLARE a :FUZZYSET;
BEGIN
       CREATE_FUZZYSET((1.0,0.0),(3,1.0),(4,0.3),(6,0.6),(8,0.0),a);
       EDIT(a);
END
```

5.10.1.2 Unscharfe Zahlen

Der Datentyp FUZZYNUMBER bildet stetige unscharfe Mengen ab. Die Zugehörigkeitsfunktion der unscharfen Zahl wird durch den Verlauf des zugrundeliegenden Polygonzugs bestimmt. Unscharfe Zahlen können arithmetisch miteinander verknüpft werden oder mittels Fuzzy-Operatoren miteinander kombiniert werden oder als unscharfe Werte Fakten zugewiesen werden. Zur Transformation einer unscharfen Zahl in einen skalaren Wert ist es möglich, eine Defuzzifikation durchzuführen. Das folgende Beispiel zur DREM-Programmiersprache zeigt, wie eine unscharfe Zahl angelegt und in einer Maske angezeigt wird.

---

196 Zu Syntax, Semantik und Pragmatik der DREM-Programmiersprache vgl. N.N., (1996).

197 Vgl. N.N., (1996).

```
DECLARE a : FUZZYNUMBER;
BEGIN
       CREATE_FUZZYNUMBER((1.0,0.0),(3,1.0),(4,0.7),(6,0.4),(8,0.0),a);
       EDIT(a);
END
```

5.10.1.3 Linguistische Fakten und Faktenbasen

Durch den Fakt-Datentyp können linguistische Variablen angelegt werden, die sich auf einen benutzerdefinierten Lingvar-Typ[198] des DREM-Systems beziehen. Fakten können scharf oder unscharf sein. Im scharfen Fall können sie ganzzahlige oder reelle Werte annehmen, im unscharfen Fall kann der Name eines linguistischen Terms oder eine unscharfe Menge zugewiesen werden.

Die Klasse FACTBASE ist ein Container für Fakt-Variablen. Eine Entscheidungssituation kann sich durch mehrere Fakten charakterisieren, die dann in einer Faktenbasis zusammengefaßt werden. Das folgende Beispiel legt zwei Fakten an und fügt sie einer Faktenbasis hinzu. Abschließend kann diese Faktenbasis editiert werden.

```
DECLARE  db : DECKUNGSBEITRAG;
         av : AUFTRAGSVOLUMEN;
         fb : FACTBASE;
BEGIN
       db = LINGVAR(DECKUNGSBEITRAG,"niedrig");     -- unscharfen Fakt einfuegen
       av = LINGVAR(AUFTRAGSVOLUMEN,3000.00);       --   scharfen Fakt einfuegen
       ADD_ENTRY(fb,db);
       ADD_ENTRY(fb,av);
       EDIT(fb);
END
```

5.10.1.4 Regelbasen

In einem Objekt der Klasse RULEBASE können Entscheidungsmuster abgelegt werden. Durch sie kann eine Faktensituation ausgewertet und eine unscharfe Schlußfolgerung gezogen werden. Das Ergebnis einer solchen Schlußfolgerung ist wiederum eine Faktenbasis. Das obige Beispiel zur Faktenbasis wird nun um eine Regelbasis mit zwei Entscheidungsregeln erweitert. Nun kann mittels einer unscharfen Inferenz die Faktensituation ausgewertet werden. Die resultierende Schlußfolgerung kann in weiteren Berechnungen verwendet werden.

198 Vgl. N.N., (1996).

```
DECLARE
               db                    : DECKUNGSBEITRAG;
               av                    : AUFTRAGSVOLUMEN;
               ap                    : AUFTRAGSPRIORITAET;
               ausgangslage          : FACTBASE;
               resultat              : FACTBASE;
               ermittle_prio         : RULEBASE;
BEGIN
       RULEBASE ermittle_prio
       rule
               level 0
               rule_certainty 0.8, fuzzy_min;
               inference fuzzy_algeb_prod;
               combine        fuzzy_min;
               preconditions  DECKUNGSBEITRAG  is "niedrig";
                              AUFTRAGSVOLUMEN  is "gering";
               conclusions    AUFTRAGSPRIORITAET is "unwichtig";
       rule
               level 0
               rule_certainty 0.9, fuzzy_min;
               inference fuzzy_algeb_prod;
               combine        fuzzy_min;
               preconditions  DECKUNGSBEITRAG  is "hoch";
                              AUFTRAGSVOLUMEN  is "gering";
               conclusions    AUFTRAGSPRIORITAET is "wichtig";
       endbase;
       db = LINGVAR(DECKUNGSBEITRAG,"niedrig");     -- unscharfen Fakt einfuegen
       av = LINGVAR(AUFTRAGSVOLUMEN,3000.00);       --   scharfen Fakt einfuegen
       ADD_ENTRY(ausgangslage,db);
       ADD_ENTRY(ausgangslage,av);
       EDIT(ausgangslage);
       resultat=apply(ermittle_prio, ausgangslage,fuzzy_max);
       Edit(resultat);
END
```

5.11 Zusammenfassung

In diesem Kapitel wurde die Struktur der Workflow-Entwicklungsumgebung DREM beschrieben. DREM stellt bereits als Basissystem viele Funktionalitäten zur Verfügung, die der realitätsnahen und, durch die Integration linguistischer Datentypen, unscharfen Modellierung von Workflow- oder PPS-Zusammenhängen dienen. Durch die Verbindung des objektorientierten Ansatzes mit der expliziten Unschärfe ermöglicht DREM nicht nur die Abbildung der Beziehungen und des Verhalten von Objekten der Realwelt, sondern auch die Darstellung der attributinhärenten Unsicherheit und Unschärfe. Basis dafür sind sowohl die in der DREM-Sprache als auch in den DREM-Datentypen zur Verfügung gestellten Konzepte zur Behandlung von Unschärfe. Die Erweiterung der DREM-Programmiersprache um Fuzzy-Sprachelemente ermöglicht es, neben Fuzzy-Informationsobjekten auch Fuzzy-Planungsobjekte[199] abzubilden.

[199] vgl. Abschnitt 4.4.

6. Anwendungsbeispiel zur industriellen Auftragsabwicklung

Dieses Kapitel zeigt ein Beispiel zur Unterstützung des Workflowmanagements innerhalb der industriellen Auftragsabwicklung anhand der Überlegungen von BECKER, REHFELDT und TUROWSKI[200], sowie von BITTERLICH, FROEBEL und LULL[201]. Die grundlegenden Gedanken sind im folgenden kurz zusammengefaßt.

Ziel dieser Beispielanwendung ist die Darstellung des Workflows innerhalb der Auftragsabwicklung. Die vom Vertrieb aufgrund vager Vertriebsinformationen erzeugten, vorläufigen Kundenaufträge enthalten unscharfe Attribute in Form von Mengen und Zeiten. Diese Integration von Unschärfe erfolgt innerhalb des Workflows vom Vertrieb über die Disposition bis in den Fertigungsbereich (siehe Bild 4.5). Die unscharfen Beziehungen zwischen den Einflußgrößen fließen als Regelbasen innerhalb von Fuzzy-(Planungs-)Objekten in Entscheidungsmodelle ein. Exemplarisch werden zwei Teilbereiche des Workflows mit Regelbasen versehen. Auf der Stufe des Vertriebs wird ausgehend von der Kundenbewertung, der Auftragsgröße und der Auftragssicherheit durch Fuzzy-Inferenz eine vorläufige Vertriebspriorität ermittelt und in dem entsprechendem Attribut des Auftrags vermerkt. Bild 6.1 zeigt diese Zusammenhänge graphisch auf.

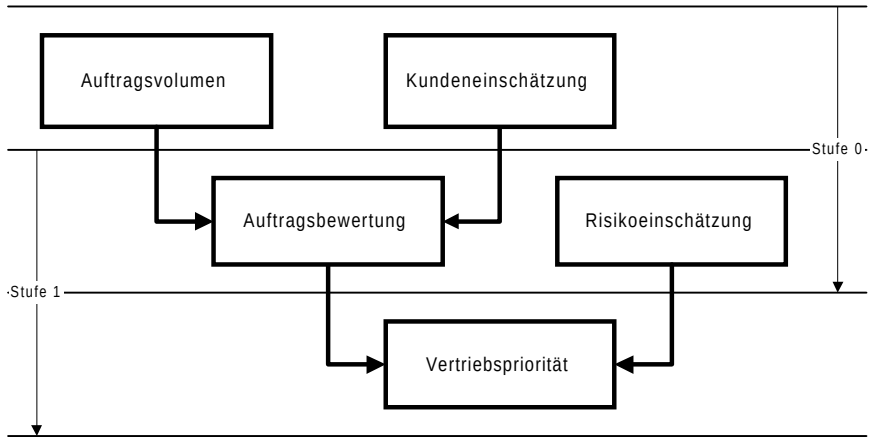


Bild 6.1: Entscheidungshierarchie zur Ermittlung der Vertriebspriorität

Der Auftrag durchfließt die Materialbedarfsplanung und wird im Fall der Eigenerstellung in einen Fertigungsauftrag transformiert. Bei der Einplanung dieses Auftrags innerhalb der Programmplanung wird auf Basis der unscharfen Entscheidungsgrößen Auftragsvolumen, Schlupfzeit und Vertriebspriorität eine unscharfe Fertigungspriorität ermittelt. Analog zur Entscheidungsfindung im Vertrieb werden auch in diesem Bereich die vorliegenden Vorbewertungen hierarchisch gemäß Bild 6.2 zu einer

200 Vgl. Becker et al. (Auftragsabwicklung) (1996), S.53.
201 Vgl. Bitterlich et al. (1996), S. 66.

Gesamtbewertung aggregiert. Anhand dieser Gesamtbewertung wird der Auftrag vorläufig eingeplant und behält bis zu einer Neuplanung seine Fertigungspriorität. Der Vertrieb informiert nun den Kunden - aufgrund der Kenntnis des vorläufigen Termins der Einlastung - über den vorläufigen Liefertermin. Im folgenden werden zuerst die Regelbasen zur Entscheidungsfindung definiert. Diese Regelbasen werden innerhalb von DREM in Fuzzy-Methoden umgesetzt und dem entsprechenden Planungsobjekttyp zugeordnet.

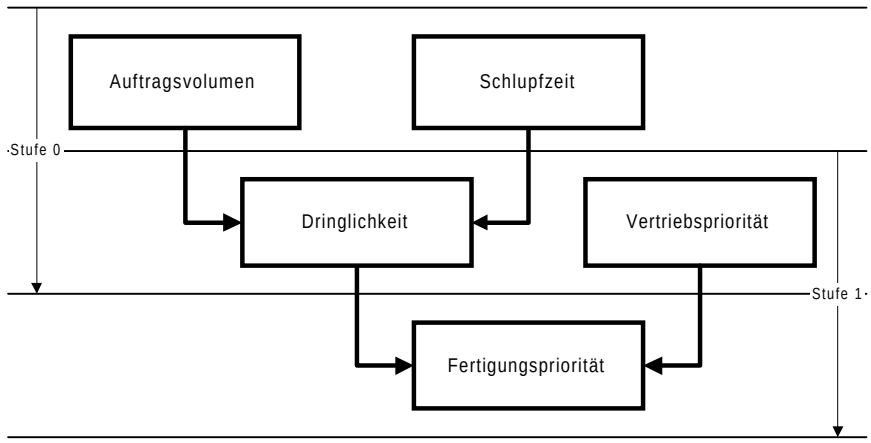


Bild 6.2: Entscheidungshierarchie zur Ermittlung der Fertigungspriorität

6.1 Ermittlung der Vertriebspriorität

Aus dem Vertrieb gehen die folgenden Parameter für die Entscheidungsfindung in das DREM-System ein:

Auf der Stufe 0 werden die Parameter Auftragsvolumen (Bild 6.3) und Kundeneinschätzung zur Ermittlung der Auftragsbewertung benötigt. Dieses Ergebnis fließt in Stufe 1 zusammen mit der Risikoeinschätzung in die Ermittlung der Vertriebspriorität ein. Die Definition dieser linguistischen Variablentypen im DREM-System wird am Beispiel des Auftragsvolumens beschrieben.

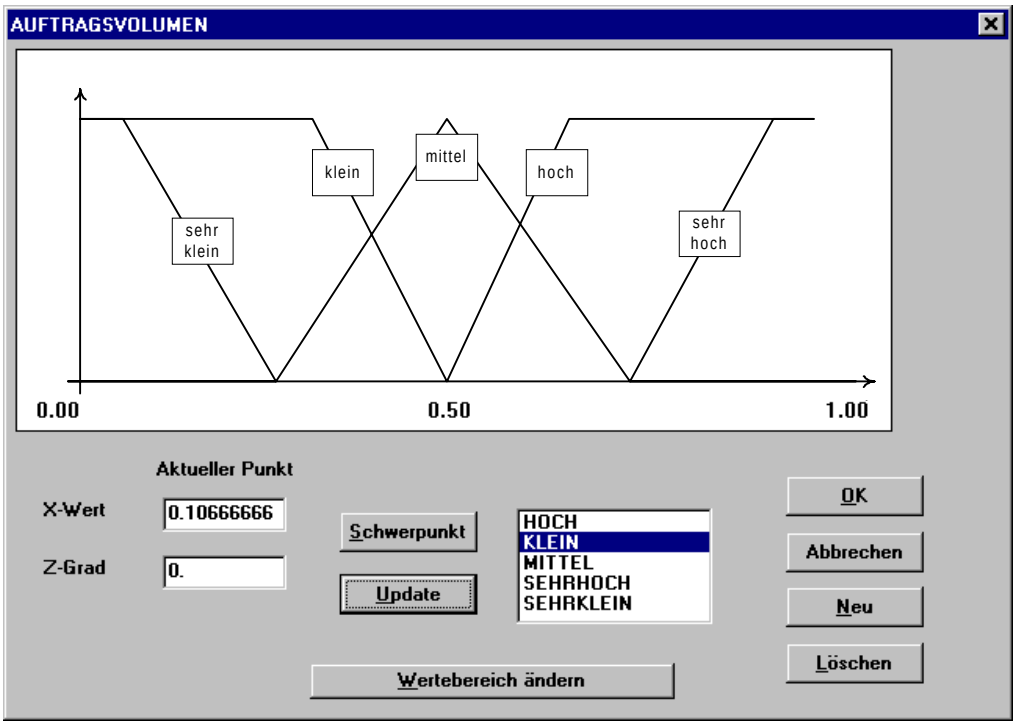


Bild 6.3: Linguistische Variable „Auftragsvolumen“[202]

Eine linguistische Variable bzw. ein Fakt des Typs Auftragsvolumen kann einen der fünf Basisterme *sehr klein*, *klein*, *mittel*, *hoch* und *sehr hoch* annehmen. Der Term *mittel* wird durch eine trianguläre Fuzzy-Zahl und die Terme *sehr klein*, *klein*, *hoch* sowie *sehr hoch* werden als trapezoidale Fuzzy-Intervalle, die links (bei *sehr klein* und *klein*) und rechts (bei *hoch* und *sehr hoch*) über den Wertebereich hinausgehen, modelliert. Sie werden aufsteigend nach ihren Gipfelpunkten auf dem Einheitsintervall angeordnet. *Klein* subsumiert *sehr klein* und *hoch* subsumiert *sehr hoch*. In ähnlicher Weise erfolgt die Definition der restlichen linguistischen Variablentypen. Die Eckwerte werden als trapezoidale Fuzzy-Intervalle beschrieben; die Werte, die nicht am Rand des Definitionsbereichs [0;1] liegen, werden als triangulär Fuzzy-Zahlen modelliert.

Die Partionierung der restlichen linguistischen Variablen befindet sich im Anhang A.

Nach der Definition der linguistischen Variablentypen können Entscheidungsmodelle in Form von Regelbasen aufgestellt werden.

---

202 Die Graphik stellt einen Ausschnitt der DREM-Benutzerschnittstelle dar. Gezeigt ist die Maske zur Definition einer linguistischen Variablen.

6.1.1 Aufstellen der Regelbasen

Die Regeln zur Ermittlung der Vertriebspriorität sind innerhalb einer Fuzzy-Methode in Form einer Regelbasis zusammengefaßt. Bild 6.4 visualisiert die Entscheidungsfindung erfolgt auf der Ebene des Vertriebs.

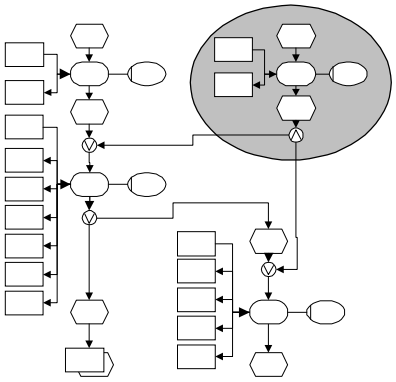


Bild 6.4: Einordnung der Ermittlung der Vertriebspriorität

Die Entscheidungsstruktur weist zwei Hierarchiestufen auf. Zuerst werden die Regeln für Stufe 0 definiert.

Auftragsbewertung		Auftragsvolumen				
	UND	sehr klein	klein	mittel	hoch	sehr hoch
Kunden-einschät-zung	normal	normal	normal	normal	wichtig	wichtig
	wichtig	normal	normal	wichtig	wichtig	sehr wichtig
	sehr wichtig	normal	wichtig	wichtig	sehr wichtig	sehr wichtig

Tabelle 20: Regelbasis zur Ermittlung der Auftragsbewertung

Vertriebspriorität		Auftragsbewertung		
	UND	normal	wichtig	sehr wichtig
Risiko-einschät-zung	klein	normal	hoch	hoch
	normal	klein	normal	hoch
	hoch	klein	klein	normal

Tabelle 21: Regelbasis zur Ermittlung der Vertriebspriorität

6.2 Ermittlung der Fertigungspriorität

Im Fertigungsbereich (Bild 6.5) wird eine vorläufige Einplanung des Auftrags vorgenommen, wobei die vom Vertrieb ermittelte Priorität mitberücksichtigt wird.

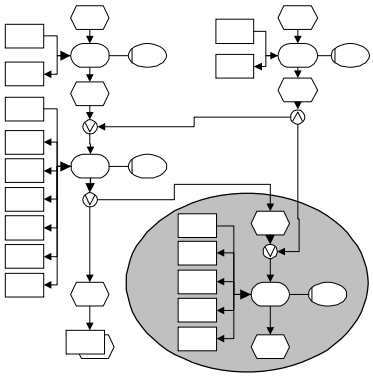


Bild 6.5: Einordnung der Ermittlung der Fertigungspriorität

Auf der Stufe 0 werden die Entscheidungsparameter Auftragsvolumen (

Bild 6.3) und Schlupfzeit (Anh.A.1) zur Ermittlung der Dringlichkeit (Anh.A.6) benötigt. Die Dringlichkeit fließt in Stufe 1 zusammen mit der Vertriebspriorität in die Ermittlung der Fertigungspriorität (Anh.A.7) ein.

6.2.1 Regelbasis zur Ermittlung der Dringlichkeit eines Auftrages

Die Regeln zur Ermittlung der Vertriebspriorität sind innerhalb einer Fuzzy-Methode in Form einer Regelbasis zusammengefaßt. Die Entscheidungsstruktur weist zwei Hierarchiestufen auf. Zuerst werden, wie in Tabelle 22 aufgeführt, die Regeln für die Stufe 0 definiert, damit werden die Auswirkungen der Eingangsgrößen Auftragsvolumen und Schlupfzeit auf die Auftragsdringlichkeit festgelegt.

Dringlichkeit		Auftragsvolumen				
	UND	sehr klein	klein	mittel	hoch	sehr hoch
Schlupf-zeit	klein	mittel	mittel	hoch	hoch	hoch
	mittel	klein	mittel	mittel	hoch	hoch
	hoch	klein	klein	klein	mittel	hoch

Tabelle 22: Regelbasis zur Ermittlung der Dringlichkeit

6.2.2 Regelbasis zur Ermittlung der Fertigungspriorität

In der zweiten Regelbasis (Tabelle 23) werden für die Stufe 1 die Regeln zur Ermittlung der Fertigungspriorität definiert.

Fertigungs-priorität		Dringlichkeit		
	UND	klein	mittel	hoch
Vertriebs-priorität	klein	klein	klein	normal
	normal	normal	normal	hoch
	hoch	normal	hoch	hoch

Tabelle 23: Regelbasis zur Ermittlung der Fertigungspriorität

6.3 Skripte zu den Regelbasen

Die Definition von Fuzzy-Methoden innerhalb des DREM-Systems erfolgt durch die Erstellung von Regelbasen innerhalb der DREM-Programmiersprache. Die in den obigen Tabellen dargestellten Regelbasen lassen sich in Skripte, die im folgenden jeweils ausschnittsweise die Definition exemplarischer Regeln jeder Hierarchiestufe der zugehörigen Regelbasis enthalten, transformieren.

6.3.1 Ermittlung der Vertriebspriorität

```
DECLARE V_PRIORITAET_ERMITTELN: RULEBASE;

BEGIN
RULEBASE V_PRIORITAET_ERMITTELN
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "sehr klein"
       conclusions           auftragsbewertung is "normal"
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "klein"
                             kundeneinschaetzung is "normal"
       conclusions           auftragsbewertung is "normal"
...
RULE
       LEVEL 1
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsbewertung is "normal"
                             risikoeinschaetzung is "klein"
       conclusions           vertriebsprioritaet is "normal"
RULE
       LEVEL 1
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsbewertung is "normal"
                             risikoeinschaetzung is "normal"
       conclusions           vertriebsprioritaet is "klein"
       ...
endbase;
....
END
```

6.3.2 Ermittlung der Fertigungspriorität

```
DECLARE F_PRIORITAET_ERMITTELN: RULEBASE;

BEGIN
...
RULEBASE F_PRIORITAET_ERMITTELN
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "sehr klein"
                             Schlupfzeit is "klein"
       conclusions           Dringlichkeit is "mittel"
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "sehr klein"
                             Schlupfzeit is "mittel"
       conclusions           Dringlichkeit is "klein"
...
       LEVEL 1
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         dringlichkeit is "hoch"
                             vertriebsprioritaet is "klein"
       conclusions           fertigungsprioritaet is "hoch"
RULE
```

```
        LEVEL 1
        RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
        combine fuzzy_min;
        inference fuzzy_algeb_prod;
        preconditions         dringlichkeit is "hoch"
                              vertriebsprioritaet is "normal"
        conclusions           fertigungsprioritaet is "normal"
        ....
endbase;
...
END
```

6.4 Struktur der Datenbank

Die zum Anwendungsbeispiel gehörige DREM-Daten- und Methodenbank ist unterteilt in Informationsobjekte und Planungsobjekte. Informationsobjekte stellen Entitäten im betrieblichen Umfeld dar während Planungsobjekte betrieblichen Aufgaben entsprechen.

6.4.1 Informationsobjekte

Die folgenden Informationsobjekte wurden zur Definition des Anwendungsbeispiels benötigt.

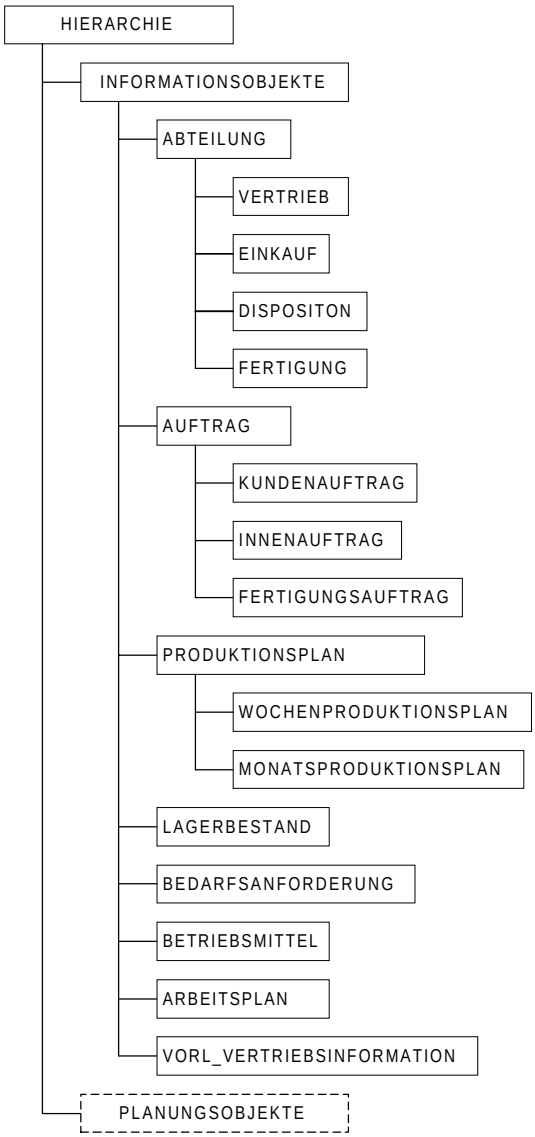


Bild 6.6: Informationsobjekttypen zur Auftragsabwicklung mit Unschärfe

In der Hierarchieebene der Informationsobjekte (Bild 6.6) finden sich die organisatorischen Einheiten und die Informationsobjekte aus der eEPK[203] zur unscharfen Auftragsabwicklung wieder. Es ist hierbei herauszustellen, daß eine explizite Fuzzifizierung durch die Verwendung von Fakten als Ausprägungen von linguistischen Variablen nicht notwendig ist, denn der DREM-Datentyp Fakt kann sowohl scharfe als auch unscharfe Werte annehmen. Wird durch Erhöhung der Informationsreife aus einem „vorläufigen Kundenauftrag“ ein „Kundenauftrag“, so sind nur die Ausprägungen der Daten in der linguistischen Variablen „Auftragsvolumen“ zu ändern. Es ist hierbei lediglich die unscharfe Mengen- oder Zeitangabe in Form eines FuzzySets in eine Realzahl umzuwandeln. Durch die Modellierung mit dem Fakt-Datentyp lassen sich auch andere Kennzahlen und Attribute von bisher konventionell scharfen Informationsobjekten „schleichend“ fuzzifizieren.

6.4.2 Planungsobjekte

In der Hierarchieebene der DREM-Planungsobjekte (Bild 6.7) finden sich die organisatorischen Einheiten und die Informationsobjekte aus der eEPK zur unscharfen Auftragsabwicklung wieder.

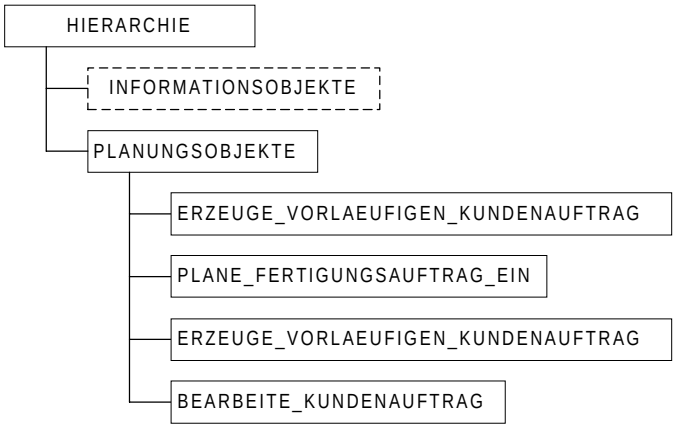


Bild 6.7: Planungsobjekttypen zur Auftragsabwicklung mit Unschärfe

[203] Vgl. Abschnitt 4.

7. Ausblick und Tendenzen

Im Rahmen dieser Arbeit wurden Verfahren aufgezeigt, die bei Einbettung in ein PPS-System in einem hohen Maß zur Komplexitätsreduktion beitragen können. Da die Verarbeitung von unscharfen Daten eine Generalisierung der Verarbeitung von scharfen Daten darstellt, kann in den meisten Fällen eine Fuzzifizierung von betriebswirtschaftlichen Informationssystemen erfolgen. Ein Zugewinn an Transparenz der Systeme wäre die Folge.

Die Kombination der mathematischen Grundlage mit der umgangssprachlichen Modellierung von Unschärfe durch die Fuzzy-Set-Theorie führt zu einer Verbesserung der Mensch-Maschine-Kommunikation und damit zu einer gesteigerten Akzeptanz des auftragsbegleitenden Systems.

Es ist zu beachten, daß auch bei Einsatz von fuzzifizierten Systemen, die Qualität der Ergebnisse vor allem von der Qualität der Modellierung der Zusammenhänge abhängt. Dies führt in logischer Konsequenz zu einem Bedarf an Methoden, unscharfe Zusammenhänge systematisch darzustellen.

Die Ausnutzung von Parallelisierungspotentialen führt zu einer Abkehr von traditionell sequentiellen Prinzipien der Ablauforganisation. Es darf nicht außer acht gelassen werden, daß Aufgabenträger für diese Optimierungspotentiale erst sensibilisiert werden müssen, denn der durch eine Parallelisierung erhöhte Koordinationsaufwand führt langfristig durch den erhöhten Bedarf an Teamarbeit auch zu Veränderungen in der Aufbauorganisation von Unternehmen.

Es ist zur Klärung des Zielkonflikts zwischen der Planungssicherheit und der Transparenz zu diskutieren, ab welchem Grad an Sicherheit Informationen in ein System eingehen dürfen. Denn die potentiell erhöhte Planungssicherheit, die als Nutzen der frühzeitigen Verwendung von unsicheren Informationen entsteht, kann durch eine Überflutung des Systems mit vorläufigen Informationen, die Transparenz gefährden.

Die in der vorliegenden Arbeit dargestellten Konzepte und Ergebnisse zur Realisation und Integration der softwaregestützten Entscheidungskomponente in eine Workflow-Entwicklungsumgebung sind keineswegs als isolierter Prozeß zu sehen. Der praktische Nutzen ergibt sich sowohl auf technischer Ebene aus der Operationalisierung unscharfer Daten als auch auf betriebswirtschaftlicher Ebene aus der Synergie der Fuzzy-Logik mit den, dem Workflowmanagement vorgelagerten, Modellierungsmethoden des Business Process Reengineering.

Literaturverzeichnis

Adam, D.: Planung und Entscheidung, 3.Auflage, Wiesbaden 1993.

Aliev, R., Bonfig, K.W., Aliew, F.: Messen, Steuern und Regeln mit Fuzzy-Logik, München, 1994.

Altrock, C., Krause.B.: Multi-criteria decision making in German automotive industry using fuzzy logic. Fuzzy Sets And Systems 63 (1994) S. 375 - 380.

Baekeland, R., Kerre, E.: Piecewise Linear Fuzzy Quantities: A Way to Implement Fuzzy Information into Expert Systems and Fuzzy Databases. In: 2nd International Conference on Information Processing and Management of Uncertainty in Knowledge Based Systems IPMU'88. Hrsg: Bouchon, B., Saitta, L., Yager, R. Berlin u. a. 1988, S. 119 - 126.

Becker, J.: CIM-Integrationsmodell - Die EDV-gestützte Verbindung betrieblicher Bereiche. Berlin u. a. 1991.

Becker, J., Rosemann, M.: Logistik und CIM. Die effiziente Material- und Informationsflußgestaltung im Industrieunternehmen, Berlin u. a. 1993.

Becker, J., Rehfeldt, M., Turowski, K.: Koordination verteilter Objekte in der PPS. CIM-Management, 11 (1995) 3 S. 33 - 37.

Becker, J., Rehfeldt, M., Turowski, K. (Auftragsabwicklung): Auftragsabwicklung mit unscharfen Daten in der Industrie. In: Hrsg.: J. Biethahn, A. Hönerloh, J. Kuhl, V. Nissen. Göttingen 1996. S. 51 - 61.

Becker, J., Rehfeldt, M., Turowski, K. (Planungsobjekte): DV-Unterstützung einer dezentralen Produktionsplanung und -steuerung durch Planungsobjekte. In: Tagungsband Werkstattmanagemet Organisation und Informationstechnik im Spannungsfeld von zentralen und dezentralen Strukturen. ETH Zürich, 1996. S.5.1 - 5.16.

Bitterlich, N., Fröbel, D., Lull, B.: Fuzzy-basierte Entscheidung zur Auftrags-reihenfolge. In: Tagungsband zum 2. Göttinger Symposium Softcomputing. Hrsg.: J. Biethahn, A. Hönerloh, J. Kuhl, V. Nissen. Göttingen 1996. S. 62 - 71.

Blieberger, J., Schildt, G.-H., Schmid, U., Stöckler, S.: Informatik. 2. Auflage, Wien u. a. 1992.

Bothe, H.-H.: Fuzzy-Logic, 2.Auflage, Berlin u. a. 1995.

Böhme, G.: Fuzzy-Logik - Einführung in die algebraischen und logischen Grundlagen, Berlin u. a. 1993.

Demandt, B.: Fuzzy-Theorie oder die Faszination des Vagen . Braunschweig, Wiesbaden 1993.

Driankov, D., Hellendoorn, H.: Chaining of fuzzy IF-THEN rules in Mamdani-Controllers. In: Proceecings of the FUZZ-IEEE / IFES'95 Conference. Hrsg: IEEE. San Francisco 1995, S. 103 - 108.

Dubois, D., Prade, H.: Fuzzy sets and probability: Misunderstandings, bridges and gaps. In: Second IEEE International Conference on Fuzzy Systems. Hrsg: IEEE. San Francisco 1993, S. 1059 - 1068.

Dubois,D., Prade,H.(Fuzzy Numbers): Fuzzy Numbers: an Overview. In: Readings in Fuzzy Sets for Intelligent Systems. Hrsg.: D.Dubois, H. Prade, R.R. Yager. San Mateo 1993, S. 112 - 148.

Eversheim, W., Roggatz, A.,Zimmermann, H.-J., Derichs, T.: Kurze Produkt-entwicklungszeiten durch Nutzung unsicherer Informationen - Modelle zur datentechnischen Umsetzung. it+ti 37 (1995) 5 S. 47 - 53.

Eversheim, W., Laufenberg, L.: Markterfolg ist planbar - Integrierte Gestaltung von Simultaneous Engineering-Projekten. VDI-Z 137 (1995) 1/2 S. 32 - 36.

Hagemeyer, J., Galler, J., Scheer, A.-W.: Koordination verteilter Workflow-modellierung mit ContAct. In: Workflowmanagement - State-of-the-Art aus Sicht von Theorie und Praxis - Proceedings zum Workshop vom 10.April 1996. Arbeitsbericht Nr. 47 des Instituts für Wirtschaftsinformatik der Westfälischen Wilhelms-Universität Münster. Hrsg.: J. Becker, H. L. Grob, U. Müller-Funk, G. Vossen. Münster 1996. S. 22 - 27.

Helfrich, C.: PPS-Praxis - Fertigungssteuerung und Logistik im CIM-Verbund. 2.Auflage, München 1993.

Hopf, W.: Fuzzy-Logik zur Steuerung auftragsorientierter Werkstattfertigung, o. O. u J. (Dissertation Regensburg 1995).

Hönerloh, A., Kalonda, N., Kuhl, J.: Unscharfe Bestandsregelung und qualitative Beurteilung von Lagerhaltungspolitiken. In: Tagungsband zum 2. Göttinger Symposium Softcomputing. Hrsg.: J. Biethahn, A. Hönerloh, J. Kuhl, V. Nissen. Göttingen 1996. S. 115 - 135.

Jaanineh, G., Maijohann, M.: Fuzzy-Logik und Fuzzy-Control. Würzburg 1996.

Kemper, H.G.: Akzeptanz. In: Lexikon der Wirtschaftsinformatik. Haupthrsg.: P. Mertens. Hrsg.:H.R. Hansen, Krallmann, H., Scheer, A.-W., Seibt, D., Stahlknecht, P., Strunz, H., Thome, R., Wedekind, H. Berlin u. a. 1987. S. 14 - 15.

Kosko, B.: Fuzzy Thinking - The New Science of Fuzzy Logic, Englewood Cliffs 1993.

Kruse, R., Gebhardt, J., Klawonn, F.: Fuzzy-Systeme 2.Auflage, Stuttgart 1995.

Kurbel,K.: Entwicklung und Einsatz von Expertensystemen- Eine anwendungsorientierte Einführung in wissensbasierte Systeme, 2.Auflage, Berlin u. a. 1992.

Kurbel, K.: Produktionsplanung und -steuerung. Methodische Grundlagen von PPS-Systemen und Erweiterungen. In: Handbuch der Informatik. Hrsg.: A. Endres, H. Krallmann, P. Schnupp. Bd. 13.2, München u. a. 1993.

Leymann, F.: The Workflow-Based/Application Paradigm. In: Workflowmanagement - State-of-the-Art aus Sicht von Theorie und Praxis - Proceedings zum Workshop vom 10.April 1996. Arbeitsbericht Nr. 47 des Instituts für Wirtschaftsinformatik der Westfälischen Wilhelms-Universität Münster. Hrsg.: J. Becker, H. L. Grob, U. Müller-Funk, G. Vossen. Münster 1996. S. 5 - 11.

Luther, W., Ohsmann, M.: Mathematische Grundlagen der Computergraphik. 2.Auflage, Braunschweig, Wiesbaden 1989.

Mayer,A., Mechler,B., Schlindwein, A., Wolke, R.: Fuzzy-Logic: Einführung und Leitfaden zur praktischen Anwendung, Bonn u. a.1993.

Mag, W.: Entscheidung und Information, München 1977.

Mechler, B.: Intelligente Informationssysteme, Bonn u. a. 1994.

Meyers, S.: Effektiv C++ programmieren, 2.Auflage, Bonn u. a. 1995.

Meyer, B.: Objektorientierte Softwareentwicklung , München u. a. 1990.

Milberg, J., Amann, W., Zetlmayer, H.: Wissensbasierte Simulation und Regelung von Produktionssystemen. CIM Management 7 (1991) 6, S. 4 - 9.

Nelles, O.: Fuzzy Rules Extraction by a Genetic Algorithm. In: EUFIT ‘96. Hsrg. H.-J. Zimmermann. Bd. 3. Aachen 1996, S. 489 - 493.

Nietsch, T., Rautenstrauch, C., Rehfeldt, M., Rosemann, M., Turowski, K.: Ansätze für die Verbesserung von PPS-Systemen durch Fuzzy-Logik. Arbeitsbericht Nr. 23 des Instituts für Wirtschaftsinformatik der Westfälischen Wilhelms-Universität Münster. Hrsg.: J. Becker, H. L. Grob, K. Kurbel, U. Müller-Funk, R. Unland, G. Vossen. Münster 1993. Kurzversion: Nietsch, T., Rautenstrauch, C., Rehfeldt, M., Rosemann, M., Turowski, K.: Verbesserung von PPS-Systemen durch Fuzzy-Logik. CIM Management 10 (1994) 6, S. 22 - 26.

N. N.: DREM. Referenz-Handbuch zur Skriptsprache. (nicht genannt) Münster 1996.

Opitz, O.: Mathematik für Ökonomen, 3.Auflage, München, Wien 1991.

Rautenstrauch, C.: Integration Engineering - Konzeption, Entwicklung und Einsatz integrierter Softwaresysteme, Bonn u. a. 1993.

Rehfeldt, M., Turowski, K. (Technology): Impact on Integrated Information Systems through Fuzzy Technology. In: EUFIT ‘94. Hrsg.: H.-J. Zimmermann. Bd. 3. Aachen 1994, S. 1637 - 1645.

Rehfeldt, M., Turowski, K. (Integration): Integration von Fuzzy-Ansätzen in PPS-Systeme. In: Fuzzy-Systeme ‘94. Hrsg.: R. Plam. München 1994, S. 1 - 10.

Rehfeldt, M., Turowski, K.: Fuzzy Objects in Production Planning and Control. In: EUFIT ‘96. Hsrg. H.-J. Zimmermann. Bd. 3. Aachen 1996, S. 1983 - 1989.

Rehfeldt, M., Turowski, K. (Coordination): Anticipating Coordination in Distributed Information Systems through Fuzzy Information. In: EUFIT ‘95. Hsrg. H.-J. Zimmermann. Bd. 3. Aachen 1995, S. 1774 - 1779.

Rehfeldt, M., Turowski, K. (Workflow): A Fuzzy Distributed Object-Oriented Database System as a Basis for a Workflow Management System. In: IFSA ‘95. Bd. 2. São Paulo 1995, S. 365 - 368.

Rembold, U., Nnaji, B., Storr, A.: CIM: Computeranwendung in der Produktion, Bonn u. a. 1994.

Rommelfanger, H. J.: Fuzzy Decision Support Systeme - Entscheiden bei Unschärfe, 2.Auflage., Berlin u. a. 1994.

Rommelfanger, H. J.: Regelbasierte Entscheidungsunterstützung mit Fuzzy-Logik. In: Tagungsband zum 2. Göttinger Symposium Softcomputing. Hrsg.: J. Biethahn, A. Hönerloh, J. Kuhl, V. Nissen. Göttingen 1996. S. 16 - 28.

Scheer, A.-W.: Wirtschaftsinformatik - Informationsmodelle für industrielle Geschäftsprozesse, 5.Auflage, Berlin 1994.

Schmidt, G., Jacob, E., Lahl, B., Meyer, J.: Fuzzy Logic für die reaktive Fertigungssteuerung. ZwF 88 (1993) 2, S. 81 - 83.

Sedgewick, R.: Algorithmen in C++ , Bonn u. a. 1992.

Shmilovici, A., Maimon, O.: Best Fuzzy Rule Selection with Orthogonal Matching Pursuit. In: In: EUFIT'96. Hrsg. H.J. Zimmermann, Bd. 1. Aachen, S. 592 - 96.

Stroustrup, B.: Die C++ Programmiersprache, 2.Auflage, Bonn u. a. 1992.

Suh, C., Suh, E.: Using human factor guidelines for developing expert systems. Expert Systems 10 (1993) 3, S. 151 - 156.

Tilli, T.: Fuzzy-Logik: Grundlagen, Anwendungen, Hard- und Software. 2. Auflage, München 1992.

Turksen, I. B.: Fuzzy expert systems for IE/OR/MS. Fuzzy Sets and Systems 51 (1992), S. 1 - 27.

Turksen, I. B., Ulguray, D., Wang, Q.: Hierarchical scheduling based on approximate reasoning - A comparison with ISIS. Fuzzy Sets and Systems 46 (1992), S. 349 - 371.

Turowski, K.: Der Beitrag der Geschäftsprozeßmodellierung bei der Entwicklung verteilter Anwendungssoftware in der Industrie. Informationssystem Architekturen 1 (1994) 2, S. 113 - 115.

Turowski, K.: Prozeßorientierung in der Produktionsplanung und -steuerung. In: Vossen, G. Becker, J. (Hrsg.): Geschäftsprozeßmodellierung und Workflowmanagement. Bonn, 1995, S. 209 - 223.

Vossen, G.: Datenmodelle, Datenbanksprachen und Datenbank-Management-Systeme, 2.Auflage , Bonn u. a. 1994.

Weber, R., Zimmermann,H.-J.: Automatische Akquisition von unscharfem Expertenwissen. KI 2 (1991), S. 20 - 26.

Womack, J., Jones, D., Roos, D.: Die zweite Revolution in der Autoindustrie, Frankfurt/Main, New York, 6.Auflage, 1992.

Zadeh, L.A.: Fuzzy Sets. Information and Control, 8 (1965), S. 338 - 353.

Zimmermann, H.-J.: Fuzzy Set Theory and its applications. 2.Auflage, Boston u. a. 1991.

Zimmermann, H.-J.: Industrielle Einsatzgebiete von Fuzzy Technologien. In Neuro+Fuzzy: Technologien / Anwendungen. Hrsg: H.J. Zimmermann, Düsseldorf, 1995.

Zimmermann, H.-J.: Fuzzy Sets, Decision Making and Expert Systems, Boston u. a. 1987.

Zimmermann, H.-J., Zysno, P.: Latent Connectives in Human Decision Making. Fuzzy Sets and Systems 4 (1980) S. 37 - 51.

Zlatareva, N. P.: A Framework for Verification , Validation and Refinement of Knowledge Bases: The VVR System. International Journal of Intelligent Systems 9 (1994), S. 703 - 737.

Anhang

A Linguistische Variablen

Kundeneinschätzung [0;1]			
Term	von	bis	Typ
normal	0,00	0,50	Trapez (links abgeschnitten)
wichtig	0,25	0,75	Dreieckszahl
sehr wichtig	0,50	1,00	Trapez (rechts abgeschnitten)

Anh.A.1: Linguistische Variable "Kundeneinschätzung"

Auftragsbewertung [0;1]			
Term	von	bis	Typ
normal	0,00	0,50	Trapez (links abgeschnitten)
wichtig	0,25	0,75	Dreieckszahl
sehr wichtig	0,50	1,00	Trapez (rechts abgeschnitten)

Anh.A.2: Linguistische Variable „Auftragsbewertung“

Risikoeinschätzung [0;1]			
Term	von	bis	Typ
klein	0,00	0,50	Trapez (links abgeschnitten)
normal	0,25	0,75	Dreieckszahl
hoch	0,50	1,00	Trapez (rechts abgeschnitten)

Anh.A.3: Linguistische Variable „Risikoeinschätzung“

Vertriebspriorität [0;1]			
Term	von	bis	Typ
klein	0,00	0,50	Trapez (links abgeschnitten)
normal	0,25	0,75	Dreieckszahl
hoch	0,50	1,00	Trapez (rechts abgeschnitten)

Anh.A.4: Linguistische Variable „Vertriebspriorität“

Schlupfzeit [0;1]			
Term	von	bis	Typ
klein	0,00	0,50	Trapez (links abgeschnitten)
mittel	0,25	0,75	Dreieckszahl
hoch	0,50	1,00	Trapez (rechts abgeschnitten)

Anh.A.5: Linguistische Variable „Schlupfzeit“

Dringlichkeit [0;1]			
Term	von	bis	Typ
klein	0,00	0,50	Trapez (links abgeschnitten)
mittel	0,25	0,75	Dreieckszahl
hoch	0,50	1,00	Trapez (rechts abgeschnitten)

Anh.A.6: Linguistische Variable „Dringlichkeit“

Fertigungspriorität [0;1]			
Term	von	bis	Typ
klein	0,00	0,50	Trapez (links abgeschnitten)
mittel	0,25	0,75	Dreieckszahl
hoch	0,50	1,00	Trapez (rechts abgeschnitten)

Anh.A.7: Linguistische Variable **„**Fertigungspriorität**“**

B Regelbasen

- Ermittlung der Vertriebspriorität

```
DECLARE V_PRIORITAET_ERMITTELN: RULEBASE;

BEGIN
...
RULEBASE V_PRIORITAET_ERMITTELN
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "sehr klein"
       conclusions           auftragsbewertung is "normal"
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "klein"
                             kundeneinschaetzung is "normal"
       conclusions           auftragsbewertung is "normal"
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "klein"
                             kundeneinschaetzung is "wichtig"
       conclusions           auftragsbewertung is "normal"
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "klein"
                             kundeneinschaetzung is "sehr wichtig"
       conclusions           auftragsbewertung is "wichtig"
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "mittel"
                             kundeneinschaetzung is "normal"
       conclusions           auftragsbewertung is "normal"
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "mittel"
                             kundeneinschaetzung is "wichtig"
       conclusions           auftragsbewertung is "wichtig"
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "mittel"
                             kundeneinschaetzung is "sehr wichtig"
       conclusions           auftragsbewertung is "wichtig"
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "hoch"
                             kundeneinschaetzung is "normal"
       conclusions           auftragsbewertung is "wichtig"
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "hoch"
                             kundeneinschaetzung is "wichtig"
       conclusions           auftragsbewertung is "wichtig"
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "hoch"
                             kundeneinschaetzung is "sehr wichtig"
       conclusions           auftragsbewertung is "sehr wichtig"
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
```

```
        preconditions         auftragsvolumen is "sehr hoch"
                              kundeneinschaetzung is "normal"
        conclusions           auftragsbewertung is "wichtig"
RULE
        LEVEL 0
        RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
        combine fuzzy_min;
        inference fuzzy_algeb_prod;
        preconditions         auftragsvolumen is "sehr hoch"
                              kundeneinschaetzung is "wichtig"
        conclusions           auftragsbewertung is "sehr wichtig"
RULE
        LEVEL 0
        RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
        combine fuzzy_min;
        inference fuzzy_algeb_prod;
        preconditions         auftragsvolumen is "sehr hoch"
                              kundeneinschaetzung is "sehr wichtig"
        conclusions           auftragsbewertung is "sehr wichtig"
RULE
        LEVEL 1
        RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
        combine fuzzy_min;
        inference fuzzy_algeb_prod;
        preconditions         auftragsbewertung is "normal"
                              risikoeinschaetzung is "klein"
        conclusions           vertriebsprioritaet is "normal"
RULE
        LEVEL 1
        RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
        combine fuzzy_min;
        inference fuzzy_algeb_prod;
        preconditions         auftragsbewertung is "normal"
                              risikoeinschaetzung is "normal"
        conclusions           vertriebsprioritaet is "klein"
RULE
        LEVEL 1
        RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
        combine fuzzy_min;
        inference fuzzy_algeb_prod;
        preconditions         auftragsbewertung is "normal"
                              risikoeinschaetzung is "hoch"
        conclusions           vertriebsprioritaet is "klein"
RULE
        LEVEL 1
        RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
        combine fuzzy_min;
        inference fuzzy_algeb_prod;
        preconditions         auftragsbewertung is "wichtig"
                              risikoeinschaetzung is "klein"
        conclusions           vertriebsprioritaet is "hoch"
RULE
        LEVEL 1
        RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
        combine fuzzy_min;
        inference fuzzy_algeb_prod;
        preconditions         auftragsbewertung is "wichtig"
                              risikoeinschaetzung is "normal"
        conclusions           vertriebsprioritaet is "normal"
RULE
        LEVEL 1
        RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
        combine fuzzy_min;
        inference fuzzy_algeb_prod;
        preconditions         auftragsbewertung is "wichtig"
                              risikoeinschaetzung is "hoch"
        conclusions           vertriebsprioritaet is "klein"
RULE
        LEVEL 1
        RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
        combine fuzzy_min;
        inference fuzzy_algeb_prod;
        preconditions         auftragsbewertung is "sehr wichtig"
                              risikoeinschaetzung is "klein"
        conclusions           vertriebsprioritaet is "hoch"
RULE
        LEVEL 1
        RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
        combine fuzzy_min;
        inference fuzzy_algeb_prod;
        preconditions         auftragsbewertung is "sehr wichtig"
                              risikoeinschaetzung is "normal"
        conclusions           vertriebsprioritaet is "hoch"
RULE
        LEVEL 1
        RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
        combine fuzzy_min;
        inference fuzzy_algeb_prod;
        preconditions         auftragsbewertung is "sehr wichtig"
                              risikoeinschaetzung is "hoch"
        conclusions           vertriebsprioritaet is "normal"
endbase;
....
END
```

- Ermittlung der Fertigungspriorität

```
DECLARE F_PRIORITAET_ERMITTELN: RULEBASE;

BEGIN
...
RULEBASE F_PRIORITAET_ERMITTELN
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "sehr klein"
                             Schlupfzeit is "klein"
       conclusions           Dringlichkeit is "mittel"
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "sehr klein"
                             Schlupfzeit is "mittel"
       conclusions           Dringlichkeit is "klein"
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "sehr klein"
                             Schlupfzeit is "hoch"
       conclusions           Dringlichkeit is "klein"
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "klein"
                             Schlupfzeit is "klein"
       conclusions           Dringlichkeit is "mittel"
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "klein"
                             Schlupfzeit is "mittel"
       conclusions           Dringlichkeit is "mittel"
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "klein"
                             Schlupfzeit is "hoch"
       conclusions           Dringlichkeit is "klein"
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "mittel"
                             Schlupfzeit is "klein"
       conclusions           Dringlichkeit is "hoch"
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "mittel"
                             Schlupfzeit is "mittel"
       conclusions           Dringlichkeit is "mittel"
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "mittel"
                             Schlupfzeit is "hoch"
       conclusions           Dringlichkeit is "klein"
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "hoch"
                             Schlupfzeit is "klein"
       conclusions           Dringlichkeit is "hoch"
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
       preconditions         auftragsvolumen is "hoch"
                             Schlupfzeit is "mittel"
       conclusions           Dringlichkeit is "hoch"
RULE
       LEVEL 0
       RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
       combine fuzzy_min;
       inference fuzzy_algeb_prod;
```

```
        preconditions         auftragsvolumen is "hoch"
                              Schlupfzeit is "hoch"
        conclusions           Dringlichkeit is "hoch"
RULE
        LEVEL 0
        RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
        combine fuzzy_min;
        inference fuzzy_algeb_prod;
        preconditions         auftragsvolumen is "sehr hoch"
        conclusions           dringlichkeit is "hoch"
RULE
        LEVEL 1
        RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
        combine fuzzy_min;
        inference fuzzy_algeb_prod;
        preconditions         dringlichkeit is "klein"
                              vertriebsprioritaet is "klein"
        conclusions           risikoeinschaetzung is "klein"
RULE
        LEVEL 1
        RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
        combine fuzzy_min;
        inference fuzzy_algeb_prod;
        preconditions         dringlichkeit is "klein"
                              vertriebsprioritaet is "normal"
        conclusions           fertigungsprioritaet is "klein"
RULE
        LEVEL 1
        RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
        combine fuzzy_min;
        inference fuzzy_algeb_prod;
        preconditions         dringlichkeit is "mittel"
                              vertriebsprioritaet is "hoch"
        conclusions           fertigungsprioritaet is "hoch"
RULE
        LEVEL 1
        RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
        combine fuzzy_min;
        inference fuzzy_algeb_prod;
        preconditions         dringlichkeit is "mittel"
                              vertriebsprioritaet is "klein"
        conclusions           fertigungsprioritaet is "normal"
RULE
        LEVEL 1
        RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
        combine fuzzy_min;
        inference fuzzy_algeb_prod;
        preconditions         dringlichkeit is "mittel"
                              vertriebsprioritaet is "normal"
        conclusions           fertigungsprioritaet is "klein"
RULE
        LEVEL 1
        RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
        combine fuzzy_min;
        inference fuzzy_algeb_prod;
        preconditions         dringlichkeit is "hoch"
                              vertriebsprioritaet is "hoch"
        conclusions           fertigungsprioritaet is "hoch"
RULE
        LEVEL 1
        RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
        combine fuzzy_min;
        inference fuzzy_algeb_prod;
        preconditions         dringlichkeit is "hoch"
                              vertriebsprioritaet is "klein"
        conclusions           fertigungsprioritaet is "hoch"
RULE
        LEVEL 1
        RULE_CERTAINTY 1.0, fuzzy_algeb_prod;
        combine fuzzy_min;
        inference fuzzy_algeb_prod;
        preconditions         dringlichkeit is "hoch"
                              vertriebsprioritaet is "normal"
        conclusions           fertigungsprioritaet is "normal"
endbase;
....
END
```

C Struktureigenschaften[204] unscharfer Mengen

Der Durchschnitt einer unscharfen Menge wird auf die Minimum-Operation „$\cap$“, die Vereinigung „$\cup$“ wird auf den Maximum-Operation zurückgeführt. Das Komplement $\neg$ einer unscharfen Menge ist mit dem Einerkomplement $\mu_{CA}(x)=1-\mu_A(x)$ erklärt. Die Konstanten 1 bzw. 0 werden der Grundmenge **G** bzw. der leeren Menge $\varnothing$ zugeordnet. μ_A, μ_B, μ_C, μ_D bezeichnen im folgenden kurz die Zugehörigkeitsfunktionen von **A**, **B, C** und **D**.Es ergeben sich folgende Struktureigenschaften:

Ungültigkeit des Komplementgesetzes

$\mathbf{A}\cap\neg\mathbf{A} \neq \varnothing$
$\mathbf{A}\cup\neg\mathbf{A} \neq \mathbf{G}$

Beispiel(μ_A=0.25):

$\min(\mu_A, 1-\mu_A)$ = $\min(0{,}25;0{,}75)$ = $0{,}25(\neq 0.0)$ und
$\max(\mu_A, 1-\mu_A)$ = $\max(0{,}25;0{,}75)$ = $0{,}75\ (\neq 1.0)$!

Neutrale Elemente

$\min(1, \mu_A) = \mu_A \quad \Rightarrow \quad \mathbf{G}\cap\mathbf{A} = \mathbf{A}$
$\max(0, \mu_A) = \mu_A \quad \Rightarrow \quad \varnothing\cup\mathbf{A} = \mathbf{A}$

Kommutativität

$\min(\mu_A, \mu_B) = \min(\mu_B, \mu_A) \quad \Rightarrow \quad \mathbf{A}\cap\mathbf{B} = \mathbf{B}\cap\mathbf{A}$
$\max(\mu_A, \mu_B) = \max(\mu_B, \mu_A) \quad \Rightarrow \quad \mathbf{A}\cup\mathbf{B} = \mathbf{B}\cup\mathbf{A}$

Assoziativität

$\min(\min(\mu_A,\mu_B,\mu_C) = \min(\mu_A,\min(\mu_B,\mu_C)) \quad \Rightarrow \quad (\mathbf{A}\cap\mathbf{B})\cap\mathbf{C} = \mathbf{A}\cap(\mathbf{B}\cap\mathbf{C})$
$\max(\max(\mu_A,\mu_B,\mu_C) = \max(\mu_A,\max(\mu_B,\mu_C)) \quad \Rightarrow \quad (\mathbf{A}\cup\mathbf{B})\cup\mathbf{C} = \mathbf{A}\cup(\mathbf{B}\cup\mathbf{C})$

Monotonie

$\mu_A \leq \mu_C \wedge \mu_B \leq \mu_D$

---

[204] Vgl. Böhme (1993), S. 38ff.

$\min(\mu_A,\mu_B) \leq \min(\mu_C,\mu_D) \quad \Rightarrow \quad \mathbf{A}\subset\mathbf{C} \wedge \mathbf{B}\subset\mathbf{D} \Rightarrow \mathbf{A}\cap\mathbf{C} \subset \mathbf{B}\cap\mathbf{D}$

$\max(\mu_A,\mu_B) \leq \max(\mu_C,\mu_D) \quad \Rightarrow \quad \mathbf{A}\subset\mathbf{C} \wedge \mathbf{B}\subset\mathbf{D} \Rightarrow \mathbf{A}\cup\mathbf{C} \subset \mathbf{B}\cup\mathbf{D}$

Distributivität

$\min(\mu_A,\max(\mu_B,\mu_C)) = \max(\min(\mu_A,\mu_B),\min(\mu_A,\mu_C)) \Rightarrow \mathbf{A}\cap(\mathbf{B}\cup\mathbf{C}) = (\mathbf{A}\cap\mathbf{B})\cup(\mathbf{A}\cap\mathbf{C})$

$\max(\mu_A,\min(\mu_B,\mu_C)) = \min(\max(\mu_A,\mu_B),\max(\mu_A,\mu_C)) \Rightarrow \mathbf{A}\cup(\mathbf{B}\cap\mathbf{C}) = (\mathbf{A}\cup\mathbf{B})\cap(\mathbf{A}\cup\mathbf{C})$

Idempotenz

$\min(\mu_A,\mu_A) = \mu_A \quad \Rightarrow \quad \mathbf{A} \cap \mathbf{A} = \mathbf{A}$

$\max(\mu_A,\mu_A) = \mu_A \quad \Rightarrow \quad \mathbf{A} \cup \mathbf{A} = \mathbf{A}$

Absorption

$\min(\mu_A,\max(\mu_A,\mu_B)) = \mu_A \quad \Rightarrow \quad \mathbf{A}\cap(\mathbf{A} \cup \mathbf{B}) = \mathbf{A}$

$\max(\mu_A,\min(\mu_A,\mu_B)) = \mu_A \quad \Rightarrow \quad \mathbf{A} \cup(\mathbf{A} \cap \mathbf{B}) = \mathbf{A}$

De Morgan[205]

$1-\min(\mu_A,\mu_B) = \max(1-\mu_A,1-\mu_B) \quad \Rightarrow \quad \neg(\mathbf{A}\cap\mathbf{B})=\neg\mathbf{A}\cup\neg\mathbf{B}$

$1-\max(\mu_A,\mu_B) = \min(1-\mu_A,1-\mu_B) \quad \Rightarrow \quad \neg(\mathbf{A}\cup\mathbf{B})= \neg\mathbf{A}\cap\neg\mathbf{B}$

Doppeltes Komplement

$1-(1-\mu_A) = \mu_A \quad \Rightarrow \quad \neg\neg\, \mathbf{A} = \mathbf{A}$

Komplement von Grundmenge und leerer Menge

$1-1 = 0 \quad \Rightarrow \quad \neg\mathbf{G}=\varnothing$

$1-0 = 1 \quad \Rightarrow \quad \neg\varnothing=\mathbf{G}$

[205] Zum Beweis vgl. Bothe(1995), S. 40.

D Quelltexte

FCSET.H

```
#ifndef _FCSET_H
#define _FCSET_H

#include "fc.h"
#include "pppo.h"
#include "fcintrvl.h"


typedef enum InsertionRule{ irBefore , irOverWrite,irAfter} InsertionRule;

typedef enum iOpType{ ArithLRNumber, ArithLRInterval,ArithExtension} iOpType;

typedef enum ZeroAllowed{ZANO = 0 ,ZAYES} ZeroAllowed;

typedef double (*arithfunc) (double , double);

typedef CInterval& (*ivfunc) (const CInterval& , const CInterval&);

class CFuzzySet:public CEinfachObjekt
{
public:
          DECLARE_SERIAL( CFuzzySet );
protected:
          CObArray pPoint;
          CInterval theRange;
          BOOL m_FuzzyNumber;
public:
    // Konstruktoren
          CFuzzySet(void);
          CFuzzySet(CFuzzySet* theFuzzySet);
          CFuzzySet(double theValue,double spread = 0.5);               // Create crisp
Singleton
          CFuzzySet(double from, double over, double to); // Create Triangular
          CFuzzySet(const CFuzzySet& theFuzzySet);
          CFuzzySet(double from, double plateau1, double plateau2, double to); // Create
Trapezoidal
          CFuzzySet(BOOL empty);
          // Destruktor
          ~CFuzzySet(void);
          // Lesende Funktionen  sortiert nach Rueckgabetyp

          BOOL IstPositiv(void) const ;
          BOOL IstNegativ(void) const ;
          BOOL IsConvex(void) const ;
          BOOL IsNormalized(void) const ;
          BOOL IsValidFuzzyNumber(void) const;

          double Height(void) const ;
//        double GetStep() const ;
          // double GetCentroid() const ;
          //double DeFuzzifycation() const ;
          double GetRMX(void) const ; // rightmost x
          double GetLMX(void) const ;  // leftmost x
          double FuzzyOperation (double a, double b, FuzzyOp theOp,  double FuzzyPar = 0.0)
const ;
          double GetDoM(double xvalue) const;
          CInterval GetRange(void) const ;
          double GetMaxDom(void) const ;
          void GetMinMax(double &min, double &max) const;

          int GetNumPoints(void) const;
          int GetMaxIndex(void) const;

          // Defuzzifikation

          double CentroidDefuzzification(void) const;
          double HeightDefuzzification(void)  const;
          double LeftOfMaxDefuzzification(void) const;
          double RightOfMaxDefuzzification(void) const;
          double MeanOfMaxDefuzzification(void) const;

          //

          CFuzzySet* ConvexContour(void) const ;
          CFuzzySet* Operation (FuzzyOp theOp ,const CFuzzySet& b,double granularity =
0.01,double FuzzyPar = 0.0) const ;
          CFuzzySet* Operation (FuzzyOp theOp ,double theValue,double granularity =
0.01,double FuzzyPar = 0.0) const ;
          void FuzzyOperation (FuzzyOp theOp ,double theValue,double granularity =
0.01,double FuzzyPar = 0.0)  ;
          CFuzzySet* GibKehrwert (void) const ;
          CFuzzySet* Reduce(void) const;
          CFuzzySet* GetNormalized(void) const;

          CFuzzySetEntry* GetLeftMostPoint(void) const ;
          CFuzzySetEntry* GetRightMostPoint(void) const ;
          CFuzzySetEntry* GetPoint(int nPos) const;

          CString GibText (void) const ;

          // virtuell ueberladenes Zeug

    virtual EinfachTyp GibEinfachTyp(void) const;
          virtual CTyp* GibTyp(void) const;
          virtual CString Display(void);

          //
          BOOL IsLRNumber(double& left, double& middle, double &right) const;
          BOOL IsLRInterval(double& left, double& middle1, double& middle2, double &right)
const;

          iOpType GetOperationType(const CFuzzySet& theFSet) const ;

          void Normalize(void);
          void DeletePoint(int i);
          BOOL DeletePoint(const CFuzzySetEntry& theEntry);
          void DeleteAllPoints(void);
          long PunkteKopieren(const CFuzzySet& theFSet);
          void Sort(void);

          void SetFuzzyNumberOn(void);
          void SetFuzzyNumberOff(void);
          void AppendPoint(const CFuzzySetEntry& theEntry,InsertionRule irRule = irBefore);
          void AppendPoint(double xval, double dom, InsertionRule irRule = irBefore);
          int EntryPositionBinary(const CFuzzySetEntry& thePoint) const ;
```

```
          void SetRange(const CInterval& tR);
          void Auseinanderziehen(double faktor); // fuer bessere numerische Verarbeitung
          void Zusammendruecken(double faktor);



//        int operator == (CFuzzySet* theFSet);
          int operator == (const CFuzzySet& theFSet) const ;
//        CFuzzySet* operator - (double theDouble);
          CFuzzySet* OperatorTemplate (const CFuzzySet& theFSet,ivfunc theIFunc,arithfunc
theFunc,ZeroAllowed ZA) const ;
          CFuzzySet* operator - (const CFuzzySet& theFSet) const ;
          CFuzzySet* operator * (const CFuzzySet& theFSet) const;
          CFuzzySet* operator / (const CFuzzySet& theFSet) const;
          CFuzzySet* operator + (const CFuzzySet& theFSet) const;

          CFuzzySet* operator - (void);

          CFuzzySet* operator *= (double fs);
          CFuzzySet& operator = (const CFuzzySet& fs);

          #ifndef _INDREM_
          friend ostream& operator <<(ostream& os , CFuzzySet& me);
          friend ostream& operator <<(ostream& os , CFuzzySet* me);
          #endif
          CObArray* GetSchnittpunkte(CFuzzySet* a);

          // für den InOutManager:
          virtual void                          Read( CInOutConv* pConv, BYTE *bError );
          // in inoutrw.cpp
          virtual CMyStringList*       Write( CInOutConv* pConv );
                    // bs 27.05.


          // virtual CString Dump(const CString& theName) const ;
          #ifdef _DEBUG
          virtual void Dump(CDumpContext& dc) const;
          #endif
          // Fuer Dope neu zu programmieren
          // Für MFC-Archive
          virtual void Serialize(CArchive& ar);
};


#endif // _FCSET_H
```

FCSET.CPP

```
////////////////////////////////////////////////////////////////
//   File:                                          fcset.cpp
//   Path:                                          q:\aktuell\fcset.cpp
//   Type:                                          Implementation
////////////////////////////////////////////////////////////////
//   Description:                       Basic Functions for CFuzzySet-Objects
////////////////////////////////////////////////////////////////
//   Created:                                   Thu May 03,1996 22:08:39 UTC
//   Author:                                    Marc Schönefeld
////////////////////////////////////////////////////////////////
//   Modified:                                  Wed Jul 21,1996 16:10:36 UTC
//   Last maintained by:      Marc Schönefeld
////////////////////////////////////////////////////////////////

#include "stdafx.h"                     // Header : MFC , Vorcompiliert
#include "fcse.h"                       // Header : CFuzzySetEntry-Funktionen
#include "fcset.h"                      // Header : CFuzzySet-Funktionen
#include "fclr.h"                       // Header : LRFuzzySet-Funktionen
#include "fchelp.h"                     // Header : Help-Funktionen
#include "fcintrvl.h"                   // Header : CIntrval-Funktionen

#include <math.h>                       // Header : Mathe-Funktionen

IMPLEMENT_SERIAL( CFuzzySet, CUnserObjekt,0)

#define new DEBUG_NEW                   // Der Feind heisst : Memory-Leak

////////////////////////////////////////////////////////////////
// Funktion : GibEinfachTyp
// Klasse   : CFuzzySet
////////////////////////////////////////////////////////////////
// Ermittelt die zu einem CFuzzySet das entsprechende
// Einfachtyp-Kennzeichen
////////////////////////////////////////////////////////////////


EinfachTyp CFuzzySet::GibEinfachTyp(void) const
{
         if (m_FuzzyNumber)
                  return ET_FuzzyZahl;
         else
                  return ET_FuzzyMenge;
};

////////////////////////////////////////////////////////////////
// Funktion : GibEinfachTyp
// Klasse   : CFuzzySet
////////////////////////////////////////////////////////////////
// Gibt zu einem CFuzzySet einen entsprechenden CTyp als
// CEinfachTyp
////////////////////////////////////////////////////////////////

CTyp* CFuzzySet::GibTyp(void) const
{
         CEinfachTyp* result;
         if (m_FuzzyNumber)
                  result = new CEinfachTyp(ET_FuzzyZahl);
         else
                  result = new CEinfachTyp(ET_FuzzyMenge);
         ASSERT (result);
         return result;
}


////////////////////////////////////////////////////////////////
// Funktion : EntryPostionBinary
// Klasse   : CFuzzySet
////////////////////////////////////////////////////////////////
// Ermittelt die zu einem uebergebenen CFuzzySet die
// potentielle Position des uebergebenen FuzzySetEntry's ...
////////////////////////////////////////////////////////////////

int CFuzzySet::EntryPositionBinary(const CFuzzySetEntry& thePoint) const
{
         int size  = GetNumPoints();
         int i               = 0;
         int j               = size - 1;
         int k       = 0;
         double x    = thePoint.GetX();
         CFuzzySetEntry* MyPoint;
         if (x < GetPoint(0)->GetX())
         {
                  return 0;
         }
         else if ( x > GetPoint(j)->GetX())
         {
                  return size;
         };
         while (i <= j)
         {
                  k = (i + j ) / 2;
                  MyPoint = GetPoint(k);
                  if (x < MyPoint->GetX())
                           j = k - 1;
                  else
                           i = k + 1;
         };
         return i; //thanx to Wirth it became not only shorter, it also became right ...
};

////////////////////////////////////////////////////////////////////////////////
// Funktion : AppendPoint
// Klasse   : CFuzzySet
////////////////////////////////////////////////////////////////////////////////
// Fuegt ein Objekt vom Typ CFuzzySetEntry in ein CFuzzySet ein
// die Einträge sind aufsteigend nach ihren X-Werten sortiert
// Der Parameter irRule gibt an , ob bei Gleichheit der X-Werte der einzufügende
// Punkt vor oder nach dem bereits bestehenden Punkt eingefügt werden soll ...
// Ist irRule auf irOverWrite gesetzt, so wird der bereits bestehende Punkt
// ueberschrieben ....
////////////////////////////////////////////////////////////////////////////////

void CFuzzySet::AppendPoint(const CFuzzySetEntry& thePoint,InsertionRule irRule )
{
         double  xval                                    = 0.0;
         int     bEntryExists                  = 0 ;
         int     bEntryExistsBinary    = 0 ;
         CFuzzySetEntry* OldPoint;
         ASSERT(thePoint.IsValidDoM());
         if (!GetNumPoints())
         {
                  bEntryExists = 0;
```

```
        }
        else
        {
                //            bEntryExists = EntryPosition(thePoint);
                //                  bEntryExistsBinary = EntryPositionBinary(thePoint) ;
                bEntryExists = EntryPositionBinary(thePoint);

                if (bEntryExists < GetNumPoints())
                {
                        OldPoint = GetPoint(bEntryExists);
                        if (thePoint == *OldPoint)
                                bEntryExists = -1;
                };
        };

        if (bEntryExists != -1) // Punkt noch nicht vorhanden ....
        {
                pPoint.InsertAt(bEntryExists,new CFuzzySetEntry(thePoint));

                xval = thePoint.GetX();

                if ( xval > theRange.GetMaxX() || (!GetNumPoints()))
                        theRange.SetMaxX(xval);

                if ( xval < theRange.GetMinX() || (!GetNumPoints()))
                        theRange.SetMinX(xval);
        }

        else
        {

        };
};

////////////////////////////////////////////////////////////////////////////
// Funktion : AppendPoint
// Klasse   : CFuzzySet
////////////////////////////////////////////////////////////////////////////
// Fuegt ein Objekt vom Typ CFuzzySetEntry in ein CFuzzySet ein, wobei die
// Werte für das CFuzzySetEntry aus den übergebenen Werten xval und dom
// erstellt wird.
// Sonst wie oben
////////////////////////////////////////////////////////////////////////////


void CFuzzySet::AppendPoint(double xval, double dom, InsertionRule irRule)
{
        CFuzzySetEntry thePoint = CFuzzySetEntry(xval,dom);
        AppendPoint(thePoint,irRule);
};

////////////////////////////////////////////////////////////////////////////
// Funktion : IsValidFuzzyNumber()
// Klasse   : CFuzzySet
////////////////////////////////////////////////////////////////////////////
// Ermittelt ob ein CFuzzySet-Objekt eine gueltige Fuzzy-Zahl ist .
// Dies ist nur dann der Fall, falls der FuzzySet normalisiert und konvex ist .
////////////////////////////////////////////////////////////////////////////


BOOL CFuzzySet::IsValidFuzzyNumber(void) const
{
        BOOL b1 = IsNormalized();
        BOOL b2 = IsConvex();
        return ( b1 && b2);
};

////////////////////////////////////////////////////////////////////////////
// Funktion : PunkteKopieren
// Klasse   : CFuzzySet
////////////////////////////////////////////////////////////////////////////
// Kopiert alle Punkte aus dem uebergebenen CFuzzySet in das aktuelle Objekt
// Gibt als Rueckgabewert die Anzahl der kopierten Punkte
////////////////////////////////////////////////////////////////////////////

long CFuzzySet::PunkteKopieren(const CFuzzySet& theFSet)
{
        int iNumPoints = theFSet.GetNumPoints();
        for (int i = 0; i < iNumPoints; i++)
        {
                CFuzzySetEntry* theEntry = theFSet.GetPoint(i);
                ASSERT(theEntry);
                pPoint.SetAtGrow(i,new CFuzzySetEntry(theEntry));
        }
        return i-1;
};



//////////////////////////
/// KONSTRUKTOREN       //
//////////////////////////



CFuzzySet::CFuzzySet(double theValue,double spread)
{
//      const double spread = 0.5;
        theRange = CInterval(theValue-spread,theValue+spread);
        AppendPoint(CFuzzySetEntry(theValue-spread,0));
        AppendPoint(CFuzzySetEntry(theValue,1));
        AppendPoint(CFuzzySetEntry(theValue+spread,0));
        m_FuzzyNumber = IsValidFuzzyNumber();
};

CFuzzySet::CFuzzySet(void)
{
        theRange = CInterval(-0.1,0.1);
        AppendPoint(CFuzzySetEntry(-.1,0));
        AppendPoint(CFuzzySetEntry(0,1));
        AppendPoint(CFuzzySetEntry(+.1,0));
        m_FuzzyNumber = IsValidFuzzyNumber();
};

CFuzzySet::CFuzzySet(CFuzzySet* theFSet)
{
        theRange = theFSet->theRange;
        PunkteKopieren(*theFSet);
        m_FuzzyNumber = theFSet->m_FuzzyNumber;
};

CFuzzySet::CFuzzySet(const CFuzzySet& theFSet)
{
```

```
          theRange = theFSet.theRange;
          PunkteKopieren(theFSet);
          m_FuzzyNumber = theFSet.m_FuzzyNumber;
};

CFuzzySet::CFuzzySet(BOOL empty)
{
          if (!empty)
          {
                    theRange = CInterval(-0.1,0.1);
                    AppendPoint(CFuzzySetEntry(-.1,0));
                    AppendPoint(CFuzzySetEntry(0,1));
                    AppendPoint(CFuzzySetEntry(+.1,0));
          }
          else
          {
                    theRange = CInterval(0.0,0.0);
          }
          m_FuzzyNumber = IsValidFuzzyNumber();
};

CFuzzySet::CFuzzySet(double from, double plateau1, double plateau2, double to )  // Create
Trapezoidal
{
          theRange = CInterval(from,to);
          AppendPoint(CFuzzySetEntry(to,0));
          AppendPoint(CFuzzySetEntry(plateau1,1));
          AppendPoint(CFuzzySetEntry(plateau2,1));
          AppendPoint(CFuzzySetEntry(from,0));
          m_FuzzyNumber = IsValidFuzzyNumber();
};

CFuzzySet::CFuzzySet(double from, double over, double to ) // Create Triangular
{
          theRange = CInterval(from,to);
          AppendPoint(CFuzzySetEntry(from,0));
          AppendPoint(CFuzzySetEntry(over,1));
          AppendPoint(CFuzzySetEntry(to,0));
          m_FuzzyNumber = IsValidFuzzyNumber();
};

//////////////////////
/// DESTRUKTOR       //
//////////////////////

CFuzzySet::~CFuzzySet(void)
{
          DeleteAllPoints();
};


////////////////////////////////////
//  O P E R A T O R E N           //
////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////
// Funktion : operator =
// Klasse   : CFuzzySet
/////////////////////////////////////////////////////////////////////////////
// Erstellt aus uebergebenem Objekt eine tiefe Kopie des Uebergabewertes
// Gibt als Rueckgabewert eine Referenz auf sich selbst (wichtig für a = b = c)
/////////////////////////////////////////////////////////////////////////////
```

```
CFuzzySet& CFuzzySet::operator = (const CFuzzySet& fs)
{
    if (this == &fs) // bin ich's selbst ?
          return *this;

    DeleteAllPoints();
    PunkteKopieren(fs);
    theRange = fs.theRange;
    m_FuzzyNumber = fs.m_FuzzyNumber;
    return *this;  // a = b = c
};

/////////////////////////////////////////////////////////////////////////////
// Funktion : operator *=
// Klasse   : CFuzzySet
/////////////////////////////////////////////////////////////////////////////
// Multipliziert alle Zugehörigkeitsgrade mit fs
/////////////////////////////////////////////////////////////////////////////

CFuzzySet* CFuzzySet::operator *= (double fs)
{
          for (int i=0;i < GetNumPoints(); i++)
          {
                    CFuzzySetEntry* p1 = GetPoint(i) ;
                    p1->SetDoM(p1->GetDoM() * fs);
          }
          return this;
};

/////////////////////////////////////////////////////////////////////////////
// Funktion : operator ==
// Klasse   : CFuzzySet
/////////////////////////////////////////////////////////////////////////////
// Stellt fest ob *this = theFSet;
/////////////////////////////////////////////////////////////////////////////


int CFuzzySet::operator == (const CFuzzySet& theFSet) const
{
          BOOL gleich = TRUE;
          gleich &= (theFSet.theRange == theRange);
          gleich &= (theFSet.m_FuzzyNumber == m_FuzzyNumber);
          gleich &= (theFSet.GetMaxIndex() == GetMaxIndex());
          if (gleich)
          for (int i = 0 ; gleich && (i <= GetMaxIndex()); i++)
          {
              CFuzzySetEntry other = GetPoint(i);
              CFuzzySetEntry me = theFSet.GetPoint(i);
                    gleich &= (me == other);
          }
          return gleich;
};

////////////////////////////////////
//  S T A T U S F U N K T I O N E N //
////////////////////////////////////

/////////////////////////////////////////////////////////////////////////////
// Funktion : GetDoM
// Klasse   : CFuzzySet
```

```
////////////////////////////////////////////////////////////////////////////
// Ermittelt aus dem aktuellen CFuzzySet-Objekt den Zugehörigkeitsgrad für die
// übebergebene Realzahl. Falls der Wert direkt in der Punktemenge gefunden werden
// kann wird der zugehörige Wert zurückgegeben , andernfalls wird interpoliert ...
////////////////////////////////////////////////////////////////////////////

double CFuzzySet::GetDoM(double xvalue) const
{
          int last = -1, next = 0 ;
          double theDom = 0;
          double m,b;
          double theVal1 , theVal2;
          double maxval = 0.0;
          double retvalue = 0.0;
          double x1,y1,x2,y2;
          double xval = 0.0;
          BOOL found = FALSE;
          CFuzzySetEntry CurPoint;
          for (int i = 0 ; i < GetNumPoints() ; i++)
          {
                    CurPoint = GetPoint(i);
                    xval = CurPoint.GetX();

                    if (found && (xvalue < xval))
                              break;
                    if ( fabs(xvalue - xval) < 1e-6)
                    {
                              found = TRUE;
                              if (CurPoint.GetDoM() > maxval )
                              {
                                        maxval = CurPoint.GetDoM();
                              }
                    }
          }
          if (found)
                    return maxval;

          found = FALSE;
          for ( i = 0 ; (!found) && (i < GetNumPoints()-1) ; i++)
          {
                    theVal1 = GetPoint(i) -> GetX();
                    theVal2 = GetPoint(i+1) -> GetX();
                    if ((theVal1 <= xvalue) && (theVal2 >= xvalue))
                    {
                              last = i;
                              next = i+1;
                              found = TRUE;
                    }
          }
          if ( last >=0 )
          {
                    y1 = GetPoint(last)->GetDoM();
                    y2 = GetPoint(next)->GetDoM();
                    x1 = GetPoint(last)->GetX();
                    x2 = GetPoint(next)->GetX();
                    ASSERT(x1 != x2);
                    m = (y2-y1)/(x2-x1);
                    b = y1-m*x1;
                    theDom = xvalue * m + b ;
          }
          if (theDom > 0.0)
          {
                    retvalue = theDom;
          }
          else
          {
                    retvalue = 0.0;
          }
          return retvalue;
};

////////////////////////////////////////////////////////////////////////////
// Funktion : GetMinMax
// Klasse   : CFuzzySet
////////////////////////////////////////////////////////////////////////////
// Ermittelt aus dem aktuellen CFuzzySet-Objekt den minimalen sowie den maximalen
// Zugehörigkeitsgrad , welche in den übergebenen Referenzen eingetragen werden
////////////////////////////////////////////////////////////////////////////

void CFuzzySet::GetMinMax(double &min, double &max) const
{
          double theVal;
          int i;

          min = 1.0;
          max = 0.0;
          for (i = 0  ; i < GetNumPoints() ; i++)
          {
                    theVal = GetPoint(i) -> GetDoM();
                    if ( max  < theVal)
                    {
                              max = theVal;
                    };
                    if ( min  > theVal)
                    {
                              min = theVal;
                    };
          }
}

////////////////////////////////////////////////////////////////////////////
// Funktion : IsNormalized
// Klasse   : CFuzzySet
////////////////////////////////////////////////////////////////////////////
// Stellt fest, ob dieser CFuzzySet normalisiert, d.h. den maximalen Zugehörigkeits-
// grad 1.0 besitzt.
////////////////////////////////////////////////////////////////////////////

BOOL CFuzzySet::IsNormalized(void) const
{
          BOOL retvalue;
          double min;
          double max;

          GetMinMax(min,max);
          retvalue = (min == 0.0) && (max == 1.0);
          return retvalue;
}

////////////////////////////////////////////////////////////////////////////
// Funktion : IsConvex
// Klasse   : CFuzzySet
////////////////////////////////////////////////////////////////////////////
// Stellt fest, ob dieser CFuzzySet eine konvexe Form besitzt.
```

```
////////////////////////////////////////////////////////////////////////////

BOOL CFuzzySet::IsConvex(void) const
{
         BOOL bConvex = FALSE;
         enum vorz {pos=1 , neg,none};
         vorz dieser,last; // vorzeichenwechsel ?
         int p=0, vzw=0;
         CFuzzySetEntry *p1,*p2;

         dieser=none;

         while ( p < GetNumPoints()-1)
         {
                  p1 = GetPoint(p+1);
                  p2 = GetPoint(p);
                  if (p1->GetDoM() < p2->GetDoM())
                           dieser = pos;
                  else
                           if (p1->GetDoM() > p2->GetDoM())
                                    dieser = neg;
                           else
                                    dieser = none;
                  if ((dieser != last ) && !(dieser!=none) && (last != none))
                  {
                           vzw++;
                  }
                  p++;
                  last = dieser;
         }
         if (vzw  <1 )
         {
                  bConvex = TRUE;
         }
         return (bConvex);
};


////////////////////////////////////////////////////////////////////////////
// Funktion : Auseinanderziehen
// Klasse   : CFuzzySet
////////////////////////////////////////////////////////////////////////////
// Expandiert ein CFuzzySet-Objekt um 'faktor' in Y-Richtung
////////////////////////////////////////////////////////////////////////////


void CFuzzySet::Auseinanderziehen(double faktor) // fuer bessere numerische Verarbeitung
{
         int p = 0;
         CFuzzySetEntry* p1;
         for (p = 0 ; p < GetNumPoints()-1;p++)
         {
                  p1 = GetPoint(p);
                  p1->SetDoM(p1->GetDoM()*faktor);
         }
};

////////////////////////////////////////////////////////////////////////////
// Funktion : Auseinanderziehen
// Klasse   : CFuzzySet
////////////////////////////////////////////////////////////////////////////
// Staucht ein CFuzzySet-Objekt um 'faktor' in Y-Richtung
////////////////////////////////////////////////////////////////////////////


void CFuzzySet::Zusammendruecken(double faktor) // fuer bessere numerische Verarbeitung
{
         int p = 0;
         CFuzzySetEntry* p1;
         if (faktor == 0.0)
                  faktor = 1.0;  // besser nicht durch 0 teilen ....
         for (p = 0 ; p < GetNumPoints()-1;p++)
         {
                  p1 = GetPoint(p);
                  p1->SetDoM(p1->GetDoM()/faktor);
         }
};

////////////////////////////////////////////////////////////////////////////
// Funktion : GetNormalized
// Klasse   : CFuzzySet
////////////////////////////////////////////////////////////////////////////
// Gibt eine normalisierte Kopie eines CFuzzySet-Objektes
////////////////////////////////////////////////////////////////////////////


CFuzzySet* CFuzzySet::GetNormalized(void) const
{

         CFuzzySet* theSet = new CFuzzySet(*this);
         theSet->Normalize();
         return theSet;
};

////////////////////////////////////////////////////////////////////////////
// Funktion : Normalize
// Klasse   : CFuzzySet
////////////////////////////////////////////////////////////////////////////
// Normalisiert ein CFuzzySet
////////////////////////////////////////////////////////////////////////////

void CFuzzySet::Normalize(void)
{
         double theMax;
         if (GetNumPoints())
         {
                  theMax = Height();
                  if ( theMax != 0.0)
                  {
                           for (int i= 0; i < GetNumPoints(); i++)
                           {
                                    CFuzzySetEntry* p1 = (CFuzzySetEntry*)
pPoint.GetAt(i);
                                    p1->SetDoM(p1->GetDoM() / theMax);
                           }
                  }
         }
};


void CFuzzySet::Sort(void)
{
         for (int i=0;i < GetNumPoints(); i++)
         {
```

```
                    for (int j=i+1; j < GetNumPoints(); j++)
                    {
                              CFuzzySetEntry* p1 = (CFuzzySetEntry*)pPoint.GetAt(i) ;
                              CFuzzySetEntry* p2 = (CFuzzySetEntry*)pPoint.GetAt(j) ;
                              if (p2->GetX() < p1->GetX())
                              {
                                        pPoint.SetAt(i,p2);
                                        pPoint.SetAt(j,p1);
                              }
                    }
          }
};

double CFuzzySet::FuzzyOperation (double a, double b, FuzzyOp theOp, double FuzzyPar) const
{
          double retvalue = 0;
          switch(theOp)
          {
                    case Fo_t_Min               : retvalue = __min(a,b);     break;
                    case Fo_s_Max               : retvalue = __max(a,b);     break;
                    case Fo_t_AlgebProd : retvalue = agp(a,b);               break;
                    case Fo_s_AlgebSum  : retvalue = ags(a,b);               break;
                    case Fo_t_BoundDiff : retvalue = bdd(a,b);               break;
                    case Fo_s_BoundSum  : retvalue = bds(a,b);               break;
                    case Fo_t_DrastProd : retvalue = drp(a,b);               break;
                    case Fo_s_DrastSum  : retvalue = drs(a,b);               break;

                    default : ASSERT (1==0);
          }
          return retvalue;
};

double CFuzzySet::Height(void) const
{
          double theMax     = 0.0;
          double theCurrent = 0.0;
          int i=0;
          CFuzzySetEntry* thePoint;
          if (GetNumPoints())
                    for (i = 0 ; i < GetNumPoints(); i++)
                    {
                              thePoint = GetPoint(i);

                              theCurrent = thePoint->GetDoM();
                              if (theMax < theCurrent)
                              {
                                        theMax = theCurrent;
                                        if (theMax == 1.0)
          //größer wirds nicht
                                                  break;
                              }
                    }
          return theMax;
};

void CFuzzySet::FuzzyOperation (FuzzyOp theOp ,double theValue,double granularity,double
FuzzyPar )
{
          double    val1,
                              val3;
          int                 i;
```

```
     int   numSteps = ((CDopeApp*)AfxGetApp())->m_pFuzzyManager->GetNumSteps();
           double  start  = GetPoint(0)->GetX();
           double  end    = GetPoint(GetNumPoints()-1)->GetX();
           double  step   = ((end-start)/(double)numSteps);
           double  x    = 0.0;
           CFuzzySetEntry *p5;

           if (granularity)
           {

                     for (i=0; i<GetNumPoints() ;i++)
                     {
                         p5 = GetPoint(i);
                              val3 = FuzzyOperation(p5->GetDoM(),theValue,theOp,FuzzyPar);
                              p5->SetDoM(val3);
                     }
                     for ( i = 0,x=start; i <= numSteps ; i++,x+=step)
                     {
                              val1 = GetDoM(x);
                              val3 = FuzzyOperation(val1,theValue,theOp,FuzzyPar);
                              p5 = new CFuzzySetEntry(x,val3);
                              AppendPoint(*p5,irOverWrite);
                              delete p5;
                     }
           }
};

CFuzzySet* CFuzzySet::Operation (FuzzyOp theOp ,double theValue,double granularity,double
FuzzyPar ) const
{
          double    val1,
                              val3;
          long    anzahl = 20;
          double  start  = GetPoint(0)->GetX();
          double  end    = GetPoint(GetNumPoints()-1)->GetX();
          double  step   = ((end-start)/(double)anzahl);
          CFuzzySet* pF3 = new CFuzzySet(TRUE);
          pF3->DeleteAllPoints();
          CFuzzySet* pF4;
          CFuzzySetEntry p5;

     int numSteps = ((CDopeApp*)AfxGetApp())->m_pFuzzyManager->GetNumSteps();

     granularity = (end-start) / numSteps;

           if (granularity)
           {

                     for ( int i=0; i<GetNumPoints() ;i++)
                     {
                              val3 = FuzzyOperation(GetPoint(i)-
>GetDoM(),theValue,theOp,FuzzyPar);
                              p5 = CFuzzySetEntry(GetPoint(i)->GetX(),val3);
                              pF3->AppendPoint(p5);
                     }
                     for ( double x=start; x <= end ; x+=granularity)
                     {
                              val1 = GetDoM(x);
                              val3 = FuzzyOperation(val1,theValue,theOp,FuzzyPar);
                              p5 = CFuzzySetEntry(x,val3);
                              pF3->AppendPoint(p5,irOverWrite);
                     }
```

```
          }
          pF4 = pF3->Reduce();
          delete pF3;
          return pF4;
};

CFuzzySet* CFuzzySet::Operation (FuzzyOp theOp,const CFuzzySet& b,double granularity, double
FuzzyPar) const
{
          double dtheMin,dtheMax;
          double val1,val2,val3;

          CFuzzySet* pF4;
          CFuzzySetEntry p5;

          int numSteps = ((CDopeApp*)AfxGetApp())->m_pFuzzyManager->GetNumSteps();

          dtheMin= __min(GetPoint(0)->GetX(),b.GetPoint(0)->GetX())-0.1;
          dtheMax= __max(GetPoint(GetNumPoints()-1)->GetX(),b.GetPoint(b.GetNumPoints()-1)-
>GetX())+0.1;

          CFuzzySet* pF3 = new CFuzzySet(TRUE);
          pF3->DeleteAllPoints();

          granularity = (dtheMax-dtheMin) / numSteps;

          if (granularity)
          {
                    for ( double x=dtheMin ;x <= dtheMax; x +=granularity)
                    {
                              val1 = GetDoM(x);
                              val2 = b.GetDoM(x);
                              val3 = FuzzyOperation(val1,val2,theOp,FuzzyPar);
                              p5 = CFuzzySetEntry(x,val3);
                              pF3->AppendPoint(p5);
                    }
          }

          pF4 = pF3->Reduce();
          delete pF3;
          return pF4;
};

CInterval& ivPlus(const CInterval& a, const CInterval& b)
{
          return CInterval(a.GetMinX()+b.GetMinX(), a.GetMaxX()+b.GetMaxX());
} ;


CInterval& ivMinus(const CInterval& a, const CInterval& b)
{
          return CInterval(a.GetMinX()-b.GetMaxX(), a.GetMaxX()-b.GetMinX());
} ;

CInterval& ivMulti(const CInterval& a, const CInterval& b)
{
          return CInterval(a.GetMinX()*b.GetMinX(), a.GetMaxX()*b.GetMaxX());
} ;


CInterval& ivDivi(const CInterval& a, const CInterval& b)
{
          ASSERT(b.GetMinX());
          ASSERT(b.GetMaxX());
          return CInterval(a.GetMinX()/b.GetMaxX(), a.GetMaxX()/b.GetMinX());
};

CFuzzySet* CFuzzySet::OperatorTemplate(const CFuzzySet& theFSet,ivfunc theIFunc, arithfunc
theFunc,ZeroAllowed ZA) const
{
          double dtheMin,dtheMax;
          double val1,val2;
          double i,j ,theval,themax;

          CFuzzySetEntry pEntry;

          int numSteps = ((CDopeApp*)AfxGetApp())->m_pFuzzyManager->GetNumSteps();

          const CFuzzySet* pF1 = this;
          const CFuzzySet* pF2 = &theFSet;

          CInterval CIa = pF1->GetRange();
          CInterval CIb = pF2->GetRange();

          CInterval CIc = theIFunc(CIa,CIb);

          CFuzzySet* retvalue;
          CFuzzySet* pF3 = new CFuzzySet(TRUE);


          pF3->SetRange(CIc);

          pF3->DeleteAllPoints();

          dtheMin= CIc.GetMinX();
          dtheMax= CIc.GetMaxX();
          for (i = dtheMin ; i <= dtheMax; i+=0.1)
          {
                    themax=0.0;
                    double jstart = pF1->GetPoint(0)->GetX();
                    double jend   = pF1->GetPoint(GetNumPoints()-1)->GetX();
                    double jvar   = (jend-jstart)/ numSteps;
                    for (int k = 0 ; k <= numSteps; k++)
                    {
                              j = jstart +  jvar * k;
                              if ((ZA) || (j !=0))
                              {
                                        val1 = pF1->GetDoM(j);
                                        val2 = pF2->GetDoM(theFunc(i,j));
                                        theval = __min(val1,val2);
                                        if (theval > themax)
                                        {
                                                  themax = theval;
                                        }
                              }
                    };

                    pEntry = CFuzzySetEntry(i,themax);
                    pF3->AppendPoint(pEntry);
          }
          retvalue = pF3->Reduce();
          delete pF3;
          return retvalue;
};
```

```
double minus (double a, double b)
{
        return a-b;
};

double plus (double a, double b)
{
        return a+b;
};

double geteilt (double a, double b)
{
        ASSERT(b);
        return a/b;
};


double mal (double a, double b)
{
        return a*b;
};

iOpType CFuzzySet::GetOperationType(const CFuzzySet& theFSet) const
{
        double a;

        if (IsLRNumber(a,a,a) && theFSet.IsLRNumber(a,a,a))
                return ArithLRNumber;
        if (IsLRInterval(a,a,a,a) && theFSet.IsLRInterval(a,a,a,a))
                return ArithLRInterval;

        return ArithExtension;
};

CFuzzySet* CFuzzySet::operator + (const CFuzzySet& theFSet) const
{

        double la,lb,ma,mb,ra,rb,ma1,ma2,mb1,mb2;
        CFuzzySet* result;
        switch (GetOperationType(theFSet))
        {
                case  ArithLRNumber :
                {
                        BOOL b1 = IsLRNumber(la,ma,ra);
                        BOOL b2 = theFSet.IsLRNumber(lb,mb,rb);
                        ASSERT(b1);
                        ASSERT(b2);
                        CLRFuzzyNumber a(la,ma,ra);
                        CLRFuzzyNumber b(lb,mb,rb);
                        CLRFuzzyNumber* c = a + b;
                        result = new CFuzzySet(c->GetFuzzySet());
                        delete c;
                        return result;
                        break;

                }
                case  ArithLRInterval :
                {
                        BOOL b1 = IsLRInterval(la,ma1,ma2,ra);
                        BOOL b2 = theFSet.IsLRInterval(lb,mb1,mb2,rb);
                        ASSERT(b1);
                        ASSERT(b2);
                        CLRFuzzyInterval a(la,ma1,ma2,ra);
                        CLRFuzzyInterval b(lb,mb1,mb2,rb);
                        CLRFuzzyInterval* c = a + b;
                        result = new CFuzzySet(c->GetFuzzySet());
                        delete c;
                        return result;
                        break;
                }
                case ArithExtension :
                        return OperatorTemplate(theFSet,ivPlus,minus,ZAYES);
                        break;
                default:
                {
                        ASSERT(FALSE);
                        return NULL;
                        break;
                }

        };
};

CFuzzySet* CFuzzySet::operator - (const CFuzzySet& theFSet) const
{

        double la,lb,ma,mb,ra,rb,ma1,ma2,mb1,mb2;
        CFuzzySet* result;
        switch (GetOperationType(theFSet))
        {
                case  ArithLRNumber :
                {
                        BOOL b1 = IsLRNumber(la,ma,ra);
                        BOOL b2 = theFSet.IsLRNumber(lb,mb,rb);
                        ASSERT(b1);
                        ASSERT(b2);
                        CLRFuzzyNumber a(la,ma,ra);
                        CLRFuzzyNumber b(lb,mb,rb);
                        CLRFuzzyNumber* c = a - b;
                        result = new CFuzzySet(c->GetFuzzySet());
                        delete c;
                        return result;
                        break;

                }
                case  ArithLRInterval :
                {
                        BOOL b1 = IsLRInterval(la,ma1,ma2,ra);
                        BOOL b2 = theFSet.IsLRInterval(lb,mb1,mb2,rb);
                        ASSERT(b1);
                        ASSERT(b2);
                        CLRFuzzyInterval a(la,ma1,ma2,ra);
                        CLRFuzzyInterval b(lb,mb1,mb2,rb);
                        CLRFuzzyInterval* c = a - b;
                        result = new CFuzzySet(c->GetFuzzySet());
                        delete c;
                        return result;
                        break;
                }
                case ArithExtension :
                        return OperatorTemplate(theFSet,ivMinus,plus,ZAYES);
                        break;
                default:
```

```
                    {
                              ASSERT(FALSE);
                              return NULL;
                              break;
                    }

          };
};

CFuzzySet* CFuzzySet::operator * (const CFuzzySet& theFSet) const
{

          double la,lb,ma,mb,ra,rb,ma1,ma2,mb1,mb2;
          CFuzzySet* result;
          switch (GetOperationType(theFSet))
          {
                    case  ArithLRNumber :
                    {
                              BOOL b1 = IsLRNumber(la,ma,ra);
                              BOOL b2 = theFSet.IsLRNumber(lb,mb,rb);
                              ASSERT(b1);
                              ASSERT(b2);
                              CLRFuzzyNumber a(la,ma,ra);
                              CLRFuzzyNumber b(lb,mb,rb);
                              CLRFuzzyNumber* c = a * b;
                              result = new CFuzzySet(c->GetFuzzySet());
                              delete c;
                              return result;
                              break;

                    }
                    case  ArithLRInterval :
                    {
                              BOOL b1 = IsLRInterval(la,ma1,ma2,ra);
                              BOOL b2 = theFSet.IsLRInterval(lb,mb1,mb2,rb);
                              ASSERT(b1);
                              ASSERT(b2);
                              CLRFuzzyInterval a(la,ma1,ma2,ra);
                              CLRFuzzyInterval b(lb,mb1,mb2,rb);
                              CLRFuzzyInterval* c = a * b;
                              result = new CFuzzySet(c->GetFuzzySet());
                              delete c;
                              return result;
                              break;
                    }
                    case ArithExtension :
                              return OperatorTemplate(theFSet,ivMulti,geteilt,ZANO);
                              break;
                    default:
                    {
                              ASSERT(FALSE);
                              return NULL;
                              break;
                    }

          };
};


CFuzzySet* CFuzzySet::operator / (const CFuzzySet& theFSet) const
{
          return OperatorTemplate(theFSet,ivDivi,mal,ZAYES);
};

double CFuzzySet::CentroidDefuzzification(void) const
{
          double    nom   = 0.0, /*zaehler*/
                              denom = 0.0, /*nenner */
                              middle = 0.0,
                              hoehe = 0.0,
                              m = 0.0,
                              b = 0.0,
                              retvalue = 0.0,
                              trap = 0.0,
                              trapsum =0.0;

          double x1,y1,x2,y2;

          CFuzzySetEntry *p1,*p2;

          for (int i=0;i <= GetMaxIndex()-1; i++)
          {
                    p1 = GetPoint(i);
                    p2 = GetPoint(i+1);

                    y1 = p1->GetDoM();
                    y2 = p2->GetDoM();
                    x1 = p1->GetX();
                    x2 = p2->GetX();

                    if (x1 !=x2)
                    {
                              m = (y2-y1)/(x2-x1);
                              b = y1-m*x1;


                              denom     +=  (1/2.0)*m*(pow(x2,2)-pow(x1,2))+b*(x2-x1);   //
<- Trapezinhalt
                              nom       += (1/3.0)*m*(pow(x2,3)-pow(x1,3))+.5*b*(pow(x2,2)-
pow(x1,2));  // <<- Moment
                              trap       = (y1+y2)/2.0 * (x2-x1);
                              trapsum += trap;
                    }
          };
          if (denom != 0.0)
                    retvalue = nom / denom;
          else
                    retvalue = GetRange().GetMean();
          return retvalue;
};


void CFuzzySet::DeleteAllPoints(void)
{
          CFuzzySetEntry *a;
          if (GetNumPoints() )
                    for (int i= 0; i < pPoint.GetSize(); i++)
                    {
                              a =  GetPoint(i);
                              delete a;
                    };
          pPoint.RemoveAll();
};
```

```
CFuzzySet* CFuzzySet::Reduce(void) const
{

          CFuzzySet* mynewone = new CFuzzySet(TRUE);
          mynewone->DeleteAllPoints();
          CFuzzySetEntry *p1, *p2;
          double a,b;
          int size = GetNumPoints();
          double* theSlopes = new double[size];
          int i ;
    if (size)
          {
                    for (i= 0; i < size-1; i++)
                    {
                              p1=GetPoint(i);
                              p2=GetPoint(i+1);
//                            p3=GetPoint(i+2);
                              theSlopes[i] = p1->GetSlope(*p2);
                    };
                    i = 0;
                    while ( i < GetNumPoints()-2)
                    {
                              mynewone->AppendPoint(*GetPoint(i));
                              while (i < GetNumPoints()-2)
                              {
                                        double r,s,t;
                                        a = theSlopes[i];
                                        b = theSlopes[i+1];
                                        r = GetPoint(i)->GetDoM();
                                        s = GetPoint(i+1)->GetDoM();
                                        t = GetPoint(i+2)->GetDoM();
                                        if ((a==b) || ((r == s) && (s== t)))
                                        {
                                                  i++;
                                        }
                                        else
                                        {
                                                  break;
                                        }
                              };
                              i++;
/*                            while ((i < GetNumPoints()-2)&&(c=fabs(fabs(a=theSlopes[i]) -
fabs(b=theSlopes[i+1])) < tolerance ))
                                        i++; */
                              //i++;
                    };
                    p1 = GetPoint(GetNumPoints()-1);
                    mynewone->AppendPoint(*p1);
                    for (i = 1 ; i < mynewone->GetNumPoints()-2; i++) // führende Nullen
weghauen
                    {
                              if ((mynewone->GetPoint(i)->GetDoM() ==0.0) && (mynewone-
>GetPoint(i-1)->GetDoM() ==0.0))
                              {
                                        mynewone->DeletePoint(i-1);
                                        i--;
                              };
                    }
          }
          delete []theSlopes;        // nicht nur einen freigeben !!!!
          return mynewone;
};
```

```
CInterval CFuzzySet::GetRange(void)  const
{
    return theRange;
};

void CFuzzySet::SetRange(const CInterval& tR)
{
    theRange = tR;
};

int CFuzzySet::GetNumPoints(void) const
{
    return pPoint.GetSize();
};

int CFuzzySet::GetMaxIndex(void) const
{
    return pPoint.GetUpperBound();
};

CFuzzySetEntry* CFuzzySet::GetRightMostPoint(void)  const
{
    return GetPoint(pPoint.GetUpperBound());
};

CFuzzySetEntry* CFuzzySet::GetLeftMostPoint(void)  const
{
    return GetPoint(0);
};

double CFuzzySet::GetRMX(void) const
{
    return GetRightMostPoint()->GetX();
}; // rightmost x

double CFuzzySet::GetLMX(void) const
{
    return GetLeftMostPoint()->GetX();
};  // leftmost x

CFuzzySetEntry* CFuzzySet::GetPoint(int nPos) const
{
          CFuzzySetEntry* result = NULL;
          if ((nPos >= 0) && (nPos < pPoint.GetSize()))
          {
                    result = ((CFuzzySetEntry*) pPoint.GetAt(nPos));
          }
          return result;
};

#ifdef _DEBUG

void CFuzzySet::Dump(CDumpContext &dc) const
{
          CObject::Dump(dc);

          dc << GetRange();

          dc << "Anzahl Punkte : " << GetNumPoints();
};
```

```
#endif

void CFuzzySet::SetFuzzyNumberOn(void)
{
         m_FuzzyNumber = TRUE;
};

void CFuzzySet::SetFuzzyNumberOff(void)
{
         m_FuzzyNumber = FALSE;
};

CFuzzySet* CFuzzySet::operator - (void)
{
         CFuzzySet* erg = operator*=(-1);
         return erg;
};


CFuzzySet* CFuzzySet::GibKehrwert (void) const
{
 return new CFuzzySet(*this);
};

CString CFuzzySet::GibText (void) const
{
         char buf[40];
         CString CrLf = "\r\n";
         sprintf(buf,"(%f-%f)", theRange.GetMinX(),theRange.GetMaxX());
         CString a = (m_FuzzyNumber) ? CString("FuzzyNumber") : CString("FuzzySet");
         a  += CString(buf);
         for (int i = 0 ; i <= GetMaxIndex(); i++)
         {
             CFuzzySetEntry theEntry = GetPoint(i);
                   CString c = theEntry.GibText();
                   a += c ;
         }
         return a;
};

BOOL  CFuzzySet::IstPositiv(void) const
{
         BOOL IstPositiv = TRUE;
         for (int i = 0 ; i <= GetMaxIndex() && IstPositiv; i++)
         {
             CFuzzySetEntry theEntry = GetPoint(i);
             IstPositiv &= (theEntry.GetX() > 0.0);
         }
         return IstPositiv;
};

BOOL  CFuzzySet::IstNegativ(void) const
{
         BOOL IstNegativ = TRUE;
         for (int i = 0 ; i <= GetMaxIndex() && IstNegativ; i++)
         {
             CFuzzySetEntry theEntry = GetPoint(i);
             IstNegativ &= (theEntry.GetX() < 0.0);
         }
         return IstNegativ;
};
```

```
void CFuzzySet::DeletePoint(int i)
{
         delete pPoint.GetAt(i);
         pPoint.RemoveAt(i);
};


BOOL CFuzzySet::DeletePoint(const CFuzzySetEntry& theEntry)
{
         BOOL IsDeleted = FALSE;
         for (int i = 0 ; i <= GetMaxIndex() && !IsDeleted; i++)
         {
             CFuzzySetEntry thePoint = GetPoint(i);
             if (theEntry == thePoint)
             {
                   DeletePoint(i);
                             IsDeleted = TRUE;
                   }
         }
         return IsDeleted;
};



void CFuzzySet::Serialize(CArchive& ar)
{
    long tmpfn;

         CUnserObjekt::Serialize(ar);

         theRange.Serialize(ar);

         if( ar.IsStoring() )
         {
                   ar << (long) m_FuzzyNumber;
         }
         else
         {
        DeleteAllPoints();
                   ar >> tmpfn;
                   m_FuzzyNumber = (BOOL) tmpfn;
         }

         pPoint.Serialize(ar);
};

CString CFuzzySet::Display(void) // sollte eigentlich const sein !?!
{
         char szBuf[30];
         sprintf(szBuf, "%f",CentroidDefuzzification());
         CString Return = "circa "+ CString(szBuf);
         return Return;
};

CFuzzySet* CFuzzySet::ConvexContour(void) const
{
         CHelperFunctions fch;
         CFuzzySet* result = new CFuzzySet();
         point p[100];
         int iNum = GetNumPoints();
         for (int i = 0; i < iNum; i++)
```

```
        {
                p[i].x = GetPoint(i)->GetX();
                p[i].y = GetPoint(i)->GetDoM();
        };

        int num = fch.wrap(p,iNum);

        result->DeleteAllPoints();

        for (i = 0; i < num ; i++)
        {
                CFuzzySetEntry theNew = CFuzzySetEntry(p[i].x,p[i].y);
                result->AppendPoint(theNew);
        };

        result->Normalize();

        return result;
};

double CFuzzySet::HeightDefuzzification(void) const
{
        double sum1         = 0;
        double sum2         = 0;
        double peak         = 0;
        double pwert        = 0;

        double dCurX        = 0.0;
        double dCurDom      = 0.0;

        double retvalue = 0.0;
    peak = GetRange().GetMinX();

    BOOL bWasPeak;
        for (int i = 0; i < GetNumPoints(); i++)
        {
                CFuzzySetEntry thePoint = GetPoint(i);
                dCurX   = thePoint.GetX();
                dCurDom = thePoint.GetDoM();
                if (dCurDom > pwert)
                {
                        bWasPeak = TRUE;
                }
                if ((dCurDom < pwert) && (bWasPeak))
                {
                        bWasPeak = FALSE;
                        sum1 += peak * pwert;
                        sum2 += pwert;
                }
                if ((dCurDom == pwert) && (dCurDom > 0))
                {
                        sum1 += peak * pwert;
                        sum2 += pwert;
                }
                pwert = dCurDom;
                peak  = dCurX;
        }
        if (sum2 != 0)      // wenn moeglich, errechne Defuzzy-Wert
                retvalue = sum1 / sum2;
        else
                {
                if (sum1 == 0)
                        retvalue = GetRange().GetMean();
                }
        return retvalue;
};

double CFuzzySet::LeftOfMaxDefuzzification(void) const
{
    double maximum = 0.0;
    double maxX    = 0.0;
          double dCurX        = 0.0;
          double dCurDom      = 0.0;


    for (int i = 0; i < GetNumPoints(); i++)
          {
                    CFuzzySetEntry thePoint = GetPoint(i);
                    dCurX   = thePoint.GetX();
                    dCurDom = thePoint.GetDoM();
                    if (dCurDom > maximum)
                    {
                            maximum = dCurDom;
                            maxX    = dCurX;
                    }
          }
          return maxX;
};

double CFuzzySet::RightOfMaxDefuzzification(void)const
{
    double maximum = 0.0;
    double maxX    = 0.0;
          double dCurX        = 0.0;
          double dCurDom      = 0.0;

    for (int i = 0; i < GetNumPoints(); i++)
          {
                    CFuzzySetEntry thePoint = GetPoint(i);
                    dCurX   = thePoint.GetX();
                    dCurDom = thePoint.GetDoM();
                    if (dCurDom >= maximum)
                    {
                            maximum = dCurDom;
                            maxX    = dCurX;
                    }
          }
          return maxX;
};


double CFuzzySet::MeanOfMaxDefuzzification(void) const
{
    double retvalue;
    retvalue = (LeftOfMaxDefuzzification()+RightOfMaxDefuzzification())/2.0; //oops , missing
brackets
    return retvalue;
};

BOOL CFuzzySet::IsLRNumber(double& left, double& middle, double &right) const
{
          if (GetNumPoints() != 3)
                    return FALSE;
          if ((GetPoint(0)->GetDoM() + GetPoint(2)->GetDoM()) != 0.0)
```

```
                return FALSE;
        if (GetPoint(1)->GetDoM() != 1.0)
                return FALSE;
        left = GetPoint(0)->GetX();
        middle = GetPoint(1)->GetX();
        right = GetPoint(2)->GetX();
        return TRUE;
}

BOOL CFuzzySet::IsLRInterval(double& left, double& middle1,double& middle2, double &right)
const
{
        if (GetNumPoints() != 4)
                return FALSE;
        if ((GetPoint(0)->GetDoM() + GetPoint(3)->GetDoM()) != 0.0)
                return FALSE;
        if ((GetPoint(1)->GetDoM() != 1.0) && (GetPoint(1)->GetDoM() != 1.0))
                return FALSE;
        left = GetPoint(0)->GetX();
        middle1 = GetPoint(1)->GetX();
        middle2 = GetPoint(2)->GetX();
        right = GetPoint(3)->GetX();
        return TRUE;
}

CLRFuzzyNumber::CLRFuzzyNumber(const CFuzzySet& theFuzzySet)
{
        double a,b,c;
        BOOL b1 = theFuzzySet.IsLRNumber(a,b,c);
        ASSERT(b1);
        left = a;
        middle = b;
        right = c;
};

CLRFuzzyInterval::CLRFuzzyInterval(const CFuzzySet& theFuzzySet)
{
        double a,b,c,d;
        BOOL b1 = theFuzzySet.IsLRInterval(a,b,c,d);
        ASSERT(b1);
        left = a;
        middle1 = b;
        middle2 = c;
        right = d;
};

CLRFuzzyInterval::CLRFuzzyInterval(const CLRFuzzyInterval& theLRFuzzyInterval)
{
        left = theLRFuzzyInterval.left;
        middle1 = theLRFuzzyInterval.middle1;
        middle2 = theLRFuzzyInterval.middle2;
        right = theLRFuzzyInterval.right;
};


CLRFuzzyNumber::CLRFuzzyNumber(const CLRFuzzyNumber& theLRFuzzyNumber)
{
        left = theLRFuzzyNumber.left;
        middle = theLRFuzzyNumber.middle;
        right = theLRFuzzyNumber.right;
};
```

```
CFuzzySet* CLRFuzzyInterval::GetFuzzySet(void) const
{
        return new CFuzzySet(left,middle1,middle2,right);
};

CFuzzySet* CLRFuzzyNumber::GetFuzzySet(void) const
{
        return new CFuzzySet(left,middle,right);
};

CLRFuzzyNumber::CLRFuzzyNumber(double a, double b, double c)
{
        left = a;
        middle = b;
        right = c;
};

CLRFuzzyInterval::CLRFuzzyInterval(double a, double b, double c,double d)
{
        left = a;
        middle1 = b;
        middle2 = c;
        right = d;
};


BOOL CLRFuzzyNumber::IstPositiv(void) const
{
        BOOL b1 = (left > 0);
        BOOL b2 = (middle > 0);
        BOOL b3 = (right > 0);
        BOOL b4 = b1 && b2 && b3;
        return b4;
};

BOOL CLRFuzzyNumber::IstNegativ(void) const
{
        BOOL b1 = (left < 0);
        BOOL b2 = (middle < 0);
        BOOL b3 = (right < 0);
        BOOL b4 = b1 && b2 && b3;
        return b4;
};


BOOL CLRFuzzyInterval::IstPositiv(void) const
{
        BOOL b1 = (left > 0);
        BOOL b2 = (middle1 > 0);
        BOOL b3 = (middle2 > 0);
        BOOL b4 = (right > 0);
        BOOL b5 = b1 && b2 && b3 && b4;
        return b5;
};

BOOL CLRFuzzyInterval::IstNegativ(void) const
{
        BOOL b1 = (left < 0);
        BOOL b2 = (middle1 < 0);
        BOOL b3 = (middle2 < 0);
        BOOL b4 = (right < 0);
```

```
	BOOL b5 = b1 && b2 && b3 && b4;
	return b5;
};


CLRFuzzyNumber* CLRFuzzyNumber::operator + (const CLRFuzzyNumber& theFSet) const
{
	return new
CLRFuzzyNumber(left+theFSet.left,middle+theFSet.middle,right+theFSet.right);
};

CLRFuzzyNumber* CLRFuzzyNumber::operator - (const CLRFuzzyNumber& theFSet) const
{
	return new CLRFuzzyNumber(left-theFSet.right,middle-theFSet.middle,right-
theFSet.left);
};

CLRFuzzyNumber* CLRFuzzyNumber::operator * (const CLRFuzzyNumber& theFSet) const
{
	double m = middle;
	double n = theFSet.middle;

	double ma = middle-left;
	double mb = right-middle;

	double nc = theFSet.middle-theFSet.left;
	double nd = theFSet.right-theFSet.middle;

	if (IstPositiv() && theFSet.IstPositiv())
		return new CLRFuzzyNumber(m*nc+ma*n,n*m,m*nc+n*mb);

	else if (IstPositiv() && theFSet.IstNegativ())
		return new CLRFuzzyNumber(n*ma-m*nd,n*m,n*mb-m*nc);

	else if (theFSet.IstPositiv() && theFSet.IstNegativ())
		return new CLRFuzzyNumber(m*nc-n*mb,n*m,m*nd-n*ma);

	else if (IstNegativ() && theFSet.IstNegativ())
		return new CLRFuzzyNumber(-n*mb-m*nd,m*n,-n*ma-m*nc);
	else
	{
		ASSERT(FALSE);
		return NULL;
	}
};

CLRFuzzyNumber* CLRFuzzyNumber::operator / (const CLRFuzzyNumber& theFSet) const
{
	double m = middle;
	double n = theFSet.middle;

	double ma = middle-left;
	double mb = right-middle;

	double nc = theFSet.middle-theFSet.left;
	double nd = theFSet.right-theFSet.middle;

	double nn = n*n;

	if (IstPositiv() && theFSet.IstPositiv())
		return new CLRFuzzyNumber((m*nd+n*ma)/nn,m/n, (m*nc+n*mb)/nn);
	else
	{
		ASSERT(FALSE);
		return NULL;
	}
};


CLRFuzzyInterval* CLRFuzzyInterval::operator * (const CLRFuzzyInterval& theFSet) const
{
	double m1 = middle1;
	double m2 = middle2;
	double n1 = theFSet.middle1;
	double n2 = theFSet.middle2;

	double ma = middle1-left;
	double mb = right-middle2;

	double	nc = theFSet.middle1-theFSet.left;
	double nd = theFSet.right-theFSet.middle2;

	if (IstPositiv() && theFSet.IstPositiv())
		return new CLRFuzzyInterval(m1*n1*nc,m1*n1,m1*n2,m2*nd+n2*mb);

	else
	{
		ASSERT(FALSE);
		return NULL;
	}

};

CLRFuzzyInterval* CLRFuzzyInterval::operator / (const CLRFuzzyInterval& theFSet) const
{
	ASSERT(FALSE);
	return NULL;
};


CLRFuzzyInterval* CLRFuzzyInterval::operator + (const CLRFuzzyInterval& theFSet) const
{
	return new
CLRFuzzyInterval(left+theFSet.left,middle1+theFSet.middle1,middle2+theFSet.middle2,right+theF
Set.right);
};

CLRFuzzyInterval* CLRFuzzyInterval::operator - (const CLRFuzzyInterval& theFSet) const
{
	return new CLRFuzzyInterval(left-theFSet.right,middle1-theFSet.middle1,middle2-
theFSet.middle2,right-theFSet.left);
};
```

FCSE.H

```
//fse.h
#ifndef _FCSE_H
#define _FCSE_H
#include "afx.h"
#include "fc.h"

//#define _byte_version_


// für den InOutManager:
class CInOutConv;   // bs 27.05.
class CMyStringList;

class CFuzzySetEntry:public CObject
{
  public:
  DECLARE_SERIAL( CFuzzySetEntry );
  protected:
  double x;
#ifdef _byte_version_
         unsigned char dom;
#else
         double dom;
#endif


  public:
    CFuzzySetEntry();
    CFuzzySetEntry(double inx,double indom);
    CFuzzySetEntry(CFuzzySetEntry* theEntry);
    CFuzzySetEntry(const CFuzzySetEntry& theEntry);
    ~CFuzzySetEntry();
    // reading functions
    double GetX() const ;
    double GetDoM() const;
    double GetSlope(const CFuzzySetEntry& me) const ;
    CString GibText() const ;
    int operator != (const CFuzzySetEntry& me) const;
    int operator == (const CFuzzySetEntry& me) const;
    BOOL IsValidDoM() const ;
    // modifying functions
    void SetDoM(double mydom);
    void SetX(double myx);


#ifndef _INDREM_
    friend ostream& operator <<(ostream& os , CFuzzySetEntry& me);
    friend ostream& operator <<(ostream& os , CFuzzySetEntry* me);
#endif
         CFuzzySetEntry* operator * (double fs);
    CFuzzySetEntry& operator = (const CFuzzySetEntry& me);
    CFuzzySetEntry& operator = (CFuzzySetEntry* me);

    BOOL IsOnTheSameLine(const CFuzzySetEntry &me1,const CFuzzySetEntry &me2) const ;
    void GetParameters(double &xval,double &dom) const ;

         // für den InOutManager:
         virtual void                          Read( CInOutConv* pConv, BYTE *bError );
         // in inoutrw.cpp
         virtual CMyStringList*        Write( CInOutConv* pConv );
                  // bs 27.05.

    //virtual CString Dump(CString theName) const ;
    virtual void Serialize(CArchive& ar) ;
#ifdef _DEBUG
         virtual void Dump(CDumpContext& dc) const;
#endif
};

#endif //_FCSE_H
```

FCSE.CPP

```
// FuzzySetEntry && Interval && BOOL - Implementation
#include "stdafx.h"
#include <math.h>

#include "fcse.h"
#include "fchelp.h"

IMPLEMENT_SERIAL( CFuzzySetEntry, CObject, 0)

#define new DEBUG_NEW

CString CFuzzySetEntry::GibText() const
{
    char buf[20];
    sprintf(buf,"(%f/%f)",GetX(),GetDoM())    ;
          return CString(buf);
}


CFuzzySetEntry::CFuzzySetEntry(double inx,double indom)
{
          x=inx;
          // ASSERT(indom >= 0.0);
          if (indom >1.0)
          {
                    dom = 1.0;

          }
          else
          {
                    if (indom <0.0)
                    {
                              dom = 0.0;
                    }
                    else
                    {
                              dom = indom;
                    }
          }
};

CFuzzySetEntry::CFuzzySetEntry(CFuzzySetEntry* theEntry)
{
          x=theEntry->x;
          ASSERT(theEntry->IsValidDoM());
          dom = theEntry->dom ;
};

CFuzzySetEntry::CFuzzySetEntry(const CFuzzySetEntry& theEntry)
{
          x=theEntry.x;
          ASSERT(theEntry.IsValidDoM());
          dom = theEntry.dom ;
};

CFuzzySetEntry::CFuzzySetEntry()
{
          x=0.0;
          dom = 0.0;
};

double CFuzzySetEntry::GetX()  const
{
    return x;
};

double CFuzzySetEntry::GetDoM() const
{
          return dom;
};

void CFuzzySetEntry::SetDoM(double mydom)
{
          dom       = mydom;
};

void CFuzzySetEntry::SetX(double myx)
{
          double tmpx,tmpy;
    tmpy = modf(myx * 1e6,&tmpx);
    x = double (tmpx / 1e6);

};


CFuzzySetEntry& CFuzzySetEntry::operator = (const CFuzzySetEntry& me)
{
    x= me.x;
    dom = me.dom;
    return *this;
};

CFuzzySetEntry& CFuzzySetEntry::operator = (CFuzzySetEntry* me)
{
    x   = me->x;
    dom = me->dom;
    return *this;
};

CFuzzySetEntry* CFuzzySetEntry::operator * (double fs)
{
    return new CFuzzySetEntry(x,dom*fs);
};


double CFuzzySetEntry::GetSlope(const CFuzzySetEntry& me) const
{
    double retvalue;
    double denom = me.GetX()-GetX();
    double temp  = me.GetDoM() - GetDoM();
    if (denom)
    {
        ASSERT(denom);
        retvalue =  temp/denom;
    }
    else
        retvalue = temp;

    return retvalue;

};
```

```
void CFuzzySetEntry::GetParameters(double &xval,double &dom) const
{
    xval = GetX();
    dom  = GetDoM();
} ;


BOOL CFuzzySetEntry::IsOnTheSameLine(const CFuzzySetEntry& me1,const CFuzzySetEntry &me2)
const
{
    double x1,x2,x3,y1,y2,y3;
    BOOL retvalue;
    GetParameters(x1,y1);
    me1.GetParameters(x2,y2);
    me2.GetParameters(x3,y3);
    ASSERT(x2 != x1);
    double slope = (y2-y1)/(x2-x1);
    double intersec1 = y1-slope*x1;
    double intersec2 = y2-slope*x2;
    retvalue = (y3 == slope*x3+intersec1);
    return retvalue;
};


int CFuzzySetEntry::operator == (const CFuzzySetEntry& me) const
{

    return (x == me.x) && (dom == me.dom);

};


CFuzzySetEntry::~CFuzzySetEntry()
{
 ;
};

BOOL CFuzzySetEntry::IsValidDoM() const
{
  BOOL b1 = (dom >= 0.0); ASSERT(b1);
  BOOL b2 = (dom <= 1.0); ASSERT(b2);
  return (b1 && b2);
};

#ifdef _DEBUG
void CFuzzySetEntry::Dump(CDumpContext &dc) const
{
//        CObject::Dump(dc);

          dc << "(X:" << x << "/";
          dc <<  "D:" << dom << ")";
};
#endif


void CFuzzySetEntry::Serialize(CArchive& ar)
{
          double theDom;

          CObject::Serialize(ar);

          if (ar.IsStoring())
          {
                    ar << x;

                    theDom = dom;

          ar << theDom;
          }
          else
          {
                    ar >> x;
                    ar >> theDom;

                    dom = theDom;
          };
};
```

FCRULE.H

```
// frule.h
#ifndef _FCRULE_H
#define _FCRULE_H

class CRulePart : public CObject
{
public:
  DECLARE_SERIAL( CRulePart );

protected:
  CLingVar* m_theVar;
//  CLingTerm* m_theTerm;
  CString   m_csTermName;
  public:
  CRulePart(void) ;
  //CRulePart(const CLingVar& theVar, const CLingTerm& theTerm);
  CRulePart(CLingVar* theVar, const CString csLingTermName);
//  CRulePart(const CRulePart& theRule);
  ~CRulePart(void);
  virtual CRulePart& operator = (const CRulePart& theRule);
  CLingVar* GetLV(void) const ;
  CLingTerm* GetTerm(void) const ;
  virtual CString Dump(CString theName);
  virtual void Serialize(CArchive& ar);
};

class CAntecedent : public CRulePart
{
  public:
  CAntecedent(const CAntecedent& theRule);
  //CAntecedent(const CLingVar& theVar, const CLingTerm& theTerm)  ;
  CAntecedent(CLingVar* theVar, const CString csLingTermName);
  ~CAntecedent(void);
  virtual CAntecedent& operator = (const CAntecedent& theRule);
};

class CConsequent : public CRulePart
{
public:
  CConsequent(const CConsequent& theRule);
  //CConsequent(const CLingVar& theVar, const CLingTerm& theTerm);
  CConsequent(CLingVar* theVar, const CString csLingTermName);
  ~CConsequent(void);
  virtual CConsequent& operator = (const CConsequent& theRule);
};

class CFakt;
class CRulebase;

class CRule:public CObject
{
friend class CRulebase;
public:
  DECLARE_SERIAL( CRule);

protected:
  double    m_Certainty;
  BOOL      m_enabled;
  int       m_iInferLevel;
  FuzzyOp   opAggregation;
  FuzzyOp   m_tNorm;
  FuzzyOp   m_tCertainty;
  FuzzyOp   m_sNorm;
  FuzzyImp  m_opImplication;

  CMapStringToOb m_Antecedents;
  CMapStringToOb m_Consequents;

public:
  void SetCertainty(double theCertainty);
  CRule(double theCertainty = 1.0,BOOL enabled=TRUE,int iInferLevel = 0,FuzzyImp theImp =
Fi_mamdani,FuzzyOp theOp = Fo_t_Min,FuzzyOp sn = Fo_s_Max,FuzzyOp tn = Fo_t_Min);
  CRule(const CRule& theRule);
  void AddAntecedent(const CAntecedent& theAnte);
  void AddConsequent(const CConsequent& theCons);

  int GetNumAntecedents(void) const ;
  int GetNumConsequents(void) const;

  CFactBase* Infer(const CFactBase& theFactBase,ReasoningMethod theReasMet = RM_Approximate,

                  FuzzyOp tNorm = Fo_t_Min,

                      FuzzyOp sNorm = Fo_s_Max,
                                              FuzzyImp ImpOp = Fi_mamdani) const;
  CFuzzySet* GeneralizedModusPonens(FuzzyOp tNorm,FuzzyOp sNorm,FuzzyImp fi_implikation,
              const CAntecedent& pA,
              const CConsequent& pB,
              const CFakt& pAstrich,
              double step =0.1) const;
  BOOL Enable(void);
  BOOL Disable(void);
  ~CRule(void);
  void SetInferLevel(int theLevel);
  int GetInferLevel(void) const ;
  FuzzyOp GettNorm(void) const ;
  FuzzyOp GettCertainty(void) const ;
  FuzzyOp GetsNorm(void) const ;

  void  SettNorm(FuzzyOp thetNorm);
  void  SettCertainty(FuzzyOp thetNorm);
  void  SetsNorm(FuzzyOp thesNorm);

  FuzzyImp GetImplication(void) const ;
  FuzzyOp GetopAggregation(void) const ;
  void SetopAggregation(FuzzyOp thesNorm);
  void SetopImplication(FuzzyImp theImp);
  virtual CString Dump(CString theName);
  virtual void Serialize(CArchive& ar);
};

class CRulebase:public CEinfachObjekt
{
public:
  DECLARE_SERIAL( CRulebase );

protected:
 FuzzyOp                    opAggregation;
 ReasoningMethod    m_iInferMethod;
 double                     m_dAggregationParam;
```

```
 CObList*                            m_Regeln;

 public:
 CRulebase(ReasoningMethod ReasMet = RM_Approximate,FuzzyOp AggOp = Fo_s_Max, double AggParam
= 1.0);
 CFactBase* CombineResults(const CFactBase& theFB1,const CFactBase& theFB2,FuzzyOp
theOp,double theOpParm) const;

 int GetNumLayers(void) const;
 FuzzyOp GetOpAggregation(void) const ;
 void  SetOpAggregation(FuzzyOp thesNorm);
 double GetParamAggregation(void) const ;
 void  SetParamAggregation(double theParam);

 CObList* GetRulesInLayer(int iLayer) const;
 CFactBase* Infer(const CFactBase& theFactBase) const;
 void AddRule(const CRule& theRule);
 void AddRule(CRule* theRule);
 void Apply(const CFactBase& theFacts,CMapStringToOb& theResults); // auf CFuzzySets
 int GetNumRules(void) const;
 CMapStringToOb* GetInferencePath(void) const;
 CMapStringToOb* GetResultLVs(void)const;
 CMapStringToOb* GetInputLVs(void)const;
 ~CRulebase(void);
 //virtual CString Dump(CString theName) const ;
 virtual void Serialize(CArchive& ar);
 virtual EinfachTyp GibEinfachTyp(void) const;
 virtual CTyp* GibTyp(void) const;
 virtual CString Display(void);
 // to do
 #ifdef _DEBUG
    virtual void Dump(CDumpContext& dc) const;
 #endif

};


#endif _FCRULE_H
```

FCRULE.CPP

```
#include "stdafx.h"
#include "fc.h"
#include "fcset.h"
#include "fclv.h"
#include "fcfacts.h"
#include "fcrule.h"
#include "fchelp.h"
#include "typframe.h"
#include <math.h>

IMPLEMENT_SERIAL( CRule, CObject,0)
IMPLEMENT_SERIAL( CRulePart, CObject,0)
IMPLEMENT_SERIAL( CRulebase, CEinfachObjekt,0)

#define new DEBUG_NEW        // Der Feind heisst : Memory-Leak

CRulePart::CRulePart(void)
{
         m_theVar = NULL;
         m_csTermName = "";

};

CLingVar* CRulePart::GetLV(void) const
{
    return m_theVar;
};

CLingTerm* CRulePart::GetTerm(void) const
{
         CLingTerm* myTerm = m_theVar->GetLingTerm(m_csTermName);
    return myTerm;
};



CRulePart::CRulePart(CLingVar* theVar, const CString csLingTermName)
{
         m_theVar = theVar;
         CLingTerm* myTerm = m_theVar->GetLingTerm(csLingTermName);
         ASSERT(myTerm);
         m_csTermName = csLingTermName;
};

CAntecedent::CAntecedent(CLingVar* theVar, const CString csLingTermName)
{
         m_theVar = theVar;
         CLingTerm* myTerm = m_theVar->GetLingTerm(csLingTermName);
         ASSERT(myTerm);
         m_csTermName = csLingTermName;
};

CConsequent::CConsequent(CLingVar* theVar, const CString csLingTermName)
{
         m_theVar = theVar;
         CLingTerm* myTerm = m_theVar->GetLingTerm(csLingTermName);
         ASSERT(myTerm);
         m_csTermName = csLingTermName;
};

CAntecedent::CAntecedent(const CAntecedent& theRule)
{
         m_theVar = theRule.m_theVar;
         m_csTermName = theRule.m_csTermName ;
};

CConsequent::CConsequent(const CConsequent& theRule)
{
         m_theVar = theRule.m_theVar;
         m_csTermName = theRule.m_csTermName ;
};

CRulePart& CRulePart::operator = (const CRulePart& theRule)
{
         m_theVar = theRule.m_theVar;
         m_csTermName = theRule.m_csTermName ;
         return *this;
};

CAntecedent& CAntecedent::operator = (const CAntecedent& theRule)
{
         m_theVar = theRule.m_theVar;
         m_csTermName = theRule.m_csTermName ;
         return *this;
};

CConsequent& CConsequent::operator = (const CConsequent& theRule)
{
         m_theVar = theRule.m_theVar;
         m_csTermName = theRule.m_csTermName ;
         return *this;
};

CConsequent::~CConsequent(void)
{
         ;
}


CAntecedent::~CAntecedent(void)
{
         ;
}


CRulePart::~CRulePart(void)
{
         ASSERT(m_theVar);
//       ASSERT(m_theTerm);
//       delete m_theVar;
//       delete m_theTerm;
};

CRule::CRule(double theCertainty,BOOL enabled,int iInferLevel,FuzzyImp theImp,FuzzyOp theOp
,FuzzyOp sn ,FuzzyOp tn )
{
  m_sNorm = sn;
  m_tNorm = tn;
  SetCertainty(theCertainty);
  m_enabled = enabled;
```

```
  opAggregation = theOp;
  m_opImplication = theImp;
  m_iInferLevel = iInferLevel;
};


CRule::CRule(const CRule& theRule)
{
  POSITION pos;
  CAntecedent* ante;
  CConsequent* cons;
  CString key;

  for (pos = theRule.m_Antecedents.GetStartPosition(); pos != NULL;)
  {
         theRule.m_Antecedents.GetNextAssoc(pos,key,(CObject*&)ante);
         ASSERT(ante);
         m_Antecedents.SetAt(key,new CAntecedent(*ante));
  };

  for (pos = theRule.m_Consequents.GetStartPosition(); pos != NULL;)
  {
         theRule.m_Consequents.GetNextAssoc(pos,key,(CObject*&)cons);
         ASSERT(cons);
         m_Consequents.SetAt(key,new CConsequent(*cons));
  };

  m_sNorm = theRule.m_sNorm;
  m_tNorm = theRule.m_tNorm;
  m_tCertainty = theRule.m_tCertainty;
  SetCertainty(theRule.m_Certainty);
  m_enabled = theRule.m_enabled;
  opAggregation = theRule.opAggregation;
  m_opImplication = theRule.m_opImplication;
  m_iInferLevel = theRule.m_iInferLevel;
};

CRule::~CRule(void)
{
  POSITION pos;
  CString key;
  CAntecedent* ante;
  CConsequent* cons;
  for (pos = m_Consequents.GetStartPosition(); pos != NULL;)
  {
         m_Consequents.GetNextAssoc(pos,key,(CObject*&)cons);
         delete cons;
  };

  for (pos = m_Antecedents.GetStartPosition(); pos != NULL;)
  {
         m_Antecedents.GetNextAssoc(pos,key,(CObject*&)ante);
         delete ante;
  };
};

void CRule::SetCertainty(double theCertainty)
{
         m_Certainty = theCertainty;
};
```

```
void CRule::AddConsequent(const CConsequent& theCons)
{
  CString csName = theCons.GetLV()->GetName();
  m_Consequents.SetAt(csName ,new CConsequent(theCons));
};

void CRule::AddAntecedent(const CAntecedent& theAnte)
{
  CString csName = theAnte.GetLV()->GetName();
  m_Antecedents.SetAt(csName ,new CAntecedent(theAnte));
};

CFactBase* CRule::Infer(const CFactBase& theFactBase,ReasoningMethod theMethod,FuzzyOp
tNorm,FuzzyOp sNorm,FuzzyImp ImpOp) const
{
  CFactBase* theReturnValues = new CFactBase(theFactBase);
  CConsequent* theCons;
  CAntecedent* theAnte;
  CString keyA,keyC;
  POSITION posA,posC;
  CFuzzySet* ptheResult = NULL ;
  CFuzzySet* pInter  = NULL ;
  CFuzzySet* theTmpFS = NULL;
  double dCompMeasure ;

  // pInter = new CFuzzySet(TRUE);

  int i=0;
  CFakt pResultFact;

  for (posC = m_Consequents.GetStartPosition();posC != NULL;) // alle Schluesse
  {
    //ptheResult = new CFuzzySet(TRUE,1.0);
    //ptheResult->DeleteAllPoints();
    m_Consequents.GetNextAssoc(posC,keyC,(CObject*&)theCons);
    CLingVar* theLingVar = theCons->GetLV();

    CString csResName = theLingVar->GetName();

    for (i=0,posA = m_Antecedents.GetStartPosition();posA != NULL;i++) // alle
Voraussetzungen
    {
       m_Antecedents.GetNextAssoc(posA,keyA,(CObject*&)theAnte);
       CLingVar* theALingVar = theAnte->GetLV();
       CString csLVName = theALingVar->GetName();
       CFakt* pThisFact = theFactBase.GetFakt(csLVName);
       ASSERT(pThisFact);
             switch (theMethod)
             {
                    case RM_Approximate:
            dCompMeasure = pThisFact->GetCompMeasure(*theAnte);
            theTmpFS =theCons->GetTerm()->GetFuzzySet() ;
            pInter = theTmpFS->Operation(tNorm,dCompMeasure);
                           break;
                    case RM_Plausible:
                           pInter = GeneralizedModusPonens(tNorm,sNorm,m_opImplication,
                                  *theAnte, // if a
                                  *theCons, // then b
                                  *pThisFact // x = a'
                                 );
                           break;
                 default:
```

```
                        ASSERT(1==2);
                    break;
      }
      if (i==0)
      {
        ptheResult = pInter;
      }
      else
      {
          CFuzzySet* pZwischen = ptheResult->Operation(opAggregation, *pInter);
          delete ptheResult;
        ptheResult = pZwischen;
        delete pInter;
        //delete pZwischen;
      }
   }
        ptheResult->FuzzyOperation(m_tCertainty,m_Certainty);
   pResultFact = CFakt(theLingVar->GetName(),theLingVar->GetName(),ptheResult);
   delete ptheResult;
        theReturnValues->AddFact(pResultFact);
  };
  return theReturnValues;
};

int CRulebase::GetNumLayers(void) const
{
  int maxlayer = 0;
  CRule* theRule = NULL;
  POSITION pos;

  for (pos =  m_Regeln->GetHeadPosition(); pos != NULL;)
    {
      theRule = (CRule*)m_Regeln->GetNext(pos);
      int checklayer = theRule->GetInferLevel();
      if (checklayer>maxlayer)
            maxlayer=checklayer;
    };
  return maxlayer;
};

CObList* CRulebase::GetRulesInLayer(int iLayer) const
{
          POSITION pos;
          CRule* theRule = NULL;

          CObList* theRules = new CObList;
          for (pos =  m_Regeln->GetHeadPosition(); pos != NULL;)
          {
                    theRule = (CRule*)m_Regeln->GetNext(pos);
                    int checklayer = theRule->GetInferLevel();
                    if (checklayer == iLayer)
                    {
                              theRules->AddTail(new CRule(*theRule));
                    }
          };
          return theRules;
};

int CRulebase::GetNumRules(void) const
{
          ASSERT(m_Regeln);
          return m_Regeln->GetCount();
};

CRulebase::CRulebase(ReasoningMethod ReasMet,FuzzyOp AggOp, double AggParam)
{
  opAggregation                         = AggOp;
  m_iInferMethod              = ReasMet;
  m_dAggregationParam         = AggParam;
  m_Regeln                                        = new CObList();
};

CFactBase* CRulebase::CombineResults(const CFactBase& theFB1,const CFactBase& theFB2,FuzzyOp
theOp,double theOpParm) const
{
    CFactBase* theFacts  = new CFactBase();
    CFakt* theFactouter;
    CFakt* theFactinner;
    CFakt theNewFuzzyFact;
    CFuzzySet* tmpFS1;
    CFuzzySet* tmpFS2;
    CFuzzySet* result;
    CString key;
    CFakt* pTheFakt;
    POSITION pos1,pos2;

    for (pos1 = theFB1.m_theFacts.GetStartPosition(); pos1 != NULL;)
    {
        theFB1.m_theFacts.GetNextAssoc(pos1,key,(CObject*&) /*pa*/ pTheFakt);
                    theFacts->AddFact(pTheFakt);

    };
    for (pos2 = theFB2.m_theFacts.GetStartPosition(); pos2 != NULL;)
    {
        theFB2.m_theFacts.GetNextAssoc(pos2,key,(CObject*&) theFactouter);
        if (theFacts->m_theFacts.Lookup(key,(CObject*&) theFactinner))
        {
            tmpFS1 = theFactouter->GetFuzzySet();
            tmpFS2 = theFactinner->GetFuzzySet();
            result = tmpFS1->Operation(theOp, *tmpFS2);
            theNewFuzzyFact = CFakt(theFactinner->GibLingVarTypName(),"",result);
                                theFacts->AddFact(theNewFuzzyFact);
                                delete result;
            delete tmpFS1;
            delete tmpFS2;
        }
        else
        {
            result = theFactouter->GetFuzzySet();
            CFakt theFuzzyFact = CFakt(theFactouter->GibLingVarTypName(),"",result);
                                delete result;
                                theFacts->AddFact(theFuzzyFact);

        };
    };
  return theFacts;

}

FuzzyOp CRulebase::GetOpAggregation(void) const
{
         return opAggregation;
} ;
```

```
void  CRulebase::SetOpAggregation(FuzzyOp thesNorm)
{
         opAggregation = thesNorm;
} ;

double CRulebase::GetParamAggregation(void) const
{
         return m_dAggregationParam;
} ;

void  CRulebase::SetParamAggregation(double theParam)
{
         m_dAggregationParam = theParam;
} ;


CFactBase* CRulebase::Infer(const CFactBase& theFactBase) const
{

 POSITION pos;
 CObList* pLayerRules;

 CRule* theRule;
 CFactBase* pIntermediateResults;
 CFactBase* pEndResults = new CFactBase(theFactBase);
 CFactBase* pZwischen   = NULL;
 CFactBase* pCombine    = NULL;
 CHelperFunctions hf;

 CString key;
 CFakt* theFakt;


 int r = GetNumLayers();
 for (int l = 0; l <= r; l++)
 {
    pLayerRules = GetRulesInLayer(l);
    for (pos =  pLayerRules->GetHeadPosition(); pos != NULL;)
    {
       theRule = (CRule*) pLayerRules->GetNext(pos); // die Regel
       pIntermediateResults = theRule->Infer(*pEndResults,m_iInferMethod,theRule-
>GettNorm(),theRule->GetsNorm(),theRule->GetImplication());
            pZwischen = pEndResults;
            pCombine = CombineResults(*pEndResults,*pIntermediateResults, opAggregation,
m_dAggregationParam);
       pEndResults = new CFactBase(*pCombine);
       delete pCombine;
       delete pZwischen;
       delete pIntermediateResults;
    };

 };

 for (pos = theFactBase.m_theFacts.GetStartPosition(); pos ; )
 {
         theFactBase.m_theFacts.GetNextAssoc(pos,key,(CObject*&)theFakt);

         pEndResults->CleanLV(key);
 }
 return pEndResults;
};
```

```
CRulebase::~CRulebase()
{
  POSITION pos;
  CRule* theRule;
  ASSERT(m_Regeln);
  for (pos =  m_Regeln->GetHeadPosition(); pos != NULL;)
  {
       theRule = (CRule*) m_Regeln->GetNext(pos); // die Regel
       delete theRule;
  };

  delete m_Regeln;
};

void CRulebase::AddRule(const CRule& theRule)
{
  ASSERT(m_Regeln);
  m_Regeln->AddTail(new CRule(theRule));
};

void CRulebase::AddRule(CRule* theRule)
{
  ASSERT(m_Regeln);
  m_Regeln->AddTail(theRule);
};

CMapStringToOb* CRulebase::GetResultLVs(void) const  // HASHVALUES = Namen der Lingvars
                                                     // POINTER    = Die LingVars selber
                                                     // Grundlage fuer die Ermittlung
                                                     // der Consequenzen
{
  CMapStringToOb* theList = new CMapStringToOb();
  CConsequent* theCon;
  POSITION posR;
  CString key;
  for (posR =  m_Regeln->GetHeadPosition(); posR != NULL;)
  {

     CRule* theRule = (CRule*)m_Regeln->GetNext(posR);
     CMapStringToOb* theCons = &theRule->m_Consequents;
     for (POSITION pos = theCons->GetStartPosition();pos !=NULL;)
     {
         theCons->GetNextAssoc(pos,key,(CObject*&) theCon);
         (*theList)[theCon->GetLV()->GetName()]= theCon->GetLV();
     }

  }
  return theList;
};

#ifdef _DEBUG
void CRulebase::Dump(CDumpContext& dc) const
{
  dc << "Rulebase" << "\n\r";
  dc << (int) opAggregation;
  dc << (int) m_iInferMethod;
  dc << m_dAggregationParam;
  dc << m_Regeln;
} ;

#endif
```

```
CString CRule::Dump(CString theName)
{
  return "RULE: "+theName + " \n";
} ;


CString CRulePart::Dump(CString theName)
{
  return "CAntecedent: "+theName + " \n";
} ;

CMapStringToOb* CRulebase::GetInputLVs() const  // HASHVALUES = Namen der Lingvars
                                                // POINTER    = Die LingVars selber
{
  CMapStringToOb* theList = new CMapStringToOb();
  CAntecedent* theAnte;
  POSITION posR;
  CString key;
  for (posR =  m_Regeln->GetHeadPosition(); posR != NULL;)
  {
     CRule* theRule = (CRule*)m_Regeln->GetNext(posR);
     CMapStringToOb* theAntes = &theRule->m_Antecedents;
     for (POSITION pos = theAntes->GetStartPosition();pos !=NULL;)
     {
         theAntes->GetNextAssoc(pos,key,(CObject*&) theAnte);
         (*theList)[theAnte->GetLV()->GetName()]= theAnte->GetLV();
     }

  }
  return theList;
};

////////////////////////////////////////////////////////////////////////
//  GENERALIZED MODUS PONENS
////////////////////////////////////////////////////////////////////////
//
// if x  = a then  y = b
//    x  = a' ==>  y = b'
////////////////////////////////////////////////////////////////////////

CFuzzySet* CRule::GeneralizedModusPonens(
               FuzzyOp tNorm,
               FuzzyOp sNorm,
               FuzzyImp fi_implikation,
               const CAntecedent& pA,    // if a
               const CConsequent& pB,    // then b
               const CFakt& pAstrich,    // wir haben aber a' und wollen b' zurueck ...
               double step) const
{

          CFuzzySet* a = pA.GetTerm()->GetFuzzySet();
          CFuzzySet* b = pB.GetTerm()->GetFuzzySet();
          CFuzzySet* retvalue = new CFuzzySet(TRUE );
          double i,j;
          double DoM;
          ASSERT(b->GetRMX() >= b->GetLMX());
          for (i = b->GetLMX(); i <= b->GetRMX(); i+=step )
          {
                   DoM =0.0;
                   ASSERT(a->GetRMX() >= a->GetLMX());
                   for (j = a->GetLMX(); j <= a->GetRMX(); j+=step )
                   {
                            DoM =     Compute_FuzzyOp(sNorm,DoM,
                                                      Compute_FuzzyOp(tNorm,pAstrich.GetDoM(j),
                                                      Compute_Implication(fi_implikation,a-
>GetDoM(j),b->GetDoM(i))));
                   }
                   retvalue->AppendPoint(i,DoM);
          }

          retvalue->Reduce();
          return retvalue;
};


int CRule::GetNumAntecedents() const
{
    return m_Antecedents.GetCount();
};

int CRule::GetNumConsequents() const
{
    return m_Consequents.GetCount();
};

BOOL CRule::Enable()
{
          BOOL temp = m_enabled;
          m_enabled = TRUE;
          return temp;
};

BOOL CRule::Disable()
{
          BOOL temp = m_enabled;
          m_enabled = FALSE;
          return temp;
};

void CRule::Serialize(CArchive& ar)
{
  long l1,l2,l3,l4,li;

  if (ar.IsStoring())
  {
          ar << m_Certainty;
          l1 = (int) m_enabled; ar << l1;
          ar << (long) m_iInferLevel;
          ar << (long) m_tNorm;
          ar << (long) m_sNorm;
          ar << (long) m_opImplication;
  }
  else
  {
          ar >> m_Certainty;
          ar >> l1 ; m_enabled = (BOOL) l1;
          ar >> li;  m_iInferLevel = (int) li;
          ar >> l2 ;  m_tNorm = (FuzzyOp) l2;
          ar >> l3 ;  m_sNorm = (FuzzyOp) l3;
          ar >> l4 ;  m_opImplication = (FuzzyImp) l4;
  };
  m_Antecedents.Serialize(ar);
```

```
  m_Consequents.Serialize(ar);
};


void CRulePart::Serialize(CArchive& ar)
{
          m_theVar->Serialize(ar);
          ar << m_csTermName;

          //m_theTerm->Serialize(ar);
};

void CRulebase::Serialize(CArchive& ar)
{
          long l1,l2;

          if (ar.IsStoring())
          {
                    l1 = (int) opAggregation;
                    l2 = (int) m_iInferMethod;
                    ar << l1 << l2 << m_dAggregationParam;
          }
          else
          {
                    ar >> l1 >> l2 >> m_dAggregationParam;
                    opAggregation  = (FuzzyOp) l1;
                    m_iInferMethod = (ReasoningMethod) l2;
          }
          m_Regeln->Serialize(ar);
};

void CRule::SetInferLevel(int theLevel)
{
          m_iInferLevel = theLevel;
};

int CRule::GetInferLevel(void) const
{
          return m_iInferLevel;
};

FuzzyOp CRule::GettNorm(void) const
{
          return m_tNorm;
};

FuzzyOp CRule::GettCertainty(void) const
{
          return m_tCertainty;
};

FuzzyOp CRule::GetsNorm(void) const
{
          return m_sNorm;
};

void CRule::SettNorm(FuzzyOp thesNorm)
{
          m_tNorm = thesNorm;
};

void CRule::SettCertainty(FuzzyOp thesNorm)
{
          m_tCertainty = thesNorm;
};

void CRule::SetsNorm(FuzzyOp thesNorm)
{
          m_sNorm = thesNorm;
};

void CRule::SetopAggregation(FuzzyOp thesNorm)
{
          opAggregation = thesNorm;
};

void CRule::SetopImplication(FuzzyImp theImp)
{
          m_opImplication = theImp;
};

FuzzyImp CRule::GetImplication(void) const
{
          return m_opImplication;
};

FuzzyOp CRule::GetopAggregation(void) const
{
          return opAggregation;
};

EinfachTyp CRulebase::GibEinfachTyp(void) const
{
          return ET_RegelBasis;
};

////////////////////////////////////////////////////////////
// Funktion : GibEinfachTyp
// Klasse   : CRulebase
////////////////////////////////////////////////////////////
// Gibt zu einem CRulebase einen entsprechenden CTyp als
// CEinfachTyp
////////////////////////////////////////////////////////////

CTyp* CRulebase::GibTyp(void) const
{
          CEinfachTyp* result;
          result = new CEinfachTyp(ET_RegelBasis);
          ASSERT (result);
          return result;
}


CString CRulebase::Display(void) // Hallo Bernhard, sollte eigentlich const sein !?!
{
          CString Return = "Regelbasis";
          return Return;
};
```

FCOP.H

```
#ifndef _fuzzyop_h_
#define _fuzzyop_h_
typedef enum {RM_Invalid = 0,
                              RM_Approximate,
                              RM_Plausible} ReasoningMethod;

typedef enum {Fo_Invalid = 0,
                              Fo_t_Min,
              Fo_s_Max,
              Fo_t_DrastProd,
              Fo_s_DrastSum,
              Fo_t_BoundDiff,
              Fo_s_BoundSum,
              Fo_t_EinstProd,
              Fo_s_EinstSum,
              Fo_t_AlgebProd,
              Fo_s_AlgebSum,
              Fo_HamachProd,
              Fo_t_p_HamachDurchschn,
              Fo_s_p_HamachVerein,
              Fo_t_YagerDurchschn,
              Fo_s_YagerVerein,
              Fo_KompensatUnd,
              Fo_t_p_WernersUnd,
              Fo_s_p_WernersOder} FuzzyOp;

// Implikationen (für gmp !!!)

typedef enum {Fi_Invalid = 0,
                              Fi_zadeh ,
              Fi_lukasiewicz,
              Fi_mamdani,
              Fi_Standard_Strict,
              Fi_goedel,
              Fi_reichenbach,
              Fi_larsen,
              Fi_goguen,
              Fi_Standard_Strict_Star,   // Die naechsten 4  : Tilli
              Fi_Standard_Star_Strict,
              Fi_Standard_Star_Star,
              Fi_Standard_Strict_Strict,
              Fi_kleene_dienes,
              Fi_gaines,
              Fi_modified_gaines,
              Fi_Kle_de_Luk,
              Fi_willmott,
              Fi_Standard_Sharp,
              Fi_Wu1,
              Fi_Wu2}FuzzyImp;

// unaere operatoren

typedef enum {Fu_identity = 1,
              Fu_sehr,
              Fu_sehrsehr,
              Fu_etwas,
              Fu_bisschen,
              Fu_nicht,
              Fu_nichtsehr}FuzzyUO;

typedef enum {FS_original,
              FS_derived} FACTSOURCE;

#endif
```

FCOP.CPP

```
#include "stdafx.h"
#include "fc.h"
#include <math.h>
// Drastisches Produkt
// Input :  double a,b Zugehörigkeitsgrade
// Output : double dra Drastisches Produkt

double drp (double a , double b)
{
 if (a==1) {return b;}
 else if (b==1) {return a;}
 else {return 0;}
}

double drs (double a , double b)
{
 if (a==0) {return b;}
 else if (b==0) {return a;}
 else {return 1;}
}

// Algebraisches Produkt  (t-Norm)
// Input :  double a,b Zugehörigkeitsgrade
// Output : double dra Drastisches Produkt

double agp (double a, double b)
{
  return (a*b);
};

// Algebraische Summe  (s-Norm)
// Input :  double a,b Zugehörigkeitsgrade
// Output : double dra Drastisches Produkt

double ags (double a, double b)
{
  return (a + b - a*b) ;
};

// Beschränkte Differenz (t-Norm)
// Input :  double a,b Zugehörigkeitsgrade
// Output : double bdd Beschränkte Differenz

double bdd (double a, double b)
{
  return ( __max(0, a + b -1));
};

// Beschränkte Summe  (s-Norm)
// Input :  double a,b Zugehörigkeitsgrade
// Output : double bds Beschränkte Summe

double bds (double a, double b)
{
```

```
  return ( __max(1, a + b));
};

double identitaet (double a) {return a;};
// Konzentrationen
double sehr (double a) {return pow(a,2);};
double sehrsehr (double a) {return pow(a,4);};
// Dilationen
double etwas (double a){return pow(a,1/2);};
double bisschen (double a){return pow(a,1/4);};
// Negationen
double nicht (double a) {return 1-a;};
double nichtsehr (double a) {return 1-sehr(a);};

double imp_zadeh(double a,double b) // boehme s. 266 auch willmott genannt
{
         return __max(__min(a,b),1-a);
};

double imp_kleene_dienes (double a,double b) // boehme s. 266
{
         return __max(1-a,b);
};

double imp_mamdani (double a,double b)           // boehme s. 266
{
         return __min(a,b);
};

double imp_goguen (double a,double b)
{
         return (a==0) ? 1 : __min(1,a/b);};     // boehme s. 266

double imp_lukasiewicz (double a,double b)
{
         return __min(1,1-a+b);
};  // boehme s. 266

double imp_reichenbach (double a,double b)
{
         return 1-a+a*b;
};     // boehme s. 266

double imp_gaines_rescher (double a,double b) // auch standard-strict-star
{
         return a <= b ? 1 : 0 ;
};     // boehme s. 266

double imp_goedel (double a,double b)
{
         return a <= b ? 1 : b ;
}; // boehme s. 266

double imp_larsen (double a,double b)
{
         return a * b ;
};          // boehme s. 266


double Compute_Implication(FuzzyImp theImp, double a, double b)
{
         switch (theImp)
         {
                  case Fi_zadeh:                          return imp_zadeh(a,b);
         break;
                  case Fi_mamdani:             return imp_mamdani(a,b);
         break;
                  case Fi_goedel:                         return imp_goedel(a,b);
         break;
                  case Fi_gaines:                         return imp_gaines_rescher(a,b);
break;
                  case Fi_kleene_dienes:   return imp_kleene_dienes(a,b); break;
                  case Fi_reichenbach:     return imp_reichenbach(a,b);   break;
                  case Fi_lukasiewicz:     return imp_lukasiewicz(a,b);   break;
                  case Fi_larsen:          return imp_larsen(a,b);        break;
                  case Fi_goguen:          return imp_goguen(a,b);        break;
                  default:
                  {
                           ASSERT(1==0);
                           return 0.0;
                  }
         }
};

double Compute_Unary(FuzzyUO theUnary, double a)
{
  switch (theUnary)
  {
     case Fu_identity:           return identitaet(a);
     case Fu_sehr:               return sehr(a);
     default: ASSERT(1==0); return 0.0;
  }
};

double Compute_FuzzyOp(FuzzyOp theOp, double a,double b,double gamma)
{
  switch (theOp)
  {
     case Fo_t_Min : return __min(a,b);
     case Fo_s_Max : return __max(a,b);
     case Fo_t_DrastProd : return drp(a,b);
     case Fo_s_DrastSum  : return drs(a,b);
     case Fo_t_BoundDiff : return bdd(a,b);
     case Fo_s_BoundSum  : return bds(a,b);
     case Fo_t_AlgebProd : return agp(a,b);
     case Fo_s_AlgebSum  : return ags(a,b);

     default: ASSERT(1==0); return 0.0;
  }
};
```

FC.H

```
#ifndef _fuzzy_h
#define _fuzzy_h

#include "fcop.h"
#include "dope.h"
//#include <stdio.h>
//#include <iostream.h>
//#include "afx.h"
#include <afxcoll.h>

#define _INDREM_  ' '

class CAntecedent;
class CConsequent;
class CFact;
class CFuzzySet;
class CFuzzySetEntry;
class CLingTerm;
class CLingVar;

//#define ALMOSTZERO 1e-9

// T und S-Normen

/*typedef enum {RM_Invalid = 0,
                              RM_Approximate,
                              RM_Plausible} ReasoningMethod;

typedef enum {Fo_Invalid = 0,
                              Fo_t_Min,
              Fo_s_Max,
              Fo_t_DrastProd,
              Fo_s_DrastSum,
              Fo_t_BoundDiff,
              Fo_s_BoundSum,
              Fo_t_EinstProd,
              Fo_s_EinstSum,
              Fo_t_AlgebProd,
              Fo_s_AlgebSum,
              Fo_HamachProd,
              Fo_t_p_HamachDurchschn,
              Fo_s_p_HamachVerein,
              Fo_t_YagerDurchschn,
              Fo_s_YagerVerein,
              Fo_KompensatUnd,
              Fo_t_p_WernersUnd,
              Fo_s_p_WernersOder} FuzzyOp;

// Implikationen (für gmp !!!)

typedef enum {Fi_Invalid = 0,
                              Fi_zadeh ,
              Fi_lukasiewicz,
              Fi_mamdani,
              Fi_Standard_Strict,
              Fi_goedel,
              Fi_reichenbach,
              Fi_larsen,
              Fi_goguen,
              Fi_Standard_Strict_Star,   // Die naechsten 4  : Tilli
              Fi_Standard_Star_Strict,
              Fi_Standard_Star_Star,
              Fi_Standard_Strict_Strict,
              Fi_kleene_dienes,
              Fi_gaines,
              Fi_modified_gaines,
              Fi_Kle_de_Luk,
              Fi_willmott,
              Fi_Standard_Sharp,
              Fi_Wu1,
              Fi_Wu2}FuzzyImp;

// unaere operatoren

typedef enum {Fu_identity = 1,
              Fu_sehr,
              Fu_sehrsehr,
              Fu_etwas,
              Fu_bisschen,
              Fu_nicht,
              Fu_nichtsehr}FuzzyUO;

typedef enum {FS_original,
              FS_derived} FACTSOURCE;         */

static const char *FuzzyOpName[] = {
    "MINIMUM",
    "MAXIMUM",
    "DRASTIC PROD",
    "DRASTIC SUM",
    "BOUNDED DIFF",
    "BOUNDED SUM",
    "EINSTEIN PROD",
    "EINSTEIN SUM",
    "ALGEBRAIC PROD",
    "ALGEBRAIC SUM",
    "HAMACHER PROD",
    "HAMACHER INTERSEC",
    "HAMACHER UNION",
    "YAGER INTERSEC",
    "YAGER UNION",
    "KOMPENS AND",
    "WERNERS INTERSEC",
    "WERNERS UNION"                 }          ;

static const char *FuzzyImpName[] = {
    "ZADEH",
    "Lukasiewicz",
    "Mamdani",
    "Standard_Strict",
    "Goedel",
    "Reichenbach",
    "Larsen",
    "Goguen",
    "Standard_Strict_Star",   // Die naechsten 4  : Tilli
    "Standard_Star_Strict",
    "Standard_Star_Star",
    "Standard_Strict_Strict",
    "Kleene_dienes",
    "Gaines",
    "Modified_gaines",
    "Kle_de_Luk",
```

```
    "Willmott",
    "Standard_Sharp",
    "Wu1",
    "Wu2"
    };


typedef enum {DEFo_LeftMax=1,
              DEFo_RightMax,
              DEFo_MidMax,
              DEFo_Centroid,
              DEFo_WeightedCentroid,
              DEFo_LevelCentroid} DeFuzzOp;

typedef enum {FACT = 0,FACTFUZZY = 1 ,FACTCRISP} FacTType;

double FuzzyOperation( double left, double right,  FuzzyOp Operation,
                       double param=1.0);

// zweistellige Operatoren

double drp (double a, double b) ;  // drastisches Produkt
double drs (double a, double b) ;  // drastische  Summe
double agp (double a, double b) ;  // Algebraisches Produkt
double ags (double a, double b) ;  // Algebraisches Summe
double bdd (double a, double b) ;  // Beschraenkte Differenz
double bds (double a, double b) ;  // Beschraenkte Summe

double Compute_FuzzyOp(FuzzyOp theOp, double a,double b,double gamma = 0.0);

// Einstellige Operatoren // Boehme S.31

double identitaet (double a);
// Konzentrationen
double sehr (double a);
double sehrsehr (double a);
// Dilationen
double etwas (double a);
double bisschen (double a);
// Negationen
double nicht (double a);
double nichtsehr (double a);

double Compute_Unary(FuzzyImp theImp, double a);


// Implikationsoperatoren

double imp_zadeh(double a,double b);   // boehme s. 266 auch willmott genannt
double imp_kleene_dienes (double a,double b);          // boehme s. 266
double imp_rescher (double a,double b);    // bouchon-meunier and yao s.30
double imp_mamdani (double a,double b);    // boehme s. 266
double imp_Brouwer_goedel (double a,double b);  // boehme s. 266
double imp_goguen (double a,double b);     // boehme s. 266
double imp_mamdani (double a,double b);
double imp_lukasiewicz (double a,double b);  // boehme s. 266
double imp_reichenbach (double a,double b); // boehme s. 266

double Compute_Implication(FuzzyImp theImp, double a, double b);

#ifndef _INDREM_

ostream& operator << (ostream& os, const BOOL &me);

#endif

class CInterval;
class CIntervalList;

class CFactBase;

class CAntecedent;
class CConsequent;

// CRULE
//
// R E G E L B A S I S
//
class CRule;
class CRulebase;

// Dieses Objekt ist im DREM - System jeweils einmal vorhanden ...

static char csStepWidth[] = "StepWidth";
static char csStepNum[] = "StepNum";
static char csStatus[] = "Fuzzy-Status";

class CFuzzyManager : public CObject
{
         protected:
         double dGranularity;
         int iNumSteps;
         public:
         CFuzzyManager(void);
         ~CFuzzyManager(void);
         double GetGranularity(void) const;
         int GetNumSteps(void) const ;
         void SetGranularity(double dtheGran);
         void SetNumSteps(int itheNum);
         void ReadINIValues(void);
         void WriteINIValues(void) const;
         CFuzzyManager* GetFuzzyManager(void) const;
};


#endif
```

FCMAN.CPP

```
// Implementation FCMAN.CPP

// Systemdaten
#include "stdafx.h"

// Fuzzy-Definitionen
#include "fc.h"

CFuzzyManager ::CFuzzyManager(void)
{
         dGranularity = 0.01;
         iNumSteps    = 200;
         ReadINIValues();
};

CFuzzyManager ::~CFuzzyManager(void)
{
         WriteINIValues();
};

double CFuzzyManager ::GetGranularity(void) const
{
         return dGranularity;
};

int CFuzzyManager ::GetNumSteps(void) const
{
         return iNumSteps;
};


void CFuzzyManager ::SetGranularity(double dtheGran)
{
         dGranularity = dtheGran;
};

void CFuzzyManager ::SetNumSteps(int itheNum)
{
         iNumSteps = itheNum;
};


void CFuzzyManager ::ReadINIValues(void)
{
         char buf[20];
         sprintf(buf,"%f",dGranularity);
         double dTmpGran = atof(AfxGetApp()->GetProfileString(csStatus,csStepWidth,buf));
         if (dTmpGran != 0.0)
         {
                  dGranularity = dTmpGran;
         };

         sprintf(buf,"%i",iNumSteps);
         int iTmpSteps = atoi(AfxGetApp()->GetProfileString(csStatus,csStepNum,buf));
         if (iTmpSteps != 0)
         {
                  iNumSteps = iTmpSteps;
         };
};

void CFuzzyManager ::WriteINIValues(void) const
{
         char buf[20];
         sprintf(buf,"%f",dGranularity);
         AfxGetApp()->WriteProfileString(csStatus,csStepWidth,buf);
         sprintf(buf,"%i",iNumSteps);
         AfxGetApp()->WriteProfileString(csStatus,csStepNum,buf);
};

CFuzzyManager* CFuzzyManager ::GetFuzzyManager(void) const
{
         return ( ( (CDopeApp *) AfxGetApp() )->m_pFuzzyManager);
};
```

FCLV.H

```
#ifndef _FCLV_H
#define _FCLV_H
#include "fcset.h"
class CLingTerm:public CObject
{
  public:
         DECLARE_SERIAL( CLingTerm);

  protected:
           CFuzzySet* m_pTermMembership;
           CString m_csName;
  public:
           CLingTerm(CString theName,CFuzzySet* theSet);
           CLingTerm(const CLingTerm& theTerm);
           CLingTerm(void);
           CFuzzySet* GetFuzzySet(void) const ;
           CString GetName(void) const ;
           CLingTerm& operator = (CLingTerm& theTerm);

           virtual void Serialize(CArchive& ar);
           ~CLingTerm(void);

#ifdef _DEBUG
           void Dump(CDumpContext &dc) const;
#endif
};

class CLingVar:public CEinfachObjekt
{
public:
DECLARE_SERIAL( CLingVar);
protected:
CString m_csName;
CObArray m_pTerm; // of CLingTerm
CString m_csUnit;
CInterval m_pUoD;
public:
CLingVar(void);
CLingVar(const CString csTypName);
CLingVar(const CString Bez,const CInterval& theInt,const CString theMassEinheit);
CLingVar(CLingVar* ptheVar);
CLingVar(const CLingVar& ptheVar);

~CLingVar(void);

void AddTerm(CLingTerm* theTerm,BOOL bCopyIt = TRUE);
void SetzeMassEinheit(const CString theMassEinheit);
BOOL Edit(const CString csTypName);
void SetRange(const CInterval& theRange);
void SetRange(double von , double bis);
void DeleteTerm(const CString csTermName);
void DeleteTerm(int i);
void DeleteAllTerms(void);

CString GetName(void) const ;
CString GetUnit(void) const ;          // bs 07.07.96
CLingTerm* GetLingTerm(const CString& theName) const ;
CLingTerm* GetLingTerm(int i) const;
int operator==(const CLingVar& comp) const;
BOOL IstLeer(void) const ;
BOOL IsMemberOf(const CString theName) const ;
CInterval GetRange(void) const ;
int GetNumTerms(void) const  ;
int GetMaxIndex(void) const ;
double GetMinX(void) const ;
double GetMaxX(void) const ;
virtual void Serialize(CArchive& ar);
#ifdef _DEBUG
          void Dump(CDumpContext &dc) const;
#endif

};
#endif
```

FCLV.CPP

```
#include "stdafx.h"
#include "fc.h"
#include "fclv.h"
#include "fcintrvl.h"
#include "fcset.h"
#include "resource.h"
#include "em_lv.h"


IMPLEMENT_SERIAL( CLingTerm, CObject, 0)
IMPLEMENT_SERIAL( CLingVar, CEinfachObjekt, 0)

#define new DEBUG_NEW
// Der Feind heisst : Memory-Leak


CLingVar::CLingVar()
{
         m_csName = "<unbenannt>";
         m_csName.MakeUpper();
         m_csUnit = "";
         m_pUoD = CInterval(0,1);
};


CLingVar::CLingVar(const CString csTypName)
{
         m_csName = csTypName;
         m_csName.MakeUpper();
         m_csUnit = "";
         m_pUoD = CInterval(0,1);
};


CLingVar::CLingVar(CLingVar* ptheVar)
{
         m_csName  = ptheVar->m_csName;
         m_csName.MakeUpper();
         m_csUnit   = ptheVar->m_csUnit;
         m_pUoD = ptheVar->m_pUoD;
         for (int i=0;i< ptheVar->m_pTerm.GetSize();i++)
                  AddTerm((CLingTerm*)ptheVar->m_pTerm.GetAt(i));
};

CLingVar::CLingVar(const CLingVar& ptheVar)
```

```
{
          m_csName  = ptheVar.m_csName;
          m_csName.MakeUpper();
          m_csUnit   = ptheVar.m_csUnit;
          m_pUoD = ptheVar.m_pUoD;
          for (int i=0;i< ptheVar.m_pTerm.GetSize();i++)
                    AddTerm((CLingTerm*)ptheVar.m_pTerm.GetAt(i));
};


CLingVar::~CLingVar()
{
  for (int i=0;i< m_pTerm.GetSize();i++)
    delete (CLingTerm*) m_pTerm.GetAt(i);
};

CLingTerm* CLingVar::GetLingTerm(const CString& theName) const
{
          CString csTmp = theName;
          csTmp.MakeUpper();
          for (int i=0;i< m_pTerm.GetSize();i++)
          {
                    CString csCurTerm = ((CLingTerm*) m_pTerm.GetAt(i))->GetName();
                    if (csCurTerm == csTmp)
                    {
                              return (CLingTerm*)m_pTerm.GetAt(i);
                    }
          };
          return NULL;
}


CLingTerm* CLingVar::GetLingTerm(int i) const
{
          CLingTerm* result = NULL;
          if (i< m_pTerm.GetSize())
          {
                    result = (CLingTerm*)m_pTerm.GetAt(i);
          };
          return result;
}


void CLingVar::SetzeMassEinheit(const CString theMassEinheit)
{
    m_csUnit =theMassEinheit;
};

CLingVar::CLingVar(const CString Bez,const CInterval& theInt,const CString theMassEinheit)
{
    m_csName        = Bez;
          m_csUnit            = theMassEinheit;
    m_pUoD                    = CInterval(theInt);
};


void CLingVar::AddTerm(CLingTerm* theTerm,BOOL bCopyIt)
{
          if (bCopyIt)
                    m_pTerm.Add(new CLingTerm(*theTerm));
          else
                    m_pTerm.Add(theTerm);
};


// CLingTerm

CLingTerm::CLingTerm(CString theName,CFuzzySet* theSet)
{
    m_csName                            = theName;
    m_pTermMembership         = new CFuzzySet(theSet);
};

CLingTerm::CLingTerm()
{
    m_csName                            = "";
    m_pTermMembership         = NULL;
};


CLingTerm& CLingTerm::operator = (CLingTerm& theTerm)
{
    if (this == &theTerm) // bin ich's selbst ?
          return *this;
          delete m_pTermMembership;
    m_csName                            = theTerm.m_csName;
    m_pTermMembership         = theTerm.m_pTermMembership;
    return *this;
};

CLingTerm::CLingTerm(const CLingTerm& theTerm)
{
    m_csName = theTerm.m_csName;
    m_pTermMembership=new CFuzzySet(theTerm.m_pTermMembership);
};


CLingTerm::~CLingTerm()
{
  delete m_pTermMembership;
};

CFuzzySet* CLingTerm::GetFuzzySet(void) const
{
    return m_pTermMembership;
};

CString CLingTerm::GetName(void) const
{
          CString csTmp = m_csName;
          csTmp.MakeUpper();
    return csTmp;
};


#ifdef _DEBUG

void CLingTerm::Dump(CDumpContext &dc) const
{
          dc << "Name : " << m_csName << "\n"
                    << "Membership: " << m_pTermMembership << "\n";
}
#endif
```

```
#ifdef _DEBUG
void CLingVar::Dump(CDumpContext &dc) const
{
    dc << m_csName ;
          dc << m_pTerm  ;
          dc << m_csUnit ;
          dc << m_pUoD   ;
};
#endif



BOOL CLingVar::Edit(const CString Typname) //#BF 20.2.96
{
          int ub;
          int i;
          CEMLingVar* EditFenster = new CEMLingVar();
          EditFenster->m_lv = new CLingVar(*this);
          EditFenster->m_editierbar = 1;
          EditFenster->m_Name = Typname;
          BOOL result = FALSE;
          //EingabeFenster aufrufen
          if (EditFenster->DoModal()==IDOK)
          {
                    //neuen Werte eintragen
                    m_csName = EditFenster->m_lv->GetName();
                    m_pUoD = EditFenster->m_lv->m_pUoD;

                    CLingTerm *elv1,
                                                  *elv2;

        DeleteAllTerms();

                    ub= EditFenster->m_lv->GetMaxIndex();

                    for (i=0; i <= ub; ++i)
                    {

                              elv1 = EditFenster->m_lv->GetLingTerm(i);
                              elv2 = new CLingTerm(*elv1);

                              ASSERT (elv2);
                              AddTerm(elv2,FALSE);
                    }

                    result = TRUE;
          }
          delete EditFenster;
    return result;
}

  int CLingVar::operator==(const CLingVar& comp) const
  {
          ASSERT(FALSE);
    return (int)TRUE;
  };

  int operator==(const CLingVar& f1,const CLingVar& f2)
  {
          ASSERT(FALSE);
    return (int)TRUE;
  };

  BOOL CLingVar::IstLeer() const
  {
   return (m_pTerm.GetSize() == 0);
  };

  BOOL CLingVar::IsMemberOf(const CString theName) const
  {
   ASSERT(FALSE);
   return TRUE;
  };

  void CLingVar::SetRange(const CInterval& theRange)
  {
    m_pUoD = theRange;
  };

  void CLingVar::SetRange(double von , double bis)
  {
    CInterval theRange(von,bis);
    SetRange(theRange);
  };

void CLingVar::DeleteTerm(const CString csTermName)
{
          CLingTerm* ptheTerm;
          CString csSearchName = csTermName;
          csSearchName.MakeUpper();
          for (int i = 0; i <= GetMaxIndex(); i++)
          {
                    ptheTerm = (CLingTerm*)m_pTerm.GetAt(i);
                    if (csSearchName == ptheTerm->GetName())
                    {
                              DeleteTerm(i);
                              break;
                    }
          }
};

void CLingVar::DeleteAllTerms()
{
          for (int i = GetMaxIndex() ; i>=0 ; i--)
          {
                    DeleteTerm(i);
          }

}

void CLingVar::DeleteTerm(int i)
{
          delete (CLingTerm*) m_pTerm.GetAt(i);
          m_pTerm.RemoveAt(i);
};

int CLingVar::GetNumTerms(void) const
{
          return m_pTerm.GetSize();
};

  int CLingVar::GetMaxIndex(void)   const
  {
          return m_pTerm.GetUpperBound();
```

```
  };

  double CLingVar::GetMinX(void) const
  {
   return m_pUoD.GetMinX();
  };

  double CLingVar::GetMaxX(void) const
  {
  return m_pUoD.GetMaxX();
  };

CInterval CLingVar::GetRange(void) const
{
  return m_pUoD;
} ;

CString CLingVar::GetName(void) const
{
          return m_csName;
};

CString CLingVar::GetUnit(void) const            // bs 07.07.96
{
          return m_csUnit;
};

void CLingVar::Serialize(CArchive& ar)
{

          CEinfachObjekt::Serialize(ar);

          if( ar.IsStoring() )
          {
              ar << m_csName;
                    ar << m_csUnit;
          }
          else
          {
                    ar >> m_csName;
                    ar >> m_csUnit;
          }
          m_pUoD.Serialize(ar);
          m_pTerm.Serialize(ar);

};


void CLingTerm::Serialize(CArchive& ar)
{

    CObject::Serialize(ar);

    if( ar.IsStoring() )
          {
                    ar << m_csName;
          }
          else
          {
                    ar >> m_csName;
                    delete m_pTermMembership;
                    m_pTermMembership = new CFuzzySet;
          }
      m_pTermMembership->Serialize(ar);

};
```

FCINTRVL.H

```
#ifndef _FCINTRVL_H
#define _FCINTRVL_H

// für den InOutManager:
class CInOutConv;   // bs 27.05.
class CMyStringList;

class CInterval:public CObject
{
  public:
         DECLARE_SERIAL( CInterval );
  protected:
     double minx,maxx;
  public:
      CInterval();
      CInterval(double myx1,double myx2);
      CInterval(const CInterval& tR1);
      CInterval& operator = ( const CInterval& me) ;
      int operator == ( const CInterval& me) const ;
      void SetMinX(double myx);
      void SetMaxX(double myx);
      double GetMinX() const ;
      double GetMaxX() const ;
      double GetSpan() const ;
      double GetMean() const ;
      CInterval& operator + ( const CInterval& fs);

          // für den InOutManager:
          virtual void                          Read( CInOutConv* pConv, BYTE *bError );
          // in inoutrw.cpp
          virtual CMyStringList*       Write( CInOutConv* pConv );
                    // bs 27.05.
          virtual void Serialize(CArchive& ar);

#ifdef _DEBUG
          virtual void Dump(CDumpContext& dc) const;
#endif
};

#endif // _FCINTRVL_H
```

FCINTRVL.CPP

```
// CInterval - Implementation


#include "stdafx.h"
#include "fcintrvl.h"

CInterval& CInterval::operator = ( const CInterval& me)
{
    minx = me.minx;
    maxx = me.maxx;
    return *this;
};

int CInterval::operator == ( const CInterval& me)  const
{
    return (minx ==  me.minx) && (maxx == me.maxx) ;
};

CInterval::CInterval()
{
    minx = 0;
    maxx = 0;
};

CInterval::CInterval(double myx1,double myx2)
{
    minx = myx1;
    maxx = myx2;
};


CInterval::CInterval(const CInterval& tR1)
{
      minx = tR1.minx;
      maxx = tR1.maxx;
};


void CInterval::SetMinX(double myx)
{
    minx = myx;
};

void CInterval::SetMaxX(double myx)
{
    maxx = myx;
};

double CInterval::GetMinX()  const
{
    return minx ;
};

double CInterval::GetMaxX()  const
{
    return maxx ;
};

double CInterval::GetSpan() const
{
    return maxx-minx;
};

double CInterval::GetMean() const
{
    return (maxx+minx) / 2.0 ;
};

CInterval& CInterval::operator + ( const CInterval& fs)
{
    CInterval retvalue = *this;
    retvalue.minx += fs.minx;
    retvalue.maxx += fs.maxx;
    return retvalue;
};
```

```
IMPLEMENT_SERIAL( CInterval, CObject, 0)


void CInterval::Serialize(CArchive& ar)
{
          if( ar.IsStoring() )
          {
                    ar << minx ;
                    ar << maxx;
          }
          else
          {
                    ar >> minx;
                    ar >> maxx;
          }
};


#ifdef _DEBUG

void CInterval::Dump(CDumpContext &dc) const
{
          dc << "[" <<minx << "-" << maxx << "]" ;
};

#endif
```

FCHELP.H

```
//#include "stdafx.h"


struct point
{
         double x;
         double y;
//       char c;
};


class CHelperFunctions : public  CObject
{
  public:
  BOOL IsAlmostZero(double a) const ;
  CString Double2CString(double a) const ;
  double theta (point p1,point p2) const;
  double length (struct point p1,struct point p2) const;
  int ccw (point p0, point p1, point p2) const;
  void swap (point p[], int posa, int posb) const;
  int wrap(point p[], int NN) const;
};
```

FCHELP.CPP

```
#include "stdafx.h"
#include "fchelp.h"
#include <math.h>

BOOL CHelperFunctions::IsAlmostZero(double a) const
{
  return a < 1e-10;
};

CString CHelperFunctions::Double2CString(double a) const
{
  char buf[30];
  sprintf(buf,"%f",a);
  return CString(buf);
};



// PRIVATE Hilfefunktionen und -Strukturen für die konvexe Huelle


double CHelperFunctions::theta (point p1,point p2) const
{
         double dx,dy,ax,ay;
         double t;
         dx = p2.x - p1.x ; ax = fabs(dx);
         dy = p2.y - p1.y ; ay = fabs(dy);
         t = ( ax + ay == 0.0) ? 0.0 : (double) dy / (ax + ay);
         if (dx < 0) t = 2 - t; else if (dy < 0) t = t + 4;
         return t * 90.0;
};
```

```
int CHelperFunctions::ccw (point p0, point p1, point p2) const
{
         double dx1,dx2,dy1,dy2;

         dx1 = p1.x - p0.x; dy1 = p1.y - p0.y;
         dx2 = p2.x - p0.x; dy2 = p2.y - p0.y;
         if (dx1 * dy2 > dy1 * dx2) return +1;
         if (dx1 * dy2 < dy1 * dx2) return -1;
         if ((dx1 * dx2 < 0) ||(dy1 * dy2 < 0)) return -1;
         if ((dx1 * dx1 + dy1 * dy1) < (dx2 * dx2 + dy2 * dy2))     return +1;
         return 0;
};


void CHelperFunctions::swap (point p[], int posa, int posb) const
{
         point a = p[posa];
         p[posa] = p[posb];
         p[posb] = a;
};

double CHelperFunctions::length (struct point p1,struct point p2) const
{
         return (p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y);
};

int CHelperFunctions::wrap(point p[], int NN) const
{
         int i, min , M;

         int N = NN;

         double th,th1,v;
         double len,len1;

         point q[100];

         for (int k = 0; k < 2 ; k++)
         {

                  BOOL self   = FALSE;
                  BOOL tausch = FALSE;

                  point pCompare, pAkt;

                  for (min = 0, i = 1; i < N; i++)
                           if (p[i].y < p[min].y)
                                    min = i;

             p[N] = p[min];
             th = 0.0;

             for (M = 0 ; M < N ; M++)
             {
                   swap (p,M,min);
                   min = N; v = th; th = 360.0; len1 =0.0;
                   pAkt = p[M];
                   self = FALSE;
                   tausch = FALSE;
                   for (i = M+1 ; i < N ; i++)
                   {
                            pCompare = p[i];
```

```
                        th1 = theta(p[M],pCompare);
                        len = length(p[M],pCompare);
                        if ((th1 >= v) && (th1 <= th))
                                {
                                        if ((th1 > v) && (th1 < th))
                                        {
                                                min       = i;
                                                th        = th1;
                                                len1 = len;
                                                self = FALSE;
                                                tausch = TRUE;
                                        }
                                        else
                                        {
                                                if ((th1 == v) || (th1 == th))
                                                        if (len)
                                                        {
                                                                min
        = i;
                                                                th
        = th1;
                                                                len1
= len;
                                                                self
= FALSE;
        tausch = TRUE;
                                                        }
                                                        else
                                                        {
                                                                self = TRUE;
                                                        }
                                        }
                                }
                        }
                        if ((!tausch ) && (!self))
                        {
                                break;
                        }
                }


                int cc = 0;
                p[M+1]=p[0];
                q[0]=p[0];
                int j = 1;
                for (i = 1; i <= M ; i++)
                {
                        cc = ccw(p[i-1],p[i+1],p[i]);
                        if (cc != 0)
                        {
                                q[j++]=p[i];
                        }
                };
                j--;

                M = j+1 ;

                for (i = 0 ; i <= j ; i++)
                {
                        p[i]=q[i];
                };
                        N =M ;
        }


        for (i = 1; i <= (M-1)/2; i++)
        {
                        swap(p,i,M-i);
              };


        return M ;
};
```

FCFACTS.H

```
#ifndef _FCFACTS_H
#define _FCFACTS_H
#include "fc.h"

/////////////////////////////////////////////////////////////////////////////
// Klasse           CFakt
/////////////////////////////////////////////////////////////////////////////

class CFuzzySet;

class CFactBase:public CEinfachObjekt
{
public:
  DECLARE_SERIAL( CFactBase );
public:
  CMapStringToOb m_theFacts;
public:
  CFactBase(void);
  CFactBase( const CFactBase& theBase);
  ~CFactBase(void);
  void AddFact(CString LVName,CString Kette, CEinfachObjekt& Wert);
  void AddFact(CString LVName,CString Typbez, CRealzahl Wert);
  void AddFact(CString LVName,CString Typbez, CGanzzahl Wert);
  void AddFact(CString LVName,CString Typbez, CZeichenkette Wert);
  void AddFact(CString LVName,CString Typbez, CFuzzySet& Wert);
  void AddFact(const CFakt& Wert);
  void AddFact(CFakt* Wert);

  CFakt* GetFakt(const CString& csFactName) const;
  CFakt* GetFirstFakt(void) const;
  void CleanAll(void);
  void CleanLV(const CString& theLV);
  int GetNumFacts(void) const;
  CString GibText(void) const;
  CFactBase& operator = ( const CFactBase& fs);
  int operator == ( const CFactBase& theFSet) const ;
  virtual EinfachTyp GibEinfachTyp(void) const;
  virtual CTyp* GibTyp(void) const;
  // Für MFC-Archive
  virtual void Serialize(CArchive& ar);
#ifdef _DEBUG
         void Dump(CDumpContext &dc) const ;
#endif

};

#endif
```

FCFACTS.CPP

```
#include "stdafx.h"
#include "fc.h"
#include "fcset.h"
#include "fclv.h"
#include "fcrule.h"
#include "fcfacts.h"
#include "typframe.h"
#include "dope.h"
#include "em_fakt.h"
#include <math.h>


IMPLEMENT_SERIAL( CFactBase , CEinfachObjekt, 0)

#define new DEBUG_NEW

CFactBase::CFactBase(void)
{
         ;
};

CFactBase::CFactBase(const CFactBase& theBase)
{
         POSITION pos;
         CString key;
         CFakt *theFact,
               *theNewFact;

         for (pos = theBase.m_theFacts.GetStartPosition();pos !=NULL;)
         {
                  theBase.m_theFacts.GetNextAssoc(pos,key,(CObject*&) theFact);
                  theNewFact = new CFakt(*theFact);
                  m_theFacts.SetAt(key,theNewFact);
         }
};


CFactBase::~CFactBase(void)
{
         CFakt *theFact;
         POSITION pos;
         CString key;

         for (pos = m_theFacts.GetStartPosition();pos;)
         {
                  m_theFacts.GetNextAssoc(pos,key,(CObject*&) theFact);
                  delete theFact;
         }
};


void CFactBase::AddFact(const CFakt& theFact)
{
         CString csIndexName = theFact.GibLingVarTypName();
         m_theFacts[csIndexName] = new CFakt(theFact);
};

void CFactBase::AddFact(CFakt* theFact)
{
         AddFact(*theFact);
};

CFakt* CFactBase::GetFakt(const CString& csFactName) const
{
         CFakt* pLocFact;
         if (m_theFacts.Lookup(csFactName,(CObject*&) pLocFact))
                  return pLocFact;
         else
                  return NULL;
};
```

```
#pragma message("Helge : wir brauchen GET_FACT mit Stringuebergabe ")
#pragma message("Faktenbasisrueckgaben nur mit einem Fakt moeglich")


CFakt* CFactBase::GetFirstFakt(void) const
{
          POSITION pos;
          CFakt* pLocFact;
          CString key;
          for (pos = m_theFacts.GetStartPosition();pos !=NULL;)
          {
                    m_theFacts.GetNextAssoc(pos,key,(CObject*&) pLocFact);
                    return pLocFact;
          }
          return NULL;
};


CFactBase& CFactBase::operator = (const CFactBase& fs)
{
          POSITION pos;
          CString key;
          CFakt *theFact;

          for (pos = fs.m_theFacts.GetStartPosition();pos !=NULL;)
          {
                    fs.m_theFacts.GetNextAssoc(pos,key,(CObject*&) theFact);
                    AddFact(theFact);
          }
          return *this;
};

int CFactBase::operator == (const CFactBase& theFSet) const
{

          BOOL result =       TRUE ;
          POSITION            pos;
          CString             key;
          CFakt                         *theFact,
                                        *theCompFact;
          for (pos = theFSet.m_theFacts.GetStartPosition();pos;)
          {
                    m_theFacts.GetNextAssoc(pos,key,(CObject*&) theFact);
                    if (theFSet.m_theFacts.Lookup(key,(CObject*&) theCompFact))
                    {
                              result != (*theCompFact == *theFact);
                              if (!result)
                              {
                                        return FALSE;
                              }
                    }
                    else
                    {
                              return FALSE;
                    };

          }
          return result;
};

void CFactBase::CleanAll(void)
{
          POSITION pos;
          CString key;
          CFakt *theFact;
          for (pos = m_theFacts.GetStartPosition();pos !=NULL;)
          {
                    m_theFacts.GetNextAssoc(pos,key,(CObject*&) theFact);
                    m_theFacts.RemoveKey(key);
                    delete theFact;
          }
};


void CFactBase::CleanLV(const CString& theLV)
{
    CFakt* myFakt;
          if (m_theFacts.Lookup(theLV,(CObject*&) myFakt))
          {
                    delete myFakt;
          }
          m_theFacts.RemoveKey(theLV);
};

void CFactBase::AddFact(CString LVName,CString Kette, CEinfachObjekt& Wert)
{
   CFakt Fakt = CFakt(LVName,Kette, Wert);
   AddFact(Fakt);
};

void CFactBase::AddFact(CString LVName,CString Typbez, CRealzahl Wert)
{
   CFakt Fakt = CFakt(LVName,Typbez, Wert);
   AddFact(Fakt);
};

void CFactBase::AddFact(CString LVName,CString Typbez, CGanzzahl Wert)
{
   CFakt Fakt = CFakt(LVName,Typbez, Wert);
   AddFact(Fakt);
};

void CFactBase::AddFact(CString LVName,CString Typbez, CZeichenkette Wert)
{
   CFakt Fakt = CFakt(LVName,Typbez, Wert);
   AddFact(Fakt);
};


void CFactBase::AddFact(CString LVName,CString Typbez, CFuzzySet& Wert)
{
   CFakt Fakt = CFakt(LVName,Typbez, Wert);
   AddFact(Fakt);
};

int CFactBase::GetNumFacts(void) const
{
  return m_theFacts.GetCount();
};

CString CFactBase::GibText(void) const
{
  CString csBack    ="";
```

```
  POSITION pos;
  CString key;
  CFakt *theFact;


  for (pos = m_theFacts.GetStartPosition();pos !=NULL;)
  {
    m_theFacts.GetNextAssoc(pos,key,(CObject*&) theFact);

    CString csLVName = theFact->GibLingVarTypName();
    CString csValue  = theFact->Display();

    csBack = csBack + csLVName + " IS " + csValue + "\r\n";
  }
  return csBack;
};

EinfachTyp CFactBase::GibEinfachTyp(void) const
{
         return ET_FaktenBasis;
};

CTyp* CFactBase::GibTyp() const
{
         CEinfachTyp* result;
         result = new CEinfachTyp(ET_FaktenBasis);
         ASSERT (result);
         return result;
};

void CFactBase::Serialize(CArchive& ar)
{
         CEinfachObjekt::Serialize(ar);
         m_theFacts.Serialize(ar);
};


#ifdef _DEBUG

void CFactBase::Dump(CDumpContext &dc) const
{
//       m_theFacts.Dump(dc);
};

#endif
```

E Abschließende Erklärung

Ich versichere hiermit, daß ich meine Diplomhausarbeit „Entwurf und Realisierung einer auf Fuzzy-Logik basierenden Entscheidungskomponente zur Integration in eine Workflow-Entwicklungsumgebung“ selbständig und ohne fremde Hilfe angefertigt habe, und daß ich alle von anderen Autoren wörtlich übernommenen Stellen wie auch die sich an die Gedankengänge anderer Autoren eng anlehnenden Ausführungen meiner Arbeit besonders gekennzeichnet habe und die Quellen zitiert habe.

Münster, den 18. September 1996