Konsistenz in hybriden wissensbasierten Systemen

Vom Fachbereich Elektrotechnik der
Bergischen Universität – Gesamthochschule Wuppertal
genehmigte Dissertation zur Erlangung des akademischen Grades
einer Doktor-Ingenieurin

von
Diplom-Ingenieurin
Sabine Hagemann
aus
Wuppertal

Konsistenz in hybriden wissensbasierten Systemen

Vom Fachbereich Elektrotechnik der
Bergischen Universität – Gesamthochschule Wuppertal
genehmigte Dissertation zur Erlangung des akademischen Grades
einer Doktor-Ingenieurin

von
Diplom-Ingenieurin
Sabine Hagemann
aus
Wuppertal

Tag der mündlichen Prüfung: 03.12.1999

Referent:	Professor Dr.-Ing. J. Heidepriem
Korreferenten:	Professor Dr.rer.nat. K.-H. Becks
	Privatdozent Dr.-Ing. R. Möller

Danksagung

Mein besonderer Dank gilt meinem Doktorvater Herrn Prof. Dr. J. Heidepriem, der mir durch großen persönlichen Einsatz die Einhaltung aller relevanten Termine ermöglichte. Ebenfalls danke ich Herrn Prof. Dr. K.-H. Becks und Herrn Privatdozent Dr. R. Möller für die Übernahme des Korreferats.

Für sein persönliches Engagement und die Bereitschaft, für alle großen und kleinen Sorgen ein aufmerksamer Ansprechpartner zu sein danke ich Herrn Prof. E. Forner sehr herzlich. Er hat mein Bild von Forschung und Lehre entscheidend mitgeprägt.
Schließlich danke ich allen Frauen, die mich auf die eine oder andere Art unterstützt und bestärkt haben, sowie all den Männern, die bei der Realisierung der Software geholfen und die eine oder andere Katastrophe miterlebt haben.

Inhaltsverzeichnis

Einleitung

1 Repräsentation von Wissen **3**

1.1 Logikbasierte Repräsentation 5

1.1.1 Aussagenlogik 5

1.1.2 Prädikatenlogik 6

1.2 Regelbasierte Repräsentation 10

1.3 Strukturierte Objekte 14

1.3.1 Semantische Netze 14

1.3.2 Frames und objektorientierte Ansätze 15

1.4 Mathematische Modelle 18

1.5 Hybride Systeme 19

1.6 Zusammenfassung 21

2 Der Konsistenzbegriff **23**

2.1 Datenbanken 24

2.1.1 Konsistenzbedingungen 24

2.1.2 Überprüfung der Konsistenzbedingungen 26

2.1.3 Deduktive Datenbanken 27

2.2 Wissensbasierte Systeme 28

2.2.1 Konsistenz in klassischen Deduktionssystemen 29

2.2.2 Konsistenz in regelbasierten Systemen 30
2.3 Konsistenz in hybriden Systemen 34
2.3.1 Regeln 34
2.3.2 Frames 35
2.3.3 Syntaktischer Test 39
2.3.3.1 Die Connection Graph Proof Procedure . . . 45
2.3.4 Prozeduren 50
2.3.5 Funktionaler Test 53
2.3.6 Bewertung und Auswahl der Testkomponenten 58
2.4 Zusammenfassung 61

3 Realisierung **63**
3.1 Die Entwicklungsumgebung 65
3.2 Klassen- und Frameeditor 69
3.3 Regeleditor 77
3.4 Prozeduren 85
3.5 Funktionaler Test 89

4 Schlußbemerkungen **97**
4.1 Zusammenfassung 97
4.2 Übertragbarkeit der Ergebnisse 99
4.3 Ausblick 100
Literaturverzeichnis **101**

A Anhang **107**
A.1 Struktur der realisierten Module 107
A.2 Das Modul User-Root 108
A.3 Das Modul Konsistenz 109
A.4 Das Modul "syntaktischer Test“ 110
A.5 Das Modul "funktionaler Test“ 121

Abbildungsverzeichnis

1.1 Graph eines Suchbaumes 13

1.2 Beispiel eines semantischen Netzes 14

1.3 Graph einer Klassenhierarchie 16

1.4 Blockvorstellung eines hybriden wissensbasierten Systems 19

2.1 Konsistenzprüfung nach Beauvieux, nach [1] 31

2.2 Struktur der Ein- und Ausgangsframes 56

2.3 Struktur des erweiterten Ausgangsframes 57

3.1 Blockbild einer G2 Applikation mit Konsistenzmodul 64

3.2 Komponenten eines Expertensystems (nach [7]) 65

3.3 Modulhierarchie einer leeren Wissensbasis mit Konsistenztest 66

3.4 Ein (sinnloses) Regel-Objekt und seine Unterarbeitsfläche 68

3.5 Beispiele für Dialogelemente in G2 70

3.6 Startbild einer leeren Wissensbasis mit Konsistenzmodul 71

3.7 Dialog zur Erzeugung und Bearbeitung von Klassendefinitionen 72

3.8 Eingabe möglicher Werte 73

3.9 Entstehung eines Objektes 74

3.10 Menü einer Klassendefinition sowie Menü und Tabelle eines Objektes . . 75

3.11 Arbeitsflächen zur grafischen Erstellung einer Regel 78

3.12 Dialog zur Eingabe einer spezifischen Regel 79

3.13 Dialog für den Eingabe-Baustein "Wert" 80

3.14 Dialog zum Setzen eines Attributwertes 81
3.15 Arbeitsflächen zur grafischen Erstellung einer Prozedur 87
3.16 Start-Arbeitsblatt des funktionalen Tests 90
3.17 Eigenschaften spezieller "Ausgangs-Attribute" 92
3.18 Tabelle der Ausgangs-Attribute für Auto-1 und Auto-2 93

A.1 Modulhierarchie einer leeren Wissensbasis mit Konsistenztest 107
A.2 Arbeitsblatt beim Start der "leeren" Wissensbasis 108
A.3 Module für den Konsistenztest mit ihren Inhalten 109
A.4 Palette der möglichen Wissenselemente 110
A.5 Legende der Symbole in den folgenden Strukturen 111
A.6 Hierarchie der Menüs und Dialoge einer Klasse 112
A.7 Hierarchie der Menüs und Dialoge eines Objektes 113
A.8 Hierarchie der Menüs einer Regel 114
A.9 Dialoge der Regel-Bausteine 115
A.10 Aktionen der Schaltflächen in der Regel-Werkstatt 116
A.11 Hierarchie der Menüs und Dialoge einer Prozedur 117
A.12 Dialoge der Prozedur-Bausteine I 118
A.13 Dialoge der Prozedur-Bausteine II 119
A.14 Aktionen der Schaltflächen in der Prozedur-Werkstatt 120
A.15 Dialog eines Arbeitsblattes 120
A.16 Lösche / lade funktionalen Test 121
A.17 Ablauf des funktionalen Tests 123

Verwendete Formelzeichen

Elementare Aussagen (Atome, Primformeln)

A1 : Die Sonne scheint.

Junktoren

$\wedge$	konjunktive Verknüpfung (*und*)
$\vee$	disjunktive Verknüpfung (*oder*)
$\neg$	Negation (*nicht*)
$\Longrightarrow$	Implikation (*impliziert*)
$\Leftrightarrow$	Äquivalenz (*ist äquivalent zu*)

Formel in Aussagenlogik

F_A1

einstelliges Prädikat

P1 : studiert_Elektrotechnik(a)

Formel in Prädikatenlogik

F_P1 : $\forall$x studiert_Elektrotechnik(x) $\Longrightarrow$ hört_Mathematik(x)

Quantoren

$\forall$x oder $\forall_x$: für alle x

$\exists$x oder $\exists_x$: es existiert ein x

Einleitung

Computer übernehmen immer mehr Aufgaben am Arbeitsplatz und haben sich auch im privaten Umfeld etabliert. Das Fachgebiet "Computational Intelligence" oder "Künstliche Intelligenz" ist ein Teilgebiet der Informatik, das sich mit der Frage intelligenter Leistungen von Computerprogrammen beschäftigt. Mit verschiedenen Darstellungsarten für Wissen und dessen Verknüpfung kommt man zu sogenannten "wissensbasierten" Ansätzen oder Systemen, die in begrenzten Bereichen wie menschliche Experten reagieren.

Solche "Expertensystem-Anwendungen" versprachen in der Anfangszeit (in den Jahren 1980 – 1985) viele Vorteile auch für industrielle Aufgabenstellungen. Es zeigten sich jedoch rasch Schwächen bei der Einbindung wissensbasierter Systeme in ein bestehendes industrielles Umfeld.

Die vorliegende Arbeit beschäftigt sich mit der Frage nach der Widerspruchsfreiheit (Konsistenz) von wissensbasierten Programmsystemen (im folgenden kurz als Systeme bezeichnet). Um ein zuverlässiges Verhalten der Systeme zu gewährleisten, muß das verwendete Wissen widerspruchsfrei repräsentiert werden. Diese Forderung kann während der Entwicklung des Systems unter Umständen auch ohne Hilfsmittel relativ gut durch den Entwickler erfüllt werden, sofern nur **ein** Entwickler tätig ist und er fundiertes Wissen und Erfahrung im abgebildeten Themenbereich besitzt.
Sobald ein wissensbasiertes System aber über Jahre oder gar Jahrzehnte hinweg kompetent sein soll, ist Wartung in Form von Ergänzungen und Änderungen des implementierten Wissens notwendig. Der mit der Wartung beauftragte Entwickler steht bezüglich der zu gewährleistenden Widerspruchsfreiheit vor einem ungleich größeren Problem. Zum einen ist ihm die Gedankenwelt des "Erstentwicklers" nicht vertraut und zum anderen wird der Umfang der Wissensbasis stetig größer und somit unübersichtlicher.
Hier ist Hilfe in Form von algorithmischen Testverfahren für die Konsistenz der Wissensbasis unabdingbar. Damit stellen sich folgende Fragen:

- Gibt es allgemein akzeptierte Testverfahren, um Konsistenz eindeutig und algorithmisch nachzuweisen?

- Kann ein solches Verfahren für **alle** theoretischen Ansätze zur Wissensrepräsentation und alle auf dem Markt befindlichen Programmierwerkzeuge gelten?

Auf diese Fragen geht die vorliegende Arbeit ein und bietet insbesondere für wissensbasierte Systeme, die mehrere Repräsentationsformen für Wissen in sich vereinigen (hybride Systeme), Ansätze zu deren Lösung an.
Die Interpretation des Konsistenzbegriffes ist stark kontextabhängig, die Feststellung von Widersprüchen muß daher bei hybriden Systemen alle verwendeten Wissensrepräsentationsformen einbeziehen. Im Gegensatz z.B. zu rein prädikatenlogischen Systemen muß dabei auch das Zusammenwirken der Repräsentationsformen untersucht werden. Für ein vorgegebenes kommerzielles Werkzeug zur Entwicklung wissensbasierter Systeme wird in der vorliegenden Arbeit jede darin angebotene Wissensrepräsentationsform untersucht, eine Interpretation des Konsistenzbegriffes gegeben, Randbedingungen für einen Konsistenztest erarbeitet sowie die Auswirkung auf die anderen verwendeten Repräsentationsformen geklärt. Abschließend wird ein komplexes, mit dem Entwicklungssystem selbst realisiertes Testverfahren vorgestellt.

Daraus ergibt sich folgende Struktur der Arbeit:

Kapitel 1 stellt die gebräuchlichsten Formen der Wissensrepräsentation von formaler Logik über strukturierte Objekte und mathematische Modelle als Grundlage für das Verständnis der hybriden Systeme vor. Hybride wissensbasierte Systeme bieten eine Kombination von Repräsentationsformen mit einer gemeinsamen Verwaltungskomponente an und sind der Gegenstand der Untersuchungen dieser Arbeit.

In **Kapitel 2** werden die verschiedenen Definitionen des Begriffs der Konsistenz für Datenbanken, allgemein wissensbasierte Systeme sowie speziell für hybride Systeme untersucht. Aus der Analyse der Konsistenzprobleme werden Ansätze zum Test auf und zur Wahrung von Konsistenz entwickelt.

In **Kapitel 3** wird die Realisierung der entwickelten Ansätze mit einem kommerziellen Werkzeug, der Expertensystemshell G2 der Firma Gensym beschrieben.

Eine Wertung der Ansätze, Vorteile und Probleme theoretischer und praktischer Art, sowie ein Ausblick auf weitere Entwicklungen schließen in **Kapitel 4** die Arbeit ab.

Kapitel 1

Repräsentation von Wissen

Die Darstellung menschlichen Wissens in einer der automatischen Bearbeitung zugänglichen Form ist eines der zentralen Anliegen der Software-Entwicklung.

Die Abbildung menschlicher Vorgehensweisen bei der Lösung von numerischen Problemen erfolgt z.B. durch eine festgelegte Folge von "Anweisungen" an den Computer. Die Anweisungen können mit Hilfe imperativer Programmiersprachen abgebildete Operationen sein, die ausgeführt werden müssen, wenn die Problemlösung gefunden werden soll. Daten in Form von Zahlen dienen einem solchen "Algorithmus" als Eingabewerte bzw. Vorgaben.
Sollen andere als numerische Probleme gelöst werden, ist eine andere Art der Abbildung menschlichen Verhaltens notwendig.

Der Mensch verfügt über verschiedene Lösungsstrategien für ein und das selbe Problem, die je nach konkreten Vorgaben (Fakten) intuitiv ausgewählt und zum Teil sogar angepaßt werden. Eine Möglichkeit, dieses Problemlösungsverhalten abzubilden ist, an die Stelle eines Algorithmus Wissen und Steuerstrategien zur Verarbeitung des Wissens treten zu lassen. Abhängig von den Fakten wird durch die Wissensverarbeitung dynamisch ein Weg zur Gewinnung der Lösung gefunden.

Im Bereich der wissensbasierten Systeme führt der Versuch einer Nachbildung des menschlichen Vorgehens zu verschiedenen Repräsentationsformen für das benötigte Wissen, die mehr oder weniger stark die Trennung von

1. Fakten (Daten, aufgabenabhängigen Ausprägungen), die Fallwissen darstellen, und

2. Folgerungen (Aktionen, Verfahren, allgemeine Lösungsstrategien) als Ausdruck von Bereichs-(Domänen-)wissen

unterstützen.

Im folgenden werden gängige Repräsentationsformen mit ihren Eigenheiten vorgestellt. Dabei können zwei Ansätze unterschieden werden. Sowohl die logik- und regelbasierte Repräsentation als auch die mathematischen Modelle betrachten vorrangig den Vorgang der Wissensverarbeitung. Die Darstellung von Fakten ist erst in zweiter Linie interessant. Strukturierte Objekte entstammen dem zweiten Ansatz, bei dem die Darstellung von Fakten (statischem Wissen) im Vordergrund steht und die Verarbeitung dieses Wissens darauf aufgebaut wird.

1.1 Logikbasierte Repräsentation

Die Grundlage logikbasierter Repräsentationsformen bildet die mathematische Logik. Das bekannteste Beispiel ist die boolesche Algebra mit den Werten "wahr" und "falsch" bzw. "0" und "1" sowie Junktoren wie "und", "oder", "nicht". Mengenalgebra und die Aussagenlogik sind andere Erscheinungsformen des selben Kalküls, bzw. isomorph zur booleschen Algebra.

Logikbasierte Systeme mit den im folgenden dargestellten Repräsentationsformen **Aussagen- und Prädikatenlogik** nennt man auch **Deduktionssysteme**, da spezielles Wissen aus allgemeinem bzw. anderem Wissen geschlußfolgert (deduziert) wird.
Desweiteren können diese Systeme als **Produktionssysteme** interpretiert werden, da die Komponenten Datenbasis, Produktionsregeln und Kontrollstrategie vorhanden sind.

1.1.1 Aussagenlogik

In der Aussagenlogik werden allgemeine Sachverhalte, d.h. **Fakten** und **Folgerungen** in einer symbolischen Form notiert.
Unter einer Aussage versteht man dabei einen Satz, der eine Behauptung über ein Objekt oder Beziehungen zwischen Objekten ausdrückt und von dem entschieden werden kann, ob er wahr oder falsch ist (zweiwertige Logik).

Elementare Aussagen (Atome, Primformeln) wie zum Beispiel

A1 : Die Sonne scheint.
A2 : Es ist *nicht* Werktag.
A3 : Gelegenheit für eine Wanderung.
A4 : Es steht Zeit zur Verfügung.

können mit den Junktoren

$\wedge$	konjunktive Verknüpfung (*und*)
$\vee$	disjunktive Verknüpfung (*oder*)
$\neg$	Negation (*nicht*)
$\Longrightarrow$	Implikation (*impliziert*)
$\Leftrightarrow$	Äquivalenz (*ist äquivalent zu*)

zu Formeln verknüpft werden:

F_A1: Die Sonne scheint *und* es ist *nicht* Werktag $\Longrightarrow$ Gelegenheit für eine Wanderung

Solche aussagenlogischen Formeln verknüpfen die Wahrheitswerte von Ausdrücken und entsprechen den Folgerungen in einem bestimmten Wissensgebiet (einer Domäne). Fakten werden als elementare Aussagen mit bekanntem Wahrheitswert implementiert und bilden zusammen mit dem Domänen-Wissen die Wissensbasis, die mit Ableitungsregeln erweitert werden kann.
Inferenz- oder Ableitungsregeln gestatten es, aus der Wahrheit gegebener Ausdrücke auf die Gültigkeit anderer Ausdrücke zu schließen.

Wichtige Inferenzregeln sind Modus Ponens, Modus Tollens und Kettenregel:

Modus Ponens
: Wenn (A1 $\Longrightarrow$ A3) gilt und A1 wahr ist, dann ist auch A3 wahr.

Modus Tollens
: Wenn (A1 $\Longrightarrow$ A3) gilt und A3 falsch ist, kann A1 nicht wahr sein.

Kettenregel
: Wenn (A2 $\Longrightarrow$ A4) und (A4 $\Longrightarrow$ A3) gelten, dann gilt auch (A2 $\Longrightarrow$ A3).

Ist eine Menge wahrer Aussagen (Fakten) bzw. Formeln (Axiome) gegeben, so kann mit Inferenzregeln geprüft werden, ob ein Ausdruck mit unbekanntem Wahrheitswert (Behauptung) aus den Axiomen geschlußfolgert werden kann, und somit wahr ist [19].
Eine Behauptung, die aus der Menge der Axiome gefolgert werden kann, wird als Theorem bezeichnet. Den Vorgang bezeichnet man daher auch als **theorem proving** oder **problem solving**. Wissen wird dabei nicht "erzeugt", sondern "bewußt gemacht", implizit vorhandenes Wissen auf Nachfrage explizit bestätigt.

Schwachpunkt der Aussagenlogik ist die fehlende Möglichkeit, allgemeine Aussagen unabhängig von genau bekannten Objekten (Individuen) zu formulieren.

1.1.2 Prädikatenlogik

Will man allgemeingültige Aussagen für einen bestimmten Gegenstandsbereich formulieren, benötigt man Variablen für die Objekte dieses Bereiches.

Erweitert man die Aussagenlogik um Variable, Funktionen und Quantoren, kommt man zur Prädikatenlogik, bei der ein Prädikat eine Eigenschaft eines Objektes oder die Beziehung zwischen mehreren Objekten beschreibt.

Wie in der Aussagenlogik besteht auch in der Prädikatenlogik die Syntax aus Primformeln, die Prädikate genannt werden, sowie prädikatenlogischen Formeln:

P1 : studiert_Elektrotechnik(a)
P2 : hört_Mathematik(B)
P3 : ist_Freund_von(c,d)
F_P1 : $\forall_x$ studiert_Elektrotechnik(x) $\Longrightarrow$ hört_Mathematik(x)
F_P2 : $\exists_y$ hört_Mathematik(y) $\wedge$ $\neg$ studiert_Elektrotechnik(y)
F_P3 : $\exists_z$ ist_Freund_von(z,K)

wobei P für Prädikat und F_P für prädikatenlogische Formel steht.

P1 wird auch als einstelliges Prädikat bezeichnet, da **eine** Variable, Konstante (z.B. B in P2) oder Funktion enthalten ist. P3 ist dementsprechend ein zweistelliges Prädikat. In den Formeln F_P werden Geltungsbereiche der verwendeten Variablen mit Hilfe von Quantoren angegeben:

$\forall_x$	All-Quantor: für alle x gilt ...
$\exists_x$	Existenz-Quantor: es existiert ein x, für das gilt ...

Die Inferenzregeln Modus Ponens, Modus Tollens und Kettenregel gelten auch in der Prädikatenlogik, wobei die beteiligten Prädikate freie Variable enthalten können. Bei Anwendung der Inferenzregeln müssen Prädikate gleichen Namens allerdings auch gleiche Terme enthalten. Aus diesem Grund wird der Algorithmus der Unifikation eingeführt, mit dem die syntaktische Gleichheit von Prädikaten erreicht werden kann.
Gilt zum Beispiel

F_P1 sowie
P1 in der Form: studiert_Elektrotechnik(**Thomas**)

so bedeutet Unifikation die Ersetzung der allgemeinen Variablen **x** in F_P1 durch die spezielle Ausprägung der Variablen a: **Thomas**.
Es ergibt sich also die Erkenntnis:

hört_Mathematik(Thomas).

Daraus folgt, daß die Unifikation wohl zu einer Spezialisierung, aber nie zu einer Verallgemeinerung führen kann.

Resolution

Logikbasierte Systeme werden verwendet, wenn es darum geht, automatisch Theoreme zu beweisen. Es soll dabei nachgewiesen werden, daß eine Behauptung aus gegebenen Prämissen geschlußfolgert werden kann. Das Resolutionsprinzip hat sich dabei als geeignete Methode erwiesen.

Zunächst werden die aussagen- oder prädikatenlogischen Formeln so umgeformt, daß sie einer konjunktiven Normalform der booleschen Algebra entsprechen. Die konjunktiv verknüpften Teile, die disjunktiv verknüpfte Aussagen (Prädikate) oder einzelne Aussagen (Prädikate) enthalten, werden Klauseln genannt.
Logische Gesetze zur Umformung sind beispielsweise die Gesetze von De Morgan, die Assoziativgesetze, die Distributivgesetze sowie die Ersetzung der Implikation über

$$(A \Rightarrow B) \Longleftrightarrow (\neg A \vee B).$$

Die Formel

$$F_A 1 \quad \text{Die Sonne scheint} \quad \textit{und} \quad \text{es ist } \textit{nicht} \text{ Werktag} \quad \Longrightarrow \quad \text{Gelegenheit für eine Wanderung}$$

wird dann zu:

$$\neg(\text{Die Sonne scheint} \quad \textit{und} \quad \text{es ist } \textit{nicht} \text{ Werktag}) \quad \textit{oder} \quad \text{Gelegenheit zu einer Wanderung}$$
$$\neg(\text{Die Sonne scheint}) \quad \textit{oder} \quad \text{Es ist Werktag} \quad \textit{oder} \quad \text{Gelegenheit zu einer Wanderung}$$
$$\underbrace{\neg(\text{Die Sonne scheint}) \ \vee \ \text{Es ist Werktag} \ \vee \ \text{Gelegenheit zu einer Wanderung}}_{Klausel}$$

Die Quantoren der Prädikatenlogik werden in der Klauselform nicht mehr dargestellt. Prädikatenlogische Formeln werden vorher so umgeformt, daß für alle Variablen Allquantoren zutreffen, die dann nicht mehr explizit geschrieben werden. Existenzquantoren werden dabei durch die Benennung des "existierenden Objekts" mit einer sogenannten "Skolem-Funktion" eliminiert. Klauseln, die nur aus einer Aussage bzw. einem Prädikat bestehen, werden auch als Literal bezeichnet.
Liegen alle Axiome in Klauselform vor, kann die Resolution mit Hilfe der Resolutionsregel durchgeführt werden.

Resolutionsregel:

Aus den Elternklauseln $(A \vee B)$ und $(\neg A \vee C)$ kann die Resolvente $(B \vee C)$ gefolgert werden.
Die Literale B und C können auch eine Disjunktion mehrerer Literale darstellen.

Das automatische Theorembeweisen geschieht mit dem Resolutionskalkül durch einen Widerspruchsbeweis. Die zu beweisende Behauptung wird negiert und zusammen mit den Prämissen der Resolution unterworfen. Es werden so lange Resolventen gebildet, bis der Widerspruch z.B. in Form einer leeren Klausel gefunden ist. Existiert kein Widerspruch, muß das für die verwendete Kontrollstrategie definierte Abbruchkriterium erfüllt werden.
Man erhält die Information, daß eine Behauptung aus den vorhandenen Prämissen geschlußfolgert werden kann, also über die Aussage, daß die Negation der Behauptung zu einem Widerspruch führt.

1.2 Regelbasierte Repräsentation

Eine Grundüberlegung beim Lösen von Problemen ist in jedem Benutzerhandbuch von technischen Geräten zu finden:

> Wenn dieser oder jener Zustand eingetreten ist, dann ist folgendes zu unternehmen ...
>
> oder
>
> Wenn ein bestimmter Wert erreicht wurde, dann bedeutet das ...

Es ist mit diesen Regeln möglich, eine bestehende Situation zu verstehen oder zielgerichtete Handlungen vorzunehmen.

Nahezu jede Reparaturanleitung enthält solche Regeln in Form von Tabellen als Diagnosehilfe. Man sucht eine passende Beschreibung der Situation, um dann mit einer neuen, genaueren Beschreibung erneut zu suchen oder mit der Handlungsanweisung sein Problem zu lösen.

Formal ergibt sich folgende Situation:

- die Situation wird durch bestimmte Tatsachen beschrieben,
- es existieren Regeln der Form:
  WENN Bedingung **DANN** Schlußfolgerung bzw. Aktion,
- man sucht nach einer Regel, deren Bedingungsteil (Prämisse, WENN) mit den gegebenen Tatsachen erfüllt wird,
- führt den Aktionsteil (Konklusion, DANN) der Regel aus und
- sucht erneut nach einer passenden Regel.

Eine Problemlösung nach diesem Ablaufschema kann sowohl durch einen Menschen, als auch durch ein Programmsystem erfolgen, das als Inferenzmaschine bezeichnet wird.
Die Formulierung der Regeln mit "Wenn ... dann ... " erinnert an die Formulierungen der aussagen- bzw. prädikatenlogischen Sätze "Aus ... folgt ... " oder "... impliziert ... ":

umgangssprachlich	formalisiert	prädikatenlogisch
Wenn die Pumpe eingeschaltet ist,	Pumpe 1 ist eingeschaltet	eingeschaltet(P1)
beträgt der Durchfluß durch	wenn ... dann	$\Longrightarrow$
das Ventil 5 Normeinheiten	Durchfluß durch Ventil1 ist 5	Durchfluß(V1,5)

Wie sieht es bei Anwendungen aus dem täglichen Leben aus?
Häufig wird allgemeingültiges Wissen als Wissen über ein Exemplar formuliert:

> Wenn diese Pumpe läuft und dieser Tank überläuft, (dann) muß diese Pumpe ausgeschaltet werden.

Man kann diese Regel aussagenlogisch, also **auf ein Individuum** (objektorientiert ausgedrückt: auf **eine Instanz**) bezogen, interpretieren:

> Pumpe1 läuft **und** Tank1 läuft über $\Longrightarrow$ Pumpe1 abschalten

Bemüht man sich um eine allgemeine (allquantisierte) Darstellung in Prädikatenlogik, kann man formulieren:

> für jede Pumpe, die in einen Tank pumpt und jeden an eine Pumpe angeschlossenen Tank gilt:
> Wenn eine Pumpe p eingeschaltet ist und der daran angeschlossene Tank t überläuft, dann wird die Pumpe p abgeschaltet.

Mit der Umsetzung der Verben in Prädikate (T0, T1 repräsentieren aufeinanderfolgende Zeitpunkte) ergibt sich:

$$\forall_p \forall_t \qquad angeschlossen(p,t) \wedge ein(p,T0) \wedge läuft_über(t,T0) \Longrightarrow aus(p,T1)$$

Regeln, die einen Zusammenhang in dieser allgemeinen Form beschreiben, werden als generische Regeln bezeichnet und beziehen sich bei objektorientierter Faktenrepräsentation auf Klassen. Für die Überprüfung der Prämisse muß diese "Kurzform" umgewandelt werden. Alle bekannten Pumpen und Tanks müssen in allen möglichen Kombinationen in die Prämisse eingesetzt werden, ähnlich wie bei einer Summenformel in der Mathematik. Wird diese Arbeit automatisch erledigt, z.B. von einer Inferenzmaschine, werden so viele "einfache" Regeln generiert, wie es Kombinationen von Pumpen und Tanks gibt. Eine generische Regel erleichtert also die Formulierung, aber nicht unbedingt die Anwendung des Wissens.

Gegenüber der logikbasierten steht bei der regelbasierten Repräsentation zusätzlich die Möglichkeit zur Verfügung, bestehendes Wissen zu verändern, da eine Konklusion (Aktionsteil) eine Handlungsanweisung zur Veränderung der Situation enthalten kann (Aktionsregel, [19]). Der Zustand der Wissensbasis nach Ausführung einer Regel kann ein grundlegend anderer sein als vor der Regelausführung.
Diese Eigenschaft bedeutet eine Annäherung an die dynamischen Verhältnisse der Welt, insbesondere der technischen und automatisierten Welt. Andererseits verläßt man damit die monotone Logik, die keine Änderungen, sondern nur Ergänzungen der Wissensbasis behandelt.

Insgesamt kann das Grundprinzip der Regeln formal als eine Sonderform der Prädikatenlogik interpretiert werden.

Ein wichtiger Unterschied besteht allerdings bei den Inferenzregeln (s. S. 6). Während Modus Ponens die Grundlage der Inferenz in regelbasierten Systemen bildet und die Kettenregel ebenfalls gültig ist, wird der Modus Tollens nicht verwendet.

Ein weiterer Unterschied ergibt sich aus der Betrachtungsweise der Regelverwendung. Bei einer Diagnose ist die Auswirkung einer unbekannten Ursache bekannt. Betrachtet man Regeln als Ursache-Wirkung-Beschreibung, ist sofort einsichtig, daß eine Diagnose in "Gegenrichtung" arbeitet.
Die Auswertung von Regeln durch Überprüfung der Prämisse wird als Vorwärtsverkettung bezeichnet. Wird dagegen die Konklusion getestet, spricht man von Rückwärtsverkettung. Im allgemeinen wird die Art der Problembeschreibung bereits der "Richtung" angepaßt. Für ein Diagnosesystem können die Regeln auch so formuliert werden, daß die Auswertung mit Hilfe der Vorwärtsverkettung geschehen kann.

Neben der Wissensrepräsentation durch Regeln benötigt ein System eine Strategie zur Auswahl der nächsten zu bearbeitenden Regel. Es ist durchaus möglich, daß mehrere Regeln in einem gegebenen Zustand ausführbar sind ("feuern" können).
Mögliche Steuerstrategien sind die Tiefensuche und die Breitensuche. Diese Strategien stammen aus der Graphensuche, wobei man einen Zustand als Knoten und alle Regelausführungen als Kanten zwischen zwei Knoten, dem Vorgängerzustand und dem Nachfolgezustand, bezeichnet. Bei regelbasierten Systemen umfaßt ein Zustand die gesamte Regelbasis des Systems. Von einem gegebenen Startzustand wird ein Weg zu dem vorgegebenen (erwünschten) Zielzustand gesucht. Die Suche ist beendet, wenn im Erfolgsfall der Zielzustand erreicht wurde oder im Mißerfolgsfall eine Abbruchbedingung erfüllt wird.

Tiefensuche

Wurde eine Regel "gefeuert" und ein neuer Knoten erzeugt, wird von diesem neuen Knoten aus eine weitere ausführbare Regel gesucht. Auf diese Weise entsteht zunächst eine Kette von Knoten, die keine Verzweigungen aufweist. Erst wenn das Abbruchkriterium "maximale Tiefe" erfüllt ist, wird rückwärts der erste Knoten aufgesucht, von

dem aus eine weitere Regel "gefeuert" werden kann.
In Abbildung 1.1 wird der Suchweg der Tiefensuche durch gestrichelte Pfeile angedeutet.

Breitensuche

Von jedem Knoten aus werden erst alle möglichen Regeln ausgeführt und damit die Nachfolgeknoten erzeugt. Man spricht auch von der vollständigen Expansion eines Knotens. Jeder neu erzeugte Knoten wird an eine Liste der nicht expandierten Knoten angehängt, an deren Spitze der nächste zu expandierende Knoten steht.
Der Suchweg dieser Suche wird in Abbildung 1.1 durch massive Pfeile angedeutet.
Abbruchbedingung kann z.B. die Suchzeit oder die maximale Arbeitsspeichergröße sein.

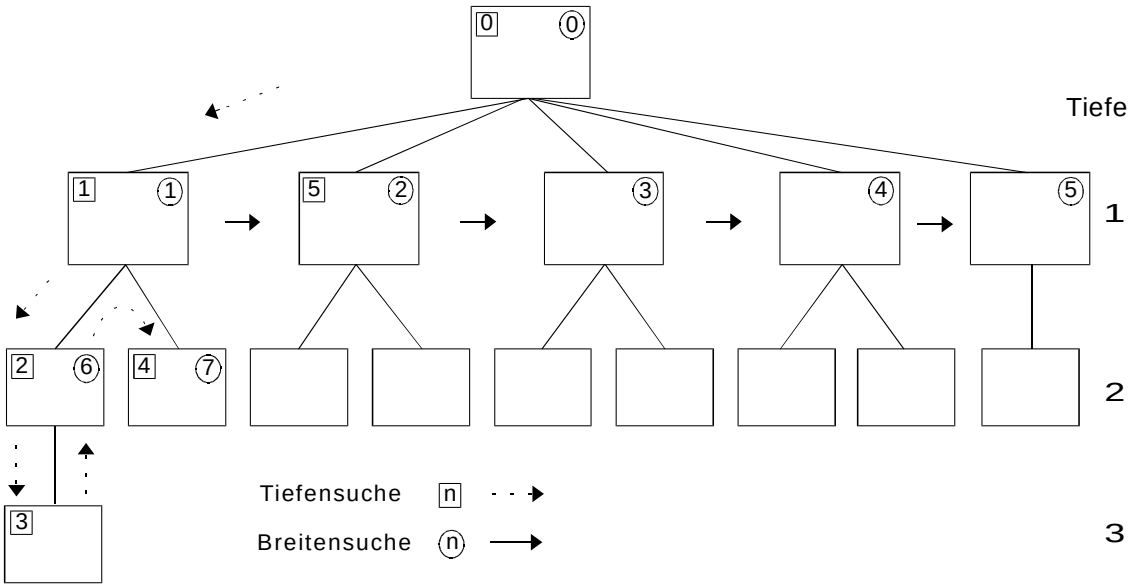


Abbildung 1.1: Graph eines Suchbaumes

1.3 Strukturierte Objekte

Während die bisher betrachteten Ansätze der Wissensrepräsentation die menschliche Vorgehensweise beim Lösen von Aufgaben als Grundlage nehmen, richtet sich das Augenmerk bei strukturierten Objekten auf die Art, wie Objekte mit ihren Eigenschaften, Zusammenhängen und Abhängigkeiten erfaßt werden.

1.3.1 Semantische Netze

Semantische oder assoziative Netze sind im Rahmen der Sprachverarbeitung entstanden. Natürlichsprachige Information kann für die automatische Verarbeitung zugänglich gemacht werden, wenn es gelingt, Satzteile und Worte grammatischen Grundstrukturen und Klassen zuzuordnen. Es werden dabei ausschließlich **Fakten** und ihre **statischen Zuordnungen** abgebildet. Die grafische Darstellung eines semantischen Netzes kann auch als die Veranschaulichung von zweistelligen Relationen interpretiert werden. Jedes Netz stellt ein eigenes, strukturiertes Objekt dar.
Schlußfolgerungen sind in semantischen Netzen nur unter Einhaltung von Randbedingungen (z.B. transitive Relationen) möglich. Bei der Sprachverarbeitung beispielsweise werden zwei Netze miteinander verglichen und so die Zuordnung der konkreten Worte zu den Wortklassen der Grammatik vorgenommen.

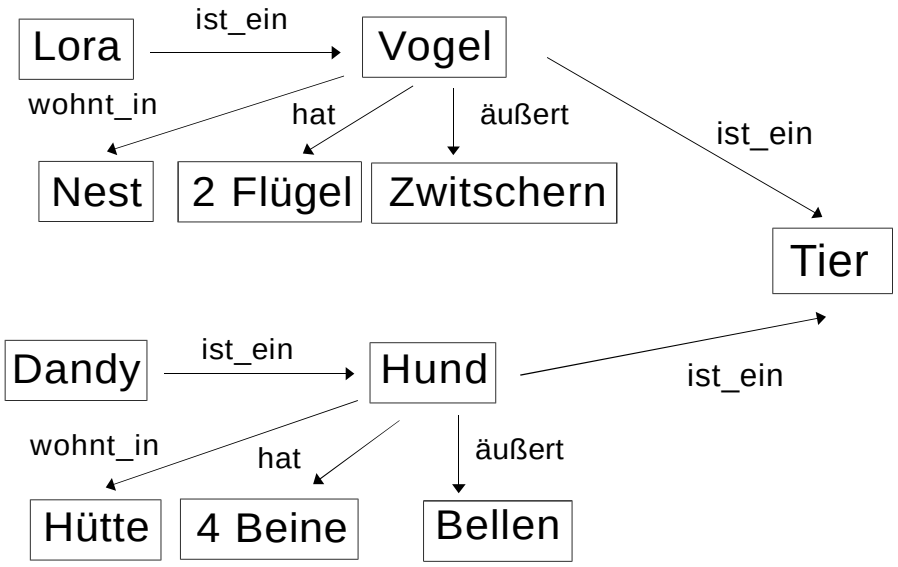


Abbildung 1.2: Beispiel eines semantischen Netzes

1.3.2 Frames und objektorientierte Ansätze

Eine andere Art, Gegenstände oder Begriffe mit ihren Eigenschaften und Beziehungen darzustellen, sind Datenstrukturen, die als Frames (Rahmen) bezeichnet werden. Sie sind mit "structures" oder "unions" in der Programmiersprache C vergleichbar. Ein ganzer Satz von Angaben (Werten, Eigenschaften) verschiedenen Typs wird in einer Einheit oder Struktur zusammengefaßt. Auf diese Weise können zusammengehörige Informationen leichter referenziert werden. Im Kontext der Wissensverarbeitung kann man sagen, daß das Wissen über ein Individuum an einer Stelle gebündelt wird. Damit kann Wissen in einer übersichtlichen Art abgelegt werden, die auch dem intuitiven Such- und Speicherverhalten des Menschen entspricht. Gegenstände können in ihren Eigenschaften genau beschrieben werden, es können aber ebenso Angaben zu Gruppenzugehörigkeiten oder weiteren Verfeinerungen gemacht werden.

Ein Beispiel:
Eine Autovermietung besitzt verschiedene Typen von Fahrzeugen. Eine Beschreibung könnte folgendermaßen aussehen:

> **konkret: Auto 13** ist ein Kombi und hat 4 Räder, 5 Türen, 75 kW Leistung, ...
>
> **Gruppenzugehörigkeit:** Auto 13 gehört zur Gruppe der Personenwagen, oder allgemeiner zur Gruppe der Kraftfahrzeuge
>
> **Verfeinerung:** Die Gruppe der PKW kann noch in die beiden Gruppen I-Autos mit Anhängerkupplung und II-Autos mit Klimaanlage unterteilt werden.

Mit Hilfe dieser Angaben kann eine Hierarchie aufgebaut werden, die bei den stärksten Verallgemeinerungen (beim größten Abstraktionsgrad) beginnt (Wurzel, Eltern) und sich dann durch immer größere Verfeinerungen (Äste, Kinder) bis zu einzelnen Individuen (Blätter, Instanzen) verzweigt. Die Hierarchieebenen können allgemein in Richtung Wurzel "Oberklassen", in Richtung Blätter "Unterklassen" genannt werden. Die Bezeichnungen "Eltern- bzw. Kindklasse" haben einen konkreten Hintergrund. So wie Kinder von ihren Eltern gewisse Eigenschaften erben und andere Eigenschaften selbst entwickeln, "erbt" auch eine "Kindklasse" die Eigenschaften der "Elternklasse" und kann neue, eigene Eigenschaften erhalten. Bild 1.3 veranschaulicht die Klassenhierarchie des Beispiels.

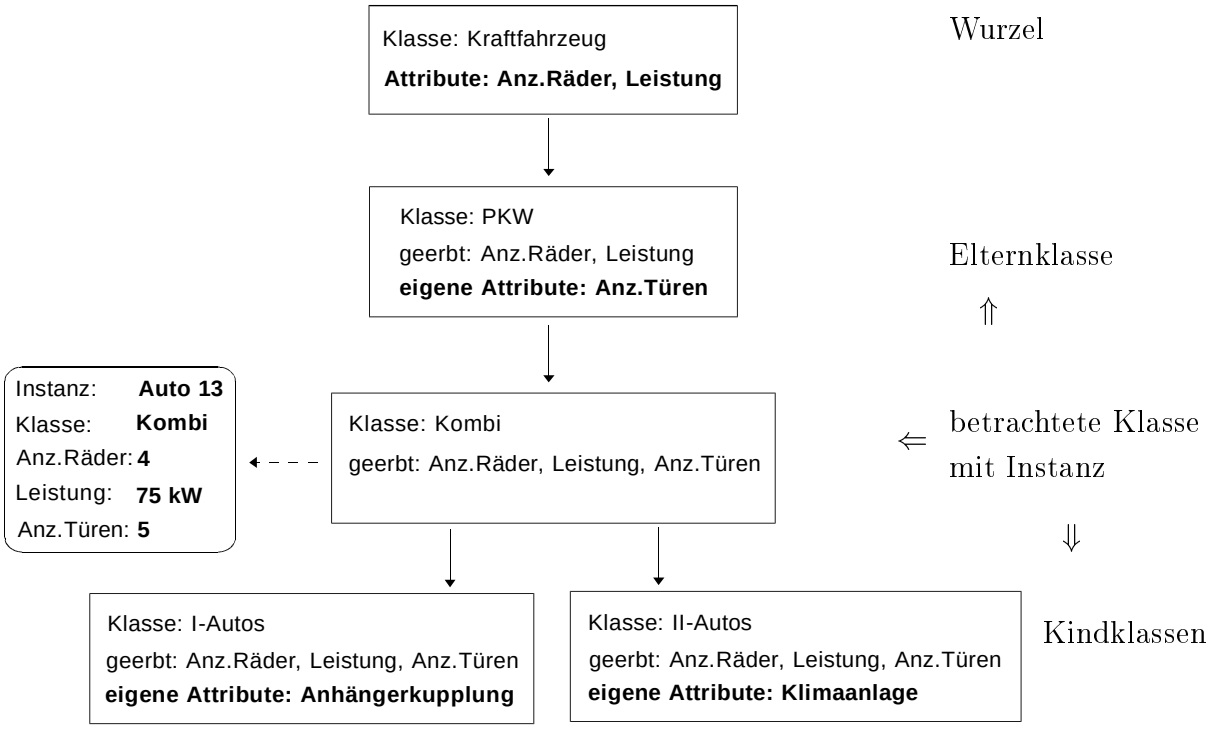


Abbildung 1.3: Graph einer Klassenhierarchie

Die betrachtete Klasse *Kombi* hat die gleichen Eigenschaften (Anzahl Räder, Anzahl Türen, Leistung) wie die "Elternklasse" PKW und diese wiederum hat die Eigenschaften der Wurzel *Kraftfahrzeug*, ergänzt um Anzahl Türen. Die Unterklasse *I-Autos* hat ebenfalls diese Eigenschaften und zusätzlich die Eigenschaft Anhängerkupplung.
Die Repräsentation einer Eigenschaft bezeichnet man auch als Attribut oder Slot. Die konkreten Ausprägungen bei einem bestimmten Gegenstand (Instanz) sind die Werte (Filler, Values). Die Klassen enthalten nur die allgemeine Bezeichnung der Eigenschaften und ihrer Werte (Initialisierungswert, Menge der möglichen Werte), während das Individuum Auto 13 jeweils einen konkreten Wert enthält.
Eine solche Hierarchie der allgemeinen Beschreibungen (Klassenhierarchie) bietet verschiedene Vorteile wie Übersichtlichkeit und leichte Änder- und Erweiterbarkeit. Gegenstände und Begriffe können leicht einer Klasse zugeordnet oder durch eine neue Klasse allgemein beschrieben werden. Änderungen oder Erweiterungen bei Klasseneigenschaften werden bei der abstraktesten Klasse vorgenommen und durch die Vererbung in alle Unterklassen weitergegeben. Auch Ausnahmen von Klassenzugehörigkeiten lassen sich leicht durch Hinzufügen einer Unterklasse mit einem Attribut für die Besonderheit repräsentieren. Ein Pinguin oder ein Strauß gehören zwar zur Klasse der Vögel, haben aber die Besonderheit, nicht zu fliegen. Wie eine Unterklasse für solche "Sonder"-Vögel

aussieht, hängt von der Verwendungsart des Wissens ab und kann sehr unterschiedlich sein.

Mit dem bisher beschriebenen Aufbau von Frames und dem Vererbungsmechanismus allein ist noch keine Verarbeitung der abgelegten Fakten und Zusammenhänge möglich. Eine Eigenschaft der Frames ist die Möglichkeit, Formeln oder Prozeduren in Attribute einzutragen, die bei einem Zugriff auf das Attribut automatisch ausgeführt bzw. gestartet werden. In diesem Zusammenhang wird eine Formel oder eine andere einfache Anweisung als "Constraint" und eine Prozedur auch als "Dämon" bezeichnet. Dieser "gute Geist" leistet unsichtbar im richtigen Moment wichtige Dienste. Wird beispielsweise der Wert eines Attributes abgefragt, der zur Zeit nicht zur Verfügung steht, wird er durch den Dämon ermittelt. An Stelle eines festen Wertes wird dazu die Funktion angegeben, mit der dieser Wert berechnet werden kann. Ebenso kann auch bei Änderung des Attributwertes ein Dämon dafür sorgen, daß andere Attribute aktualisiert werden.
Komplexere Inferenzmechanismen, die in Frames abgelegte Fakten und Beziehungen verarbeiten, sind vom Entwickler selbst anzulegen. Frames sind also eine reine Repräsentationsmöglichkeit für Fakten und Beziehungen.

Objektorientierte Ansätze basieren auf strukturierten Objekten, die als Erweiterung der Frames verstanden werden können. Gibt es bei Frames für einzelne Attribute Dämonen, werden bei objektorientierten Ansätzen alle Zugriffe auf Attribute über "objekteigene" Funktionen (Prozeduren), die hier Methoden genannt werden, vorgenommen. Ein Objekt stellt quasi eine Kapsel mit definierten Schnittstellen dar, in der das Wissen geschützt wird. Ein Attributwert kann nur verwendet werden, indem das entsprechende Objekt mittels einer Nachricht über den Wert für eine "Eigenschaft" befragt wird. Ein mit diesem Ansatz erstelltes Programm repräsentiert also Fragen und Antworten, die in Form von Nachrichtenaustausch zwischen Objekten realisiert werden.
Programmiersprachen und Entwicklungsumgebungen für wissensbasierte Systeme mit objektorientierter Wissensrepräsentation bieten viele Zwischenstadien zwischen Frames und streng interpretierter Objektorientierung an. Häufig wird das Schlagwort "objektorientiert" benutzt und eine komfortable Frameverarbeitung angeboten. Genaugenommen sollte man solche Verfahren "frame-" oder "objekt**basiert**" nennen.

1.4 Mathematische Modelle

Im Umfeld technischer Anwendungen werden Computer z.B. bei der Anlagenüberwachung oder Qualitätskontrolle eingesetzt. Unteranderem werden dabei verschiedene Kenn- und Steuergrößen unter berücksichtigung komplexer Zusammenhänge berechnet. Die Darstellung dieser Zusammenhänge kann durch mathematische Modelle geschehen, in denen Beschreibungen von Zusammenhängen aus Physik und Chemie in Form von Formeln zusammengefaßt werden.

Die Modellbildung kann auf der Grundlage von bereits realisierten Teilkomponenten aber auch ohne eine konkrete Anlage unter Verwendung von theoretischen Ansätzen erfolgen. Vorteil der Modelle ist die Möglichkeit, die Parameter der Anlagen sowohl später während des Betriebs berechnen als auch schon in der Planung vorherberechnen zu können, oder die gesamte Anlage mit neuen Parametern zu Untersuchungszwecken zu simulieren.

Wie genau die mathematischen Modelle die Realität abbilden, kann frühestens nach der Realisierung der Anlage oder der modellierten Teilkomponenten anhand von Meßwerten der Prozessgröße überprüft werden. Ohne diese Prüfmöglichkeit muß der Entwickler des Modells besonders darauf achten, daß die verwendeten Formeln in sich korrekt sind, alle Randbedingungen für ihre Gültigkeit erfüllt sind und sich keine Widersprüche durch die Verknüpfung vieler Formeln zu einem komplexen Modell ergeben.

Ist ein Modell erstellt worden, muß seine Korrektheit, also die Übereinstimmung der weiter zu verwendenden Ergebnisse mit der Realität getestet werden. Nach der Bestätigung der Korrektheit stellt ein Modell eine komplexe, abgeschlossene und in sich konsistente Repräsentation des Wissens über die modellierte Anlage dar.

Unter dem Aspekt der Wissensverarbeitung kann ein bereits bestehendes mathematisches Modell wie ein "Wissensblock" in einem wissensbasierten System eingesetzt werden, ohne daß der Entwickler des Systems den inneren Aufbau des mathematischen Modells kennen muß. Mathematische Modelle können dabei z.B. als Prozeduren in einem Frame realisiert werden.

1.5 Hybride Systeme

Während der Entwicklung von industrietauglichen wissensbasierten Systemen wurde erkannt, daß eine Repräsentationsform allein nicht genügend Flexibilität bietet, um für ein gegebenes Problem in jedem Fall eine gute Lösung zu ermöglichen. Marktfähige Entwicklungswerkzeuge wurden auf diese Anforderungen zugeschnitten. Neben verschiedenen Repräsentationsformen stellen sie auch in weiten Bereichen konfigurierbare Inferenzmechanismen zur Verfügung. Systeme, die aus der "Kreuzung" mehrerer Ansätze zur Wissensrepräsentation entstehen, bezeichnet man als hybride Systeme.

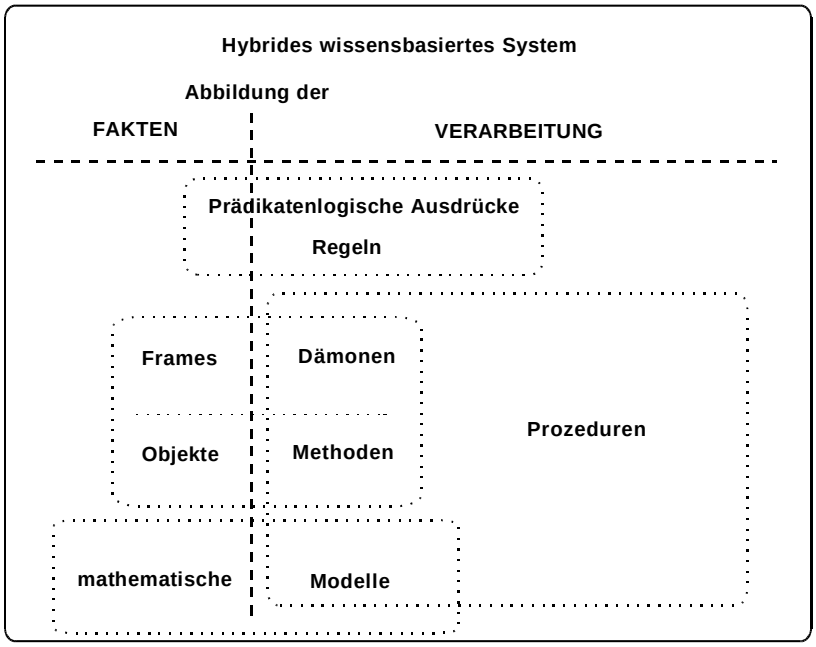


Abbildung 1.4: Blockvorstellung eines hybriden wissensbasierten Systems

Eine häufig verwendete Kombination von Wissensrepräsentationsformen umfaßt Regeln und einen frame- bzw. objektbasierten Ansatz. In den Objekten wird Faktenwissen zusammengefaßt, das mit Regeln erweitert oder verändert wird (siehe Abb. 1.4) [3]. Die Inferenzmaschine steuert das Prüfen und Ausführen der Regeln sowie die Ausführung der Methoden.
Reichen die den Objekten direkt zugeordneten Methoden zur Verarbeitung nicht aus, stehen auch Entwicklungswerkzeuge zur Verfügung, die zusätzlich konventionelle Prozeduren erlauben. Diese Werkzeuge sind ein Kompromiß zwischen der bekannten Programmierweise und dem wissensbasierten Ansatz. Mathematische Modelle, die in vielen technischen Anwendungen die Grundlage für Entscheidungen bilden, sind mit dem Hilfsmittel Prozedur auch in wissensbasierten Anwendungen integrierbar.

Leider lassen sich auch hier nicht nur die Vorteile beider Welten verbinden. Einerseits erlaubt die Verwendung von Prozeduren eine übersichtlichere Gestaltung von Regeln und einen exakt planbaren Ablauf. Andererseits sind Prozeduren selbst oft unübersichtlich, komplex und somit schwer nachzuvollziehen.

Die Kombination von Wissensrepräsentationsformen erfordert hinsichtlich der Widerspruchsfreiheit eines Systems mehr als nur die "Addition" eventuell vorhandener, einzelner Definitionen und Testverfahren zum Begriff der Konsistenz.
Die Analyse und Lösung dieses Problems ist Gegenstand der vorliegenden Arbeit.

1.6 Zusammenfassung

Die formale Darstellung von Wissen kann mit verschiedenen Ansätzen erfolgen. Mit der klassischen Logik, den Regeln, strukturierten Objekten und mathematischen Modellen bis hin zu den hybriden Systemen stehen Repräsentationsformen zur Verfügung, die für bestimmte Anwendungstypen zugeschnitten sind. Die unter der Überschrift "wissensbasierte Systeme" angesiedelten Anwendungen in technischen Umfeldern lassen sich mit den hybriden Entwicklungssystemen am besten realisieren, obwohl neben den Vorteilen auch die Nachteile der einzelnen Ansätze berücksichtigt werden müssen (siehe auch Erläuterungen in Kapitel 2.2 und Kapitel 3). Bei allen Unterschieden zur Entwicklung imperativer Programme bleibt doch die Verantwortung für Übersichtlichkeit und allgemein gute Struktur beim Entwickler.
Welche Entwicklungsmethodik aus dem klassischen Softwareengineering für wissensbasierte Systeme günstig ist, wird im folgenden Kapitel näher betrachtet.

Kapitel 2

Der Konsistenzbegriff

Wissensbasierte Systeme werden oft mit komplexen Programmen verglichen. Wissenschaftliche Arbeiten zur Konsistenz gibt es jedoch nicht nur in diesem Zusammenhang, sondern auch auf dem Gebiet der Datenbanksysteme. Es muß daher geprüft werden, in wie weit diese Konsistenzbetrachtungen auf wissensbasierte Systeme übertragen werden können.

Nach einer kurzen Abhandlung über Datenbanksysteme (Datenbanken) werden Gedanken zur Überwachung bzw. Wahrung der dort definierten Konsistenz vorgestellt. Deduktive Datenbanken bieten eine Erweiterung des klassischen Datenbankansatzes mittels zusätzlicher Deduktionsregeln. Der damit eingeschlagene Entwicklungsweg läßt es sinnvoll erscheinen, wissensbasierte Systeme als eine erweiterte Form von Datenbanken zu interpretieren. Für hybride Systeme mit den Repräsentationsformen Regeln, Objekte und im weiteren Sinne Prozeduren wird daher die Übertragbarkeit bzw. die Notwendigkeit zur Erweiterung des Konsistenzbegriffes erarbeitet. Es wird versucht, regelbasierte Systeme auf Deduktionssysteme abzubilden, um möglichst die bekannten Testverfahren für Konsistenz bzw. Inkonsistenz auch für regelbasierte Systeme zugänglich zu machen.
Es werden komplexe Lösungsansätze für die einzelnen Bereiche Regelkonsistenz, Frame- oder Objektkonsistenz und Prozedurkonsistenz und deren Kombination in einer "Gesamtkonsistenz" sowie den dabei auftretenden Randbedingungen vorgestellt.

Die Abschnitte zur Konsistenz in Datenbanken beruhen auf den Arbeiten von Gähler [5], Leikauf [18] und Moerkotte [20].

2.1 Datenbanken

Datenbanksysteme bestehen aus einer Ansammlung logisch zusammengehörender Daten, der eigentlichen Datenbank (Datenbasis), und **Programmen zu ihrer Verwaltung**, dem Datenbankverwaltungssystem [26]. Die Daten sind in **Datensätzen** organisiert und können mittels einfacher Elementaroperationen (Einfügen, Ändern und Löschen von Daten; auch "Mutationen" genannt) oder komplexeren Abläufen aus mehreren Elementaroperationen (hier kurz Mutations**folgen** genannt), manipuliert werden. Der Zugriff auf die Daten geschieht dabei durch Anwendungsprogramme, die auf Funktionen des Verwaltungssystems zugreifen.

Aus den gespeicherten Daten können von mehreren Anwendern zur gleichen Zeit beliebige Auswertungen gewonnen werden. Selbstverständliche Forderung ist dabei, daß die Daten in der Datenbank "richtig" sind, wenn auf sie zugegriffen werden kann. Der Ausdruck "richtig" steht dabei für zwei Aspekte: die "Korrektheit" und die "Konsistenz".
Unter "Korrektheit" einer Datenbasis wird die Übereinstimmung zwischen der realen Welt und ihrer Abbildung durch die Daten verstanden. Was unter "Konsistenz" zu verstehen ist, wird in den folgenden Abschnitten behandelt.

2.1.1 Konsistenzbedingungen

Unter dem Begriff "Konsistenz" werden in Datenbanksystemen ganz allgemein zwei verschiedene Sachverhalte verstanden. Erstens das Auftreten unterschiedlicher Werte für ein und das selbe Attribut und zweitens, nach Quiel [26] , die Abwesenheit von logischen Widersprüchen innerhalb der Datenbank, die nach Zehnder [33] auch als "semantische Integrität" bezeichnet und im weiteren genauer betrachtet wird.
Um Konsistenz zu gewährleisten, soll die Aufnahme widersprüchlicher Daten vermieden bzw. verhindert werden. Neben den Daten werden dazu "Konsistenzbedingungen" formuliert, die bei Manipulationen an der Datenbasis eingehalten werden müssen.

Als Datenmodell wird hier das "erweiterte Relationenmodell" nach Zehnder [33] benutzt, welches die Beschreibung von Beziehungen zwischen Gruppen von Datenobjekten ("Entitätsmengen") zuläßt. Je zwei "Kardinalitäten" beschreiben eine Beziehung. Eine Kardinalität K_{12} ist die Anzahl der Entitäten (Datenobjekte) der Entitätsmenge E2, die einer Entität der Menge E1 zugeordnet werden können. Angegeben werden im allgemeinen Unter- (u) und Obergrenze (o).

Wird ein Modell gebildet, in dem Daten in eine vorher festgelegte Datenstruktur eingefügt werden, sind bereits "Konsistenzbedingungen" verschiedener Art zu beachten:

- Ein Datum darf nur dann übernommen werden, wenn der Datentyp des betreffenden Feldes eingehalten wird.
- Ein Datensatz darf nur dann erzeugt werden, wenn die für einen solchen Datensatz definierten Kardinalitäten eingehalten werden.
- Werden in einem neuen Datensatz z.B. Schlüsselfelder anderer Datensätze als Werte in Nutzfeldern benötigt, müssen diese Datensätze existieren, damit aktuell gültige Werte eingetragen werden können.

Konsistenzbedingungen repräsentieren also Beschränkungen der zulässigen Zustände bestimmter Datenelemente oder deren Kombinationen. Neben den durch die Datentypen, -elemente und Kardinalitäten bedingten, sogenannten "impliziten" Konsistenzbedingungen können auch weitere Bedingungen für die Daten als "modellexterne" Konsistenzbedingungen angegeben werden. Eine solche Bedingung wäre z.B. die Forderung, daß nur Datensätze des Typs A eingefügt werden, die im Nutzfeld 3 einen Wert größer 4711 aufweisen.

Die impliziten Bedingungen werden durch die Definition der Datensätze der Datenbasis vorgegeben und sollten durch das Verwaltungssystem bei der Eingabe der Daten ("Eingabekonsistenz") geprüft werden. Modellexterne Konsistenzbedingungen können beliebig komplex sein und sich je nach Auswertung der Daten unterscheiden. Prinzipiell kann jedes Anwendungsprogramm eigene Bedingungen enthalten, die unabhängig von der Datenbank gespeichert und verarbeitet werden.

Damit kann definiert werden:
"Eine Datenbank ist konsistent, falls sie die im Datenschema beschriebene Struktur hat und zudem einer Menge von Regeln (Konsistenzbedingungen, kurz KB) genügt, welche sich auf die im Datenschema beschriebenen Strukturtypen beziehen."
(Zitat aus Gähler [5], Seite 11)

2.1.2 Überprüfung der Konsistenzbedingungen

Die Konsistenzprüfung wird durch die Prüfung der Einhaltung der formulierten Konsistenzbedingungen realisiert. Wird eine verletzte Bedingung ("Inkonsistenz") entdeckt, muß entweder eine Mutationsfolge ausgelöst werden, die dafür sorgt, daß die Bedingung erfüllt wird, oder die Ursache der Verletzung (die verursachende Mutationsfolge) muß rückgängig gemacht werden.
Für beide Möglichkeiten müssen gewisse Bedingungen erfüllt sein. Eine Konsistenzbedingung muß aus den Komponenten "Prüfzeitpunkt", "Bedingung" und "Reaktion bei Verletzung" bestehen. Als Prüfzeitpunkt kommen dabei die Werte "vor einer Mutationsfolge" oder "sofort nach einer Mutationsfolge" in Frage. Die eigentliche Bedingung wird im allgemeinen als prädikatenlogischer Ausdruck formuliert. Die Reaktion bei Verletzung der Bedingung kann ein einfacher Abbruch der Mutationsfolge und das Rückgängigmachen bereits erfolgter Änderungen in der Datenbank sein, oder auch der Aufruf eines Unterprogrammes, das für die Behebung der Inkonsistenz sorgt. Eine weitere Voraussetzung ist die Verwaltung der beteiligten Originaldatensätze, der aktuellen Mutationsfolge und der "temporären" Datensätze durch das Datenbankverwaltungssystem.

Werden die Konsistenzbedingungen in den Anwendungsprogrammen angesiedelt, ist kein umfassender Überblick über alle geltenden Bedingungen möglich. Sinnvoll ist daher, die Verwaltung durch das Datenbankverwaltungssystem vorzunehmen. Auf diese Weise können Anwendungsprogramme unabhängig von Konsistenzbedingungen entworfen und die Konsistenzbedingungen ohne Auswirkung auf die Programme geändert werden. Ein weiterer Vorteil dieser Zuordnung ist die Möglichkeit, die sogenannte "Meta-Konsistenz" zu überwachen. Dabei handelt es sich um die Konsistenz der einzelnen Konsistenzbedingungen untereinander.

In den betrachteten Arbeiten wird die Meta-Konsistenz nicht behandelt.
Leikauf bemerkt dazu:
"Es ist aber noch offen, wie man die Konsistenz von Restriktionen zeigen soll, die als prädikatenlogische Ausdrücke oder gar mittels Triggerprozeduren (*Unterprogrammen*) formuliert wurden; die dazu nötigen Techniken müßten wohl aus der Welt der Programmverifikation stammen." (Zitat aus Leikauf [18], Seite 37)

Konsistenzbedingungen in der hier vorgestellten Art repräsentieren einen Teil der Semantik der Daten. Dehnt man diesen Ansatz weiter aus, kommt man zu deduktiven Datenbanken.

2.1.3 Deduktive Datenbanken

In der realen Welt existieren vielfältige Gesetzmäßigkeiten oder Abhängigkeiten von Daten untereinander. Bildet man zusätzlich zu den Daten auch Gesetzmäßigkeiten beispielsweise mit Hilfe von Ableitungsregeln in der Datenbank ab, kann sich die zu speichernde Datenmenge reduzieren. Fakten, die sich aus der Verknüpfung anderer Daten ableiten lassen, müssen nicht explizit in die Datenbasis aufgenommen werden. Es reicht die Formulierung der Abhängigkeit, um im Bedarfsfall das Faktum explizit zu bilden.

In der Arbeit von Moerkotte [20] wird als Beschreibungssprache für die Ableitungsregeln (Deduktionsregeln, oder kurz: Regeln) die Prädikatenlogik erster Stufe verwendet. Eine Datenbasis besteht hier also aus den Daten, Deduktionsregeln und den schon früher beschriebenen Konsistenzbedingungen.
Alle Fakten, die nicht aus der Datenbasis ableitbar sind, gelten in negierter Form ("closed-world-assumption") und die Regeln werden in Form von Horn-Klauseln dargestellt (siehe Moerkotte [20], Seite 21).

Konsistenztests finden bei Moerkotte während bzw. unmittelbar nach einer Mutationsfolge statt. Soll beispielsweise ein Faktum neu in die Datenbasis aufgenommen werden, muß zunächst geprüft werden, ob die Negation des Faktums durch die Anwendung der Regeln aus den vorhandenen Fakten abgeleitet werden kann. Des weiteren ist zu prüfen, ob Regeln auf das neue Faktum angewandt werden können, die eine Negation eines Faktums aus der Datenbasis ableiten lassen. Als letzter Punkt ist noch die Einhaltung der formalisierten Konsistenzbedingungen für das neue Faktum zu prüfen. Hier wird deutlich, daß deduktive Datenbanken und wissensbasierte Systeme fast fließend ineinander übergehen, da das beschriebene Vorgehen bereits dem Resolutionskalkül entspricht.

Wird bei dem Test der Datenbasis eine Inkonsistenz durch eine Mutationsfolge festgestellt, soll auch hier, wie bereits früher beschrieben, die Mutationsfolge abgebrochen und alle Änderungen aus dieser Mutationsfolge rückgängig gemacht, oder eine "Reparatur-Mutationsfolge" durchgeführt werden. Eine Reparatur setzt sehr genaues Wissen über mögliche Gründe einer Inkonsistenz und die Ermittlung der Inkonsistenzursachen (Fakten, Regeln, Konsistenzbedingungen oder eine beliebige Kombination dieser Elemente) voraus. Die Konsistenz der Regeln und der Konsistenzbedingungen untereinander wird auch in der Arbeit von Moerkotte nicht untersucht.

Als weiterer Entwicklungsschritt kann die Möglichkeit angesehen werden, Regeln und Konsistenzbedingungen mit weniger komplexer Struktur zu verwenden, wenn z.B. ein objektorientiertes Datenmodell verwendet wird (Moerkotte; Seite 3).

2.2 Wissensbasierte Systeme

Datenbanken dienen im allgemeinen der Speicherung großer Datenmengen und dem schnellen Auffinden zusammengehöriger Fakten bei Anfragen von Benutzern. Wissensbasierte Systeme dienen der Speicherung von Wissen, wie Probleme gelöst werden und können auch unsicheres Wissen nutzen, also z.B. Aussagen machen wie: "...könnte mit einer Sicherheit von 0.6 zusammengehören ...".

Ein wissensbasiertes System unterscheidet sich in den folgenden Punkten von Datenbanksystemen:

- Fakten werden **unabhängig** von einander gespeichert, es existiert also keine vorgegebene Verknüpfung wie bei den Datenbankmodellen (hierarchisch, vernetzt oder relational).
- Aktionsregeln beschreiben die Verknüpfung von Fakten und lösen Aktionen aus, die neue Daten generieren können. Sie repräsentieren das Wissen, das benötigt wird, um einen Lösungsraum aufzuspannen, in dem ein komplexes Problem zu lösen ist.
- Eine Kontrolleinheit (Inferenzmaschine) steuert die Anwendung der Aktionsregeln. Ein Inferenzschritt ist mit einer "Mutationsfolge" bei einer Datenbank vergleichbar. Eine Mutationsfolge ist eine Menge von Aktionen, die zur Durchführung einer Datenmanipulation benötigt wird, die vom Benutzer über ein Anwendungsprogramm angefordert wurde. Bei einem Inferenzschritt wird die Menge von Regeln ausgeführt, deren Prämissen in einem gegebenen Zustand erfüllt ist. Die Überprüfung der Prämissen wird durch eine Änderung der Fakten der Wissensbasis ausgelöst, wobei diese Änderung entweder durch den Benutzer oder einen Inferenzschritt verursacht wird.

Diese Unterschiede sind so gravierend, daß sie eine eigenständige Betrachtung der Konsistenz in wissensbasierten Systemen notwendig machen. In der vorliegenden Arbeit kommen noch weitere wichtige Punkte hinzu: Anwendungen wissensbasierter Systeme in der Automatisierungstechnik erfordern eine **andere Regelstruktur** ("Aktionsregeln") und die Verwendung von **Prozeduren**.
Die in deduktiven Datenbanken eingesetzten Schlußfolgerungsregeln dienen der Reduktion der Basisdatenmenge. Die zusätzlich bei wissensbasierten Systemen vorhandenen **Aktionsregeln** sind in der Lage, außer dem Hinzufügen neuer Daten, wie durch Deduktionsregeln, auch bestehende Daten zu verändern, oder zu löschen. Dabei kann es sich um feste Ersetzungen genauso handeln, wie um zustandsabhängige Änderungen, die durch

Prozeduren realisiert werden. Verwendet man bei Aktionsregeln die Konsistenzansätze für Datenbanken, muß entweder in jedem Prädikat ein Gültigkeitszeitpunkt enthalten sein, oder auf eine andere Art muß sichergestellt werden, daß der "neue" Datenzustand als Ausgangspunkt für die Konsistenzuntersuchungen verwendet wird.

Abschließend kann also festgestellt werden, daß wissensbasierte Systeme durchaus mit Datenbanken in Verbindung gebracht werden können, daß aber für die Interpretation des Konsistenzbegriffes besonders in hybriden Systemen durch die Verwendung von Prozeduren neue Ansätze gefunden werden müssen.

Konsistenz

Die "Übersetzung" von Konsistenz mit Widerspruchsfreiheit läßt grundsätzlich zwei Interpretationsmöglichkeiten offen, die sich darauf beziehen, wozwischen Widersprüche auftreten können. Global betrachtet existiert die reale Welt (der betrachtete Weltausschnitt) und als Abbild oder Modell dessen Repräsentation in einer formalen Sprache. Widersprüche zwischen diesen beiden Dingen sind nur dann erkennbar, wenn der Betrachter den Weltausschnitt und die formale Repräsentation mit der jeweiligen Bedeutung erfaßt hat und vergleichen kann. Die so überprüfte Widerspruchsfreiheit wird in der Literatur (z.B. [9]) auch mit Korrektheit bezeichnet.
Betrachtet man die formale Repräsentation allein, sozusagen die künstliche Welt, sind Widersprüche zwischen einzelnen Wissenselementen z.B. zwei Aussagen oder Regeln möglich. Konsistenz kann als Widerspruchsfreiheit des bereits formalisierten Wissens in sich interpretiert werden. Die Interpretation des Begriffes Widerspruch erfährt bei den untersuchten verschiedenen Repräsentationsformen wiederum unterschiedliche Konkretisierungen.

Im folgenden wird zunächst die spezielle Interpretation des Konsistenzbegriffes in klassischen Deduktionssystemen, die auf den Repräsentationsformen Aussagen- bzw. Prädikatenlogik aufbauen, untersucht.

2.2.1 Konsistenz in klassischen Deduktionssystemen

Da die Aussagenlogik, wie bereits in Kapitel 1.1 dargelegt, die Grundlage von Deduktionssystemen ist, soll auch der Begriff Konsistenz zunächst in diesem Zusammenhang betrachtet werden.

Konsistenz wird in der **Aussagenlogik** definiert als das Vorhandensein mindestens einer Belegung mit Wahrheitswerten, für die alle betrachteten Implikationen erfüllt werden ([1] , [38]).

Für die beiden Implikationen

$$P \implies Q \qquad\qquad P \wedge R \implies \neg Q$$

ergibt sich mit Hilfe der Wahrheitstabellen

P	$\neg P \vee Q$
1	0/1
0	1

P	R	$\neg P \vee \neg R \vee \neg Q$
1	1	0/1
0	1	1
1	0	1
0	0	1

daß sie konsistent sind, da mit $P = 0$ beide erfüllt werden können.

In einem Deduktionssystem der Aussagenlogik wird eine Datenbasis, die die Elemente P, R, $P \implies Q$ und $P \wedge R \implies \neg Q$ enthält, als "nicht korrekt" bezeichnet, da der Widerspruch bereits aufgrund der Modellbildung vorhanden ist. Unkorrektheit ist also eine Form semantischer (d.h. nicht syntaktisch prüfbarer) Inkonsistenz.

2.2.2 Konsistenz in regelbasierten Systemen

Eine andere Definition des Konsistenzbegriffes wird bei regelbasierten Systemen notwendig. Das Domänenwissen repräsentiert das allgemein formulierte Wissen über Zusammenhänge in einem bestimmten Diskursbereich (Domäne). Durch Hinzufügen von Fakten, die einen aktuellen Zustand der Welt abbilden, kann man weiteres Wissen über diesen aktuellen Zustand durch Spezialisierung des Domänenwissens ableiten (deduzieren, siehe Kapitel 1.1). Jeder neue Zustand (Wissensstand) enthält das unveränderte Wissen des Vorgängerzustandes, "alte" Fakten und deduzierte neue Fakten.
Verwendet man obige Implikationen als Regeln in einer Wissensbasis, kann bei der Belegung $P = 1$ und $R = 1$ sowohl Q als auch $\neg Q$ geschlußfolgert werden.
Q kann nicht gleichzeitig sowohl wahr als auch falsch sein. Sehr wohl können aber P und R gleichzeitig wahr sein. Eine der Regeln muß daher unzulässig sein. Da für Q kein Wahrheitswert vorgegeben ist (dieser soll ja erst durch die Regeln ermittelt werden), kann nicht überprüft werden, welche der Regeln zulässig ist und welche nicht. Konsistenz muß in regelbasierten Systemen also enger gefaßt werden, als in der formalen Aussagenlogik.

Konsistenz liegt bei regelbasierten Systemen vor, wenn es keine Belegung mit Wahrheitswerten gibt, die die Bedingungsteile von Regeln erfüllt, deren Schlußfolgerungen oder Aktionen aber widersprüchlich sind (nach [1]).

Auf dieser Definition der Konsistenz von regelbasierten Systemen baut die Arbeit von Beauvieux ([35] und [1]) auf. Darin wird ein Verfahren entwickelt, mit dem eine aus Regeln in Aussagenlogik bestehende Wissensbasis nach und während ihrer Erstellung auf Konsistenz getestet werden kann. In Abbildung 2.1 ist das Ablaufdiagramm des Tests angegeben.

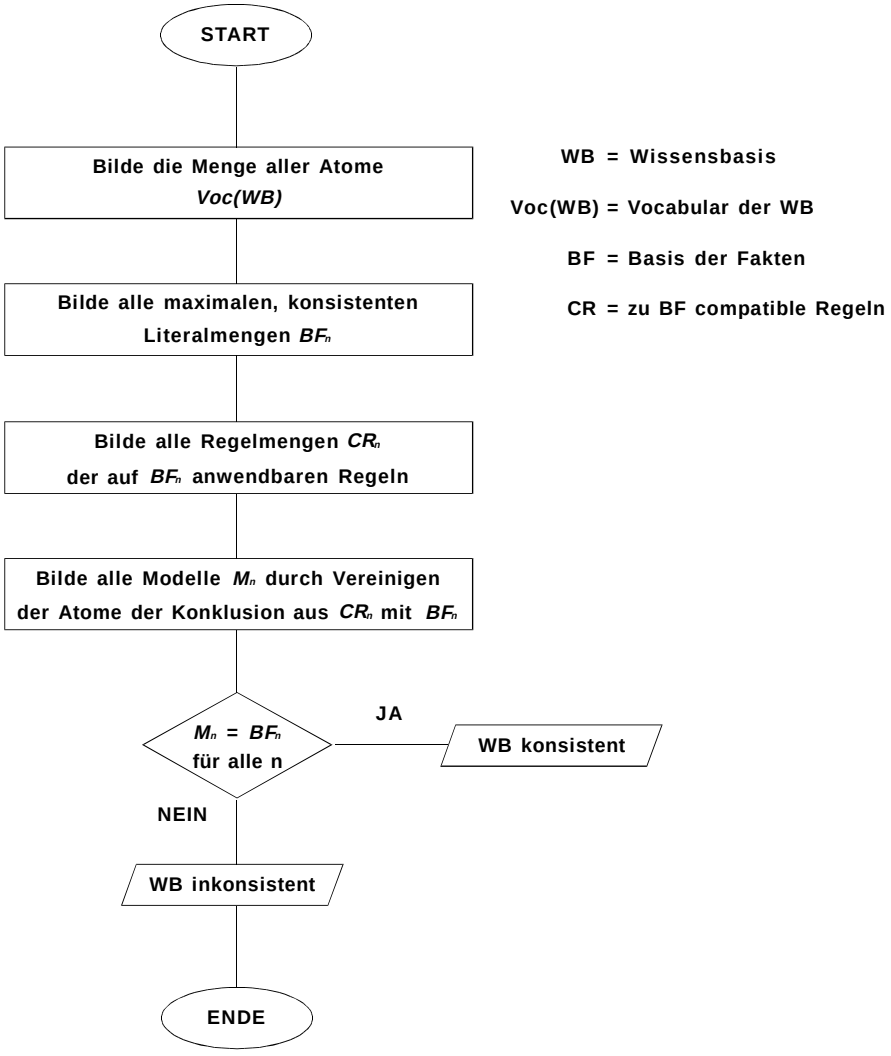


Abbildung 2.1: Konsistenzprüfung nach Beauvieux, nach [1]

Alle in den Regeln auftretenden Atome (siehe Kap. 1.1.1, Seite 5; in Angele/Studer [1] im Gegensatz zur gebräuchlichen Praxis in der Computational Intelligence mit "Literal" übersetzt) bilden zusammen das Vokabular $Voc(WB)$ der Wissensbasis (WB). Da auch komplementäre Atome im Vokabular auftreten können, werden alle maximalen, konsistenten Atommengen (Faktenmengen, "Base of facts" BF) gebildet. Konsistent bedeutet hier: frei von komplementären Atomen. Maximal heißt, daß jede Obermenge von BF inkonsistent ist. Zu jeder Atommenge BF_n wird die Menge aller Regeln ("maximal compatible rules" CR_n) gebildet, die auf BF_n angewendet werden (feuern) können. Die Atome der Bedingungsteile aus CR_n sind dementsprechend in BF_n enthalten. Durch die Vereinigung der Menge BF_n mit der Menge der Atome der Konklusionen aus CR_n entsteht ein "Modell" M_n (eine Instanz) der Wissensbasis.

Konsistenz ist dann gegeben, wenn alle Modelle M_n konsistent sind, was durch $M_n = BF_n$ gezeigt wird.

Wird eine neue Regel einer konsistenten (getesteten) Wissensbasis zugefügt, muß der Konsistenztest erneut durchgeführt werden. Die bisher gebildeten Modelle werden um die Atome der neuen Regel erweitert und dann als Vokabular interpretiert, das dem Konsistenztest nach Abbildung 2.1 unterworfen wird. Die um die neue Regel erweiterte Wissensbasis ist dann konsistent, wenn alle neuen Modelle M'_n konsistent sind.

Einsatzgebiete solcher Deduktionssysteme sind Beratungs-, Informations- und Diagnosesysteme. Hier wird mit bekanntem Basiswissen gearbeitet und anhand der gegebenen Situation implizites Wissen für den Benutzer explizit dargestellt (ausgegeben). Man könnte diese Systeme als automatische Handbücher interpretieren, da sie auf bestimmte Wissensgebiete und Situationen zugeschnitten sind.

Innerhalb der Automatisierungstechnik sind zusätzlich solche Systeme interessant, die "allgemeine" ("Das ist bei allen ... so" oder "Bei irgend einer ... ") Aussagen verarbeiten, also eher mit einem prädikatenlogischen Ansatz in Verbindung gebracht werden können:

> Wenn **eine** Pumpe eingeschaltet ist, steigt der Füllstand **eines** angeschlossenen Tankes
> $\Longrightarrow$
> für alle Pumpen und alle daran angeschlossenen Tanks gilt: wenn die Pumpe **x** eingeschaltet ist, steigt der Füllstand des Tankes **y**.

Außerdem soll die Dynamik (Zeitbezug) der technischen Systeme in der Wissensbasis abgebildet werden können d.h., daß Änderungen der Wahrheitswerte durch Schlußfolgerungen möglich sein müssen:

> Wenn eine Pumpe P1 eingeschaltet und ein Tank an ihrem Ausgang voll ist, dann wird diese Pumpe P1 abgeschaltet.

Dazu kann die Implikation in aussagen- oder prädikatenlogischen Sätzen statt als rein logische Verknüpfung als zusätzlich zeitliche Verknüpfung interpretiert werden:

> Wenn *vorher* A gültig ist, dann ist *nachher* B gültig, wobei B = ¬ A erlaubt ist.

Erfolgt die Beschreibung des betrachteten Weltausschnittes (technischen Systems) mit prädikatenlogischen Sätzen, ist nach einer Umwandlung der Sätze in eine Klauselmenge (Kapitel 1.1.2) der Konsistenztest für diese Systeme mit der **C**onnection **G**raph **P**roof **P**rocedure (CGPP) anwendbar [41] [17].
Mit Hilfe der CGPP kann für eine Klauselmenge eine Inkonsistenz mit einer endlichen Anzahl von Schritten festgestellt werden. Für technische Systeme existieren trotz der "Semi-Entscheidbarkeit" der Prädikatenlogik erster Stufe Verfahren, mit denen auch die Konsistenz mit einer endlichen Anzahl von Schritten nachgewiesen werden kann [15].

2.3 Konsistenz in hybriden Systemen

Die Repräsentationsformen **Regeln**, **Frames** bzw. **strukturierte Objekte** und **Prozeduren** stehen in den hybriden Systemen nicht unabhängig nebeneinander, sondern greifen auf unterschiedliche Art und Weise ineinander.

In den folgenden Abschnitten wird jeweils eine Repräsentationsform betrachtet und überprüft, in wie weit sich eine dort gültige Definition des Konsistenzbegriffes auf eine bereits bekannte Definition abbilden läßt. Des weiteren wird untersucht, ob und in wie weit bekannte Testverfahren für die Kombination der Repräsentationsformen anwendbar sind.

Daran anschließend wird zusammengestellt, welche Anforderungen ein "gutes" Werkzeug erfüllen muß, um ein hohes Maß an Zuverlässigkeit oder Konsistenz des entstehenden Systems zu ermöglichen.

2.3.1 Regeln

Die Formulierung von Wissen in Form von Regeln erreicht eine hohe Akzeptanz sowohl bei den Entwicklern als auch bei den Benutzern von wissensbasierten Systemen. Regelbasierte Systeme waren daher in der Informatik Gegenstand von Untersuchungen zu verschiedenen Themen, unter anderem auch zur Konsistenz.

In einigen Arbeiten wie z.B. Ginsberg [38], Moerkotte [20], Beauvieux [35] und Angele/Studer [1] wird versucht, Bedingungen für konsistente Wissensbasen zu erarbeiten und Sprachen oder Kalküle zu entwickeln, die Konsistenz sicherstellen.
In diesen Arbeiten werden Wissens- oder Datenbasen untersucht, die aus "aussagenlogischen Regeln" oder Hornregeln bzw. Horn-Klauseln bestehen. Die Regeln werden dabei ausschließlich als "Schlußfolgerungsregeln" (siehe Lunze [19], S. 78) verwendet.

Moerkotte formuliert dazu:
"Die Regeln dienen der Reduktion des Datenbestandes, da nur die expliziten Daten in die Datenbasis aufgenommen werden müssen."
(Zitat aus Moerkotte [20], Seite 2).

Die Abbildung technischer Systeme mit Regeln erfordert aber die Verwendung von Aktionsregeln wie z.B. die Regel aus Kapitel 1.2 (Seite 11):

$$\forall_p \forall_t \qquad angeschlossen(p,t) \wedge ein(p,T0) \wedge läuft_über(t,T0) \implies aus(p,T1)$$

Zum einen wird hier die prädikatenlogische Formulierung mit Quantoren verwendet und zum anderen ist ein Zeitbezug in den Prädikaten enthalten ($T0, T1$). Die oben zitierten Ansätze müßten dafür erweitert werden.

Für diese Gegebenheiten ist ein Konsistenztest mit der CGPP (siehe Kapitel 2.2.2) ohne große Erweiterungen anwendbar. An die Prädikate bzw. die für die CGPP benötigten Klauseln werden aber zusätzliche Anforderungen gestellt, die bei hybriden Systemen mit der Abbildung von Objekten zusammenhängen, die im weiteren näher betrachtet werden.

2.3.2 Frames

In den Bedingungsteilen der Regeln werden Eigenschaften des momentanen Zustandes (der Datenbasis) abgefragt. Dieser Zustand wird durch Fakten beschrieben, deren Repräsentationsform den Benutzerwünschen weitgehend entsprechen soll.

Jeder Mensch neigt dazu, Fakten zu sinnvollen Gruppen zusammenzufassen. Sei es, daß Eigenschaften einem physikalischen Gegenstand oder einzelne Fakten einem abstrakten Begriff (z.B. Arbeitsgruppe) zugeordnet werden.
Diesem Verhalten kann man in Expertensystemen durch die Verwendung von frame- oder objektbasierten Repräsentationsformen entgegenkommen (Kapitel 1.3.2). Frames können als "kleinste Gemeinsamkeit" von Frames und strukturierten Objekten verstanden werden und werden daher weiter untersucht.

Die an einem Problem beteiligten physikalischen Objekte lassen sich leicht finden und benennen. Auch die notwendigen Eigenschaften können formuliert und zugeordnet werden. Betrachtet man ganz allgemein die Repräsentation von Fakten mit Hilfe von Frames, können bei der Implementierung und Verwendung der Frames verschiedene Inkonsistenzen auftreten.

Ein Problempunkt ist die mehrfache Implementierung ein und desselben realen Objektes, beispielsweise durch verschiedene Entwickler zu verschiedenen Zeiten. Des weiteren kann eine Eigenschaft eines Objektes durchaus durch unterschiedliche Bezeichnngen (Attributnamen) abgebildet werden und schließlich kann einem Attribut ein Wert zugewiesen

werden (z.B. durch eine Berechnung), der in der realen Welt nicht vorkommt und daher für dieses Attribut nicht erwartet und weiterverarbeitet wird.

Alle drei Punkte müssen in einem konsistenten System verhindert werden, oder als **Forderungen** formuliert:

- ein physikalisches Objekt darf nur einmal implementiert werden, **(F-Fra. 1)**
- eine Eigenschaft muß durch genau ein Attribut mit genau einem Namen abgebildet werden, **(F-Fra. 2)**
- einem Attribut darf nur ein Wert zugewiesen werden, der in der erlaubten Wertemenge liegt. **(F-Fra. 3)**

Bei der Repräsentation von Objekten mit ihren Attributen (Eigenschaften) gibt es kaum Einschränkungen bzw. Vorgaben, die bei der Erfüllung dieser Forderungen helfen. Ein rotes Auto kann z.B. folgendermaßen als Frame implementiert werden:

a)	FRAME:	Auto	b)	FRAME:	Auto
	ATTRIBUT:	rot		ATTRIBUT:	Farbe
	WERT:	{ja,nein}		WERT:	{rot,grün,...}

Die erste Formulierung gestattet nur boolesche Attributwerte, während die zweite eine freie Formulierung, sowie eine beliebige Anzahl von Werten zuläßt.

Niemand kommt auf die Idee, einen Autobesitzer zu fragen: "Ist das Auto rot? Ist das Auto grün? ...". Die normale Fragestellung ist sicher: "Welche Farbe hat das Auto?" . Die angenehmere Form der Abbildung eines Objekts wird für einen "normalen" Benutzer daher die natürlichere Form b) sein.

Für einen Konsistenztest kommt im Hinblick auf die Verwendung bekannter Verfahren aus logik- oder regelbasierten Systemen zunächst nur die Formulierung a) in Frage. Im folgenden werden daher beide Formulierungen weiter untersucht.

Je nach gewählter Formulierung (im folgenden auch als Form oder Muster bezeichnet) muß sich die Syntax der Regeln im Bedingungsteil unterscheiden:

a) **Wenn** das Auto rot ist **dann** ...
b) **Wenn** die Farbe des Autos rot ist **dann** ...

Betrachtet man nun den Aktionsteil folgender Regel

> **Wenn** das Auto neu lackiert wird **und** die Farbe des Lackes grün ist,
> **dann** ist die Farbe des Autos (nachher) grün.

in beiden Schreibweisen, so werden unterschiedliche Abläufe notwendig.

Bei einer Implementierung nach Muster a) bedeutet diese Regel eine Änderung von zwei Attributen des Frames Auto:

ATTRIBUT: **rot** WERT: **ja → nein**
ATTRIBUT: **grün** WERT: **nein → ja**

Außerdem wird vorausgesetzt, daß das Attribut grün existiert!

Eine einfache Wertänderung erfüllt bei Muster b) den gleichen Zweck:

ATTRIBUT: **Farbe** WERT: **rot → grün**

Hier muß sichergestellt sein, daß grün ein zulässiger Wert des Attributes ist.

Der Aufwand bei der Frameerstellung sowie der Umfang des realisierten Systems unterscheiden sich für die beiden Muster sehr stark. Um verschiedene Farben eines Autos verarbeiten zu können, muß im Fall a) jede erlaubte Farbe als eigenständiges Attribut des Frames vorhanden sein, während im Fall b) ein einziges Attribut mit einer begrenzten, aber erweiterbaren Wertemenge ausreicht. Die Implementierung b) ist also bei der Erstellung des wissensbasierten Systemes hinsichtlich des Arbeits- und Speicheraufwandes von Vorteil.
Die Erweiterung eines bestehenden Systems ist bei Verwendung von Muster a) nur unter erheblichem Aufwand möglich. Zum einen muß die Klasse um Attribute erweitert werden, zum anderen muß diese Erweiterung durchgängig bei allen Instanzen vorgenommen werden. Bei Verwendung von Muster b) muß lediglich ein neuer zulässiger Wert für das Attribut in die Klasse eingetragen werden.

Welche Auswirkungen hat die Form der Implementierung von Frames auf einen Konsistenztest für Regeln?
Die Connection Graph Proof Procedure als ein mögliches Testverfahren benötigt beispielsweise die Darstellung des Wissens in Prädikaten. Die im Frame enthaltenen Attribute müssen in Prädikate übersetzbar sein.
Das kann, nach Frameformulierung getrennt, so aussehen:

a) $ist_rot(Auto)$ b) $Farbe(Auto, rot)$

Für negierte Prädikate, wie sie bei der Umwandlung in die Klauselform erzeugt werden (Kapitel 1.1.2) bzw. Regeln, die bei nicht zutreffenden Eigenschaften feuern sollen, gilt:

a) $\neg ist_rot(Auto)$ $\hat{=}$ OBJEKT: **Auto** ATTRIBUT: **rot** WERT: **nein**
b) $\neg Farbe(Auto, rot)$ $\hat{=}$ OBJEKT: **Auto** ATTRIBUT: **Farbe** WERT: **?? ; nicht rot**

Je nach dem, ob die Negationen im Bedingungs- oder Aktionsteil einer Regel auftreten, sind verschiedene Effekte zu beachten:

- Aktionsteil:

  a) Die Negation entspricht einer Wertänderung des Attributes von **ja** nach **nein**, also einer booleschen Negation, während bei

  b) die Negation nicht eindeutig in Bezug auf den Wert des Attributes Farbe ist. Werden nur zweiwertige Attribute mit frei formulierten Werten zugelassen, kann man eine Analogie zu Fall a) schaffen.

OBJEKT:	Auto		
ATTRIBUT:	Farbe	UND	$\neg Farbe(Auto, rot) \Leftrightarrow Farbe(Auto, grün)$
WERT:	{rot,grün}		

- Bedingungsteil:

  a) Der Wahrheitswert des Literals ist sofort durch den Wert des Attributes **rot** gegeben.

  b) Es kann nur dann ein Wahrheitswert ermittelt werden, wenn auch hier die Beschränkung auf zwei Werte gilt.

  oder die Forderung *F-CGPP* erfüllt wird:
  Verwendung einer begrenzten Wertemenge mit mehr als zwei Werten **und** Formulierung von Ausschlußregeln der Form:

  Wenn die Farbe des Autos x ist, **dann** ist die Farbe des Autos **nicht "alle anderen möglichen Farben".** **(F-CGPP)**

Bei vielen Eigenschaften ist es sicher unproblematisch, nur zwei Ausprägungen für den Wert zu erlauben.
Technische Anwendungen erfordern bei ihrer Implementierung oft erweiterte Möglichkeiten. So bietet sich beispielsweise die Verwendung von "Prozeß"-Frames mit dem mehrwertigen Attribut **Prozeßzustand** an, um kontinuierliche Prozesse zu beschreiben. Analog dazu eignen sich "Chargen"-Frames mit dem mehrwertigen Attribut **Bearbeitungszustand** oder **Status** bei automatisierten Chargenprozessen zur Repräsentation.

Zusammenfassend kann gesagt werden:

- Objekte werden in Frames der Form

  OBJEKT: Auto
  ATTRIBUT: Farbe
  WERT: {rot,grün,blau,...}

  implementiert und
- die Syntax der Regeln wird im Aktionsteil eingeschränkt auf das direkte (nicht negierte) Setzen von Attributwerten. **(F-Regel)**
- Literale werden in der Form $Attribut(Objekt, Wert)$ gebildet.

Diese Einschränkungen sind im Hinblick auf einen Konsistenztest für Regeln unabdingbar, die Forderungen *F-Fra. 1, F-Fra. 2* und *F-Fra. 3* sind damit aber noch nicht erfaßt. Da physikalische Objekte von den meisten damit vertrauten Menschen auf die gleiche Weise beschrieben werden, liegt das Problem eher bei logischen oder virtuellen Objekten wie z.B. einem Bearbeitungsschritt oder einem Arbeitsablauf. Formal gibt es keine Möglichkeiten, auf dieser Ebene Widersprüche zu verhindern, da das wissensbasierte System keinen Zugang zur Bedeutung der Objekte hat. Die Widerspruchsfreiheit der Objektdefinitionen muß der Wissensingenieur sicherstellen, der die Frames implementiert.

Als Hilfsmittel können eine Einordnung der Frames in eine hierarchische Struktur sowie Listen mit bereits implementierten Frames und deren Attributnamen dienen. Um die Vorteile framebasierten Vorgehens nutzen zu können, ist es sinnvoll, nicht beliebig viele einzelne und von einander unabhängige Frames zu formulieren, sondern eine Frame- bzw. Klassenhierarchie aufzubauen. Ähnliche Objekte werden in Gruppen (Klassen) zusammengefaßt, die durch eine allgemeine Beschreibung der gemeinsamen Eigenschaften charakterisiert wird. Durch spezielle Eigenschaften werden Unterklassen von der Ursprungsklasse und anderen Unterklassen abgegrenzt. Damit fällt es leichter, bei großen Systemen die Übersicht zu bewahren und nicht etwa ein Frame doppelt zu implementieren.

2.3.3 Syntaktischer Test

Die Betrachtungen zur Widerspruchsfreiheit in wissensbasierten Systemen im technischen Umfeld haben die Äquivalenz von Regeln und prädikatenlogischen Sätzen verdeutlicht.

Die in diesem Zusammenhang durchgeführte Betrachtung der Faktenrepräsentation mittels Frames führte bereits zu Einschränkungen bzw. Randbedingungen, die bei der Formulierung der Regeln (F-CGPP und F-Regel) und Erzeugung der Frames (F-Fra 1 bis 3) beachtet werden müssen.

Eine komfortable Entwicklungsumgebung sollte dem Entwickler die Prüfung solcher Grundbedingungen abnehmen, da sie nicht speziell zu dem Weltausschnitt (der Domäne) gehören, der im wissensbasierten System abgebildet werden soll.
Die Menge der vom System geleisteten Prüfungen bei der Erstellung der "Wissenselemente" (s. u.) kann als erweiterter syntaktischer Test interpretiert werden. Eine solche Testkomponente kann aufgrund des Entwicklungszeitpunktes und des Vorgehens an dieser Stelle auch als "Editor zur Akquisition des Wissens" bezeichnet werden.

Im folgenden werden die Anforderungen an eine solche Komponente nach Wissensrepräsentationsformen, deren implementierte Einheiten (z.B. eine Regel oder ein Frame) auch als **Wissenselemente** bezeichnet werden, getrennt zusammengestellt.

Regeln

An dieser Stelle kann das wissensbasierte System eine effektive Hilfe bei der Entwicklung bieten, indem bei der Eingabe von Regeln Hilfestellungen gegeben werden. Neben Vorschlägen zur reinen Regel**syntax** müssen dabei auch relevante Teile des bereits implementierten domänenspezifischen Wissens angezeigt werden.

Der Entwickler der Wissensbasis (auch als Wissensingenieur bezeichnet) wird beispielsweise dadurch unterstützt, daß zu einem neu eingegebenen Bedingungsteil einer Regel alle Stellen angezeigt werden, an denen die abgefragten Attribute ebenfalls Verwendung finden. Auf diese Weise können Bedingungsteile erkannt werden, die Ober- oder Untermengen der neuen Bedingungen enthalten und somit redundantes Wissen verkörpern.

Es werden durch die Anzeige der zugehörigen Aktionsteile aber auch Widersprüche erkannt. Beispielsweise kann so die Eingabe der beiden Implikationen aus Kapitel 2.2.1 (Seite 30) als Regeln vermutlich verhindert werden.
Sei die Implikation "$P \Rightarrow Q$" bereits als Regel "*R1*" eingegeben worden.
Wird nach der Eingabe des Bedingungsteils "$P \wedge R$" "*R1*" angezeigt, weil in deren Bedingungsteil ebenfalls das Attribut "P" enthalten ist, hat der Wissensingenieur die

Möglichkeit, die Inkonsistenz, die durch den Aktionsteil "$\neg Q$" entstehen würde, zu erkennen und entsprechend zu reagieren. Da bei einer großen Zahl von Regeln die Anzeige unübersichtlich werden kann, sollte zusätzlich ein formaler Test der gesamten Regelbasis vorgenommen werden.

Wie bereits in Kapitel 2.3.2 dargelegt wurde, ist die **C**onnection **G**raph **P**roof **P**rocedure (CGPP) unter bestimmten Bedingungen (Erfüllung der Forderungen F-CGPP und F-Regel) als Konsistenztest für Regelbasen verwendbar. Die Testkomponente muß die Aufbereitung der Regeln in eine für die CGPP verwendbare Form übernehmen. Konkret bedeutet das eine syntaktische Analyse der Regeln mit gleichzeitiger Umwandlung in prädikatenlogische Formeln und dann deren Umwandlung in Klauseln. Die Klauseln schließlich können mit Hilfe der CGPP auf Konsistenz getestet werden.

Weiterhin muß eine adäquate Reaktion auf eine nachgewiesene Inkonsistenz erfolgen.
Werden Redundanzen bei der Erzeugung von Klauseln vermieden, indem eine bestimmte Klausel immer nur einmal existiert, egal aus wievielen prädikatenlogischen Formeln sie entstehen kann, ist nur schwer feststellbar, wo die Ursache der Inkonsistenz liegt. Alle Regeln, deren Umwandlung zu einer Klausel führt, die an der Inkonsistenz beteiligt ist, sind potentiell inkonsistent. Wird erst nach der Implementierung vieler Regeln ein Konsistenztest mit der CGPP gestartet, können mehrere Inkonsistenzen vorliegen, die alle behoben werden müssen.

Sinnvoll erscheint daher der sofortige Test nach Eingabe einer neuen Regel. Auf diese Weise wird zu einer Regelmenge, die als konsistent erkannt wurde, eine einzelne neue Regel zugefügt, die beim Auftreten einer Inkonsistenz als deren (Teil-)Ursache feststeht. Nachteil dieses Verfahrens ist der zusätzliche Aufwand für einen kompletten Konsistenztest nach jeder Regeleingabe. Wird von jeder Regel die einmal erstellte Klauselform gespeichert, muß nur die letzte Regel transformiert und der Test mit der CGPP durchgeführt werden.

Frames

Die Erzeugung neuer Frames setzt sehr viel Überblick über bereits vorhandene Wissenselemente sowie ihre Hierarchie voraus.
Eine einfache und wirkungsvolle Hilfe ist die Anzeige der vorhandenen Klassen-, Frame- und Attributnamen. Dadurch ist der Wissensingenieur in der Lage, auf vorhandene Frames oder einzelne Attribute als Vorlage oder als Erinnerung zur besseren sprachlichen

Abgrenzung zurück zu greifen. Verwirrende Namensvielfalt für ein und die selbe Eigenschaft bei verschiedenen Frames wie z.B. "laufende_Nummer", "Nummer", "Nr.", "ID" oder ähnliches für eine Kennziffer oder einen Index können vermieden werden.
Damit ist die Forderung *F-Fra. 2* (Seite 36) erfüllt!

Gleichzeitig kann eine Hierarchie der Frames durch Bildung neuer Klassen oder Zuordnung zu bereits bestehenden Klassen mit Unterstützung des Systems aufgebaut werden. Der Wissensingenieur kann aus den Vorschlägen des Systems eine zutreffende Zuordnung auswählen oder mit Hilfe dieses Überblickes eine neue Klasse innerhalb der Hierarchie anordnen.
Eine hierarchische Strukturierung der Klassen erleichtert teilweise auch die Formulierung von Regeln, da mit der Verwendung von Oberklassen allgemeingültigere (generische) Regeln aufgestellt werden können. Statt eine Regel beispielsweise für LKW, eine für PKW und eine für KRAD aufzustellen, kann eine generische Regel "für alle Kraftfahrzeuge" formuliert werden. Das verringert die Schreibarbeit und somit die Fehlerwahrscheinlichkeit und erhöht gleichzeitig die Übersichtlichkeit.
Durch eine Analyse der im aktuellen Frame benutzten Attribute und die Auswertung der bereits vorhandenen Klassendefinitionen ist eine formale Einordnung in die Klassenhierarchie automatisch durch die Entwicklungsumgebung möglich.
Somit ist auch die Forderung *F-Fra. 1* (Seite 36) erfüllt!

Attribute

Die Fehlerminimierung hinsichtlich der Attributnamen wird durch die Wahlmöglichkeit aus bestehenden Namen unterstützt. Die möglichen Wertemengen der Attribute bedürfen aber noch der Betrachtung (Forderung *F-Fra. 3*, Seite 36).

Wie bereits früher begründet wurde (Kap.2.3.2), sind Attribute mit mehr als einem Wert und seiner Negation schwierig zu behandeln, aber sehr wünschenswert und werden daher hier weiter betrachtet.
Bereits beim Anlegen der Attribute müssen einige zusätzliche Aktionen systemintern ablaufen. Für den Wissensingenieur ist nur von Interesse, welche Werte ein Attribut annehmen kann bzw. darf. Implizites Wissen über die Beziehungen der Werte untereinander muß dem System zusätzlich mitgeteilt werden. Ein "Attribut-editor", der einen syntaktischen Test ermöglichen und einen Zulässigkeitstest sofort durchführen soll, muß dieses Wissen sozusagen "selbst akquirieren".

Bei den Wertemengen für Attribute sind zwei Fälle zu unterscheiden:

1. sich ausschließende Werte,
2. Werte, die in einer definierten Reihenfolge vorliegen.

Zu 1: Die Farbe eines einfarbigen Gegenstandes kann immer nur genau einen Wert annehmen. Sind also die Werte ROT, BLAU und GRÜN als zulässige Werte deklariert, so bedeutet das für jede Farbe den Ausschluß der beiden anderen Farben.

Wird also im Verlauf einer Anwendung des Systems von verschiedenen Stellen die Farbe des selben Gegenstandes auf unterschiedliche Werte gesetzt, so entspricht das einer Inkonsistenz, da bei gleichen Bedingungen (dem gleichen Systemzustand) zwei verschiedene Werte geschlossen werden.
Durch die zusätzliche Information "gegenseitiger Ausschluß" können z.B. Regeln als inkonsistent erkannt werden, die bei gleichem Bedingungsteil verschiedene Werte im Aktionsteil setzen (siehe Kapitel 2.3.3.1).

Zu 2: Als Attributwert eines Attributs "Temperatur" wird ein Symbol für einen Temperaturbereich verwendet . Sind beispielsweise die Symbole {"kalt", "kühl", "warm", "heiß"} als Wertemenge angegeben, kann die Reihenfolge als implizite Angabe einer Relation (z.B. "ist-günstiger-als" bei Lesart von links nach rechts oder "ist-höher-als" bei Lesart von rechts nach links) zwischen den Symbolen verstanden werden.

Weiterhin ist hier nichts über die Zuordnung eines numerischen Wertes zu einem oder eventuell auch mehreren Symbolen ausgesagt. Dies hat Konsequenzen für die Konsistenzbetrachtungen, was im Unterpunkt "Symbolfolgen" näher untersucht wird.

Bei beiden Arten der Wertemenge ist es für den Konsistenztest wichtig, daß alle möglichen Werte als Elemente der Menge angegeben werden. Andernfalls kann der Wissensingenieur beispielsweise durch einen Schreibfehler im Bedingungsteil einer Regel Attribute auf Werte abfragen, die niemals auftreten. Andererseits bedeutet ein Schreibfehler im Aktionsteil einer Regel die indirekte Deklaration eines neuen Attributwertes, der niemals ausgewertet wird, z.B. Attribut FARBE wird auf BLAI statt BLAU gesetzt.

Symbolfolgen

Wie bereits in Kapitel 2.3.2 begründet, sind frei formulierbare Attributwerte in einem anwenderfreundlichen Expertensystem rein booleschen Werten vorzuziehen. Einige sich daraus ergebende Probleme wurden dort ebenfalls bereits angesprochen.

Einen Sonderfall stellen nun implizite Relationen zwischen Symbolen dar. Wird beispielsweise für das Attribut "Temperatur" des Frames "Motor" die Wertemenge { "kalt", "kühl", "warm", "heiß" } angegeben, sind die einzelnen Symbole zulässig. Setzt man weiterhin voraus, daß es verschiedene Regelgruppen (Gruppe 1, Gruppe 2) gibt, die für ein solches Attribut einen Wert setzen, kann folgendes Szenario entwickelt werden:

Gruppe 1:
WENN A ∧ B ∧ C
DANN wird der "Temperatur" von "Motor" das Symbol "warm" zugewiesen.

Gruppe 2:
WENN A ∧ E
DANN wird der "Temperatur" von "Motor" das Symbol "heiß" zugewiesen.

Formal können die beiden Regeln entweder durch eine sinnvolle Kombination der Prämissen in eine Regel zusammengefaßt oder durch Ergänzung der Prämissen mit der negierten Prämisse der jeweils anderen Regel gegeneinander verriegelt werden:

Gruppe 1:
WENN A ∧ B ∧ C ∧ ¬E
DANN wird der "Temperatur" von "Motor" das Symbol "warm" zugewiesen.

Gruppe 2:
WENN A ∧ E ∧ ¬B ∧ ¬C
DANN wird der "Temperatur" von "Motor" das Symbol "heiß" zugewiesen.

Der Entwickler kann aber auch bewußt die beiden Regelgruppen unabhängig von einander erstellt haben, um beispielsweise einen kritischen Zustand durch eine Reihe von "Sicherheitsregeln" (z.B. Gruppe 2) zu erkennen und entsprechend zu handeln. In diesem Fall ist es notwendig, dem System für einen "Konfliktfall" Wissen über dessen Lösung zur Verfügung zu stellen.

Dabei sind zwei Vorgehensweisen denkbar:

1. das Attribut "Temperatur" wird in der Prämisse mit abgefragt und nur dann neu gesetzt, wenn der neue Wert "besser" im Sinne von "sicherer für die korrekte Funktion des Systems" ist. Soll beispielsweise ein Motor vor dem Überhitzen geschützt

werden, ist es besser, eine bestimmte Temperatur zu hoch zu bewerten ("worst case"). Dem Attribut "Temperatur" wird also "im Zweifel" (wenn mehrere Regeln feuern) besser der höhere Wert zugewiesen:

WENN A ∧ E ∧ "Temperatur" von "Motor" ist "kalt"
DANN wird der "Temperatur" von "Motor" das Symbol "warm" zugewiesen.

WENN A ∧ E ∧ "Temperatur" von "Motor" ist "kühl"
DANN wird der "Temperatur" von "Motor" das Symbol "warm" zugewiesen.

Die implizite Relation "warm" ist-höher-als ... zwischen den Symbolen wird so auch in den Regeln berücksichtigt, es muß aber für jede mögliche Kombination der Symbole in Abhängigkeit von der Relation eine eigene Regel erzeugt werden. Bei einer Erweiterung der Wertemenge muß der Entwickler (oder die Akquisitionskomponente) alle betroffenen Regeln überarbeiten, bzw. neue Regeln erzeugen.

2. Umwandlung des Attributes "Temperatur" mit einer Wertemenge bestehend aus Symbolen oder Texten in ein Attribut "Temperatur ist heiß" mit einem Werteintervall { 1 , 4 }, {0 , 10 } oder ähnlichem. Das weitere Vorgehen ist dann analog zu Punkt 1, bietet aber den Vorteil der leichteren Abfrage (z.B. mit ≤) und besseren Erweiterbarkeit der Wertemenge.
   An dieser Stelle können prinzipiell auch Werteintervalle der Form { -1.0 , 1.0 } oder { 0 , 1.0 } zugewiesen werden, wenn die Entwicklungumgebung die Verwendung von Fuzzy-Logic unterstützt. Diese Variante findet hier keine weitere Berücksichtigung.

Insgesamt ist die Verwendung von Symbolfolgen mit impliziten Relationen als Wertemenge eines Attributes ein Sonderfall, der im Rahmen dieser Arbeit nicht berücksichtigt wurde.

2.3.3.1 Die Connection Graph Proof Procedure

In Kapitel 2.2.2 wurde bereits darauf hingewiesen, daß mit Hilfe der CGPP mit einer endlichen Anzahl von Schritten Konsistenz und Inkonsistenz einer Wissensbasis nachgewiesen werden kann, wenn sie in Klauselform vorliegt, und auf prädikatenlogischen Sätzen beruht, die bestimmte Bedingungen hinsichtlich der Konfluenz usw. erfüllen (siehe Kleberhoff [15]). Dieses Verfahren liefert nur dann als Konsistenztest der gesamten Wissensbasis korrekte Aussagen, wenn tatsächlich alles verfügbare Wissen in der Wissensbasis repräsentiert und daraus alle Klauseln (siehe Kapitel 2.3.2) entwickelt wurden.

Das folgende Beispiel zeigt Schritt für Schritt, welches Wissen für den Nachweis einer Inkonsistenz mindestens erforderlich ist. Außerdem wird gezeigt, welche Klauseln bei einer Wertemenge mit mehr als zwei sich gegenseitig ausschließenden Werten notwendig und hinreichend für die CGPP sind.

In den Beispielen wird folgende Symbolik verwendet:

- Konstante werden vollständig in Großbuchstaben geschrieben (ROT)
- Variable beginnen mit einem Großbuchstaben und setzen sich sonst nur aus Kleinbuchstaben zusammen (Person)
- eine Negation wird durch ein "$\neg$" vor dem Literal gekennzeichnet
- Literale werden in der Form $\boxed{\text{Farbe(Objekt, ROT)}}$ geschrieben
- Klauseln aus mehreren disjunktiv verknüpften Literalen werden als Kette der Literale dargestellt: $\boxed{\text{Farbe(Objekt, ROT)}}\boxed{\text{Farbe(Objekt, BLAU)}}$

  wobei alle Variablen einem Allquantor unterliegen (siehe Kapitel 1.1.2)

Beispiel 1

Inkonsistente Aussagen:

Kaufabsicht(Person, Objekt) $\Rightarrow$ Farbe(Objekt, ROT)

Kaufabsicht(Person, Objekt) $\Rightarrow$ Farbe(Objekt, BLAU)

Kaufabsicht(EVA, AUTO) ist wahr

Aufgrund der Löschregel für isolierte Literale ist der Connection Graph dieser Aussagen leer. Ein leerer Connection Graph ist aber ein Zeichen für **Konsistenz** (Die Begründungen für die hier getroffenen Aussagen finden sich in [15] Kapitel 3.3).

Offensichtlich wurde nicht alles Wissen, das zur Erkennung der Inkonsistenz notwendig ist, in Klauseln umgewandelt. Der leere Connection Graph hat seine Ursache in den fehlenden Links der Literale **Farbe(Objekt, BLAU)** und **Farbe(Objekt, ROT)**. Der gegenseitige Ausschluß der Farben ROT und BLAU kann entsprechende Literale liefern.

Zusätzliche Aussage zum gegenseitigen Ausschluß:

Kaufabsicht(Person, Objekt) $\Rightarrow$ Farbe(Objekt, ROT)

Kaufabsicht(Person, Objekt) $\Rightarrow$ Farbe(Objekt, BLAU)

Kaufabsicht(EVA, AUTO) ist wahr

Farbe(Objekt, ROT) $\Rightarrow$ ¬Farbe(Objekt, BLAU)

Die CGPP liefert einen Connection Graph, der nur eine leere Klausel enthält. Die leere Klausel (NIL) ist das Kennzeichen einer Inkonsistenz.

Durch die zusätzliche Formulierung der Aussage "ROT ist **nicht** BLAU" liefert die CGPP also das erwartete Ergebnis. Besteht die Wertemenge eines Attributes aus zwei Werten, entspricht diese Aussage der Deklaration von beliebigen Begriffen (oder Symbolen) als Synonyme für die booleschen Werte true und false. Enthält die Wertemenge beliebig viele Werte, muß die Aussage "Es gilt immer nur **genau** ein Wert der Menge" formuliert werden.

Für das Attribut "Farbe" eines beliebigen "Objektes" sei die Wertemenge $\{\text{ROT, BLAU}\}$ gegeben.

Aussagen über einzelne Zusammenhänge:

a) Farbe(Objekt, ROT) $\Longrightarrow$ ¬Farbe(Objekt, BLAU)

b) Farbe(Objekt, BLAU) $\Longrightarrow$ ¬Farbe(Objekt, ROT)

c) Farbe(Objekt, ROT) $\vee$ Farbe(Objekt, BLAU)

d) Farbe(Objekt, ROT) $\Longleftrightarrow$ ¬Farbe(Objekt, BLAU)

Klauseln:

a) | ¬Farbe(Objekt, ROT) | ¬Farbe(Objekt, BLAU) |

b) | ¬Farbe(Objekt, BLAU) | ¬Farbe(Objekt, ROT) |

c) | Farbe(Objekt, ROT)) | Farbe(Objekt, BLAU |

d) i) | ¬Farbe(Objekt, ROT) | ¬Farbe(Objekt, BLAU) |

ii) | Farbe(Objekt, BLAU) | Farbe(Objekt, ROT) |

Es ist ersichtlich, daß für die CGPP die Äquivalenz (Aussage d)) als vollständige Abbildung des gegenseitigen Ausschlusses in Frage kommt, da hier sowohl Aussage a) und c) als auch Aussage b) und c) enthalten sind. Dadurch ist, wie in der Mathematik über die Formulierung der Äquivalenz als "dann und nur dann", die Forderung nach "genau" einem gültigen Wert in den Klauseln ausgedrückt worden. Die Klausel i) drückt die Bedingung "höchstens" ein gültiger Wert aus, in der Klausel ii) ist "mindestens" ein Wert formalisiert.

Für eine Wertemenge der Form $\{$ROT, BLAU, GRÜN, GELB$\}$ ergeben sich folgende **Aussagen für alle Ausschlüsse:**

e) Farbe(Obj., ROT) $\iff$ ¬Farbe(Obj., BLAU) $\wedge$ ¬Farbe(Obj., GRÜN) $\wedge$ ¬ Farbe(Obj., GELB)

f) Farbe(Obj., BLAU) $\iff$ ¬Farbe(Obj., ROT) $\wedge$ ¬Farbe(Obj., GRÜN) $\wedge$ ¬ Farbe(Obj., GELB)

g) Farbe(Obj., GRÜN) $\iff$ ¬Farbe(Obj., ROT) $\wedge$ ¬Farbe(Obj., BLAU) $\wedge$ ¬ Farbe(Obj., GELB)

h) Farbe(Obj., GELB) $\iff$ ¬Farbe(Obj., ROT) $\wedge$ ¬Farbe(Obj., BLAU) $\wedge$ ¬Farbe(Obj., GRÜN)

Zur Formalisierung einer Wertemenge mit gegenseitigem Ausschluß sind alle Klauseln mit negierten Literalen zu erzeugen, die sich aus der Kombination aller n Elemente der Menge zu Klassen mit r Elementen (zur r-ten Klasse) ohne Wiederholung ergeben. Zusätzlich muß eine Klausel mit den nicht negierten Literalen für alle Werte erzeugt werden. Bei dem hier gezeigten Beispiel sind also aus 4 Werten bzw. den damit gebildeten negierten Literalen Klassen (Klauseln) mit 2 Elementen (Literalen) zu erzeugen.

Damit ergeben sich:

$C_r(n) = \binom{n}{r} = \binom{4}{2} = 6$ Klauseln mit negierten Literalen

plus eine Klausel mit allen vier nicht negierten Literalen.

Bisher ist sicher gestellt, daß Inkonsistenzen durch Regeln, die das gleiche Attribut eines Objektes auf verschiedene Werte setzen, erkannt werden können.
Die Klauseln, die den Systemzustand repräsentieren (Fakten), bestehen jeweils aus einem Literal, das aus dem bekannten Wert eines Attributes eines konkreten Objektes erzeugt wird. Auf diese Weise müssen alle Attribute aller Objekte mit bekanntem Wert in die Klauselform überführt werden. Im ersten dargestellten Connection Graph ist daher das für dieses Beispiel relevante Faktum "Eva hat die Kaufabsicht für ein Auto" als Klausel $\boxed{\text{Kaufabsicht(EVA, AUTO)}}$ enthalten.

In den klassischen logischen Systemen wird immer eine Menge wahrer Aussagen auf eine Menge bekannter (wahrer) Fakten angewandt. Die Schlußfolgerungen sind neue Fakten, die der vorhandenen Menge an Fakten hinzugefügt werden (siehe Kapitel 1.2).
Eine Erweiterung um einen Zeitbezug für die sinnvolle Anwendung der Prädikatenlogik bei technischen Systemen wurde bereits in Kapitel 2.2.1 angesprochen und soll hier anhand eines Beispiels verdeutlicht werden.

Beispiel 2

Eine Verkehrsampel zeigt ein rotes Signal. Nachdem eine Wartezeit abgelaufen ist, wird die Ampel auf Grün geschaltet.

Farbe(Ampel, ROT) $\wedge$ Wartezeit(Ampel, 0) $\Longrightarrow$ Farbe(Ampel, GRÜN)

Farbe(Objekt, ROT) $\Longrightarrow$ $\neg$Farbe(Objekt, GRÜN)

Farbe(A1, ROT) und Wartezeit(A1, 0) sind wahr

Die CGPP liefert für diese Aussagen eine leere Klausel. Die Änderung eines Attributwertes, wie das Umschalten der Ampel, wird mit der CGPP also als inkonsistenter Zustand interpretiert. Das kann nur verhindert werden, wenn die zusätzliche Information der zeitlichen Abfolge in den Klauseln formalisiert wird.

Die Aussagen lauten dann:

Farbe(Ampel, ROT, T0) $\wedge$ Wartezeit(Ampel, 0, T0) $\Longrightarrow$ Farbe(Ampel, GRÜN, T1)

Farbe(Objekt, ROT, Zeit) $\Longrightarrow$ $\neg$Farbe(Objekt, GRÜN, Zeit)

Farbe(A1, ROT, T0) und Wartezeit(A1, 0,T0) sind wahr

Die CGPP liefert hierbei einen nicht leeren Connection Graph, der keine leere Klausel enthält. Die Aussagen werden als konsistent erkannt.
Die Einführung eines Zeitbezuges zur Formalisierung einer "vorher – nachher" Beziehung mit Hilfe der Zeitpunkte T0 (vorher) und T1 (nachher) sowie der Variablen "Zeit" für einen beliebigen, aber in einer Aussage gleichen Zeitpunkt (gleichzeitig), erweitert den Einsatzbereich der CGPP auf Systeme, die technische Abläufe abbilden.

Für die vollständige Untersuchung einer Wissensbasis auf Konsistenz müssen alle denk- bzw. realisierbaren Systemzustände als Startzustand für die Wissensbasis einmal getestet werden.

Die Anzahl der pro Startzustand zu erzeugenden Klauseln ist von der Gesamtzahl der Attribute in der Wissensbasis abhängig. Die Zahl der theoretisch möglichen Startzustände ist wiederum von dem jeweiligen Wertebereich bzw. der Menge möglicher Werte für jedes Attribut abhängig. Zur Begrenzung dieser Menge an Klauseln ist Wissen über die Verknüpfung der einzelnen Attribute und Frames bzw. der abgebildeten Objekte mit ihren Eigenschaften, untereinander erforderlich. Es ist denkbar, ein eigenes wissensbasiertes System zu entwickeln, das die Aufgabe hat, alle technisch sinnvollen Startzustände zu erzeugen. Damit wäre ein vollständiger Konsistenztest vor der Auslieferung des wissensbasierten Systems an den Kunden möglich.

Mit der in hybriden Systemen verwendeten Repräsentationsform der Prozeduren beschäftigen sich die folgenden Abschnitte. Die beim objektbasierten Ansatz verwendeten "Methoden" (siehe Kapitel 1.3.2) bedürfen keiner gesonderten Betrachtung, da Sie aufgrund ihrer Struktur wie Prozeduren behandelt werden können.

2.3.4 Prozeduren

Im Aktionsteil von Regeln können mit den bisher vorgestellten Mitteln Attributwerte einzelner Frames neu gesetzt werden. Schon die Veränderung von mehr als zwei oder drei Werten zieht eine unübersichtliche Syntax der Regel nach sich. Sind zudem die konkreten Werte nicht bekannt, sondern durch Berechnungen zu ermitteln, wächst der Wunsch, Aktionen zusammenfassen zu können. Prozeduren sind das passende Werkzeug zur Erfüllung dieses Wunsches. Eine Regel, die einen Prozeduraufruf im Aktionsteil enthält, kann einen ganzen Komplex von Aktionen auslösen.

Prozeduren können:

1. Attributwerte setzen,
2. Werte berechnen und als Attributwerte setzen,
3. Aktionen in einer vorgegebenen Reihenfolge ausführen,
4. weitere Prozeduren als "Unterprogramme" aufrufen,
5. mathematische Modelle repräsentieren oder enthalten.

Desweiteren können Prozeduren in unterschiedlichen Zusammenhängen und mit unterschiedlichen Parametern durch die Inferenzmaschine aufgerufen werden.

Werden Attributwerte mit Hilfe von Prozeduren gesetzt, verbessert das primär die Übersichtlichkeit der Regeln. Statt in einem Aktionsteil mehrere Wertzuweisungen einzeln aufzulisten, kann eine Prozedur mit möglichst "sprechendem" Namen (der Name deutet die Funktion an) eingesetzt werden. Wertzuweisungen zu verschiedenen Attributen des selben Frames können im Aktionsteil mit nur einem Prozeduraufruf (z.B. *übernimm_werte*(*Auto*, *Leihwagen*1)) erledigt werden. Auch Berechnungen, die auf mathematische Funktionen wie z.B. trigonometrische Funktionen oder Funktionen zum Rechnen mit Kalenderdaten zurückgreifen, lassen sich durch "sprechende" Prozedurnamen abkürzen, wie z.B.

gesamt_leihgebühr(*Leihwagen*1, 24.12.96, 3.1.97)

zur Berechnung der Leihgebühr für den Leihwagen1 in der Zeit vom 24.12.96 bis zum 3.1.97 unter Berücksichtigung von Feiertag- und Wochenendpauschalen.

Bei der Ableitung neuen Wissens kommen Prozeduren besonders dann zum Einsatz, wenn z.B. die Erzeugung neuer Frames mit mehreren Aktionen verbunden ist. Die einfache Information des Benutzers durch Ausgabe eines Textes ist noch mit wenigen Anweisungen realisierbar, die Erzeugung eines komplexen Frames mit neu zu ermittelnden Attributwerten (Punkt 2) und unter Umständen die Anordnung auf einer grafischen Oberfläche dagegen erfordert komplexere Aktionen, die in einer bestimmten Reihenfolge (Punkt 3) erfolgen müssen.

Für den Einsatz von wissensbasierten Systemen in der Prozeßführung ist die Möglichkeit, mathematische Modelle durch Prozeduren in die Anwendungen zu integrieren eine wichtige Eigenschaft (Punkt 5). Funktionale Zusammenhänge zwischen Prozeßgrößen bilden oft die Grundlage für Steuergrößen und sind allein wissensbasiert (regelbasiert) nicht abzubilden. An dieser Stelle findet die Integration von klassischen Modellen in die wissensbasierte Arbeitsweise statt. Wie in den meisten Fällen müssen die Vorteile auch hier durch Nachteile erkauft werden.

Innerhalb der Prozeduren kann üblicherweise auf alle Frames direkt zugegriffen werden, was sowohl für das Auslesen als auch das Setzen von Werten gilt. Geschieht dieser Zugriff über Aufruf- und Rückgabeparameter der Prozedur, können Verknüpfungen verhältnismäßig leicht erkannt werden. Häufiger sind aber Zugriffe innerhalb der Prozeduren. Als Folge davon wird die Definition des Konsistenzbegriffes, sowie die Prüfung und Wahrung der Konsistenz komplexer.

Prozeduren müssen mindestens die folgenden Forderungen erfüllen:

- die Prozedur muß wie ein konventionelles Programm auf Korrektheit überprüft werden [9] [28] und **(F-Proz. 1)**
- das Zusammenwirken der Prozedur mit Regeln und Frames darf nicht zu Widersprüchen führen. **(F-Proz. 2)**

Für die Korrektheit der Prozeduren ist ausschließlich der Programmierer verantwortlich. Eine erste Hilfe bietet bei einigen kommerziellen Systemen ein Editor, der nur syntaktisch korrekte Eingaben zuläßt (kontextsensitiver Editor). Entweder wird schon während der Eingabe die Syntax geprüft, oder nach dem Beenden des Editors wird der gesamte Prozedurtext auf syntaktische Zulässigkeit geprüft.
Korrektheit bedeutet aber mehr als korrekte Syntax, vielmehr ist darunter die Gewährleistung der Funktionssicherheit zu verstehen. Allgemein wird die richtige Funktion von Prozeduren (Programmen) durch praktische Tests überprüft. Die Testdaten werden, je nach Art des Tests (White Box, Black Box, statistischer Test, [22]), auf unterschiedliche Art gewonnen. Egal, ob statistisch Daten generiert werden, die Spezifikation der Prozedur oder ihre implementierte Struktur zugrunde gelegt werden, ein praktischer Test ist nur selten vollständig. Auch ein erfolgreich bestandener Test bedeutet nur, daß kein Fehler gefunden wurde. Um diese Lücke zu schließen, gibt es einige Ansätze zur Verifikation von Programmen. Hierbei werden mathematisch exakte Methoden gesucht und formale Sprachen entwickelt, um die Programme mit ihren Spezifikationen beschreiben zu können. Der Aufwand einer solchen Verifikation ist für Prozeduren als eine von mehreren Repräsentationsformen in einem wissensbasierten System zu hoch bzw. nur vertretbar, wenn besondere Anforderungen an die Qualitätsstufe des entwickelten Systems gestellt werden [9] [11].
Ist eine Prozedur in sich korrekt, werden die Wechselwirkungen mit Regeln und Frames interessant. Wie bereits in Kapitel 2.3.2 für Regeln erläutert, muß auch bei Prozeduren die

- Einschränkung auf direkte Wertzuweisungen **(F-Proz. 3)**

beachtet werden, um technischen Systemen gerecht zu werden.
Die Konsistenz des gesamten Systems ist damit aber noch nicht sichergestellt. Erst während der Ausführung der Prozeduren bzw. des gesamten Systems kann das Zusammenspiel von Frames, Regeln und Prozeduren überprüft werden und unerwünschte Aktionen und Widersprüche können erkannt und untersucht werden. Kommerzielle Systeme bieten zur Zeit keinerlei Ansätze in dieser Richtung.
Im folgenden werden daher alle Bereiche der Wissensrepräsentation unter dem Aspekt des Verhaltens während einer normalen Nutzung des Systems betrachtet.

2.3.5 Funktionaler Test

Ziel einer Expertensystementwicklung ist ein korrekt funktionierendes System, das in der erwarteten Weise auf gegebene Zustände und Anfragen reagiert. Die jeweilige Reaktion kann alle Möglichkeiten des verwendeten Werkzeuges umfassen. Im allgemeinen werden Werte für Attribute erzeugt und gesetzt, um so weitere Schlußfolgerungen zu initiieren. Will man dem wissensbasierten Ansatz gerecht werden, sind in hybriden Systemen hauptsächlich die Regeln die Auslöser für Aktionen.

Während der "Laufzeit" des Systems werden Regeln überprüft und ausgeführt, die den Zustand des Systems, der unter anderem durch die Attributwerte repräsentiert wird, verändern. Unter "Laufzeit" ist hier die Zeit der Benutzung des wissensbasierten Systems zur Lösung seiner eigentlichen Aufgabe gemeint. Während der Systementwicklung "läuft" die Entwicklungsumgebung (das Programm) auch, im allgemeinen werden die entwickelten Regeln und Prozeduren aber nicht bzw. nicht als Gesamtsystem ausgeführt. Bei komplexen Systemen ist für den Entwickler nicht immer zu überblicken, ob sich auch im Zusammenspiel aller Komponenten ausschließlich zulässige Zustände (Belegungen von Attributen, Existenz von Frames) ergeben und die Reaktion des Gesamtsystems in allen Fällen korrekt ist.

An dieser Stelle setzt der funktionale Test an.
Zunächst muß geklärt werden, was für einen funktionalen Test überhaupt wichtig ist. Macht es einen Unterschied, ob es sich bei dem entwickelten wissensbasierten System um ein Diagnose- oder Beratungssystem oder ein System zur Steuerung einer verfahrenstechnischen Anlage handelt? Diagnosesysteme liefern dem Anwender die Lösung des Problems oder die Antwort auf die gestellte Frage, bei automatisierungstechnischen Anwendungen ist dagegen der Ablauf, also der Weg vom Start- zum Zielzustand wichtig. Selbst wenn den Anwender nur die Lösung des Problems interessiert, ist im Falle einer Fehlfunktion ("falsche" Antwort) des wissensbasierten Systems der Grund für den Fehler im Ablauf der Wissensverarbeitung zu suchen. Ein funktionaler Test ist also von der Art des überprüften Systems unabhängig und dient der

- Überwachung des tatsächlichen Ablaufes der Schlußfolgerungen (Inferenzen).
  (F-funk. 1)

Eine solche Überwachung und Darstellung der Inferenzen kann beispielsweise mit einem sogenannten "Inferenzbrowser" erfolgen. Einige kommerzielle Systeme bieten eine solche Komponente an, die den zeitlich/logischen Ablauf der Inferenzen grafisch in einem "Inferenzbaum" darstellt. Da mit Hilfe dieser Grafik die Entscheidungen des Systems erklärt

werden können, wird eine solche Komponente teilweise auch als Erklärungskomponente bezeichnet. Erklärung bedeutet hier, daß die aktiven Elemente (Regeln, Prozeduren) mit den zum Zeitpunkt ihrer Aktivierung gültigen Attributwerten dargestellt werden. Es ist wohl leicht verständlich, daß ein solcher Inferenzbaum bei komplexen Systemen unüberschaubare Ausmaße annimmt.

Ein weiteres Problem tritt bei hybriden Systemen durch die Verwendung von Prozeduren auf. Die Angabe der Aufruf- und Rückgabe-Parameter der aktivierten Prozedur reicht nicht immer aus, da Prozeduren intern auch direkt auf Attributwerte sowohl lesend als auch schreibend zugreifen können. Diese Zugriffe werden in Inferenzbäumen nicht erfaßt.

Eine sinnvolle Ergänzung zu einem Inferenzbrowser stellt ein Werkzeug dar, mit dem

- die Belegung der Attribute überwacht **(F-funk. 2)**

werden kann. Dabei sind die folgenden Informationen interessant:

- Welches Attribut in
- welchem Frame wird von
- welcher Regel oder Prozedur
- bei welchem Systemzustand gesetzt ?

Erfolgt neben der Erfassung dieser Informationen auch eine Auswertung im Hinblick auf die Konsistenz des Systems, kann dieses Werkzeug als dynamische Komponente in einem Konsistenztest verwendet werden.

Zusätzlich zu der in Kapitel 2.3.4 angesprochenen Problematik der Generierung von Test-Startzuständen für Prozeduren ist auch die Gestaltung des weiteren Ablaufes nicht trivial. Ausgehend von einem Startzustand müssen alle Veränderungen an Attributwerten mit noch zu bestimmenden Zusatzinformationen protokolliert werden. Die Datenmenge, die parallel zu einer "Standardnutzung" des Systems anfällt, muß dann unter Berücksichtigung des Zeitablaufes ausgewertet werden.
Wird eine Überprüfung der Ergebnisse erst als letzter Schritt einer "Testnutzung" des wissensbasierten Systems durchgeführt, tritt insbesondere dann eine Zeit- und Speicherplatzverschwendung auf, wenn bereits nach wenigen Inferenzschritten ein Fehler aufgetreten ist. Je nach Umfang der automatischen Auswertung benötigt der Entwickler Informationen über Handlungsmöglichkeiten oder zumindest über das Umfeld des erkannten (potentiellen) Fehlers. Die gleichen Ansprüche werden bei der Entwicklung konventioneller Programme gestellt und durch das sogenannte "Tracing", das schrittweise

Abarbeiten von Anweisungen, befriedigt. Im Umfeld der wissensbasierten Systeme muß geklärt werden, was man unter "schrittweise" zu verstehen hat.

Wissensverarbeitung mit Hilfe von Regeln bedeutet die Prüfung der Regelbedingung und gegebenenfalls die Ausführung des Schlußfolgerungsteiles (oder Aktionsteiles, die Regel hat oder wurde "gefeuert"). Durch die Änderung von Attributwerten können Bedingungsteile anderer Regeln erfüllt und so weitere Schlußfolgerungen ausgelöst werden. Der Übergang vom Startzustand zu einem neuen Zustand kann als ein Schlußfolgerungsschritt verstanden werden.

Ein Schlußfolgerungsschritt (Inferenzschritt) beinhaltet im allgemeinen mehr als das Feuern **einer** Regel. Schrittweises Abarbeiten oder Ausführen eines wissensbasierten Systemes meint das Ausführen aller möglichen Aktionen (Regeln feuern, Prozeduren werden aufgerufen) bei einem gegebenen, unveränderlichen Systemzustand.

Eine Analogie zu dieser Verfahrensweise findet sich bei speicherprogrammierbaren Steuerungen (SPS). Die Signale an den Eingangsklemmen einer SPS-Baugruppe werden zu einem definierten Zeitpunkt als Eingangsabbild gespeichert. Die daraus abgeleiteten Steuersignale werden in einem anderen Speicherbereich, dem Ausgangsabbild gespeichert, sodaß das Eingangsabbild unverändert für alle Aktionen innerhalb einer Bearbeitungsfolge (eines Zyklus) zur Verfügung steht. Nachdem alle Programmschritte ausgeführt wurden, wird das Ausgangsabbild komplett an die Ausgangsklemmen übertragen. Analog hierzu kann ein wissensbasiertes System aufgebaut werden, indem die Bedingungsteile von Regeln und die Parameter von Prozeduren auf Attributwerte von Frames zugreifen, die zu der Oberklasse "Eingang" gehören, während alle Aktionen Attributwerte von Frames der Oberklasse "Ausgang" verändern.
Sind mit einer Belegung der "Eingangs-"Attribute alle möglichen Schlußfolgerungsschritte vollzogen worden, "steht" das System. Ein nächster "Schritt" kann begonnen werden, wenn alle Wertänderungen aus den "Ausgangs-" in die "Eingangsframes" umkopiert wurden. Auf diese Weise wird ein "schrittweises" Arbeiten realisiert, das eine wichtige Voraussetzung für den Konsistenztest bildet, da nur dann die "Quellen" einer Inkonsistenzen sofort ermittelt werden können. Der Speicheraufwand für das Protokoll ist über die Anzahl der im System verwendeten Attribute abschätzbar, da nur ein einzelner Schritt protokolliert werden muß.

Die Kombination des oben beschriebenen schrittweisen Vorgehens mit den Erfordernissen eines funktionalen Konsistenztests hat Auswirkungen auf die Art, wie die "Ausgangsframes" zu implementieren sind. Fehlerursachen können nur erkannt werden, wenn zum einen erfaßt werden kann, was ein Fehler ist und zum anderen, wenn der Verursacher eines

Objekt Ampel A1	
Attributname	Wert
Farbe	rot
Wartezeit	5 sec.
Status	aktiv

Eingangsobjekt

*Objekt *Ampel *A1		
Attributname	Wert	Quellenname
Farbe		
Wartezeit	4 sec.	Regel Z1
Status		

Ausgangsobjekt

Abbildung 2.2: Struktur der Ein- und Ausgangsframes

potentiellen Fehlers ermittelt werden kann. Geht man davon aus, daß Fehler direkt oder indirekt mit Attributwertbelegungen zusammenhängen, muß **gefordert** werden, daß

- die Verursacher von Attributwertänderungen erfaßt werden. **(F-funk. 3)**

Konkret kann man z.B. zu jedem Attribut nicht nur den jeweiligen Momentanwert abspeichern, sondern auch ein eindeutiges Kennzeichen für die Regel oder Prozedur, die diesen Wert liefert. Bei objektbasierten Entwicklungswerkzeugen wird beispielsweise auch jede Regel als ein spezielles Objekt aufgefaßt, dem ein Name zugeordnet wird. Prozeduren werden generell über ihren Namen referenziert, so daß als Kennzeichen für eine "Attributwertquelle" ein Name gespeichert werden kann (Abb. 2.2). Da die Herkunft von Attributwerten somit eindeutig bestimmt ist, können Prozeduren in der auch später im Anwendungsfall auftretenden Datenumgebung getestet werden.

Mit Hilfe der Ausgangsframes kann zu jedem Attribut die Quelle des aktuellen Wertes festgestellt werden. Durch die Unabhängigkeit der Wissenselemente (s. S. 40) ist auch die Wertänderung eines einzelnen Attributes durch zwei verschiedene Quellen möglich. Hier sind dann zwei Fälle zu unterscheiden:

1. alle Quellen liefern den gleichen Wert , oder
2. für das Attribut werden verschiedene Werte geliefert.

Den ersten Fall kann man als zunächst unkritische Redundanz auffassen. Eine Redundanz als solche ist ein Hinweis auf nicht komplett strukturiertes Wissen, aber kein direkter,

sondern nur ein potentieller Fehler. Potentieller Fehler deshalb, weil bei einer Änderung eines Wissenselementes die anderen an der Redundanz beteiligten Elemente nicht mit in Betracht gezogen werden. Dadurch kann es zum zweiten oben aufgeführten Fall kommen.

Werden für ein Attribut bei einem bestimmten Systemzustand verschiedene Werte geschlußfolgert, ist das ein Zeichen für einen Widerspruch in den implementierten Wissenselementen; das Wissen ist inkonsistent. Für die Erkennung dieser Inkonsistenz ist es erforderlich, Mehrfachbelegungen eines Attributes in einem Schritt zuzulassen und zu erfassen. Die Implementation der Ausgangsframes muß an diese Anforderungen angepaßt werden.

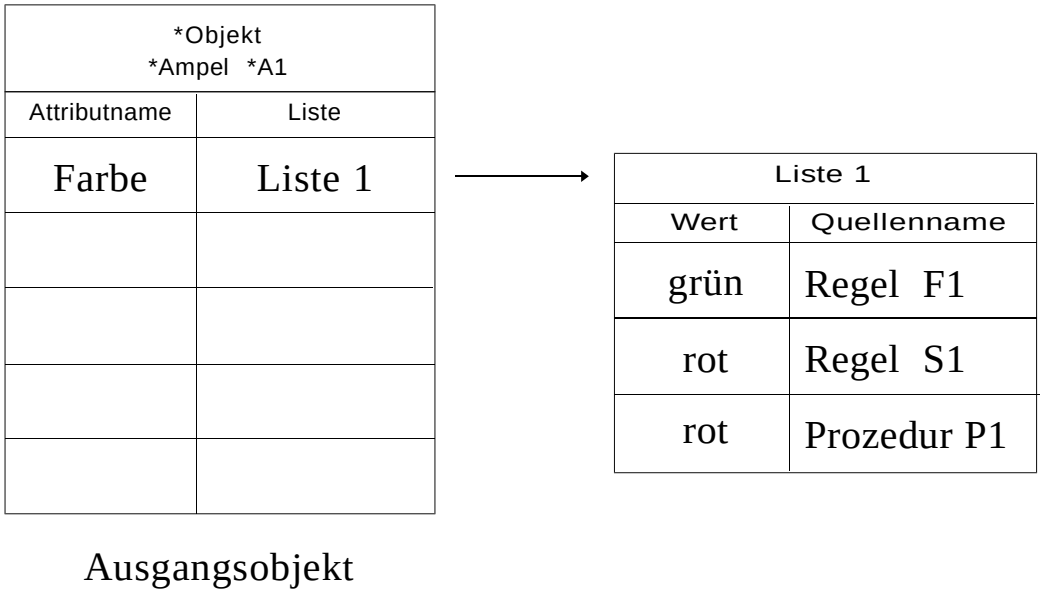


Abbildung 2.3: Struktur des erweiterten Ausgangsframes

Anstelle eines Wertes mit Quellenangabe muß es möglich sein, mehrere Werte mit Quellenangaben in der Reihenfolge des Auftretens abzulegen, ohne daß die alten Werte einfach überschrieben werden. Statt einer einfachen Variablen kann beispielsweise eine Liste für Wert/Quelle-Paare für jedes Attribut vorgesehen werden (Abb. 2.3). Enthält eine solche Attributwertliste mehr als einen Eintrag, müssen die Werte auf obige Fallunterscheidung hin untersucht werden.
Die Auswertung der festgestellten Mehrfachbelegung kann mehr oder weniger aufwendig gestaltet werden. Der Entwickler benötigt als **minimale** Ausgabe eine Meldung, daß im letzten Schritt ein unerwünschter Zustand aufgetreten ist. Ein komfortables Werkzeug kann die Quellen der Attributwerte nennen oder sofort zur Bearbeitung zur Verfügung stellen.

2.3.6 Bewertung und Auswahl der Testkomponenten

Anhand der Repräsentationsformen in einem hybriden System wurden unterschiedliche Komponenten für einen Konsistenztest erarbeitet. Der Aufwand für den Konsistenztest ist erheblich, wenn alle angesprochenen Komponenten in vollem Umfang notwendig sind. Im folgenden wird eine Bewertung der Komponenten

- syntaktischer Test; Regel- und Frameeditor,
- syntaktischer Test; Connection Graph Proof Procedure und
- funktionaler Test

vorgenommen, um eine "komfortable" und vertretbare Kombination der notwendigen Komponenten zu ermöglichen.

Editoren zur benutzerfreundlichen Eingabe der Wissenselemente bieten außer für Konsistenz und Kompaktheit auch am Markt Vorteile. Ein intuitiv zu benutzendes Entwicklungssystem erleichtert dem Wissensingenieur den Einstieg in die Implementierung des Wissens und wird mehr Akzeptanz finden als ein System mit kryptischen Befehlen. Statt alle notwendigen Randbedingungen der korrekten Implementierung z.B. eines Objekts selbst nachhalten zu müssen, kann der Wissensingenieur sicher sein, die system- und konsistenzbedingten Mindestanforderungen durch die Hilfe der Editoren zu erfüllen.

Der Aufwand, solche Editoren zu implementieren, hängt sehr stark von der gewählten Entwicklungsumgebung ab. Je mächtiger das Werkzeug ist und je mehr Freiheiten dem Applikationsentwickler vom Hersteller zugestanden werden, um so aufwendiger ist die Gestaltung "benutzerfester" Komponenten. Die ursprünglich vorhandenen Freiheiten müssen sinnvoll und korrekt eingeschränkt werden, ohne die Funktionalität zu beeinträchtigen. Hinsichtlich der Konsistenz kann mit Hilfe des Frameeditors die Erfüllung der Forderungen *F-Fra. 1* und *F-Fra. 2* erleichtert und die Forderung *F-Fra. 3* für Regeln garantiert werden (Seite 36). Der Regeleditor muß so gestaltet werden, daß die Forderungen *F-CGPP* und *F-Regel* zuverlässig erfüllt werden (Seite 38 und 39).

Die Geschwindigkeit des Entwicklungssystems und somit die Arbeitsgeschwindigkeit des Entwicklers hängt ebenfalls vom verwendeten Werkzeug und der Arbeitsoberfläche ab. Der Aufbau eines ansprechenden Erscheinungsbildes der Editoren kann unter Umständen zeitintensiv sein. Bei den notwendigen Überprüfungen von Syntax und Datentypen hängt die Belastung zusätzlich vom Geschick des Werkzeugentwicklers ab. Der zusätzlich benötigte Speicherplatz schließlich ist je nach Implementierung über die Anzahl der Frames,

Attribute und "möglichen Werte" vom zu entwickelnden System abhängig.
Insgesamt sind die Editoren, besonders im Zusammenhang mit einem funktionalen Test, notwendig.

Die Connection Graph Proof Procedure ist, wie in Kapitel 2.3.3.1 gezeigt wurde, prinzipiell für Regeln in einem hybriden System anwendbar, weist aber noch eine Schwäche auf. Wird eine Inkonsistenz im Gesamtsystem erkannt, sollte eine Angabe der potentiellen Ursachen analog zu den Quellframes im funktionalen Test erfolgen (siehe Forderung *F-funk. 3*, Seite 56). Dazu muß zu jedem Literal der Name der Ursprungsregel erfaßt werden. Da verschiedene Regeln zu gleichen Literalen bzw. Klauseln mit gleichen Literalen führen können, müßten Redundanzen in Kauf genommen und bei der Auswertung berücksichtigt werden. Eine vereinfachte Auswertung kann dann erfolgen, wenn nicht ein komplettes System, sondern immer der letzte Entwicklungsschritt geprüft wird. Nach jeder Regeleingabe oder Frameerzeugung wird die Regel bzw. die Ausschlußbedingungen in die Klauselform umgewandelt und der bereits bestehenden Klauselmenge hinzugefügt. Tritt bei der nun folgenden Anwendung der Connection Graph Proof Procedure eine Inkonsistenz auf, muß der Einfluß der "neuen" Literale untersucht werden. Dieses Vorgehen hat allerdings gravierende Nachteile. Die Konsistenz muß auch dann geprüft werden, wenn ein Attribut einen zusätzlichen "möglichen Wert" erhält, da dadurch neue Ausschlußbedingungen und somit auch neue Klauseln erzeugt werden müssen.

Durch die Verzahnung der CGPP mit den Editoren steigt der Implementierungsaufwand, und eine funktionale Trennung der Komponenten z.B. in verschiedene Module der Entwicklungsumgebung ("Zusatzpakete") ist nicht mehr möglich. Der Zeitaufwand für die Klauselerzeugung während der Wissenseingabe hindert auch ungeübte Benutzer an einer zügigen Arbeit. Der Speicherplatzverbrauch für die Klauseln ist sehr groß, wie an den Beispielen in Kapitel 2.3.3.1 deutlich wird.
Ein funktionaler Test des aktuellen Entwicklungszustandes zeigt auch die Inkonsistenzen auf, die in der Formulierung der Regeln begründet sind. Wird auf jeden Fall ein funktionaler Test durchgeführt, kann also auf einen "syntaktischen" Test der Regeln mit Hilfe der Connection Graph Proof Procedure verzichtet werden. Wird die CGPP zusätzlich zum funktionalen Test eingesetzt, beschleunigt das die Entdeckung von Inkonsistenzen der Regeln untereinander. Dieser Geschwindigkeitsvorteil muß aber mit einem erheblichen Aufwand bei Regeländerungen erkauft werden. Nach jeder Änderung müssen die Klauseln der "alten" Regel unter Beachtung möglicher Redundanzen aus der Klauselmenge entfernt und die "neue" Regel in Klauseln umgewandelt werden. Für den funktionalen Test müssen trotzdem alle Regeln im Hinblick auf die Ein- und Ausgangsframes überarbeitet werden. Eine Implementierung der CGPP zusätzlich zu einem funktionalen Test wird daher von einer "Aufwand zu Nutzen"-Betrachtung abhängig sein.

Die Komponente für den funktionalen Test ist für die Überprüfung des Gesamtsystems unabdingbar. Neben Inkonsistenzen können auch inhaltliche Fehler, wie "falsche" Wahl eines an sich zulässigen Attributwertes oder Zugriff auf einen "unbeteiligten" Frame, durch die schrittweise Abarbeitung des Inferenzablaufes gefunden und leicht korrigiert werden.

Je nach Gestaltung der funktionalen Testkomponente steigt der Implementierungsaufwand mit dem Komfort, der dem Benutzer geboten werden soll. Die Geschwindigkeit in der Testphase hängt von der Anzahl der zu überwachenden Frames und ihrer Verknüpfung ab. Die Verwendung von Ausgangsframes erhöht außerdem den Speicherplatzbedarf wiederum in Abhängigkeit der Frame- und Attributanzahl.
Mit dem funktionalenTest kann die Erfüllung der für die Konsistenz erhobenen Forderungen *F-Fra. 3* (Seite 36), *F-Proz. 2* (Seite 52) sowie *F-funk. 1 bis 3* (Seite 53, 54, 56) garantiert werden.

2.4 Zusammenfassung

In Datenbanken fallen zwei Sachverhalte unter den Begriff Konsistenz:

- "Wertkonsistenz"
  Für ein und das selbe Attribut muß auch bei mehrfachem Auftreten im Datenbestand bei einer Abfrage ein einheitlicher Wert auftreten.
  Bei Wertänderungen darf der Datenbestand erst dann für weitere Manipulationen freigegeben werden, wenn alle betroffenen Datensätze geändert wurden.

- "logische (semantische) Konsistenz"
  Semantische Zusammenhänge werden durch Datenmanipulationen nicht verletzt. Konsistenzbedingungen, die die Semantik der Daten repräsentieren, werden während oder direkt nach einer Manipulation überprüft und entsprechende Reaktionen ausgelöst.

In hybriden wissensbasierten Systemen kann Konsistenz als Komplex aus mehreren Bereichen interpretiert werden:

- Regelkonsistenz
  Die Formulierung der Regeln muß bestimmten Bedingungen unterliegen. Syntaktisch kann dann die Konsistenz mit Hilfe der Connection Graph Proof Procedure geprüft werden.

- Frame- oder Objektkonsistenz
  Aus den Bedingungen für die Regelformulierung ergeben sich Vorgaben für die Framestruktur. Aus Gründen der Übersicht ist eine klare Hierarchie anzustreben. Inkonsistenzen werden durch die Verwendung zugeschnittener Eingabemöglichkeiten mit Überprüfung weitgehend vermieden.

- Prozedurkonsistenz
  Syntaktische Korrektheit wird durch kontextsensitive Editoren sichergestellt. Funktionale Korrektheit kann nur im Zusammenspiel mit den anderen Wissenselementen dynamisch geprüft werden.

- "Gesamtkonsistenz"
  Zugeschnittene Editoren für alle Wissenselemente stellen syntaktische Konsistenz sicher. Das Zusammenspiel der Wissenselemente muß dynamisch-funktional, also während der "normalen" Nutzung des Systems, überprüft werden. Zusätzliche syntaktische Tests einzelner Wissenselemente sind daher nicht notwendig.

Kapitel 3

Realisierung

Während der Entwicklung eines neuen wissensbasierten Systemes sollten alle Möglichkeiten genutzt werden, um die Konsistenz der Wissensbasis von Beginn an zu wahren. Die kommerziell verfügbaren Werkzeuge stellen dem Wissensingenieur sehr verschiedene Hilfsmittel zur Verfügung. Die hybride Expertensystemshell G2 der Firma GENSYM GmbH in der Version 5.0 für das Betriebssystem OpenVMS 6.2 bildet die Basis für die im Rahmen dieser Arbeit vorgeschlagene Erweiterung der vorhandenen Hilfsmittel zu einer "Konsistenzprüfungskomponente".

Die Entwicklungsumgebung G2 wird vom Hersteller als "objektorientiert" bezeichnet, bietet als Repräsentationsform für den Benutzer strenggenommen aber Frames, Regeln und Prozeduren mit deren Sonderform "Methode" an. Regeln oder Variable beispielsweise sind in diesem Zusammenhang allerdings auch als Objekte im formalen Sinn interpretierbar, da eine Kapselung besteht und Methoden (Systemprozeduren) zur Verfügung stehen. Um nicht unnötig zu verwirren, wird im folgenden die Bezeichnung "Objekt" auch für die vom Benutzer erzeugten Frames verwendet.

Die in diesem Kapitel beschriebene Realisierung einer Konsistenzprüfungskomponente liefert ein Zusatzmodul zum G2 Basissystem, durch dessen Einsatz ein konsistentes Expertensystem entwickelt werden kann. Ein Schichtenmodell der Software ist in Abbildung 3.1 zu sehen.

Da ein und dieselbe Person je nach Sichtweise (oder "Schicht") sowohl ein "Anwender", als auch gleichzeitig ein "Entwickler" sein kann, werden zunächst die hier verwendeten Bedeutungen erläutert.

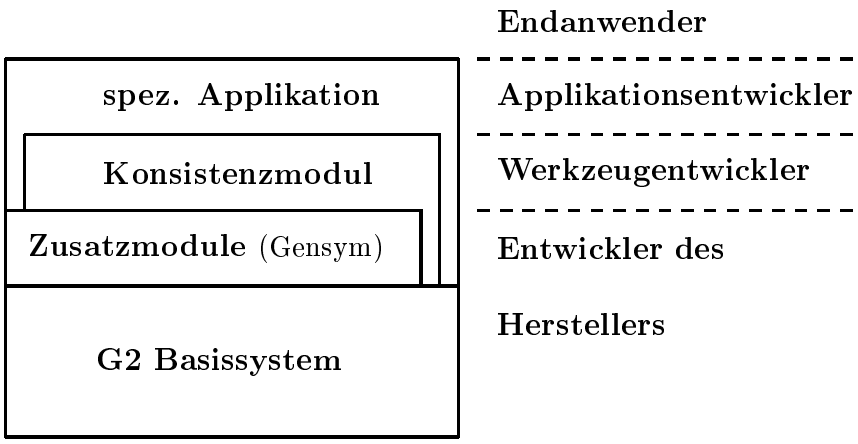


Abbildung 3.1: Blockbild einer G2 Applikation mit Konsistenzmodul

- Der "Endanwender" ist der Benutzer der fertigen Applikation,
- die der "Applikationsentwickler" (Wissensingenieur) mit Hilfe der erweiterten Expertensystemshell realisiert hat. Während der Entwicklung ist der "Applikationsentwickler" der "Benutzer" der Shell.
- Die Teile der Shell, die nicht zum G2 Basissystem gehören, wurden von "Werkzeugentwicklern" entworfen und realisiert.
- Die Entwickler des G2 Basissystems werden in dieser Arbeit nicht erwähnt, da sie nicht bekannt sind.

Dieses Kapitel beschreibt die Realisierung eines Konsistenztestes, der die Ausführungen in Kapitel 2.2 berücksichtigt. Nach einem Überblick über die Möglichkeiten der Expertensystemshell G2 wird die Umsetzung der einzelnen Abschnitte aus Kapitel 2.2 beschrieben.

Die Realisierung erfolgte durch "Werkzeugentwickler", im folgenden kurz **"Entwickler"** genannt, am **Lehrstuhl für Automatisierungstechnik** an der Bergischen Universität – Gesamthochschule Wuppertal im Rahmen dieser Arbeit durch die Verfasserin sowie in Form von durch sie betreuten Projekt- und Diplomarbeiten. Dabei wurden alle Dialoge mit Hilfe der in Abbildung 3.5 dargestellten Grafikelemente selbst entworfen und für jedes Element das Verhalten bei verschiedenen Benutzerreaktionen mit Hilfe komplexer, selbstentwickelter Methoden festgelegt.
Das Konsistenzmodul wird von **Wissensingenieuren** während einer Applikationsentwicklung benutzt, die im folgenden kurz **"Benutzer"** oder "Nutzer" genannt werden.

3.1 Die Entwicklungsumgebung

Die in der Literatur angegebene Trennung von Wissensbasis, Inferenzmaschine und Dialogkomponente mit Akquisitions- und Erklärungsteil (Abbildung 3.2) ist in der Entwicklungsumgebung G2 (Shell, oder kurz: G2) durch verschiedene Mechanismen realisiert. Die Inferenzmaschine ist integraler Bestandteil der Entwicklungsumgebung und kann durch den Entwickler in bestimmten Grenzen über Parameter gesteuert werden. Alle implementierten Klassendefinitionen und Instanzen bilden die Wissensbasis des Systems, und die grafische Benutzeroberfläche vereinigt alle Anforderungen an die Dialogkomponente in sich.

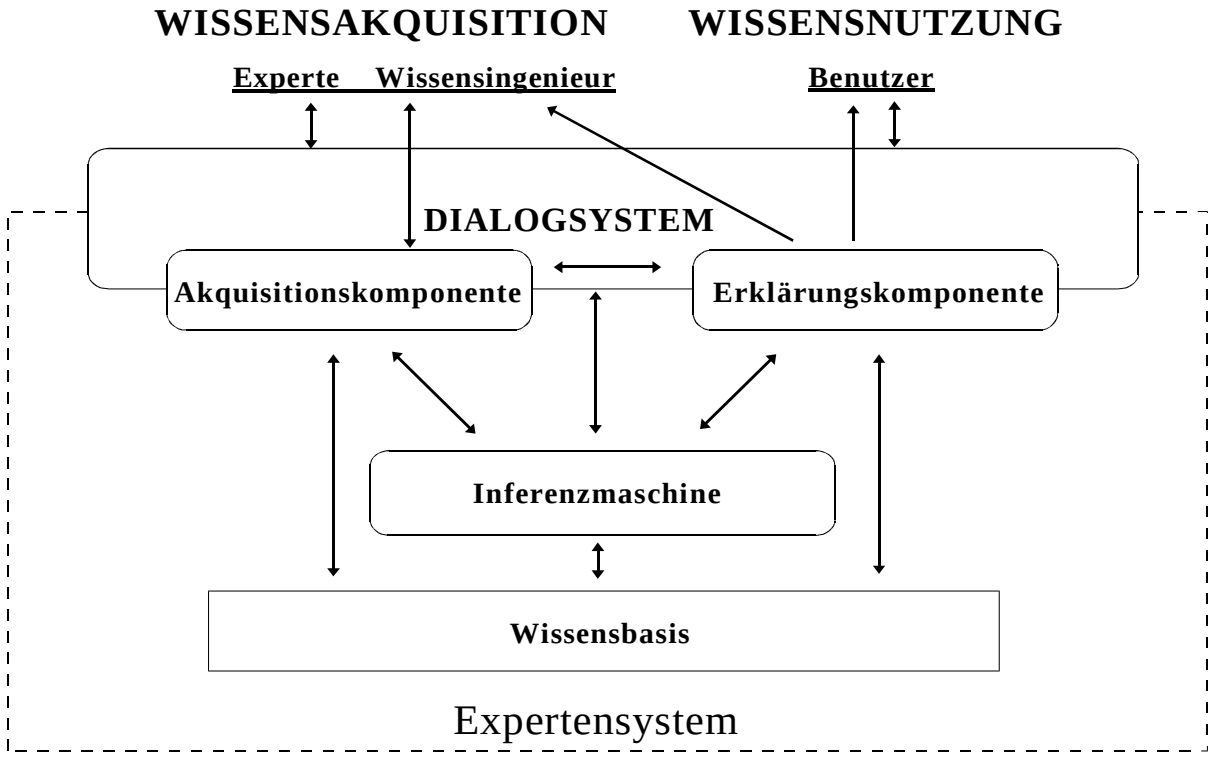


Abbildung 3.2: Komponenten eines Expertensystems (nach [7])

Um in der Benutzeroberfläche verschiedenen Benutzertypen auch verschiedene Werkzeuge zur Verfügung zu stellen, können Benutzergruppen wie z.B. Administrator, G2-erfahrener Entwickler (Developer), Entwickler (Knowledge Engineer), Benutzer oder beliebige andere deklariert werden. Der Administrator als Benutzer mit den größten Befugnissen kann über Konfigurationsparameter und Beschränkungen in verschiedenen Ebenen ein unterschiedliches Verhalten der Oberfläche für jede Gruppe vorgeben. Diese Möglichkeit

wird z.B. genutzt, um dem Wissensingenieur zum einen die Nutzung der Konsistenzkomponente zu ermöglichen und zum anderen, um ihn am Umgehen der Komponente zu hindern. Diese Eigenschaft ist besonders wichtig, denn wird bei einer Entwicklung auch nur ein Teil der Strukturen ohne Verwendung der Konsistenzkomponente implementiert, ist kein sinnvoller Konsistenztest mehr möglich.

Die Entwicklungsumgebung G2 stellt eine grafische Benutzeroberfläche mit beliebig vielen Arbeitsflächen (Arbeitsblatt, G2-Terminus: workspace) zur Verfügung, die an die Oberfläche verschiedener Betriebssysteme erinnert. Jede Arbeitsfläche kann als grafische Zusammenfassung von Wissenselementen (Objekten, G2-Terminus: objects) genutzt werden, wobei eine weitere logische Zusammenfassung von Arbeitsflächen in Teilwissensbasen (G2-Terminus: module) erfolgen kann. Die Modularisierung macht es möglich, ein vollständiges Expertensystem aus verschiedenen allgemeinen und einigen speziell für eine Anwendung erstellten Modulen zusammenzusetzen. Besondere Vorteile ergeben sich für die Wiederverwendbarkeit von Teilwissensbasen, die allgemeine Werkzeuge enthalten, sowie für die Größe des an die Kunden ausgelieferten Expertensystems. Werkzeuge wie Dialoge zum bequemen Speichern oder Ausdrucken von Informationen oder auch allgemeingültige Komponenten können für andere Entwicklungen zur Verfügung gestellt werden. Werden Entwicklungswerkzeuge (-module) nach Fertigstellung des Gesamtsystems entfernt, kann die für den Endanwender notwendige Speichergröße ohne Funktionsverlust reduziert werden.

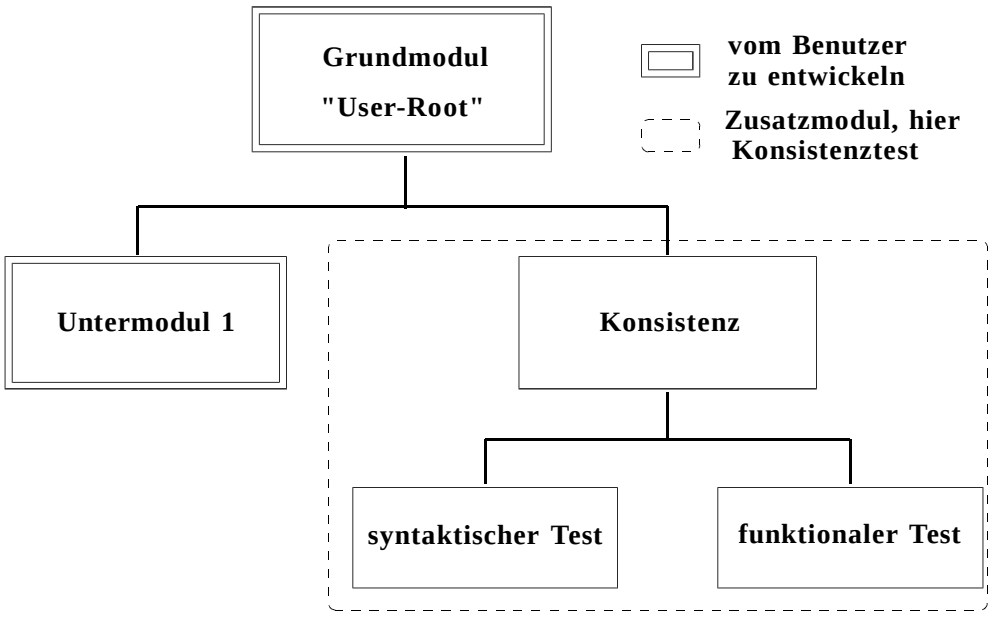


Abbildung 3.3: Modulhierarchie einer leeren Wissensbasis mit Konsistenztest

Die im Rahmen dieser Arbeit entwickelte Konsistenzkomponente ist in einem eigenen Modul (mit Untermodulen) zusammengefaßt und kann nach Abschluß der Entwicklung entfernt werden, um den Umfang der zur Ausführungszeit benötigten Wissensbasis klein zu halten. Werden später Ergänzungen der Wissensbasis notwendig, kann das Modul wieder eingebunden werden.

Die Basiskonfiguration einer leeren Wissensbasis mit Konsistenzprüfung (Abbildung 3.3) wurde so angelegt, daß ein "Grundmodul" für die zu entwickelnde Applikation zur Verfügung gestellt wird, dem in der Modulhierarchie bereits das Konsistenzmodul untergeordnet ist. Mit dem Start der Inferenzmaschine steht damit sofort ein "Konsistenz-Experte" zur Verfügung, der bei der Entwicklung der Applikation hilft (Abbildung 3.6). Neben einem Arbeitsblatt, das dem gleichnamigen Grundmodul "User-Root" zugeordnet ist, wird das Arbeitsblatt "Palette" angezeigt. Hier wird durch Anklicken mit der Maus ausgewählt, was erzeugt werden soll.

Als Repräsentationsform für Wissen stellt G2 einen objektbasierten Ansatz zur Verfügung (siehe Kapitel 1.3.2 sowie die Einführung zu diesem Kapitel), in dem zusätzlich Prozeduren als Objektklasse integriert sind. Neben den vom Entwicklungssystem vorgegebenen Objektklassen (Systemklassen) wie Regel, Prozedur, Variable, Parameter usw. können durch den Benutzer eigene Klassen erzeugt werden, deren Instanzen (Objekte) das Faktenwissen aufnehmen. Jede Instanz kann direkt über ihren Namen und indirekt über die Klassenzugehörigkeit referenziert werden.

Ein Vorteil von G2 liegt in der grafischen Repräsentation der einzelnen Objekte (Sinnbild, icon, Ikone). Einerseits ist damit eine Visualisierung der Wissenskomponenten und über Gruppierungen der Sinnbilder die Darstellung ihrer Verknüpfung gegeben. Andererseits sind die Bearbeitungsmöglichkeiten für die Objekte durch Auswahl (Anklicken) ihres Sinnbildes (icons) mit der Maus aktivierbar. Der Entwickler hat vielfältige Möglichkeiten, die Informations- und Bearbeitungsmöglichkeiten zu strukturieren und an das jeweilige Objekt zu binden. Informationen zu einem Sinnbild sind in erster Linie die Eigenschaften des dadurch symbolisierten Objektes, die üblicherweise in tabellarischer Form (G2-Terminus: table) angezeigt werden (siehe Abbildung 3.4 links).

Weitere Objekte wie Prozeduren und Variablen, die mit diesem Objekt (Regel "PS-Motor-1") zusammenhängen, können auf einer Arbeitsfläche angeordnet werden, die direkt der Regel "PS-Motor-1" zugeordnet wird (Unterarbeitsfläche, subworkspace, Abbildung 3.4 rechts). In diesem Beispiel sind auf der Unterarbeitsfläche eine grafische Darstellung der Regel sowie der Regeltext in G2 Syntax abgelegt.

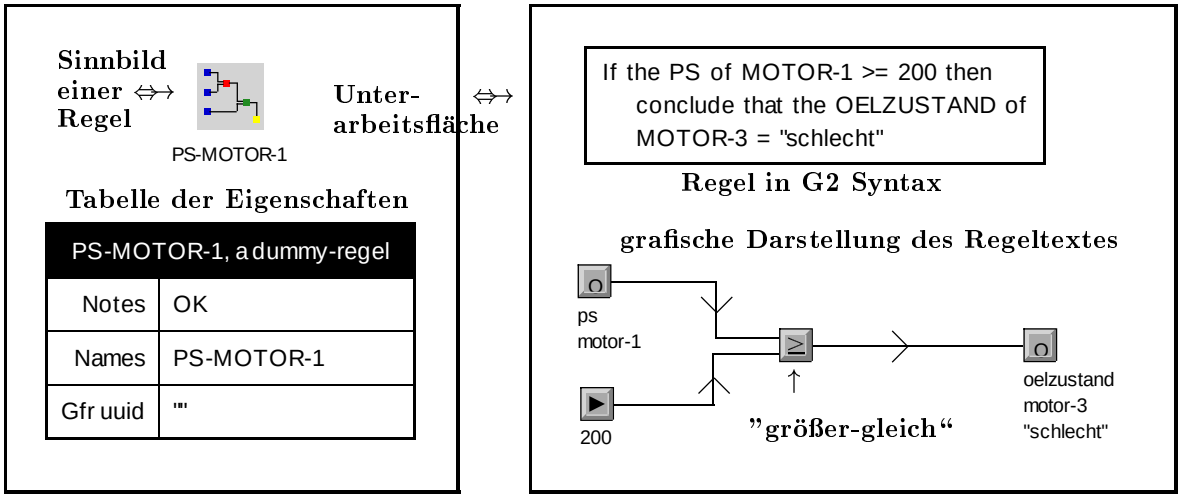


Abbildung 3.4: Ein (sinnloses) Regel-Objekt und seine Unterarbeitsfläche

Auf die Bedeutung der einzelnen Angaben, wie z.B. PS, MOTOR-1 und MOTOR-3 wird in Kapitel 3.3 näher eingegangen.
Sind die Informationen auch bei anderen Objekten (z.B. allen vom Benutzer erzeugten Regeln) relevant, kann eine Arbeitsfläche auch so angelegt werden, daß von jedem betroffenen Objekt ein Verweis erfolgen kann. Auf diese Weise ist eine flexible Strukturierung des Wissens möglich.

Der objektbasierte Ansatz von G2 ermöglicht auch im Zusammenhang mit einem Test der Wissensbasis eine komfortable Benutzung, da jedes Objekt einzeln deaktiviert bzw. aktiviert werden kann. Die Inferenzmaschine verwendet nur aktives Wissen und "ignoriert" deaktivierte Objekte. Auf diese Weise kann die aktuell zur Verfügung stehende Wissensmenge verändert und der Einfluß des deaktivierten Wissens auf das Gesamtsystem schnell erkannt werden, denn neu auftretende bzw. fehlende Werte oder Aktionen hängen mit dem deaktivierten Wissen zusammen.

Für die Gestaltung der Benutzeroberfläche bietet G2 mit verschiedenen, vorgefertigten Grafikelementen ein Skelett, das durch die Werkzeugentwickler für jegliche Aktion mit und durch die Grafikelemente mit Leben gefüllt werden muß. Für den Aufbau der dazu notwendigen, komplexen Prozedurstrukturen bietet G2 Systemprozeduren der "User Interface Library" (Benutzerschnittstellenbibliothek, Zusammenstellung von Grundprozeduren zur Manipulation der Grafikelemente) als Minimalelemente an, eine weitere Unterstützung gibt es jedoch nicht.

3.2 Klassen- und Frameeditor

Startpunkt jeder Implementierung in einer hybriden Entwicklungsumgebung sind die Objekte. In ihnen wird das Faktenwissen repräsentiert. Auf der Benutzeroberfläche erscheinen Objekte als Sinnbilder, mit deren Hilfe weitere Informationen zu dem Objekt abgerufen werden können. Die allgemeine Beschreibung der Objekteigenschaften wird nicht im Objekt selber vorgenommen, sondern in der Klassendefinition der Klasse abgelegt, deren Instanz das Objekt ist. Ohne Klassendefinition kann kein Objekt existieren.

Will ein Nutzer ein neues Objekt erzeugen, gibt es zwei Möglichkeiten:

1. es existiert bereits eine Klasse, die die Eigenschaften des gewünschten Objektes beschreibt, oder

2. es soll ein Objekt mit bisher unbekannten Eigenschaften erzeugt werden.

Fall 1 ist für jeden Benutzertyp gleich einfach. Von der bestehenden Klasse wird eine neue Instanz (Objekt) gebildet, der ein eindeutiger Name zugewiesen wird. Wie komfortabel eine Instanz erzeugt werden kann, ist vom Benutzertyp und den Vorbereitungen durch den Administrator abhängig.
Fall 2 erfordert bereits Vorwissen über objektbasiertes Vorgehen und die verwendeten Strukturen. Für den Benutzer bedeutet das, nicht einfach ein Objekt zu erzeugen, sondern zuerst eine Klassendefinition anzulegen. Dazu muß er schon vorher mit der Syntax und den Systemwerkzeugen vertraut sein, um wenigstens die unabdingbaren Eintragungen wie Klassenname und Oberklasse sofort festzulegen. Die Eigenschaften (Attribute), die das neue Objekt schließlich haben soll, können dann nach und nach eingetragen und geändert werden. Auch die Form des Sinnbildes kann nachträglich festgelegt bzw. geändert werden. Für all diese Aktionen ist in der Grundversion von G2 Erfahrung im Umgang und Wissen über die vorhandenen Klassen erforderlich.
Diesem Umstand haben die Entwickler des Lehrstuhls für Automatisierungstechnik an der Bergischen Universität – Gesamthochschule Wuppertal Rechnung getragen, indem Editoren mit zugeschnittenen Dialogen für die Erzeugung von Objekten bzw. Klassendefinitionen auf Basis der Grundfunktionen der "User Interface Library" gestaltet wurden, die auch einem unerfahrenen Benutzer ausreichend Hilfestellungen geben.

Durch diese Dialoge wird die Erfüllung folgender Randbedingungen erzwungen:

- das System verlangt als Mindesteingaben einen Klassennamen und eine oder mehrere Oberklassen,

- der Benutzer benötigt Hilfen bzgl. möglicher Typen von Attributen sowie sinnvolle Angaben zu vorhandenen Namen und Objekten (Forderungen *F-Fra. 1* und *2*), da ihm die Suchmöglichkeiten innerhalb von G2 nicht bekannt sind, und schließlich
- sollte für die Konsistenzkomponente die Möglichkeit bestehen, Wertemengen für Attribute festzulegen (Forderung *F-Fra. 3*).

Die Systemforderungen müssen in jedem Fall erfüllt werden, da G2 sonst mit einer Fehlermeldung reagiert oder, was schwerere Folgen nach sich ziehen könnte, ohne Fehlermeldung in einen unsicheren Systemzustand läuft. Auch die Anforderungen der Konsistenzkomponente müssen so weit wie möglich beachtet werden, damit bei der Regelerzeugung (siehe Kapitel 3.3) auf optimale Hilfestellung zurückgegriffen werden kann.
Einzig die Feststellung der "Benutzerfreundlichkeit" ist ein subjektiver Maßstab, für den es in der Literatur zur Mensch-Maschine-Kommunikation einige Anhaltspunkte, aber keine Vorschriften gibt. Die weite Verbreitung des Betriebssystems Windows [1] gibt grafische Elemente und Bedienweisen vor, die für eine intuitive Bedienbarkeit sorgen. Die Dialoge der Editoren wurden in Anlehnung an diesen "Quasistandard" gestaltet, indem z.B. Schaltflächen, Eingabefelder und Listenfelder verwendet wurden.

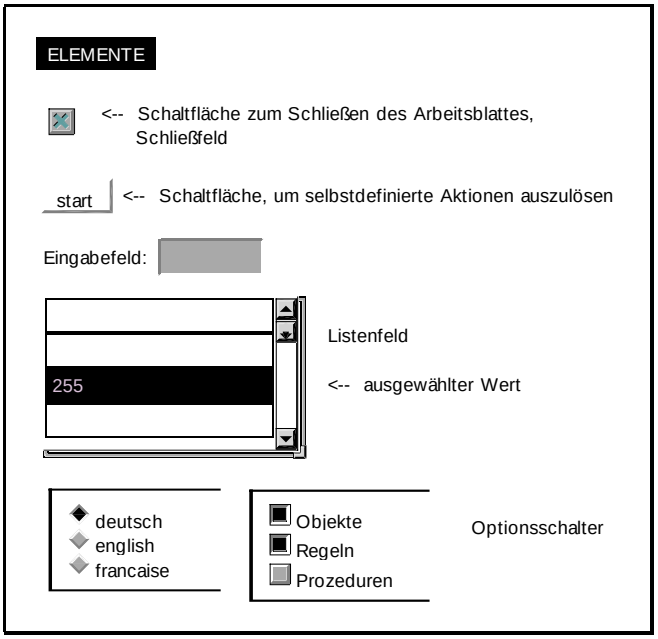


Abbildung 3.5: Beispiele für Dialogelemente in G2

[1] Warenzeichen der Microsoft Corporation

Das erste Objekt

Ohne einen Editor mit zugeschnittenen Dialogen muß der Benutzer über die Systemmenüs eine Klassendefinition auswählen und wenige Einträge in einer langen Tabelle der Eigenschaften mit dem Texteditor ändern. Der Texteditor von G2 zeigt während der Bearbeitung Syntaxblöcke an, die an der betreffenden Stelle möglich sind. Teilweise werden auch notwendige Angaben mit Standardwerten ergänzt, wenn sie nicht explizit eingegeben werden. Für einen ungeübten Benutzer kann das allerdings eher verwirrend als hilfreich sein. Eine Überprüfung der notwendigen Eingaben erfolgt außerhalb des Editors nicht, sodaß z.B. auch namenlose Klassen existieren können, was zum Abbruch der gesamten Entwicklungsumgebung führen kann.
Diese Probleme und die unnötige Datenflut werden durch die Dialoge von Klassen- und Frameeditor vom Benutzer ferngehalten. Das zur neu entwickelten Konsistenzkomponente gehörende Arbeitsblatt "Palette" (Abbildung 3.6) wurde durch die Entwickler so konfiguriert, daß für den Benutzer (Wissensingenieur) die selbsgestalteten Sinnbilder von Klassen, Objekten, Regeln usw. wie Schaltflächen reagieren.

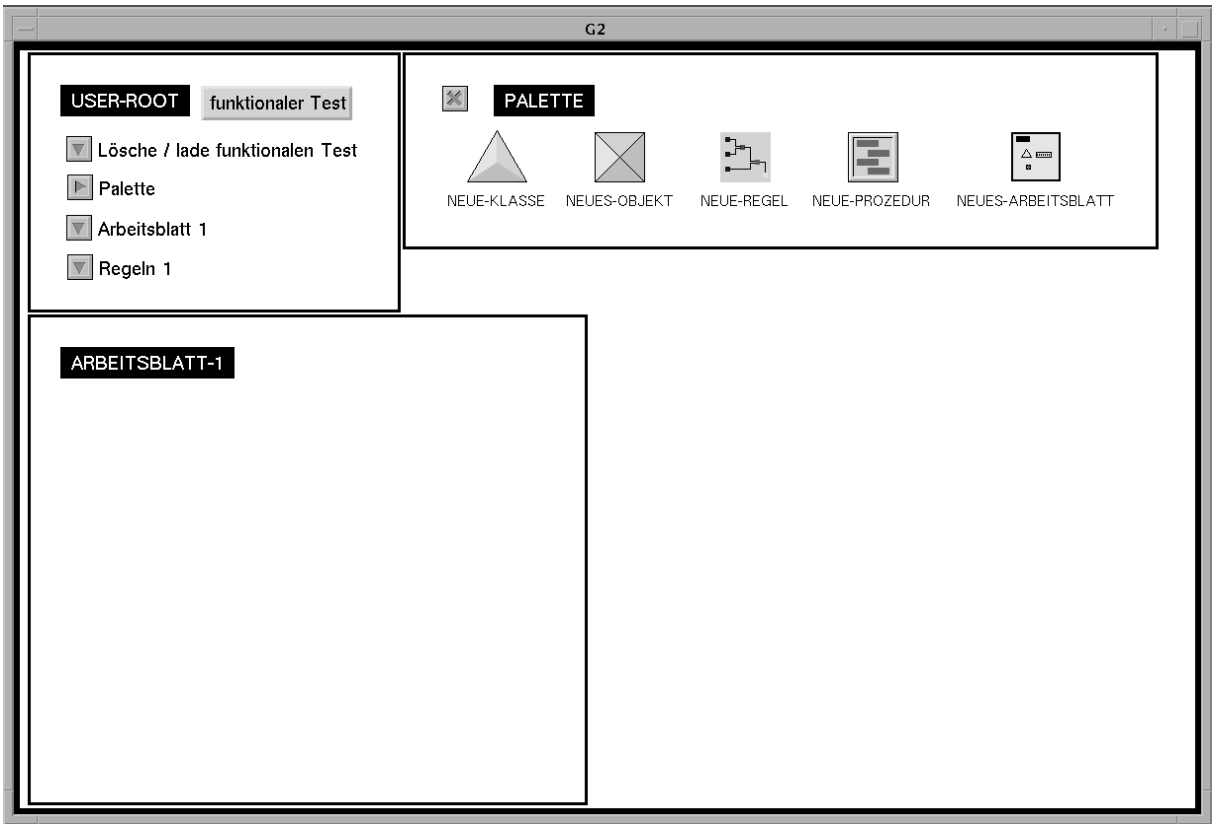


Abbildung 3.6: Startbild einer leeren Wissensbasis mit Konsistenzmodul

Nach dem Anklicken des Sinnbildes "Neues-Objekt" wird eine Kopie des Sinnbildes an den Mauszeiger geheftet, und der Benutzer kann das so symbolisierte Objekt auf einem Arbeitsblatt ablegen. Durch den sofort danach gestarteten Frameeditor wird der erste selbst gestaltete Dialog geöffnet. Dieser bietet die Schaltflächen "Bestehende Klasse" und "Neue Klasse" an, um das Objekt einer Klasse zuzuordnen. Die Reihenfolge "erst ein Objekt erzeugen, dann eine Klasse dafür definieren" kann nur scheinbar durch den Frameeditor realisiert werden. Tatsächlich wird zunächst die Absicht des Benutzers erfaßt und dann in die notwendigen Aktionen in korrekter Reihenfolge umgesetzt.
Da noch keine Klassen durch den Benutzer definiert worden sind, ist die Schaltfläche "Bestehende Klasse" inaktiv und kann nicht betätigt werden. Der mit der Schaltfläche "Neue Klasse" gestartete Dialog erlaubt das Erzeugen und Bearbeiten einer Klassendefinition (Abbildung 3.7).

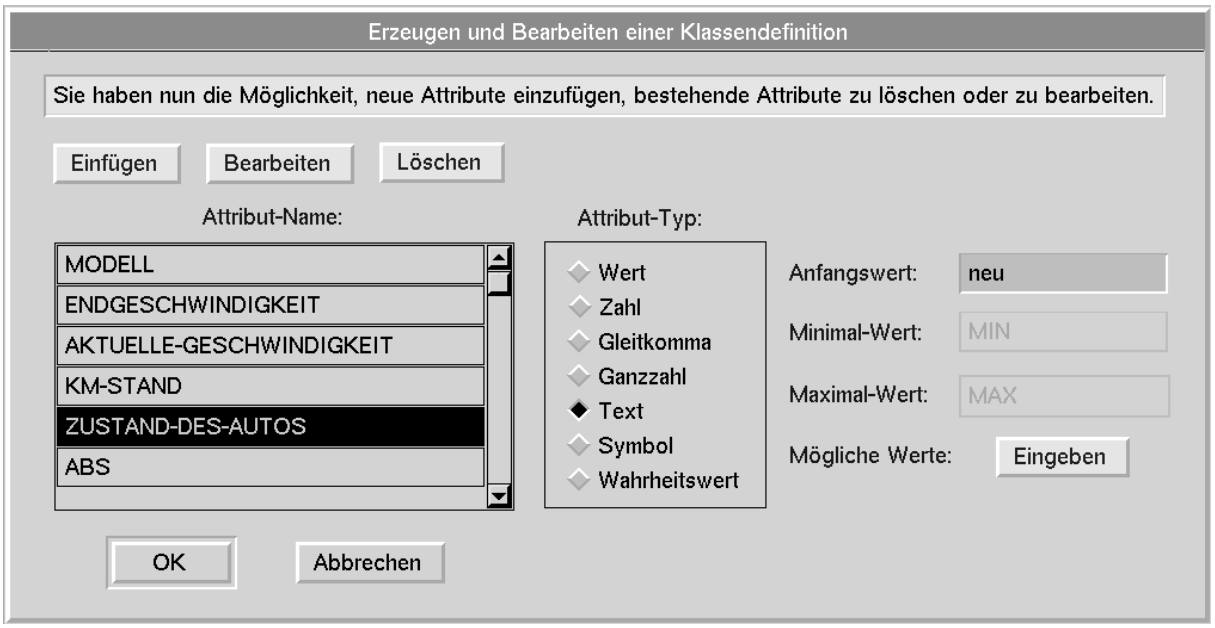


Abbildung 3.7: Dialog zur Erzeugung und Bearbeitung von Klassendefinitionen

Mit dem Start des Dialoges "Erzeugen und Bearbeiten einer Klassendefinition" wird gleichzeitig im Hintergrund überprüft, ob die zu bearbeitende Klasse bereits einen Namen hat und anderenfalls der Name (Neue-Klasse-<Nr.>) zugewiesen.
Eine neue Klasse enthält zunächst keine Attribute, die für den Benutzer direkt zugänglich sind. Das Listenfeld "Attribut-Name" ist daher leer. Der Benutzer muß mit der Schaltfläche "Einfügen" zunächst einen Platzhalter erzeugen und einen beliebigen Attributnamen eingeben. Der Datentyp des Attributes wird durch Auswahl eines Optionsschalters im Block "Attribut-Typ" festgelegt. Hier wurde zunächst eine Auswahl einfacher Datentypen vorgesehen.

Auf der rechten Seite des Dialoges sind Erweiterungen für den Konsistenztest untergebracht. Zusätzlich zum Eingabefeld "Anfangswert" gibt es die Eingabefelder "Minimal-" und "Maximal-Wert" sowie die Schaltfläche "Eingeben" für den Punkt "Mögliche Werte". Die Eingabefelder und die Schaltfläche sind nur dann aktiv, wenn die Eingaben für den gewählten Datentyp sinnvoll sind. Die Funktion des Expertensystems wird ohne Eingabe dieser Werte nicht beeinträchtigt. Auch der funktionale Konsistenztest greift nicht direkt auf die hier angegebenen Werte zurück. Trotzdem ist die Eingabe sehr empfehlenswert, denn bei der Erzeugung von Regeln baut darauf eine effektive Hilfestellung durch den Regeleditor (siehe Kapitel 3.3) auf, die zur Einhaltung der Forderung *F-Fra. 3* dient (S. 36).

Die Schaltfläche "Eingeben" führt auf einen Dialog mit einem Listenfeld für die Werte, die der Benutzer für das Attribut zulassen möchte und drei Schaltflächen für deren Bearbeitung (Abbildung 3.8).

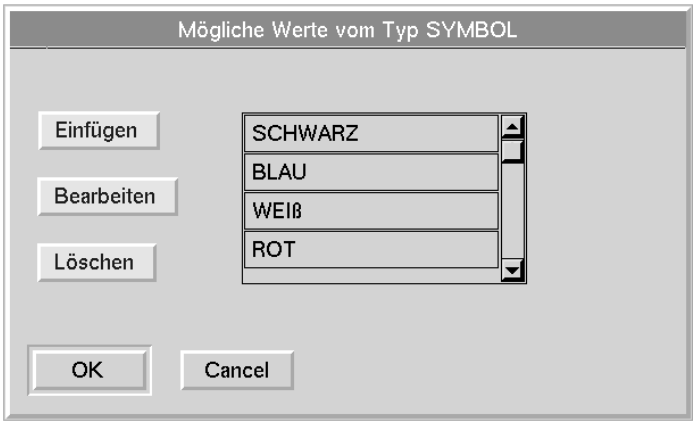


Abbildung 3.8: Eingabe möglicher Werte

Jedes neu erzeugte Listenelement enthält als Vorbelegung einen Wert des erwarteten Datentyps als Beispiel. Die Einträge werden bei Abschluß des Dialoges (OK-Schaltfläche) überprüft. Ist der Datentyp nicht korrekt, erscheint eine Fehlermeldung und der Dialog wird nicht geschlossen. Beendet der Benutzer den Hauptdialog (Abbildung 3.7), findet die gleiche Überprüfung für die dort eingegebenen Werte statt.
Da der Benutzer primär ein neues Objekt erzeugen wollte, wird die Klassendefinition und automatisch eine Instanz dieser Klasse erzeugt.

Werden weitere Objekte erzeugt, steht jetzt die Schaltfläche "Bestehende Klasse" im Startdialog zur Verfügung. Der Auswahldialog bietet alle vom Benutzer erzeugten Klassen in einem Listenfeld an. Der Klassenname kann mit der Maus ausgewählt oder

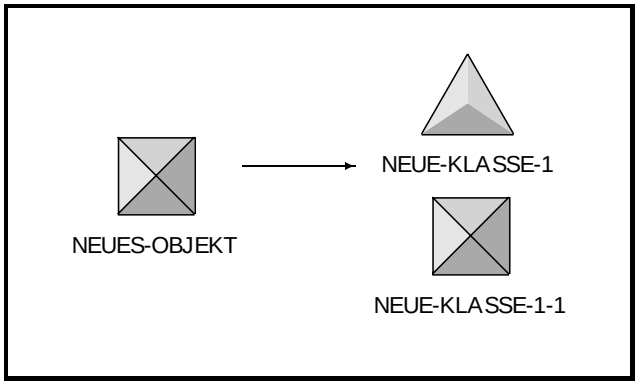


Abbildung 3.9: Entstehung eines Objektes

in einem Eingabefeld direkt eingegeben werden. Nach einer direkten Eingabe wird der Name mit den vorhandenen Klassennamen verglichen und eine Fehlermeldung angezeigt, wenn kein übereinstimmender Eintrag gefunden wird. Auf diese Weise können einerseits Suchvorgänge in langen Listenfeldern vermieden werden, wenn der Klassenname bekannt ist, andererseits werden Schreibfehler abgefangen. Über die Schaltfläche "Neues Objekt erzeugen" wird eine Instanz der gewählten Klasse generiert und erhält den provisorischen Namen "<Klassenname>-<Nr.>" (siehe Abbildung 3.9). Wird der Dialog abgebrochen, weil die gesuchte Klasse nicht vorhanden ist, erscheint der vorherige Dialog und die Schaltfläche "Neue Klasse" kann gewählt werden.

Weitere Aktionen wie die Änderung des Klassennamens oder die ansprechende Gestaltung des Sinnbildes für ein Objekt einer neuen Klasse werden durch Anklicken der Klassendefinition und Auswahl aus dem Menü eingeleitet (siehe Abbildung 3.10). Der Punkt "bearbeiten" öffnet den schon bekannten Dialog "Erzeugen und Bearbeiten einer Klassendefinition", über den Punkt "umbenennen" kommt man zu einem Eingabefeld auf dem Dialog "Änderung des Klassennamens". Sobald hier ein Name eingegeben und bestätigt wurde, werden sowohl die Klassendefinition als auch alle Objekte dieser Klasse umbenannt, aus "**Neue-Klasse-1**" wird z.B. "**Auto**" und aus "**Neue-Klasse-1**-1" wird "**Auto**-1". Selbstverständlich kann jedem Objekt ein individueller Name gegeben werden, indem der Punkt "umbenennen" aus dem Menü des Objektes verwendet wird.

Mit dem Menüpunkt "bearbeite-grafik" wird der systemeigene Grafikeditor für Sinnbilder (Icon-Editor) geöffnet. Einfache Grafikelemente können in mehreren Ebenen (layer) miteinander kombiniert werden, um ein ansprechenderes Sinnbild als das graue Quadrat zu erzeugen (siehe Abbildung 3.10 "Auto-1" gegenüber Abbildung 3.9 "Neues-Objekt" bzw. "Neue-Klasse-1-1").

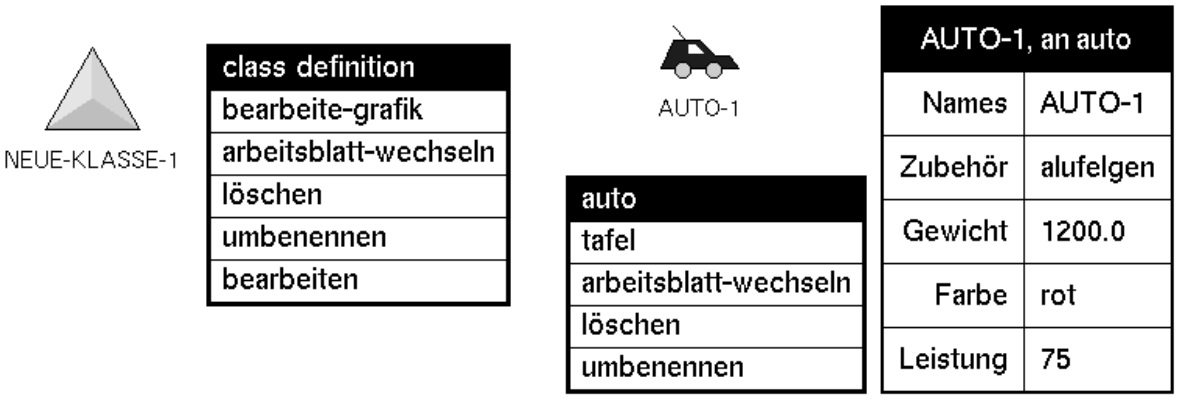


Abbildung 3.10: Menü einer Klassendefinition sowie Menü und Tabelle eines Objektes

Um ein Objekt von einer Arbeitsfläche auf eine andere zu übertragen, muß der Menüpunkt "Arbeitsblatt wechseln" gewählt werden. Bewegt der Benutzer ein Sinnbild durch Anklicken und Gedrückthalten des Mausknopfes auf einer Arbeitsfläche, können die Grenzen der Arbeitsfläche nicht überschritten werden. Statt dessen wird die Fläche vergrößert, wenn der Rand erreicht wird.
Das Menü von Objekten unterscheidet sich vom dem der Klassendefinitionen dadurch, daß der Punkt "bearbeiten" fehlt, neu ist der Punkt "tafel". Der Aufruf dieses Punktes blendet die tabellarische Darstellung der Objekteigenschaften (Attribute) und deren Werte ein (Abbildung 3.10 rechts). Die Konfiguration der Oberfläche beschränkt die Anzeige der Attribute auf solche, die der Benutzer selbst erzeugt hat, und das Systemattribut "Names".

Das Sinnbild "Neue-Klasse" auf dem Arbeitsblatt "Palette" dient zur Erzeugung einer neuen Klasse, von der **zunächst keine Objekte** existieren. Durch Anklicken des Sinnbildes und Ablegen der am Mauszeiger erscheinenden Kopie wird sofort der Dialog "Erzeugen und Bearbeiten einer Klasse" (Abb. 3.7) angezeigt. Nach einem solchen Aufruf wird die neue Klassendefinition, aber kein Objekt der neuen Klasse erzeugt.

Interne Darstellung

Vergleicht man die interne Darstellung der "klassisch" oder mit den Editoren erzeugten Klassendefinitionen und Objekte, ist kein Unterschied feststellbar. Die zusätzlichen Informationen Minimal-, Maximal- und mögliche Werte erscheinen nicht in der Klassendefinition, sondern werden in speziellen Listen abgelegt. Auch die anderen für die Konsistenzkomponente benötigten Informationen wie Klassen-, Objekt- und Attributnamen werden in Listen verwaltet. Die Einträge sind so angeordnet, daß bei Anordnung der Listen nebeneinander eine Tabelle entsteht, in deren Zeilen jeweils zusammengehörige Werte zu finden sind.

Klassen-namen	Attribut-namen	Datentyp	Anfangs-werte	Minimal-wert	Maximal-wert	mögliche Werte
Auto	Zubehör	value	Alufelgen	Min	Max	"Keine"
Auto	Gewicht	float	1200.0	600.0	2000.0	"Keine"
Auto	Farbe	symbol	rot	Min	Max	"schwarz, blau, weiß, rot"
Auto	Leistung	integer	75	60	150	"60, 75, 90, 120, 150"
Fahrrad	Farbe	symbol	blau	Min	Max	"rot, gelb, blau"
Fahrrad	Gänge	integer	3	1	24	"1, 3, 5, 7, 12, 14, 16, 21, 24"

Die Listen mit Klassen- und Attributnamen, Datentyp und Anfangswert enthalten Informationen, die auch in den Klassendefinitionen zur Verfügung stehen. Diese Redundanz bringt jedoch Geschwindigkeitsvorteile (je nach Größe der Wissensbasis und Leistungsfähigkeit der Plattform) durch die Verkettung der Daten über die Listenindizes.

Prinzipiell ist eine Speicherung der Daten in einer erweiterten Klassendefinition möglich. Denkbar ist, das Format der Attributdeklaration in der Klassendefinition um die neuen Informationen zu erweitern.

Statt

Gewicht is a float, initially is 1200.0; ...

kann

Gewicht is a float, initially is 1200.0, min is 600.0, max is 2000.0 oder
Gewicht is a float, initially is 1200.0, "600.0, 800.0, 1000.0, 1100.0, 1500.0, 2000.0"

eingetragen werden.
Dafür sind Änderungen an der internen Datenstruktur notwendig, die nur vom Hersteller der Entwicklungsumgebung vorgenommen werden können.

Eine andere Möglichkeit besteht für den Entwickler der Konsistenzkomponente. Der Dialog zur Erzeugung und Bearbeitung von Klassendefinitionen kann so verändert werden, daß nicht eine originale G2-Klassendefinition erzeugt wird, sondern hier bereits eine zugeschnittene Klassendefinition Verwendung findet. Statt eines vordefinierten Datentyps wird darin einem Attribut der Typ "Konsistenz-Attribut" zugewiesen. Dieser Typ ist eine Klassendefinition, die den eigentlichen Datentyp, Minimum, Maximum und eine Liste der möglichen Werte als Attribute erhält. Nachteil dieses Vorgehens ist, daß die Syntax der Regeln wesentlich komplexer wird (siehe auch Kapitel 3.3, Seite 83).

Nachdem der Benutzer in die Lage versetzt wurde, Objekte zu erzeugen und zu manipulieren, also Fakten in der Wissensbasis zu implementieren, soll auch Verarbeitungswissen in Form von Regeln abgelegt werden.

3.3 Regeleditor

Die Repräsentation von Regeln in wissensbasierten Systemen erfolgt hauptsächlich durch alphanumerische Eingabe des Regeltextes. Die Syntax orientiert sich dabei an Hornklauseln (analog zur Programmiersprache Prolog) oder an aussagen- bzw. prädikatenlogischen Sätzen, wie bei G2.

Der kontextsensitive Texteditor von G2 bietet dem Benutzer Schlüssel**begriffe**, aber keine komplexeren Syntaxblöcke an. Die Syntax der Regeln muß dem Benutzer also zumindest in ihrer groben Struktur bekannt sein, um sinnvolle Begriffe zu wählen. Wird die Eingabe abgeschlossen, überprüft G2 die Regel hinsichtlich der Syntax und der verwendeten Objekte, Klassen und Attribute. Wurden in einer Regel Attribute oder ganze Objekte benutzt, die nicht vorhanden sind, erzeugt G2 Unkorrektheitsmeldungen im Systemattribut "Notes" der Regel und in einem "Operator-Logbook". Beide Stellen müssen vom Benutzer aber aktiv kontrolliert werden, um die detaillierten Hinweise und Meldungen zu erhalten. Ein Benutzer könnte zwar bewußt die Reihenfolge "Regelerzeugung vor Objekterzeugung" gewählt haben, Hauptgrund für diese Meldungen zu fehlenden Objekten ist jedoch ein Namensirrtum oder Schreibfehler.

Der Regeleditor der Konsistenzkomponente dagegen nutzt die syntaktischen Strukturen und den objektbasierten Ansatz von G2, um dem Benutzer Syntaxblöcke in Form von eigens dafür erzeugten grafischen Objekten als Bausteine zur Verfügung zu stellen.
Wie bei der Erzeugung von Objekten und Klassen klickt der Benutzer das Sinnbild "Neue-Regel" des Arbeitsblattes "Palette" an und legt die Kopie des Sinnbildes auf einer beliebigen Arbeitsfläche ab. Das Menü einer Regel besteht aus den Punkten "Arbeitsblatt wechseln", "löschen", "umbenennen" und "bearbeiten". Der letzte Punkt öffnet zwei Arbeitsblätter. Das Arbeitsblatt "Regel-Palette" enthält Bausteine, die auf dem zunächst leeren Arbeitsblatt "Regel-Werkstatt" grafisch zu Regeln kombiniert werden können. In Abbildung 3.11 ist das Vorgehen beispielhaft angedeutet.

Der Benutzer kopiert durch einfaches Anklicken Regelbausteine von der Regel-Palette und legt sie, wie im oberen Teil der Regel-Werkstatt zu sehen, grob ausgerichtet ab. Es ergeben sich dabei drei "gedachte Spalten": ganz links die Eingabe-Bausteine mit Fakten, die logisch verknüpft werden können, in der Mitte ein Vergleichsbaustein und rechts ein Ausgabe-Baustein. Der Aktionsteil (die Konklusion) der Regel wird durch die rechte Spalte repräsentiert, während der Bedingungsteil (die Prämisse) immer durch den Vergleich von Fakten gebildet wird, der hier durch die linke und mittlere Spalte zusammen repräsentiert wird. Sind alle notwendigen Bausteine plaziert, werden sie durch Anklicken

der "Anschlußstellen" an den Blöcken mit einander verbunden, wie es im unteren Teil der Regel-Werkstatt zu sehen ist. Nach der Wertzuweisung für die einzelnen Blöcke über den jeweiligen Menüpunkt "bearbeiten", kann die Erzeugung der "internen" Regel über die Schaltfläche "Regel erzeugen" gestartet werden. Eine erneute Bearbeitung der Regel beginnt dann mit der bereits erstellten Regelgrafik.

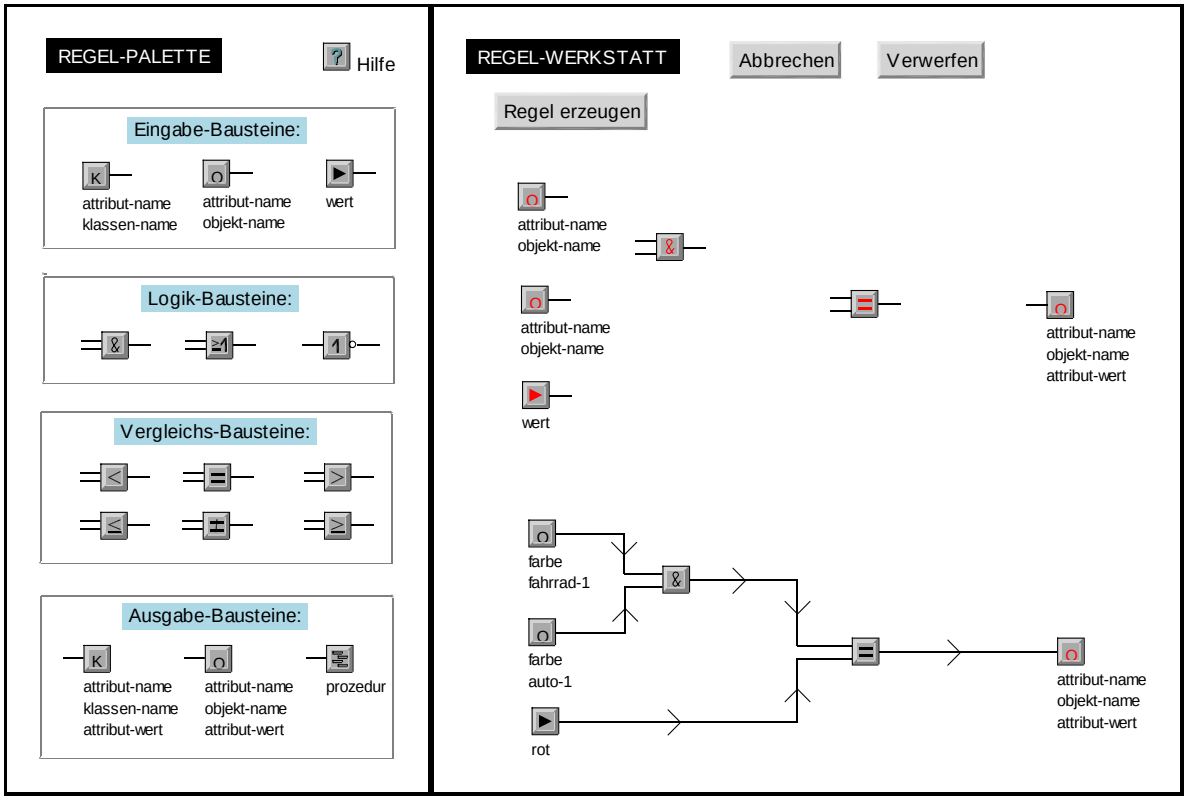


Abbildung 3.11: Arbeitsflächen zur grafischen Erstellung einer Regel

Mit der Regel-Palette werden dem Benutzer vier Bausteingruppen angeboten. **Eingabe-Bausteine** repräsentieren Attribute von Objekten und konstante Werte, die mit Hilfe der **Logik-Bausteine** verknüpft und durch die **Vergleichs-Bausteine** zu Prämissen kombiniert werden können. Aktionen für die Konklusion stehen mit den **Ausgabe-Bausteinen** zu Verfügung.

Objektorientierte Faktenrepräsentation führt zu zwei Ansätzen bei der Formulierung von Regeln. Die Prämissen können auf einen Attributwert eines speziellen Objektes zugreifen (spezifische Regel) oder auf den Attributwert aller Objekte einer Klasse (generische Regel). Für beide Regeltypen steht ein Eingabe-Baustein bereit. Der Baustein für generische Prämissen ist durch ein "K" für Klassen im Sinnbild gekennzeichnet, für spe-

zifische Prämissen wird der Baustein mit der "O"-Kennzeichnung (für Objekt) angeboten. Informationen zu den Bausteinen kann der Benutzer über die " ? "-Hilfe Schaltfläche erhalten.

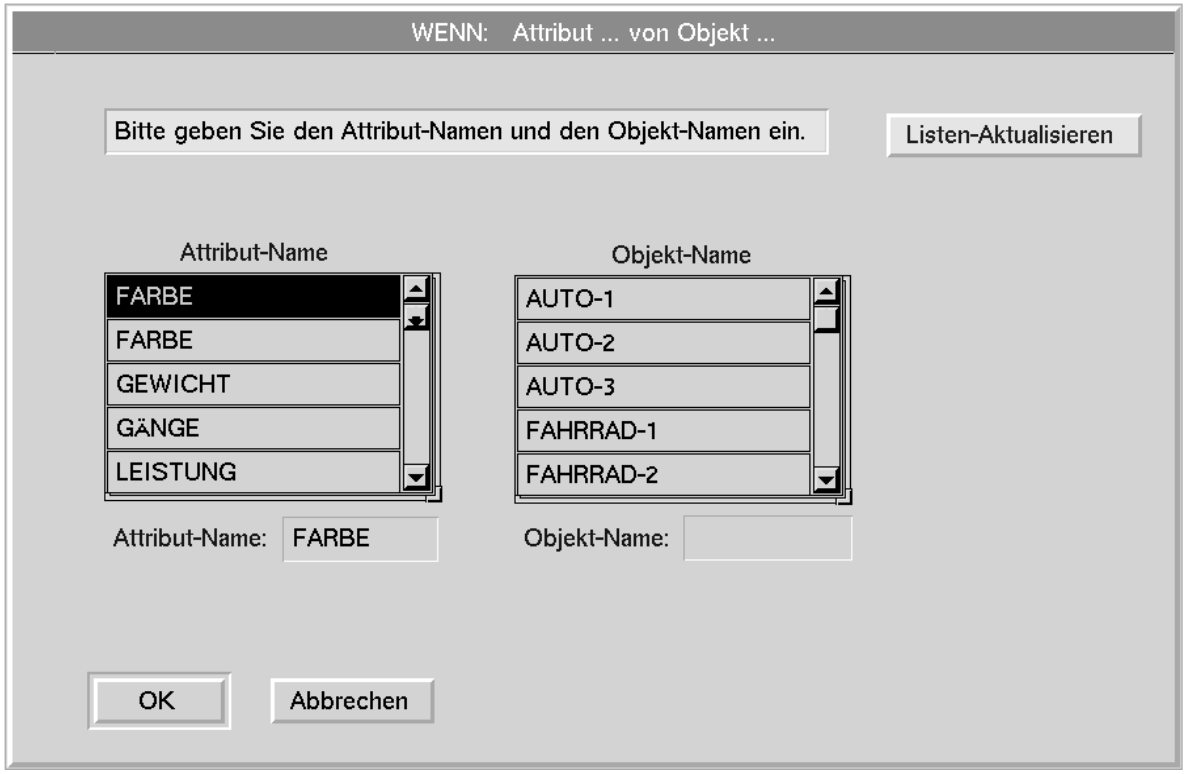


Abbildung 3.12: Dialog zur Eingabe einer spezifischen Regel

Der Dialog für die Eingabe der notwendigen Informationen zu einem **Eingabe-Baustein** bietet zwei Listenfelder an, die alle vom Benutzer erzeugten Attribut- und Objekt- bzw. Klassennamen enthalten. Hier besteht nicht die Möglichkeit, über die Tastatur einen Namen vorzugeben, es können nur bestehende Einträge mit der Maus angeklickt werden (Abbildung 3.12). Wurde in einem Listenfeld eine Wahl getroffen, wird sofort das andere Listenfeld so aktualisiert, daß nur die noch möglichen Einträge angezeigt werden. Wird beispielsweise der Attribut-Name "Gänge" gewählt, erscheinen im rechten Listenfeld nur noch die Objekt-Namen von Fahrrädern, da nur in dieser Klasse das Attribut "Gänge" vorhanden ist. Auch bei diesem Dialog wird also auf die jeweiligen Klassendefinitionen zurückgegriffen. Wählt der Benutzer den Objekt-Namen aus, werden im linken Feld alle Attribut-Namen des Objektes angezeigt. Bei einem Irrtum des Benutzers kann über die Schaltfläche "Listen-Aktualisieren" der Ausgangszustand mit allen Einträgen wieder hergestellt werden. Wie in Abbildung 3.12 können mehrere gleiche Attribut-Namen im Listenfeld angezeigt werden. Diese scheinbare Redundanz wurde bewußt nicht verhin-

dert, da der Benutzer so erfährt, daß beispielsweise das Attribut "Farbe" in zwei Klassen vorkommt. Wird einer dieser Einträge ausgewählt, enthält das rechte Listenfeld in jedem Fall alle Objekte mit diesem Attribut.

Im Beispiel in Abbildung 3.11 soll das Attribut "Farbe" der Objekte "Fahrrad-1" **und** "Auto-1" auf einen konstanten Wert ("rot") überprüft werden. Durch die Verbindung der Bausteine kann für den Dialog zur Werteingabe (Abbildung 3.13) über den "="-Vergleichs-Baustein auf Informationen über die verknüpften Attribute zugegriffen werden. Statt nur einen beliebigen Wert eines beliebigen Datentyps mittels Texteditor einzugeben, bekommt der Benutzer umfangreiche Informationen und Auswahlmöglichkeiten. Durch die (indirekte) Verbindung zum gewählten Attribut ist bekannt, um welchen Datentyp (hier SYMBOL) es sich handeln muß. Sofern an der entsprechenden Stelle Eingaben gemacht wurden, stehen auch der Minimal- und Maximalwert oder die Menge der möglichen Werte (Abbildung 3.13) zur Verfügung. Die Hinweistexte werden den Gegebenheiten jeweils angepaßt und die Schaltfläche "Auswählen" ist nur aktivierbar, wenn tatsächlich Werte zur Verfügung stehen.

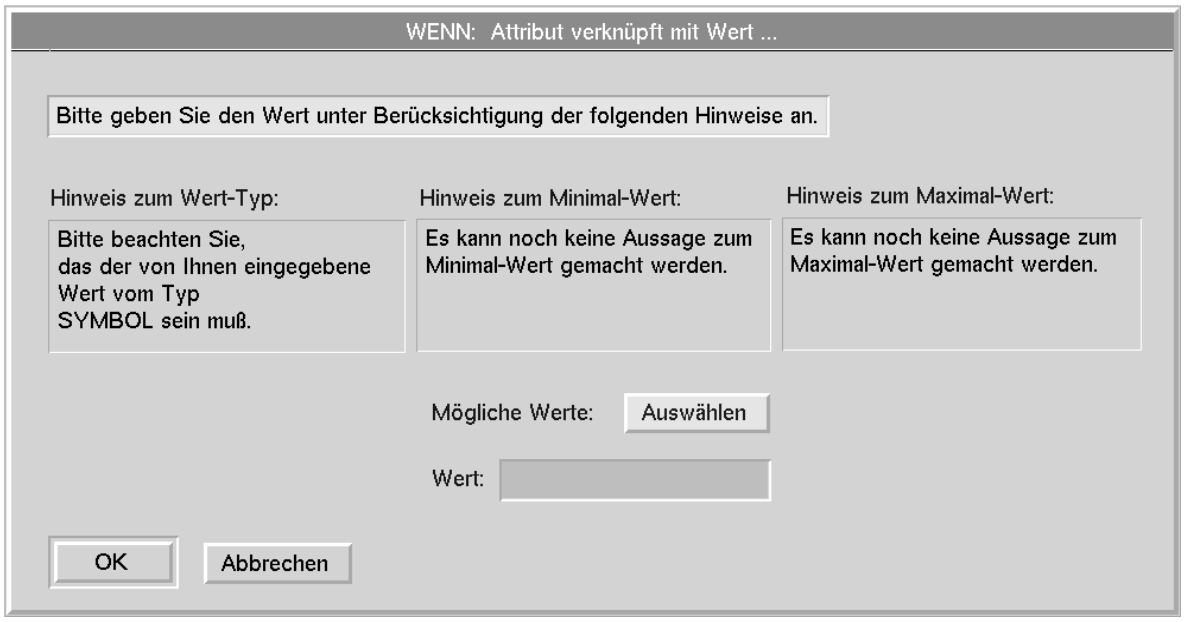


Abbildung 3.13: Dialog für den Eingabe-Baustein "Wert"

Hier macht sich die zusätzliche Arbeit bezahlt, Minimal- und Maximalwerte oder eine Menge möglicher Werte bei der Klassendefinition anzugeben. Der Dialog zur Auswahl möglicher Werte bietet besonders bei komplexeren Prämissen eine wertvolle Hilfe, was an dem Beispiel in Abbildung 3.11 deutlich wird.

In der Prämisse der Regel tritt die Konjunktion des Attributes "Farbe" von "Fahrrad-1" und "Auto-1" auf und soll mit einem konstanten Wert verglichen werden. Die möglichen Werte für "Auto-1" sind "SCHWARZ, BLAU, WEIß, ROT", für "Fahrrad-1" "BLAU, GELB, ROT".
Die Bedingung, daß die Farbe eines Autos **und** eines Fahrrades gleich einem bestimmten Wert sein müssen, enthält indirekt die Bedingung, daß die Farben gleich sind. Andere Werte als solche aus der Konjunktion der Wertemengen sind nicht sinnvoll, da die Prämisse dann nicht erfüllt werden kann.
An dieser Stelle wird **sogar für die Verknüpfung von Attributwerten** die Konsistenzforderung *F-Fra. 3* (S. 36) berücksichtigt und ihre Einhaltung durch die entwickelten Dialoge erleichtert bzw. erzwungen. Im Listenfeld der möglichen Werte des "Wert"-Eingabe-Bausteins erscheint dazu die Konjunktion der Wertemengen: "BLAU, ROT" !

Der Dialog zu einem **Ausgabe-Baustein** ist eine Zusammenfassung der Attribut- und Wert-Dialoge eines Eingabe-Bausteins (Abbildung 3.14). Nach der Auswahl von Attribut- und Objekt-Name steht die Information über das Attribut in komprimierter Form als Hinweistext zur Verfügung. Der Wert, der dem Attribut zugewiesen werden soll, kann direkt in einem Eingabefeld oder aus den "möglichen Werten" bestimmt werden.

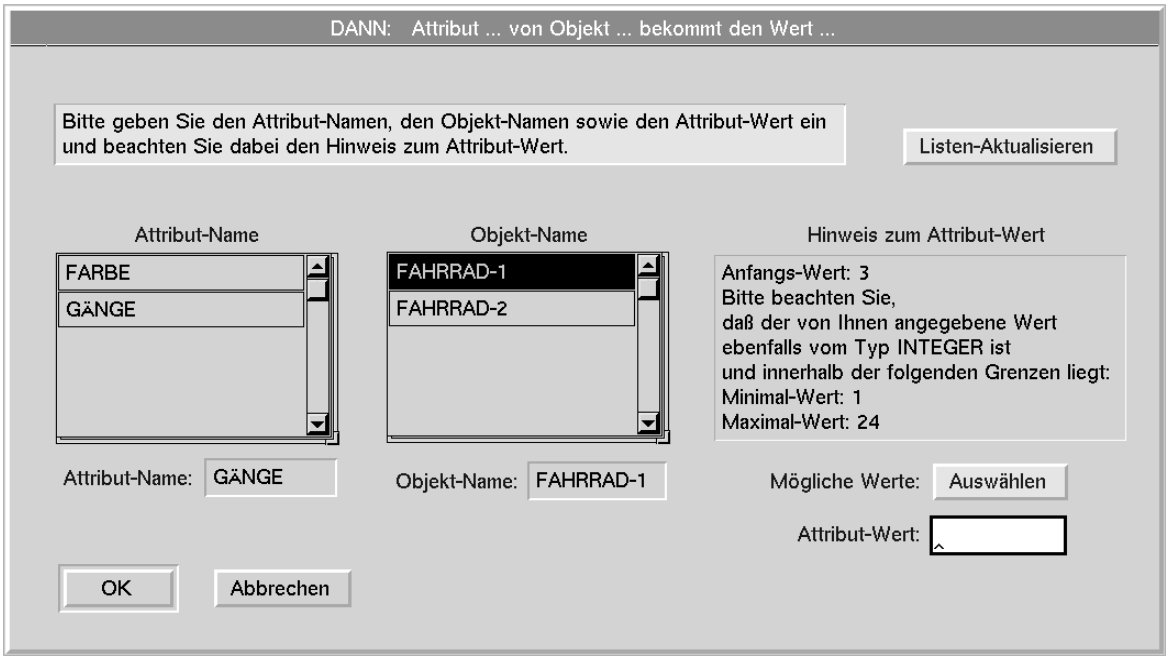


Abbildung 3.14: Dialog zum Setzen eines Attributwertes

Die Kombination aus möglichen Regelbausteinen, deren Auswahldialogen und den darin angezeigten Informationen erfüllt die Forderung *F-Regel* (S. 39) und ermöglicht dem Be-

nutzer, etwaige Verstöße gegen die Forderungen *F-Fra. 1* bis *3* an verschiedenen Stellen zu entdecken. Somit ist eine gute "manuelle" Kontrolle der Abbildung realer Objekte durch den Benutzer gegeben.

Die bisher vorgestellten Dialoge beziehen sich auf spezifische Prämissen und Konklusionen (Bausteine mit dem "O"-Symbol). Selbstverständlich wurde die gleiche Funktionalität auch für die generischen Bausteine ("K"-Symbol) realisiert. An Stelle der Auswahllisten für Objekt-Namen enthalten die Dialoge dann Listen mit den Namen der vom Benutzer erzeugten Klassendefinitionen.

Um komplexere Aktionen durch eine Regel ausführen zu lassen, steht neben der direkten Wertzuweisung als dritter Ausgabe-Baustein ein **Prozeduraufruf** zur Verfügung. Der Auswahldialog bietet in einem Listenfeld die Namen aller vom Benutzer erstellten Prozeduren an. Sobald ein Name ausgewählt wurde, erscheint ein zugehöriger Hilfe-Text (siehe Kapitel 3.4, Seite 86) und wenn nötig wird die Schaltfläche "Parameter" aktiviert. Entscheidet sich der Benutzer für eine Prozedur mit Parameterübergabe, werden nach Anklicken der "Parameter"-Schaltfläche alle Werte mit Angabe ihres Datentyps abgefragt. Die prozedurinterne Bedeutung der Parameter sollte aus dem Hilfe-Text hervorgehen, den der Benutzer bei der Prozedurerstellung formulieren kann.

Durch die Art der Informationsauswertung im "Regeleditor" werden sowohl Schreibfehler als auch logische Fehler mit geringem Aufwand verhindert. Zusätzlich wäre es über die Auswertung aller Regelgrafiken möglich, Regeln anzuzeigen, die mindestens ein Attribut enthalten, das auch in der gerade bearbeiteten Regel verwendet wird. Bei großen Regelmengen ist es aber nicht mehr praktikabel, die entsprechenden Regelgrafiken auf eigenen Arbeitsblättern anzuzeigen. Es muß eigens eine geeignete Anzeigeform entwickelt werden, die sowohl eine schnelle Übersicht als auch einen einfachen Zugriff erlaubt. Entstehen ohne diese Hilfe Redundanzen, die zu Inkonsistenzen führen, werden diese durch den funktionalen Test gefunden und können dort behoben werden. Hier kann Entwicklungsaufwand eingespart werden, wenn "nur" Konsistenz und nicht Benutzerfreundlichkeit gefordert wird.

Hat der Benutzer eine Regelgrafik vollständig aufgebaut und alle notwendigen Informationen eingegeben, kann das Arbeitsblatt mit der Schaltfläche "Regel erzeugen" geschlossen werden. Die damit gestartete Überprüfung stellt sicher, daß alle Bausteine die notwendigen Verbindungen besitzen, alle Informationen eingegeben wurden und die Datentypen der Werte zu den Attributen passen. Wird einer der Punkte nicht erfüllt, erscheint

eine Fehlermeldung und das Arbeitsblatt "Regel-Werkstatt" wird erneut angezeigt. Ein aufmerksamer Benutzer kann allerdings schon bei der Erstellung der Regelgrafik Fehler vermeiden. Die Sinnbilder der verwendeten Bausteine zeigen rote Symbole, solange Verbindungen oder Informationen fehlen. Wurde ein verbundener Baustein bearbeitet und alle Angaben sind vorhanden, erscheint das Symbol in schwarz.

Es gibt zwei weitere Möglichkeiten, das Arbeitsblatt zu schließen. Soll die Regel-Werkstatt verlassen werden, ohne eine Änderung vorzunehmen, geschieht dies über die Schaltfläche "Abbrechen". Eine bereits vorher erzeugte Regel bleibt dabei erhalten und es findet keine erneute Prüfung statt. Die Schaltfläche "Verwerfen" löscht dagegen die eventuell bestehende Regel und ermöglicht so einen Neuaufbau auf einem leeren Arbeitsblatt.

Interne Darstellung

Mit der Schaltfläche "Regel erzeugen" wird nicht nur eine Überprüfung der Regelgrafik gestartet, sondern auch der G2-interne Regeltext erzeugt. Die Klasse "rule" besitzt ein Attribut ohne Namen vom Typ Text, das als Wert die eigentliche Regel enthält. Wird der Text mit dem kontextsensitiven Editor direkt eingegeben, muß der Benutzer einige syntaktische Feinheiten in Abhängigkeit von den Datentypen beachten. Da die Regelsyntax nach bekannten Regeln aufgebaut ist, kann aus der Regelgrafik des Benutzers automatisch ein Regeltext mit der korrekten Syntax generiert werden.
Die Inferenzmaschine greift für die Wissensverarbeitung auf die so erzeugte "original G2-Regel" zu, während für den Benutzer die Regelgrafik das Wissen repräsentiert. Die durch den Benutzer erzeugten Regel-Objekte besitzen jeweils ein Unterarbeitsblatt, das sowohl die Regelgrafik als auch die "original G2-Regel" enthält. Für die Bearbeitung der Regel durch den Benutzer wird die zugehörige Regelgrafik auf das Arbeitsblatt "Regel-Werkstatt" übertragen.

Die automatische Erzeugung der Regeltexte wird um so komplexer, je komplexer die syntaktischen Strukturen sind, die verwendet werden müssen. Kritisch sind dabei die Zugriffe auf Attributwerte, sei es um sie für einen Vergleich abzufragen, sei es um neue Werte zu setzen. Die einfachste Struktur ist ein Zugriff der Form

die *Eigenschaft* von *Objekt* oder in G2-Notation z.B. the *farbe* of *Auto-1*

Voraussetzung dafür ist die "einfache" Struktur der Klassendefinition mit

class:	Auto
class specific attributes:	farbe is a text

In Kapitel 3.2 wurde bei der internen Darstellung der Objekte die Möglichkeit angesprochen, zugeschnittene Klassendefinitionen zu verwenden (siehe Seite 76).
Eine Klassendefinition mit einem um eine Stufe weiter geschachtelten Attribut wie z.B.

class:	Auto
class specific attributes:	farbe is a konsistenz-attribut-text

und

class:	Konsistenz-Attribut-Text
class specific	wert is a text;
attributes:	min is a text;
	max is a text;
	mgl-werte is a text-list;

macht auch eine Schachtelung des Zugriffs notwendig:

die *Eigenschaft* von (der *Eigenschaft* von **Objekt**)

in G2-Notation: the *wert* of the *farbe* of *Auto-1*

Der normale Benutzer der Konsistenzkomponente soll von der internen Struktur der Datenorganisation Vorteile haben. Die Auswahlmöglichkeiten auf den Dialogen enthalten bereits alle notwendigen Informationen, dürfen bei der Verwendung zugeschnittener Klassendefinitionen also nicht umfangreicher werden. Neben den Angaben "Klassen-Name", "Objekt-Name" und **ein** "Attribut-Name" (z.B. *farbe* in obiger G2-Notation) müsste nach obiger Schachtelung noch ein **zweiter** "Attribut-Name" (hier *wert*) durch den Benutzer ausgewählt werden. Dies ist nicht im Sinne der Benutzerfreundlichkeit und daher werden keine zugeschnittenen Klassendefinitionen sondern zusätzliche Listen wie in Kapitel 3.2 (Seite 75) erläutert wurde, gewählt.

Die Einträge in den Listenfeldern der Dialoge stammen aus speziellen Listen, die bei Verwendung der Konsistenzkomponente zur Objekt- und Klassenerzeugung angelegt bzw. aktualisiert werden. Es ist möglich, diese Listen bei jedem Dialogaufruf neu zu erzeugen, indem alle egal auf welche Weise erstellten Objekte und Klassen gesucht und ihre Eigenschaften extrahiert werden. Bei großen Wissensbasen wird dann jedoch viel Zeit benötigt was den Vorteil zunichte macht, auch ohne die Konsistenzkomponente erstellte Objekte und Klassen zu erfassen.

3.4 Prozeduren

Als Entwicklungswerkzeug für hybride wissensbasierte Systeme bietet G2 die Systemklasse "Prozedur" an. Innerhalb der "Prozedur"-Objekte ist eine Verknüpfung von Wissen in einer genau vorgegebenen Reihenfolge realisierbar. Die innere Struktur lehnt sich an imperative Programmiersprachen wie Fortran oder Pascal an, wobei das Hauptaugenmerk auf den Umgang mit Objekten gerichtet ist.
Im einfachsten Fall dient eine Prozedur als Zusammenfassung von Anweisungen zum spezifischen Setzen von Attributwerten, da solche Aktionen unnötig große und unübersichtliche Regeln erfordern würden. Die Berechnung eines Attributwertes auf der Basis anderer Attributwerte kann innerhalb einer Prozedur deutlich komplexere Verfahren beinhalten als in einer Regel, und ebenso wie dort sind generische Anweisungen des Inhaltes "für alle Objekte der Klasse ..." möglich. Prozeduren können außerdem mit Parametern aufgerufen werden und Werte an die aufrufende Stelle (im allgemeinen eine andere Prozedur) zurückgeben.

Im G2 Basissystem kann der Benutzer mit dem kontextsensitiven Editor einen Prozedurtext erzeugen, der während der Eingabe auf syntaktische Fehler geprüft wird. Beim Abschluß der Eingabe wird zusätzlich die Anzahl der öffnenden und schließenden Klammern verglichen und kontrolliert, ob alle lokalen Variablen im Prozedurkopf deklariert wurden. Man kann die Funktion des Basissystems als ein Mittelding ansehen zwischen einem reinen Texteditor und einem Editor mit nachfolgendem Compiler, wie sie bei der Entwicklung imperativer Programme verwendet werden. Für den Benutzer ergeben sich daraus die gleichen "Probleme" wie für alle Softwareentwickler. Die Übersicht z.B. über die funktional korrekte Anordnung von Klammern wird durch das System nicht unterstützt.

Auch für Prozeduren wurde daher ein neuer dialogorientierter Editor geschaffen, der den Benutzer so weit wie möglich unterstützt. Wie bei Klassen, Objekten und Regeln wird auch eine Prozedurerstellung durch Anklicken des Sinnbildes auf dem "Palette"-Arbeitsblatt und Ablegen der Kopie auf einem beliebigen anderen Arbeitsblatt eingeleitet.

Der damit geöffnete Dialog dient der Eingabe eines Prozedurnamens, eventuell benötigter Parameter und eines Textes, der als Kommentar zur Erklärung der Prozedur**funktion** verwendet wird. Dieser Text wird im Dialog für einen Prozeduraufruf im Aktionsteil einer Regel als Hilfe-Text angezeigt. Zur Erleichterung der Prozedurauswahl sollte wenigstens ein kurzer Hinweis eingegeben werden. Benötigt die Prozedur Parameter, sollte deren Verwendungszweck beschrieben werden. Dieser Dokumentations-Kommentar wird im Prozedurkopf abgelegt.

Mit einer Schaltfläche "Vorhandene Prozeduren" kann der Benutzer ein Listenfeld anzeigen lassen, das die Namen der bereits erzeugten Prozeduren enthält. Damit ist es einfacher, gut unterscheidbare, "sprechende" Namen zu wählen.
Werden voraussichtlich Parameter für die Prozedur benötigt, wird mit der Schaltfläche "Parameter" ein Eingabedialog geöffnet. Hier kann, ähnlich wie bei der Eingabe von "möglichen Werten" für Attribute, ein Parametername in ein Listenfeld eingetragen und der zugehörige Typ mit Optionsschaltern festgelegt werden. Diese Angaben können auch noch nachträglich verändert oder ergänzt werden.
Bei Abschluß des Dialoges über die "OK"-Schaltfläche wird geprüft, ob der notwendige Prozedurname eingegeben wurde und danach werden wie bei einer Regelbearbeitung die Arbeitsblätter "Prozedur-**Palette**" und "Prozedur-**Werkstatt**" geöffnet.

Die Palette bietet verschiedene funktionale Blöcke für eine Prozedur an, die zum Aufbau einfacher Strukturen verwendet werden können. Jeder Block stellt eine Syntaxstruktur dar, die ein oder mehrere weitere Angaben benötigt. Der Benutzer klickt einen Block auf der Palette an und legt die entstehende Kopie in der Werkstatt ab. Nachdem der Eingang des Blockes mit dem Ausgang des vorhergehenden Blocks oder des "Start"-Punktes verbunden wurde, kann der Block durch einfaches Anklicken bearbeitet werden.

"Kommentar"-Blöcke erlauben nach dem Anklicken die Eingabe eines einfachen Textes. Es ist dem Benutzer überlassen, ob er Stichworte verwendet, oder umfangreiche Beschreibungen zur Funktion der folgenden Anweisungen eingibt. Der Kommentar zur Prozedurfunktion kann über den "Start"-Punkt auch von dieser Stelle aus bearbeitet werden.

Der Dialog des "Setze Attributwert"-Blocks entspricht dem des "O"-Ausgangsblocks bei Regeln (Abbildung 3.14). Attribut- und Objektname werden aus den angezeigten Listen ausgewählt und ein Wert wird direkt eingegeben oder aus "möglichen Werten" bzw. "lokalen Variablen" ausgewählt.
Eine Besonderheit ergibt sich dann, wenn ein Attributwert innerhalb einer generischen "*For*-Schleife" gesetzt wird. Wird als Objektname die lokale Variable (z.B. X) gewählt, über die mit der Schleife (Für alle Objekte der Klasse "Auto" oder in G2-Notation: **for** X = **each** AUTO **do**) iteriert wird, erscheinen in der Liste der Attributnamen auch nur diejenigen Attribute, die in der Definition der gewählten Klasse angegeben sind. Voraussetzung dafür ist die grafische Verbindung der Blöcke **vor** deren Bearbeitung.

Der funktionale Block "hole Wert" ist bei einer Prozedurentwicklung mit dem Texteditor nicht unbedingt erforderlich. Innerhalb einer Prozedur kann mit der Syntax "die **Eigen-**

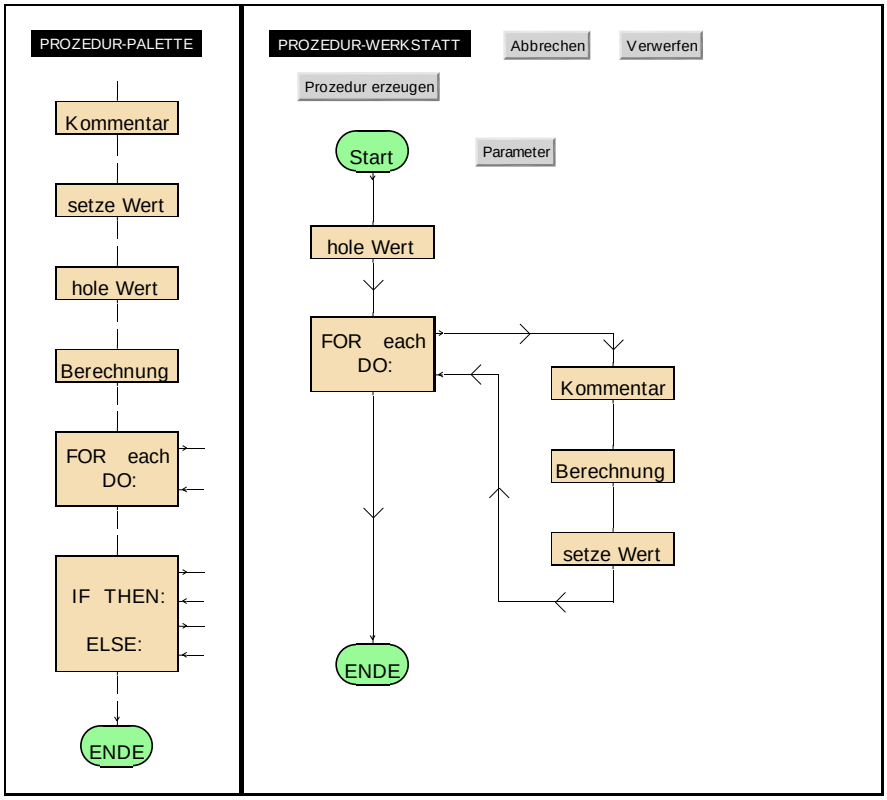


Abbildung 3.15: Arbeitsflächen zur grafischen Erstellung einer Prozedur

schaft von **Objekt**“ direkt auf den Wert eines beliebigen Attributes eines Objektes zugegriffen werden. Eine Formel zur Berechnung eines neuen Wertes kann dadurch aber auch sehr unübersichtlich werden. Besser ist es, jeden benötigten Attributwert ”in die Prozedur zu holen“ und einer lokalen Variable zuzuweisen um damit die Berechnungen durchzuführen. Diese Zuweisung kann mit dem ”hole Wert“-Block vorgenommen werden.

Berechnungen jeglicher Art werden mit dem ”Berechne“-Block realisiert. Einer lokalen Variable kann eine Berechnungsformel mit beliebigen Rechenschritten zugeordnet werden. In einem darauf folgenden ”Setze Attributwert“-Block steht als Wert dann auch diese Variable zur Verfügung.

Um die schon angesprochenen ”*For*-Schleifen“ aufzubauen, steht ein weiterer Block zur Verfügung. Kennzeichen ist der zusätzliche seitliche Ein- und Ausgang. Eine Schleife besteht aus Kopf, Rumpf und Fuß oder Schluß. Der ”*For*-Schleife“-Block enthält nur Kopf- und Schlußteil. Der Rumpf kann aus einer beliebigen Kombination funktionaler Blöcke aufgebaut werden, die zwischen dem seitlichen Ein- und Ausgang ”angeschlossen“ sein müssen.

Mit dem gleichen Verfahren werden auch die Rumpfteile einer bedingten Anweisung aufgebaut. Der "If-then-else"-Block enthält die zu erfüllende Bedingung. Die Aktionen für den Fall der Erfüllung ("then") oder Nichterfüllung ("else") werden mit funktionalen Blöcken gebildet, wobei für den "else"-Teil auch eine direkte Verbindung von Aus- und Eingang zulässig ist. Der "else"-Teil ist somit leer. Als letzte Funktion muß ein "Ende"-Block gesetzt werden, um die Prozedur abzuschließen.
Eine weitere Hilfestellung für den Benutzer ist das Arbeitsblatt "Übersicht", das neben der "Prozedur-Palette" und der "Prozedur-Werkstatt" geöffnet wird. Hier werden in Listenfeldern alle Klassen-, Objekt- und Attributnamen sowie der Datentyp, Minimal-, Maximal- und mögliche Werte angezeigt. Während der Prozedurentwicklung kann so "in den Bestandslisten geblättert" werden, um Schreibfehler und Irrtümer zu vermeiden.

Werden während der Entwicklung der Prozedur zusätzliche Parameter benötigt, kann über die Schaltfläche "Parameter" neben dem "Start"-Punkt der gleiche Eingabedialog aufgerufen werden, wie bei dem ersten Prozedur-Dialog.
Nach Abschluß der Prozedurentwicklung wird durch die "Prozedur erzeugen"-Schaltfläche die grafische Repräsentation in die G2-interne Form umgesetzt und der Dialog geschlossen. Eine spätere Bearbeitung der Prozedur ist durch den Menüpunkt "bearbeiten" möglich, der das gleiche Arbeitsblatt wie bei der Prozedurerzeugung öffnet. Als weitere Schaltflächen stehen noch "Abbrechen" und "Verwerfen" zur Verfügung um die Bearbeitung ohne Änderungen zu beenden, bzw. eine ganz neue Prozedur-Grafik aufzubauen.

Interne Abläufe
Bei Prozeduren ist die Prüfung der Struktur im Gegensatz zu Regeln eher rudimentär, da hier die Theorien zur Korrektheitsprüfung von Programmen ([28]) eingebracht werden müssen, was nicht Gegenstand dieser Arbeit ist. Die Umsetzung der einzelnen Blöcke erfolgt unter Beachtung der in Kapitel 3.5 auf Seite 94 begründeten Randbedingungen für Kommentare und Schleifen.
Die Inferenzmaschine greift nicht auf die grafische Repräsentation der Prozedur zu, sondern auf den in Maschinencode übersetzten Prozedurtext. Die bei der Übersetzung eventuell auftretenden Fehlermeldungen des Compilers werden dem Benutzer auf einem neuen Arbeitsblatt angezeigt.

3.5 Funktionaler Test

Die Voraussetzungen für einen Inferenzvorgang sind mit der Implementierung der Fakten und ihrer Verknüpfungen geschaffen worden. Der Benutzer kann nun testen, ob alle Verknüpfungen (Regeln und Prozeduren) auch zum gewünschten Zeitpunkt (genauer: bei einem bestimmten Zustand der Wissensbasis) das gewünschte Ergebnis liefern. Um bei diesem Funktionstest auch gleich die Konsistenz des Systems zu prüfen, muß der Benutzer die Schaltfläche "funktionaler Test" auf dem Arbeitsblatt "User-Root" betätigen, bevor er den geplanten "normalen" Start der Inferenz auslöst.

Das Arbeitsblatt "Funktionaler Test" (Abbildung 3.16) bietet unter einem Begrüßungstext zwei Gruppen von Optionsschaltern und eine Schaltfläche "Start" an. Die Optionsschalter zur Sprachauswahl ermöglichen die Ausgabe von Mitteilungen zum Testablauf in englischer oder deutscher Sprache. Der Ablauf des Tests kann mit den Optionsschaltern zur Ablaufart beeinflußt werden. "Einzelschrittbestätigung" erfordert durch die "schrittweise" Inferenz (siehe Kapitel 2.3.5, S. 55) mehr Aktionen vom Benutzer, macht aber den internen Ablauf deutlich, der bei der "automatischen Weiterschaltung" im Hintergrund abläuft. Ist die Auswahl getroffen, starten mit Betätigung der "Start"-Schaltfläche die Vorbereitungen zum Beginn der schrittweisen Inferenz. Der Benutzer wird durch Mitteilungen auf einem gesonderten Arbeitsblatt über zeitintensive Vorgänge informiert und zu bestimmten Handlungen aufgefordert.

Die Komponente für den funktionalen Test verfolgt die Änderungen der Attributwerte und entdeckt Inkonsistenzen. Dazu ist ein schrittweiser Ablauf der Inferenz notwendig, der mit Hilfe der in Kapitel 2.3.5 erarbeiteten Objektstruktur ($\hat{=}$ Framestruktur, siehe Kapiteleinleitung S. 63) realisiert wurde. Lesende Zugriffe auf Attributwerte erfolgen weiterhin auf die vom Benutzer erstellten Originalobjekte (Eingangsobjekte). Schreibende Zugriffe hingegen greifen auf Ausgangsobjekte zu, die ohne Beteiligung des Benutzers erstellt und verwaltet werden und die den Aufbau besitzen, der zur Erfüllung der Forderung *F-funk. 3* notwendig ist (Abbildung 2.3).

Üblicherweise initiiert der Benutzer durch bestimmte Aktionen die Schlußfolgerungskette der entwickelten Applikation. Häufig wird als "Startsignal" die Betätigung einer Schaltfläche vorgesehen, mit der entweder direkt durch Anweisungen oder indirekt durch einen Prozedur- oder Regelaufruf Attributwerte verändert werden. Sind alle "Startaktionen" durch den Benutzer vorgenommen worden, werden die Ausgangsobjekte auf Problemkennzeichen wie Doppelbelegung von Attributen oder Oszillationen überprüft (Forderung *F-funk. 2*, S. 54).

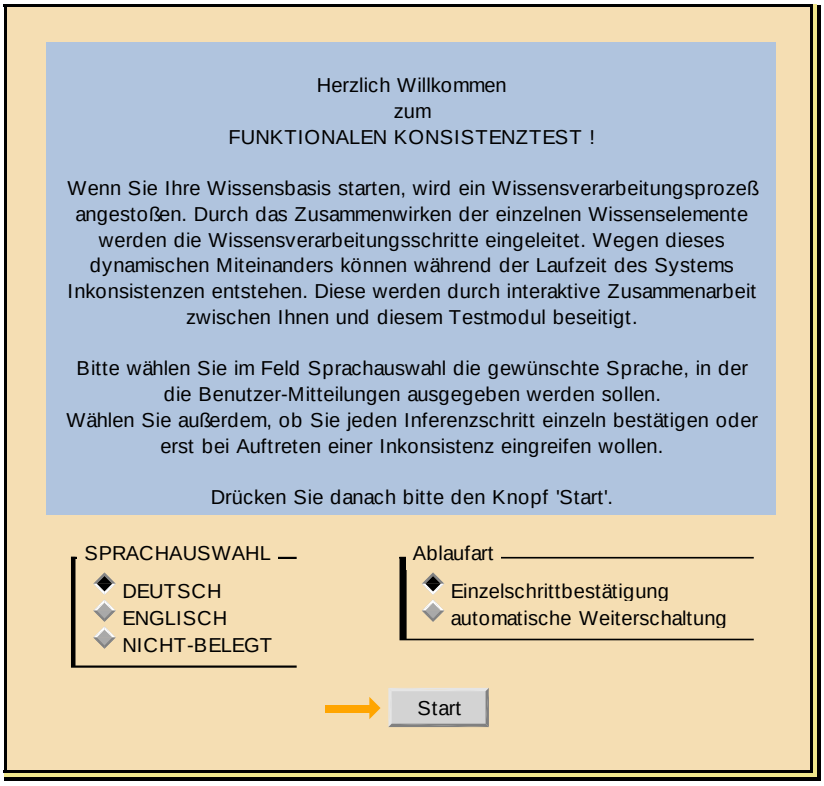


Abbildung 3.16: Start-Arbeitsblatt des funktionalen Tests

Die Inferenzkette wird durch Betätigen der Schaltfläche "Nächster Schritt" gestartet, die in der rechten unteren Ecke des Bildschirms auf einem neuen Arbeitsblatt eingeblendet wird. Bei "automatischer Weiterschaltung" wird die Inferenz erst dann gestoppt, wenn möglicherweise ein Problem vorliegt. Wurde "Einzelschrittbestätigung" gewählt, muß der Benutzer immer den nächsten Inferenzschritt anstoßen, egal ob ein Problem erkannt wurde oder nicht. Beide Varianten erfüllen die Forderung *F-funk. 1* (S. 53).

Sobald die Konsistenzkomponente eine Inkonsistenz erkennt, werden entsprechende Meldungen auf dem zusätzlich erscheinenden Arbeitsblatt "Konsistenz-Informationen" ausgegeben. Dort werden auch die als Quelle der Inkonsistenz festgestellten Objekte plaziert und stehen sofort für eine Überarbeitung zur Verfügung. Ein speziell generierter Hinweistext gibt an, welches Attribut welchen Objektes in allen Quellobjekten vorkommt und unterschiedliche Werte erhält. Dies ist besonders bei Beteiligung von Prozeduren wichtig, da für jede Prozedur einzeln der Editor aufgerufen werden muß, um Einblick in den "Programmcode" zu erhalten. Ein schnelles Erkennen der Inkonsistenzursache ist daher auch für einen geübten Programmierer nicht ohne Hilfe möglich. Der Benutzer kann selbst entscheiden, wieviele und welche Quellobjekte er verändert. Mit den Schaltflächen

"Editierung abgeschlossen" und "Nächster Schritt" wird das Arbeitsblatt geschlossen und mit den korrigierten Objekten der letzte Inferenzschritt wiederholt. Gleiches geschieht bei Auftreten einer Oszillation.

Am Ende der Inferenzkette muß die Kette noch einmal von vorne gestartet werden, wenn Änderungen vorgenommen wurden, um sicher zu stellen, daß dadurch nicht an anderer Stelle Probleme auftreten. Nach einem fehlerfreien Durchlauf hat der Benutzer die Wahl, das Modul des funktionalen Konsistenztests aus seiner Wissensbasis zu entfernen, oder für weitere Tests deaktiviert in den Hintergrund zu legen.

Der Vorteil der "Einzelschrittbestätigung" liegt in der Möglichkeit, Attributwerte zu verfolgen. Der Benutzer kann sich von einem Objekt die Tabelle der Eigenschaften anzeigen lassen und nach jeder Weiterschaltung die neuen Attributwerte kontrollieren. Dies ist die einzige Möglichkeit, das Auftreten definitiv falscher Werte zu entdecken, die beispielsweise durch eine nicht korrekte Prozedur gesetzt werden. An dieser Stelle besteht noch Entwicklungsbedarf, um dem Benutzer Zugriff auf die Quellen eines bestimmten Attributwertes auf einfache Weise zu ermöglichen.
Neben der durch Mehrfachbelegung eines Attributes gekennzeichneten Inkonsistenz erkennt die Komponente auch eine "einstufige" Oszillation. Da ein ständiger Wechsel eines Attributwertes auch Absicht des Entwicklers sein kann, werden Oszillationen zwar erkannt und die Quellobjekte angezeigt, es muß aber keine Bearbeitung erfolgen. Für diesen Fall steht dem Benutzer die Schaltfläche "Objekte sollen so bleiben" zur Verfügung.

Interne Abläufe

Hauptpunkt des funktionalen Tests ist die schrittweise Verfolgung der Inferenz. Ein Schritt ist immer dann abgeschlossen, wenn alle Regeln gefeuert haben, deren Prämisse vor dem Schritt erfüllt war. Damit für alle Regeln der gleiche Ausgangszustand gilt, werden Attributänderungen nicht bei den "Original-Objekten" vorgenommen, sondern bei gesondert erzeugten "Ausgangs-Objekten".

Frames (Objekte)

Zu Beginn des funktionalen Tests muß für jedes Attribut ein "Ausgangs-Attribut" erzeugt werden, das alle notwendigen Informationen zur Erkennung und Bearbeitung einer Inkonsistenz aufnehmen kann. Die Datenstrukturen dieser "Ausgangs-Attribute" können nach unterschiedlichen Kriterien angelegt werden. Es kann ein exaktes Abbild der Klassendefinitionen und Objekte erzeugt und z.B. mit einem "*" gekennzeichnet werden. Statt der einfachen Attributtypen werden neue "Typ-Klassen" verwendet, die alle Anforderungen für die Auswertung erfüllen.

Vorteil dieser Methode ist die starke Ähnlichkeit der zu verwendenden Syntax, um Werte zu setzen.

Aus die *Eigenschaft* von **Objekt** wird damit die *Eigenschaft* von ***Objekt**

Da ein Benutzer der Konsistenzkomponente keinen Zugriff auf die internen Strukturen und Abläufe hat, kann aber auf dieses gute "Aussehen" zugunsten besserer Handhabbarkeit verzichtet werden.
Jeder einfache Datentyp erfordert ein "Ausgangs-Attribut", das mehrere Werte dieses Typs aufnehmen kann. In Abbildung 3.17 sind die Tabellen der Eigenschaften von zwei erstellten "Ausgangs-Attributen" dargestellt.

_ZEILE-1

_ZEILE-1, a _zeile-symbol	
Notes	OK
Names	_ZEILE-1
Objektname	auto-1
Objektklasse	auto
Attribname	farbe
Wert	a symbol-list
Eintragendes objekt	a symbol-list

_ZEILE-25

_ZEILE-25, a _zeile-integer	
Notes	OK
Names	_ZEILE-25
Objektname	fahrrad-1
Objektklasse	fahrrad
Attribname	gänge
Wert	an integer-list
Eintragendes objekt	a symbol-list

Abbildung 3.17: Eigenschaften spezieller "Ausgangs-Attribute"

Diese Form ist nicht sehr anschaulich, läßt sich jedoch auf übersichtliche Weise innerhalb einer Prozedur erzeugen. Grundlage hierfür sind die durch die Editoren erzeugten Listen "Klassennamen", "Attributnamen" und "Datentyp". In Kapitel 3.2 wurde bei der Erläuterung der internen Darstellung bereits auf die Struktur der Listen hingewiesen und zur Veranschaulichung eine Tabelle (Seite 76) gebildet. Eine weitere Liste enthält alle vom Benutzer erzeugten Objekte. Sie wird Objekt für Objekt abgearbeitet, um für jedes einzelne Attribut ein eigenes Ausgangs-Attribut zu erzeugen.
Mit dem Namen der Klasse, zu der ein Objekt gehört, werden alle Einträge der Liste "Klassennamen" verglichen. Bei Übereinstimmung wird der Index des Eintrags benutzt,

um direkt auf die Elemente der anderen Listen zuzugreifen, die für den Aufbau der Ausgangs-Attribute (Oberklasse "__Zeile") dieses Objekts benötigt werden. Die Liste der Datentypen wird verwendet, um die Klasse der __Zeilen mit dem passenden Typ (__Zeile-Integer, __Zeile-Float, usw.) zu bestimmen.
Hat der Benutzer beispielsweise zwei Autos (Auto-1, Auto-2) erzeugt, sind die in Abbildung 3.18 dargestellten Ausgangs-Attribute erforderlich.

Zeilen-name	__Zeile-klasse	Objekt-name	Objekt-klasse	Attribut-name	Wert	eintragendes Objekt
__Zeile-1	__Zeile-value	Auto-1	Auto	Zubehör	**Value-Liste**	Symbol-Liste
__Zeile-2	__Zeile-float	Auto-1	Auto	Gewicht	**Float-Liste**	Symbol-Liste
__Zeile-3	__Zeile-symbol	Auto-1	Auto	Farbe	**Symbol-Liste**	Symbol-Liste
__Zeile-4	__Zeile-integer	Auto-1	Auto	Leistung	**Integer-Liste**	Symbol-Liste
__Zeile-5	__Zeile-value	Auto-2	Auto	Zubehör	**Value-Liste**	Symbol-Liste
__Zeile-6	__Zeile-float	Auto-2	Auto	Gewicht	**Float-Liste**	Symbol-Liste
__Zeile-7	__Zeile-symbol	Auto-2	Auto	Farbe	**Symbol-Liste**	Symbol-Liste
__Zeile-8	__Zeile-integer	Auto-2	Auto	Leistung	**Integer-Liste**	Symbol-Liste

Abbildung 3.18: Tabelle der Ausgangs-Attribute für Auto-1 und Auto-2

Alle Wertzuweisungen an Attribute, die im Aktionsteil von Regeln oder innerhalb von Prozeduren auftreten, werden überarbeitet, damit die Ausgangs-Attribute an Stelle der Original-Attribute verwendet werden.
Formulierungen der Form der **Eigenschaft** von **Objekt** wird **der Wert X** zugewiesen
müssen durch solche der Form

der Liste **Wert** von **__Zeile-<Nr>** wird **der Wert X** angehängt **UND**

der Liste **Eintragendes Objekt** von **__Zeile-<Nr>** wird **der Name dieser Regel** angehängt

ersetzt werden (Erfüllung der Forderung *F-funk. 3*, S. 56).

Regeln

Die mit dem Regeleditor erzeugten Regel**texte** müssen im Aktionsteil nach obigem Beispiel umgewandelt werden, während die Regel**grafiken** unverändert bleiben. Zur besseren Trennbarkeit von syntaktischer und funktionaler Komponente wird jedoch nicht der Text der Originalregel, sondern eine Kopie verändert. Die Inferenzmaschine muß dann daran gehindert werden, die Originalregel zu verwenden. Zu diesem Zweck wird jedes "originale" Regelobjekt mit einer Systemanweisung als *zu ignorieren* (verborgene, unbenannte Eigenschaft des Objektes; *disabled*) gekennzeichnet, was den gewünschten Effekt erzeugt.

Regeln mit generischen Aktionsteilen erfordern ein spezielles Vorgehen bei der Überarbeitung. In einer Iteration über alle Objekte einer Klasse wird für das jeweils aktuelle Objekt ein Attributwert gesetzt. In den neuen Anweisungen wird auf eine _Zeile zugegriffen, die sowohl von dem verwendeten Attribut, als auch von dem aktuellen Objekt abhängt. Da die Nummerierung der _Zeilen nur von der Reihenfolge bei ihrer Erzeugung anhängt, muß für jedes Objekt bei der Iteration die zugehörige _Zeile erst bestimmt werden.

Eine Regel mit generischem Aktionsteil ist eine "Kurzschreibweise" für den Benutzer, die von der Inferenzmaschine bei Bedarf durch eine Reihe von Regeln mit gleicher Prämisse ersetzt wird, in deren Aktionsteil jeweils nur auf ein spezielles Objekt der angegebenen Klasse zugegriffen wird. Der funktionale Test erfordert diese Konkretisierung in spezifische Regeln bereits vor dem Start des Inferenzprozesses für die entwickelte Applikation.

Prozeduren

Ebenso wie Regeln können Prozeduren Wertzuweisungen in einfacher Form oder in Iterationen über alle Objekte einer Klasse enthalten. Auch hier ist eine Änderung der Formulierung und eine Ersetzung der Iteration durch konkrete Zuweisungen erforderlich. Die dafür notwendige Analyse des Prozedurtextes ist sehr aufwendig, wenn nicht einige Randbedingungen bezüglich der Syntax eingehalten werden.

Die Syntax für Prozeduren in G2 enthält als Kennzeichen eines Kommentares die einfache geschweifte Klammer. Leider wird diese Klammer auch als Kennzeichen für Mengen bzw. Reihen verwendet. Um die beiden Bedeutungen genau zu unterscheiden, muß der Kontext analysiert werden. Einfacher ist es, dem Benutzer als Kommentarkennzeichen eine {** Text **} -Struktur vorzuschreiben.

Ein ähnliches Problem ergibt sich bei Iterationen in Form von *For*-Schleifen. Eine Schleife wird immer mit der Anweisung **end** abgeschlossen. Wird nach diesem Text gesucht, um das Ende des Schleifenblocks zu identifizieren, kann man unterschiedlich vorgehen.

- Ein einfacher Zähler für alle "öffnenden" Strukturen (*begin, if-then-else, for-do*) kann bei sequentieller Abarbeitung des Prozedurtextes durch jede "Öffnung" inkrementiert und durch jedes "*end*" dekrementiert werden. Auf diese Weise ist eine eindeutige Zuordnung der betrachteten öffnenden Struktur zu "ihrem *end*" möglich.

- Eine andere, auch für den Benutzer hilfreiche Möglichkeit besteht in der Kennzeichnung der *end*-Anweisung durch einen Kommentar.
  Eine Anweisung der Form

  for < lokale Variable> = **each** <Klassenname> **do** wird beispielsweise mit

  end; {## of for each < lokale Variable> ##} abgeschlossen.

Jede *For*-Schleife muß wie ein generischer Aktionsteil einer Regel in eine Folge von Anweisungsblöcken für spezielle Objekte umgewandelt werden. Um auch hier, wie bei den Regeln, den Originalzustand zu erhalten, wird eine neue Prozedur erzeugt, die die gleiche Funktion erfüllt wie die Originalprozedur. Zur Unterscheidung wird dem "alten" Namen das Kennzeichen "Kons_" vorangestellt und durch die Anpassung aller Aufrufe an den "neuen" Namen ist die Originalprozedur quasi deaktiviert.

Schaltflächen (G2 spezifisch)

Wie bereits zu Beginn dieses Kapitels gesagt wurde, wird häufig die Betätigung einer Schaltfläche als Startsignal für die Inferenzkette der Applikation verwendet. Dabei werden Aktionen ausgelöst wie das Setzen eines oder mehrerer Attributwerte oder ein Prozeduraufruf. Auch diese Aktionen werden bei der Vorbereitung zur schrittweisen Inferenz geprüft und bei Bedarf umgewandelt.

Schaltflächen stellen für die Umwandlung eine besondere Herausforderung dar, da G2 hier eine Reihe von Gestaltungsmöglichkeiten bietet [8] . Ein praktikabler Weg ist die Nachbildung der Aktionen durch Regeln, die nach den besprochenen Methoden aufgebaut werden. Diese "Knopf-Regeln" existieren nur in der G2-internen Darstellung und werden bei Betätigen der Schaltfläche über den Mechanismus der Regelkategorie aufgerufen.
Regeln können durch die Angabe einer Kategorie in eine Gruppe zusammengefaßt werden, wobei die Abarbeitung aller Regeln der Gruppe durch die Inferenzmaschine mit dem Befehl

invoke <Kategorie-Name> **rules**

ausgelöst wird. Auf diese Weise werden die von den Regeln ausgelösten Aktionen an die Betätigung der Schaltfläche gebunden, aus deren Aktion die Regeln generiert wurden. Der Originaltext der Schaltflächenaktion wird als Textkopie abgelegt, um nach dem funktionalen Test den Urzustand wieder herzustellen.

Wurde der funktionale Test komplett durchlaufen, dabei Inkonsistenzen festgestellt und deren Quellobjekte editiert, muß die gesamte Schlußkette noch einmal von vorne gestartet

werden. Aus diesem Grund werden alle Attributwerte und die Farben der Sinnbilder sofort bei Beginn der Vorbereitungen zum funktionalen Test gespeichert. Ein weiterer Satz dieser Informationen wird nach einem Inferenzschritt gespeichert, um nach einer eventuell notwendigen Änderung von Quellobjekten den gerade vollzogenen Schritt erneut mit den gleichen Startwerten auszuführen.

Nach erfolgreichem Durchlauf (es wurden keine Probleme detektiert) müssen theoretisch alle Änderungen an den Regeln, Prozeduren und Schaltflächen (im folgenden "Inferenz-Objekte" genannt) wieder rückgängig gemacht werden.
Es ist naheliegend, daß der Benutzer den funktionalen Test nicht nur einmal, am Ende der Gesamtentwicklung aufrufen wird. Vor einem erneuten Durchlauf müssen alle Änderungen, wie beschrieben, erneut vorgenommen werden, wenn nicht die bereits umgewandelten Strukturen deaktiviert und gespeichert wurden. Nach der Veränderung eines Inferenz-Objekts wegen einer Inkonsistenz muß es ebenfalls neu umgewandelt werden. Zur Verkürzung der Umwandlungszeit werden alle Inferenz-Objekte, die noch zu wandeln sind, in eine Liste eingetragen. Vor dem ersten Aufruf des funktionalen Tests enthält die Liste alle vom Benutzer erzeugten Inferenz-Objekte. Im Zuge der Umwandlung werden die bearbeiteten Inferenz-Objekte aus der Liste entfernt. Treten beim Test Inkonsistenzen oder Oszillationen auf und werden deshalb Inferenz-Objekte editiert oder erzeugt der Benutzer nach einem funktionalen Test neue Inferenz-Objekte, werden diese in die Liste aufgenommen.

Ist die Entwicklung der wissensbasierten Applikation abgeschlossen, kann zur Verringerung des benötigten Speicherumfanges der funktionale Test mit allen für seine Durchführung erzeugten Objekten aus der Wissensbasis entfernt werden. Die Editoren können ebenfalls entfernt oder auch im System belassen werden. Soll zu einem späteren Zeitpunkt der funktionale Test doch wieder verwendet werden, kann die Teilkomponente wieder in die Modulhierarchie aufgenommen werden.

Kapitel 4

Schlußbemerkungen

4.1 Zusammenfassung

Im Rahmen dieser Arbeit ist ein Test auf Widerspruchsfreiheit von hybriden Wissensbasen entwickelt und erprobt worden der es ermöglicht, eine hybride Wissensbasis zu ergänzen oder zu ändern und dabei auftretende Widersprüche innerhalb der Wissensbasis zu entdecken und zu beheben.
Bisher ist es dem Entwickler eines wissensbasierten Systems selbst überlassen, wie er die Korrektheit und Funktionssicherheit der Applikation gewährleistet. Bereits bei der Erstellung, insbesondere aber bei Wartungsarbeiten nach längerer "Arbeitszeit" des Systems kann unbemerkt widersprüchliches Wissen implementiert werden.

Ausgehend von bestehenden Definitionen der Konsistenz in Datenbanken, klassischen Deduktionssystemen und regelbasierten Systemen wird untersucht, in wie weit Testverfahren aus diesen Bereichen für hybride wissensbasierte Systeme anwendbar sind. Es wird gezeigt, daß die bei der Prädikatenlogik vorhandene Möglichkeit, eine Menge von prädikatenlogischen Sätzen mit Hilfe der Connection Graph Proof Procedure auf Konsistenz zu prüfen, unter bestimmten Bedingungen auch für regelbasierte Systeme anwendbar ist. Die Frage, ob durch Erweiterungen oder zusätzliche Bedingungen die CGPP auch bei hybriden Systemen einsetzbar ist, führt zur Untersuchung der einzelnen Wissensrepräsentationsformen.
Für die Bereiche Regelkonsistenz, Frame- oder Objektkonsistenz und Prozedurkonsistenz werden jeweils spezifische Konsistenzbedingungen erarbeitet und daraus Anforderungen an die Struktur der Frames sowie die Syntax der Regeln und Prozeduren abgeleitet.

Durch die komplexe Verzahnung der Faktenrepräsentation durch Objekte und ihrer Verarbeitung durch Regeln, insbesondere aber durch den Einsatz von Prozeduren ist eine getrennte Betrachtung der Konsistenzbereiche nicht ausreichend, um auf die Konsistenz des Gesamtsystems während der Nutzung der wissensbasierten Applikation zu schließen.

Parallel zur daher notwendigen Definition einer "Gesamtkonsistenz" erfolgt die Entwicklung eines geeigneten Testverfahrens. Dabei ergeben sich weitere Forderungen an die Struktur der Wissenselemente sowie die Forderung nach einem "schrittweisen" Ablauf der Inferenz. Dies ist erforderlich, um durch das Testverfahren nicht nur eine Inkonsistenz zu erkennen, sondern dem Entwickler auch die notwendigen Informationen zu den Quellen der Inkonsistenz zur Verfügung zu stellen. Durch das hier entwickelte Verfahren der schrittweisen Inferenz ist es einem Entwickler möglich, während der Konsistenzprüfung die Quellen von erkannten Inkonsistenzen zu bearbeiten und die Auswirkungen auf die Gesamtkonsistenz sofort zu testen.

Die Realisierung eines Zusatzmoduls zu einer Entwicklungsumgebung für hybride wissensbasierte Systeme zeigt beispielhaft die Anwendung des entwickelten Konsistenzprüfungsverfahrens.
Die für dieses Modul am Lehrstuhl fürAutomatisierungstechnik der Bergischen Universität – Gesamthochschule Wuppertal entwickelten Dialoge befähigen auch einen ungeübten Benutzer, schnell und fehlerfrei Objekte, Regeln und Prozeduren zu erzeugen. Informationen für einen strukturierten Aufbau der Wissensbasis werden gesammelt und in einer dem jeweiligen Kontext angepaßten Form dargestellt. Zusätzlich wurde die Möglichkeit geschaffen, in jedem Stadium der Entwicklung oder Wartung einen funktionalen Konsistenztest der gesamten Wissensbasis durchzuführen. Der Test wurde so gestaltet, daß der Benutzer beim Auftreten von Inkonsistenzen optimal bei deren Behebung unterstützt wird, auch ohne mit der Wissensbasis vertraut zu sein.

4.2 Übertragbarkeit der Ergebnisse

Die Betrachtungen zur Konsistenz in hybriden wissensbasierten Entwicklungsumgebungen wurden in Kapitel 2.2 so allgemein wie möglich gehalten. Jedes kommerzielle Werkzeug hat Eigenheiten sowohl in der Bedienbarkeit der Oberfläche als auch in der Art der Wissensrepräsentation. Einige Forderungen, die in Kapitel 2.3 formuliert wurden, können in verschiedenen Werkzeugen bereits erfüllt sein, ein langzeitig (über Jahre) anwendbarer Test auf Widerspruchsfreiheit ist bisher aber in keinem System realisiert.

Die beispielhafte Realisierung einer Konsistenztest-Komponente wurde unter anderem deshalb mit dem Werkzeug G2 vorgenommen, weil das Grundkonzept dieser Entwicklungsumgebung auf Erweiterungen ausgerichtet ist. Es existieren einzelne Werkzeug-Module, die hierarchisch in eine Wissensbasis integriert werden können, und es besteht die Möglichkeit, eigene Module zu entwickeln. Des weiteren ermöglicht die grafische Oberfläche mit den dafür zur Verfügung stehenden Werkzeug-Modulen den Aufbau von benutzerfreundlichen Dialogen.
Abhängig davon, wie offen eine Entwicklungsumgebung für die Implementierung einer neuen oder veränderten Benutzerschnittstelle ist, können Ansätze der vorgestellten Realisierung ebenfalls übernommen werden. Grundsätzlich sind aber nur die erarbeiteten Forderungen entsprechend den Möglichkeiten übertragbar. In vielen Fällen wird das ein Arbeitsbereich des Softwareherstellers bleiben.

4.3 Ausblick

In den Anfängen der "Expertensystem-Entwicklung" erforderten sehr komplexe und spartanische Werkzeuge einen Team von Spezialisten für die Analyse und Aufarbeitung der zu lösenden Aufgabe sowie der Vorbereitung und Realisation der Implementierung. Inzwischen werden die Werkzeuge auch für den Entwickler komfortabler und so gut bedienbar, daß keine Spezialisten, sondern angelernte Fachkräfte aus dem Arbeitsgebiet, aus dem die Aufgabe stammt, die Implementierung des Wissens selbst vornehmen können.

Während der Untersuchungen und der Entwicklung der Teilkomponenten der Konsistenzprüfung wurde das verwendete Entwicklungssystem nicht zuletzt auch wegen der diesbezüglichen Fragen und Probleme in Richtung von Möglichkeiten der Inferenzverfolgung verbessert. Dem Entwickler stehen immer bessere Eingabemöglichkeiten zur Verfügung, die beispielsweise eine Typprüfung für Attribute beinhalten.
Einen nicht zu vernachlässigenden Einfluß haben auch die Bestrebungen zur Software-Zertifizierung. Noch vor einigen Jahren war nur wichtig, daß die Applikation überhaupt die erwartete Funktionalität zeigte. In automatisierungstechnischen Aufgabenbereichen wurden wissensbasierte Systeme parallel zu herkömmlichen Prozeßführungssystemen zur Visualisierung oder Dokumentation der Prozesse eingesetzt. Doch so, wie auch das Vertrauen in Computer und herkömmliche Software erst nach und nach gewachsen ist, mußte mit verschiedenen Applikationen erst die Brauchbarkeit und Zuverlässigkeit von wissensbasierten Systemen gezeigt werden. Schließlich führen die Bestrebungen im Bereich des Qualitätsmanagements und der Zertifizierung dazu, daß sich auch Endkunden immer mehr Gedanken zu wissensbasierten Systemen machen müssen.

Literaturverzeichnis

[1] Angele, Jürgen; Studer, Rudi:
Konsistenzprüfung in wissensbasierten Systemen.
Institut für angewandte Informatik und formale Beschreibungsverfahren
Forschungsbericht Nr. 239
Universität Karlsruhe (TH), Januar 1992

[2] Bibel, Wolfgang; Hölldobler, St.; Schaub, T.:
Wissensrepräsentation und Inferenz: eine grundlegende Einführung.
Vieweg, Braunschweig 1993

[3] Curth, Michael; Bölscher, Andreas; Raschke, Bernhard:
Entwicklung von Expertensystemen.
Carl Hanser Verlag, München 1991

[4] Ester, Martin:
Konsistenzwerkzeuge für PROLOG-Wissensbasen.
ETH Zürich, Dissertation Nr.8776. 1989
auch: Informatik-Dissertationen ETH Zürich Nr.14
Verlag der Fachvereine, Zürich 1989

[5] Gähler, Felix:
Überwachung von Konsistenzbedingungen.
ETH Zürich, Dissertation Nr.9325. 1990

[6] Harmon, Paul; King, David:
Expertensysteme in der Praxis; Perspektiven, Werkzeuge, Erfahrungen.
Oldenbourg Verlag, München 1989

[7] Heidepriem, Jürgen:
Arbeitblätter zur Vorlesung "Computational Intelligence".
Lehstuhl für Automatisierungstechnik, Universität Wuppertal

[8] Hinz, Michael:
Entwicklung von Datenstrukturen und Methoden einer Akquisitionskomponente zur dynamischen Überprüfung der Widerspruchsfreiheit eines wissensbasierten Systems.
Diplomarbeit, Universität Wuppertal 1998

[9] Hohlfeld, Bernhard; Struckmann, Werner:
Einführung in die Programmverifikation: Theorie und Anwendung der Hoareschen Methode am Beispiel der Programmiersprache Pascal.
B.I. Wissenschaftsverlag, Mannheim u.a. 1992

[10] Hohlfeld, Bernhard:
Zur Verifikation von modular zerlegten Programmen.
Universität Kaiserslautern, Dissertation 1988

[11] IT-Sicherheitskriterien:
Kriterien für die Sicherheit von Systemen der Informationstechnik.
Bundesanzeiger Köln 1989 (41, 99a)

[12] IT-Evaluationshandbuch:
Handbuch für die Prüfung der Sicherheit von Systemen der Informationstechnik (IT).
Bundesanzeiger Köln 1990 (42, 151a)

[13] Jakobi, Anja; Friedrich, Jürgen:
Expertensysteme: Anwendungen, Auswirkungen und Gestaltung.
Verlagsanstalt Courier GmbH, Stuttgart 1993

[14] Karbach, W.; Linster, M.:
Wissensakquisition für Expertensysteme: Techniken, Modelle und Softwarewerkzeuge.
Carl Hanser Verlag, München, Wien 1990

[15] Kleberhoff, Ralf:
Einsatz des Varianten-Connection-Graphs für realzeitfähige Anlagenüberwachungssysteme.
BUGH Wuppertal, Dissertation 1991

[16] Koussev, Traytcho:
Entwicklung einer Erklärungskomponente für ein Expertensystem.
Fortschritt-Berichte VDI Reihe 9 Nr. 101
VDI-Verlag, Düsseldorf 1990

[17] Kowalski, Robert:
Logic for Problem Solving.
North Holland, New York 1979

[18] Leikauf, Peter:
Konsistenzsicherung durch Verwaltung von Inkonsistenzen.
ETH Zürich, Dissertation Nr.9208. 1990

[19] Lunze, Jan:
Künstliche Intelligenz für Ingenieure.
Band 1: Methodische Grundlagen und Softwaretechnologie.
R. Oldenbourg Verlag, München 1994

[20] Moerkotte, Guido:
Konsistenzprüfung in deduktiven Datenbanken: Diagnose und Reparatur.
Informatik-Fachberichte Nr. 248
Springer-Verlag, Berlin 1990

[21] Moritz, Werner:
Entwicklung von Modulen einer Akquisitionskomponente zur benutzerfreundlichen Eingabe von Klassen, Objekten und Regeln in ein wissensbasiertes System.
Diplomarbeit, Universität Wuppertal 1998

[22] Myers, Glenford J.:
Methodisches Testen von Programmen.
R. Oldenbourg Verlag, München 1987

[23] Partridge, Derek:
Künstliche Intelligenz: KI und das Software Engineering der Zukunft.
McGraw-Hill, Hamburg, New York 1988

[24] Platz, Heiko:
Entwurf und Realisierung einer Erklärungskomponente für die Expertensystem-Shell G2.
Unveröffentlichte Studienarbeit
Lehrstuhl für Automatisierungstechnik, BUGH Wuppertal 1994

[25] Pritschow, G.; Spur, G.; Weck, M. (Hrsg.):
Künstliche Intelligenz in der Fertigungstechnik.
Carl Hanser Verlag, München, Wien 1989

[26] Quiel, Gerd:
Datenbanksysteme: Grundlagen von Informationssystemen.
Verlagsgesellschaft Rudolf Müller, Köln-Braunsfeld 1981

[27] Richter, M.; Maurer, F. (Hrsg.):
Expertensysteme 95, Beiträge zur 3. Deutschen Expertensystemtagung (XPS-95).
Proceedings in Artificial Intelligence Bd.2
Infix, Sankt Augustin 1995

[28] Richter, R.; Sander, P.; Stucky, W.:
Problem – Algorithmus – Programm.
Grundkurs angewandte Informatik II
B.G. Teubner Verlag, Stuttgart 1993

[29] Rutz, Peter:
Zweiwertige und mehrwertige Logik.
Franz Ehrenwirth Verlag, München 1973

[30] Sinowjew, Alexander A.:
Über mehrwertige Logik; Ein Abriß.
Deutscher Verlag der Wissenschaften, Berlin 1968

[31] Sinowjew, Alexander A.:
Komplexe Logik; Grundlagen einer logischen Theorie des Wissens.
Deutscher Verlag der Wissenschaften, Berlin 1970

[32] VDI/VDE-Gesellschaft Meß- und Automatisierungstechnik:
Wissensverarbeitung in der Automatisierungstechnik.
GMA-Aussprachetag; Tagung Langen
VDI-Berichte 897
VDI-Verlag GmbH, Düsseldorf 1991

[33] Zehnder, Carl A.:
Informationssysteme und Datenbanken.
Verlag der Fachvereine, Zürich, und Teubner-Verlag, Stuttgart, 1989

Artikel, Aufsätze, Konferenzbeiträge

[34] Ahrens, Wolfgang:
Expertensysteme für die Prozeßführung; Erfahrungen aus dem Verbundprojekt TEX-I.
In: Chemie Ingenieur Technik Bd. 62 (1990)
H. 8, S.635-644

[35] Beauvieux, Alain:
A Method to Check Knowledge Base Consistency.
In: Nori, K.V.; Kumar, S. (Hrsg.)
Foundations of Software Technology and Theoretical Computer Science.
Lecture Notes in Computer Science 338
S.455-468
Springer Verlag, Berlin 1988

[36] Beuschel, J.; Böhme, B.:
Ein Zuverlässigkeitsmodell für Expertensysteme in der Produktionsautomatisierung.
In: msr; Messen, Steuern, Regeln
Wissenschaftlich-technische Zeitschrift für die Automatisierungstechnik 33 (1990)
H. 9, S.438–442

[37] Brewka, Gerhard:
Nichtmonotone Logiken – Ein kurzer Überblick.
In: KI (1989); H. 2, S.5–12

[38] Ginsberg, A.:
Knowledge-base Reduction: a new approach to checking knowledge bases for inconsistency and redundancy.
In: Gupta, Uma G.:
Validating and Verifying Knowledge-Based Systems.
S.585-589
IEEE Computer Society Press, Los Alamitos u.a. 1991

[39] Heinzel, Werner; Schrobar, Michael:
Realisierung eines Expertensystems zur Fehlerdiagnose in LAN.
In: ntz Bd. 47 (1994)
H. 1, S.16–21

[40] Klapproth, Uwe; Perner, Petra; Böhme, Berndt:
Einsatz eines Expertensystems zur Diagnose von Druckfehlern im Offsetdruck.
In: msr; Messen, Steuern, Regeln 34 (1991)
H. 3, S.116–120

[41] Kowalski, Robert:
A Proof Procedure Using Connection Graphs.
In: Journal of the Association for Computing Machinery 22 (1975)
S.572–595

[42] Krebs, Volker; Respondek, Thomas:
Automatisierung komplexer chemischer Prozesse durch Echtzeit-Expertensysteme.
In: atp - Automatisierungstechnische Praxis 33 (1991)
H. 2, S.82–86

[43] Omar, Rosli:
Artificial Intelligence through Logic?
In: AiCommunications AICOM 7 (1994)
H. , S.161–174

[44] Owsnicki-Klewe, B.:
Ein integriertes System zur Repräsentation von Wissen.
In: Philips; Unsere Forschung in Deutschland (1988)
Bd. 4, S.100–104

[45] Reinfrank, Michael:
Formeln und Modelle: Wissensrepräsentation mit Logik.
In: Informationstechnik it 31 (1989)
H. 2, S.102–112

[46] Rosenbaum, Oliver:
Expertensystem contra Datenbank.
In: miniMicro magazin (1991)
H. 12, S.58–59

[47] Struß, Peter:
Wissensrepräsentation – Abschied vom Programmieren?
In: Informationstechnik it 31 (1989)
H. 2, S.91–94

Anhang A

A.1 Struktur der realisierten Module

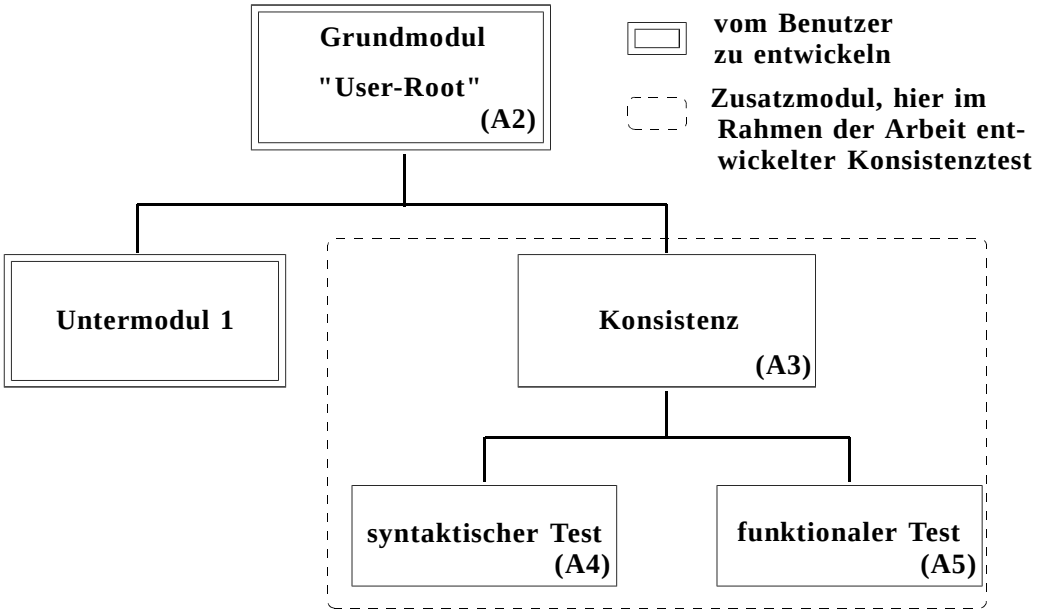


Abbildung A.1: Modulhierarchie einer leeren Wissensbasis mit Konsistenztest

Die eingerahmten Module "Konsistenz", "syntaktischer Test" und "funktionaler Test" werden im folgenden in ihrer für den Benutzer sichtbaren Struktur beschrieben. Die Angaben (A2) bis (A5) in Abbildung A.1 verweisen auf die Kapitelnummern der Unterabschnitte von Anhang A.

A.2 Das Modul User-Root

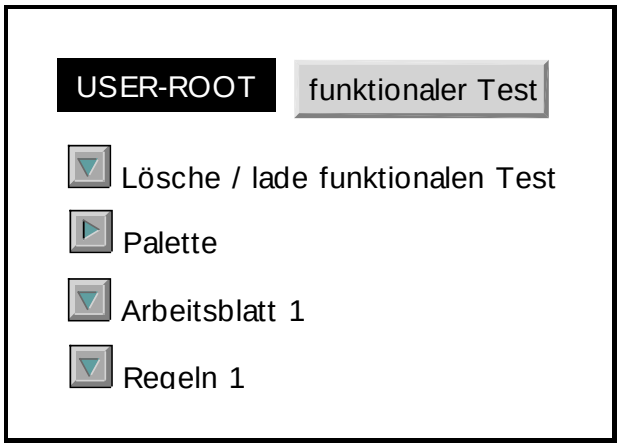


Abbildung A.2: Arbeitsblatt beim Start der "leeren" Wissensbasis

Als Basis für eine konsistente Applikationsentwicklung wird dem Anwender in einer neuen Wissensbasis bereits ein Arbeitsblatt ("User-Root") mit einer Schaltfläche ("funktionaler Test") und vier Navigationsknöpfen zur Verfügung gestellt.
Die Navigationsschaltflächen **Lösche / lade funktionalen Test** und **Palette** führen auf Arbeitsblätter, die nicht zum Grundmodul User-Root gehören, für den Anwender aber direkt zugänglich sein sollen. Die über **Arbeitsblatt 1** und **Regeln 1** erreichbaren Arbeitsblätter hingegen sind wie das Arbeitsblatt User-Root selbst dem Grundmodul zugeordnet.

Alle vom Anwender während der Entwicklung der Applikation erzeugten Objekte (incl. neuer Arbeitsblätter) gehören automatisch zum Grundmodul.

A.3 Das Modul Konsistenz

Das Modul Konsistenz stellt die Verbindung zwischen den Untermodulen syntaktischer Test und funktionaler Test her. Das Modul enthält die Arbeitsblätter mit den sowohl für die Eingabe, als auch für den funktionalen Test benötigten Wissenselementen und Verwaltungsroutinen.

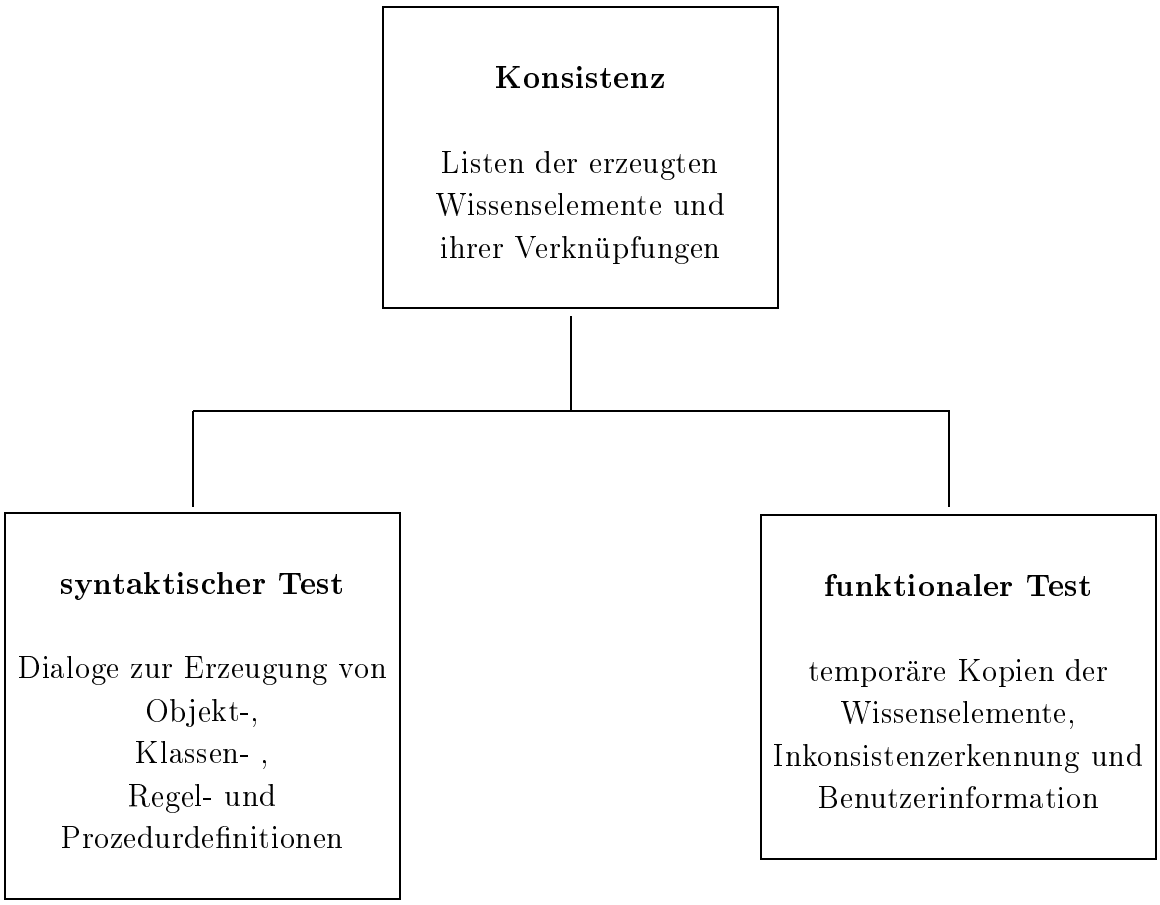


Abbildung A.3: Module für den Konsistenztest mit ihren Inhalten

A.4 Das Modul "syntaktischer Test"

In diesem Modul ist alles zusammengefaßt, was der Benutzer an zusätzlicher Funktionalität zu den G2-Grundfunktion benötigt, um eine konsistente Wissensbasis erzeugen zu können. Alle zur Erzeugung und Bearbeitung von Wissenselementen notwendigen Aktionen werden durch Sinnbilder, Auswahlmenüs und Dialoge für den Benutzer übersichtlich zusammengestellt.

Das wichtigste Hilfsmittel ist dabei das Arbeitsblatt "Palette", das sofort nach dem Start der Entwicklungsumgebung sichtbar ist (siehe Abbildung 3.6 auf Seite 71) oder durch einen Navigationsknopf auf dem ersten Arbeitsblatt des Grundmoduls aufgerufen werden kann (siehe Abb. A.2).

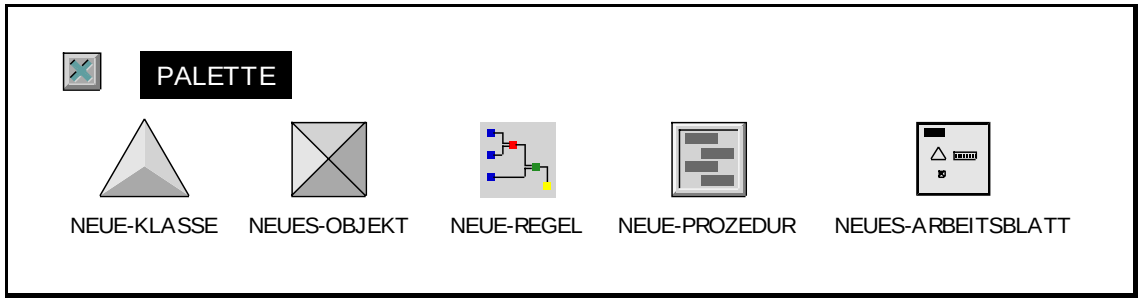


Abbildung A.4: Palette der möglichen Wissenselemente

Von dieser Palette kann der Benutzer durch Anklicken der Sinnbilder ein neues Wissenselement des dargestellten Typs erzeugen und auf einem Arbeitsblatt ablegen. Je nach Wissenselement sind verschiedene Benutzereingaben erforderlich, die durch Dialoge abgefragt werden.

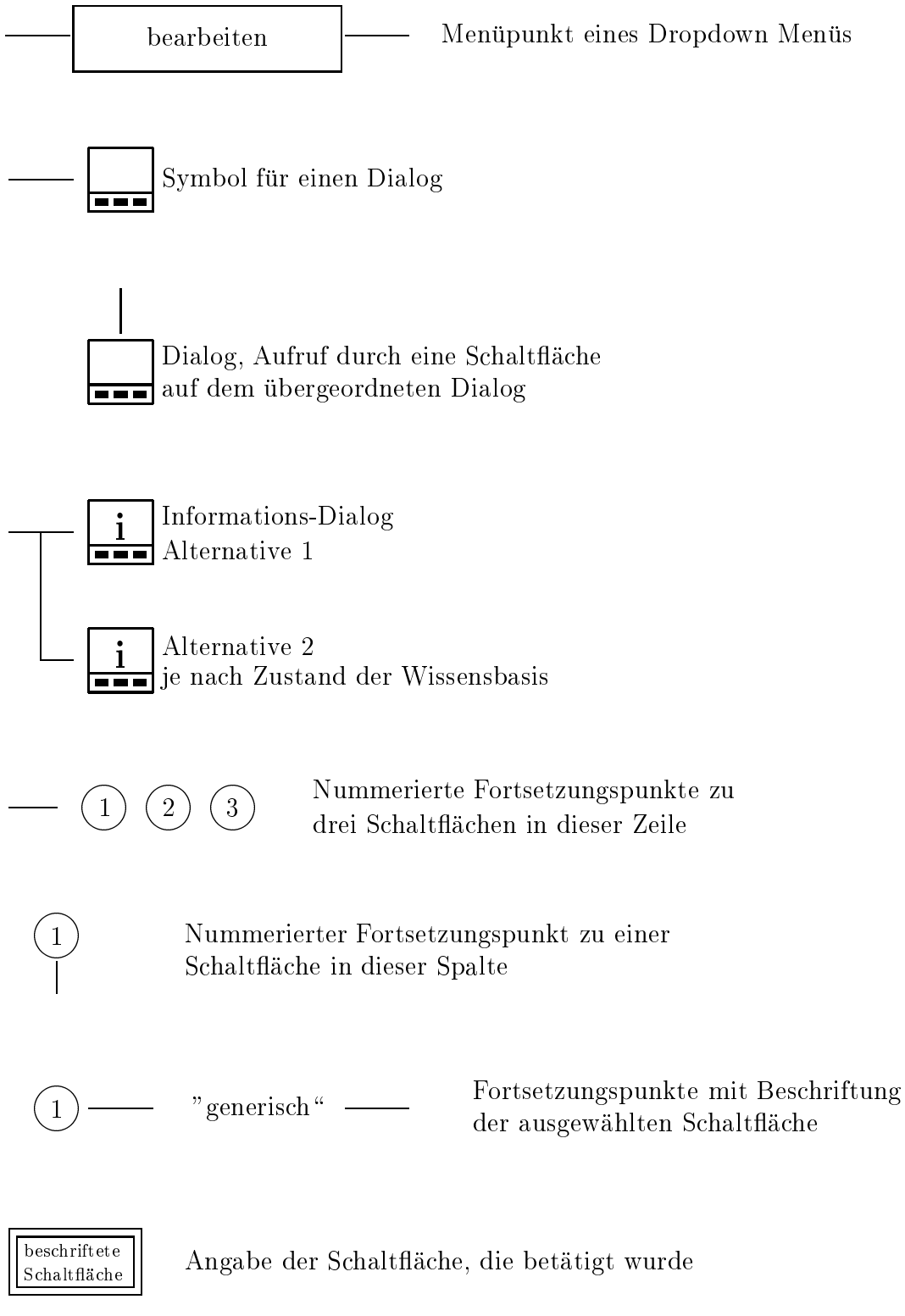


Abbildung A.5: Legende der Symbole in den folgenden Strukturen

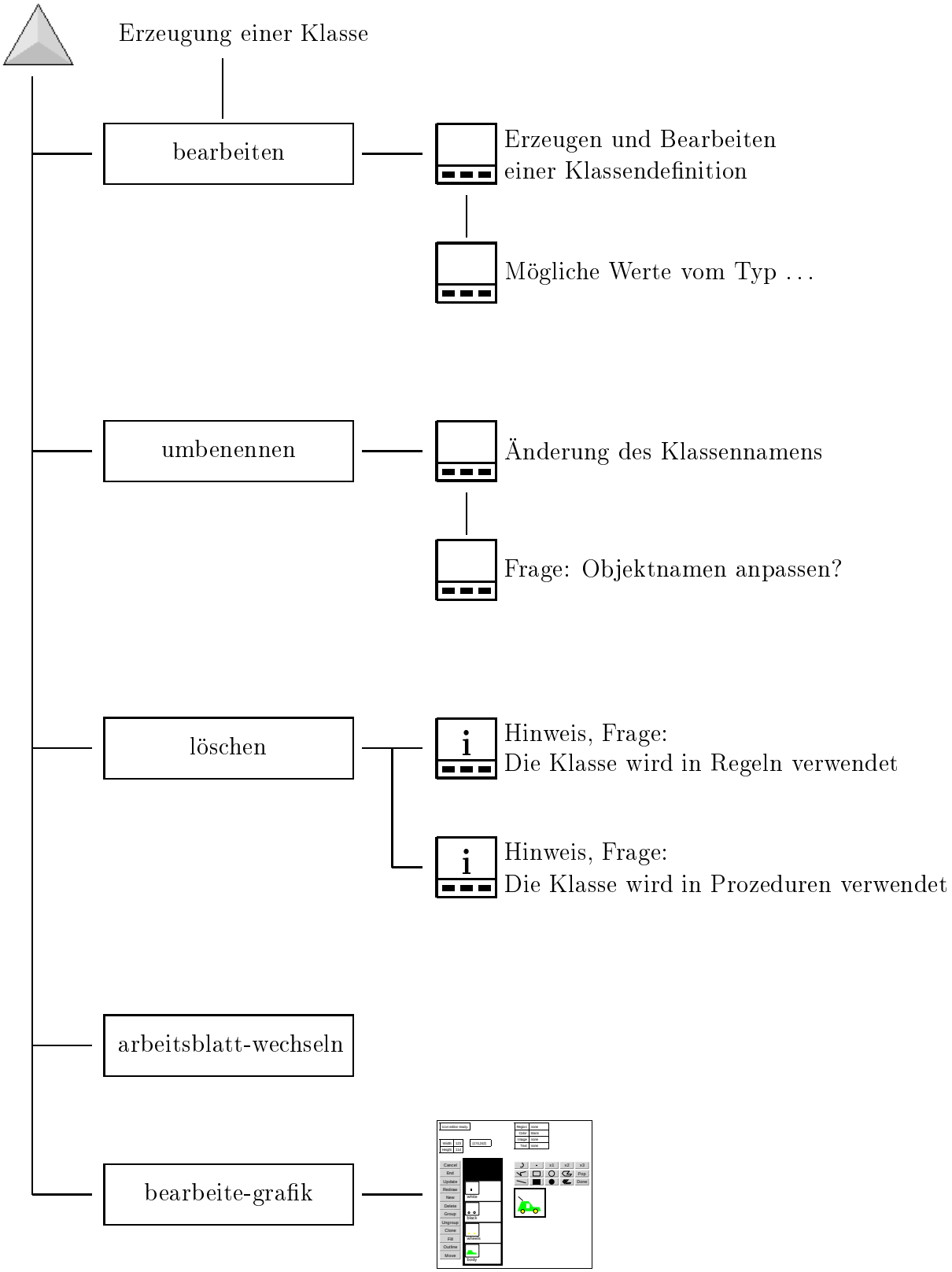


Abbildung A.6: Hierarchie der Menüs und Dialoge einer Klasse

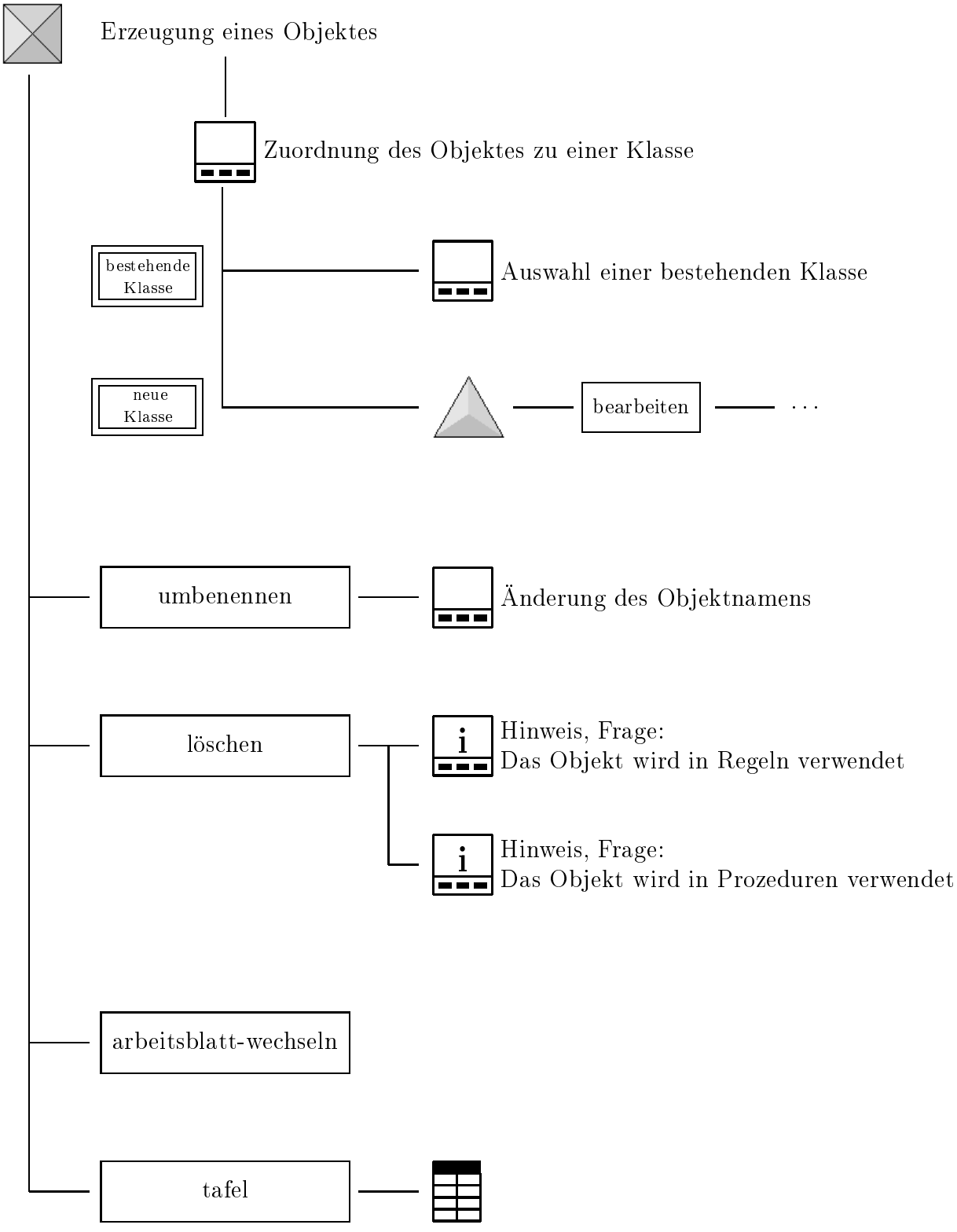


Abbildung A.7: Hierarchie der Menüs und Dialoge eines Objektes

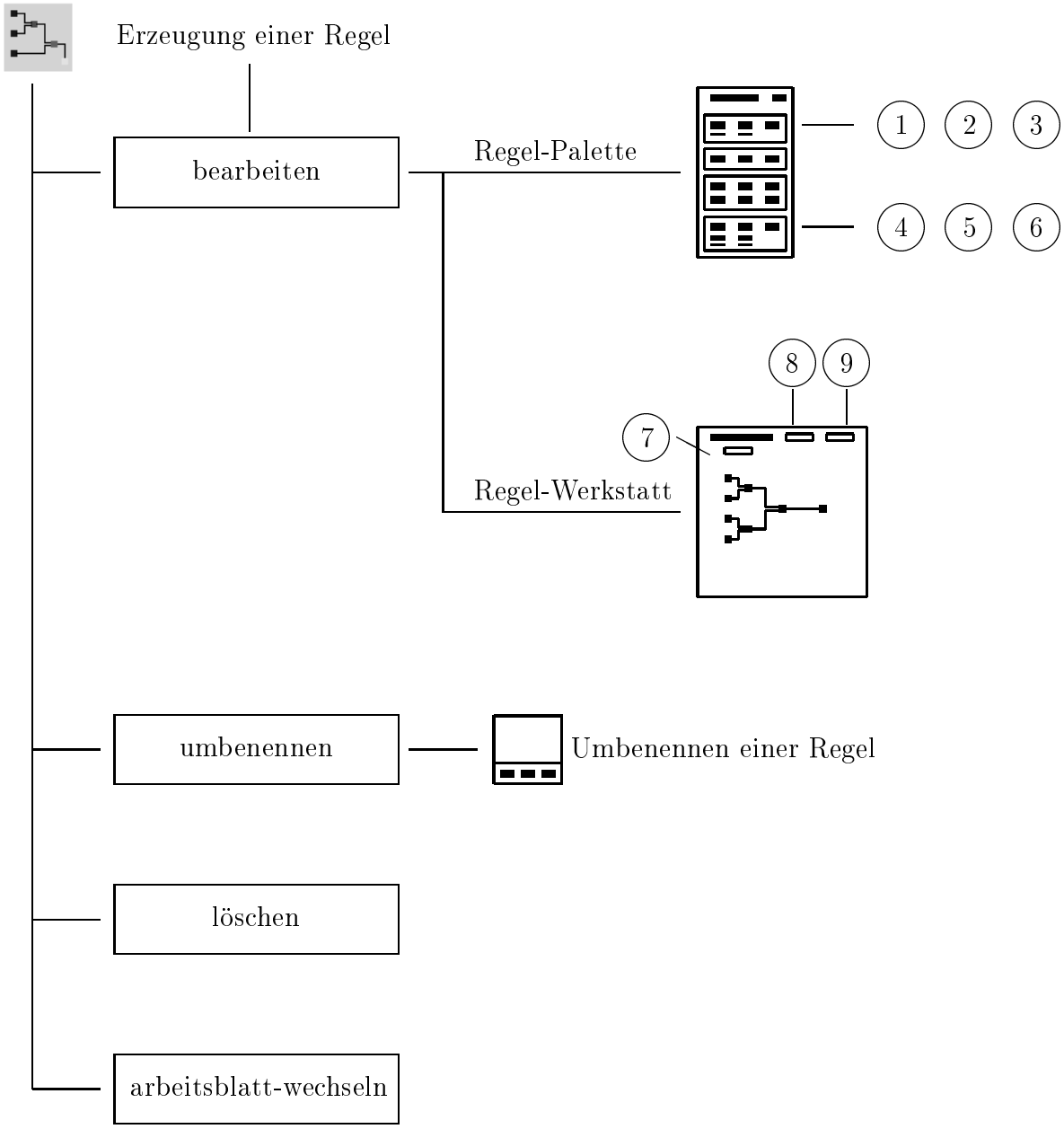
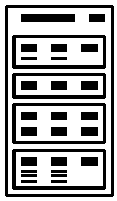
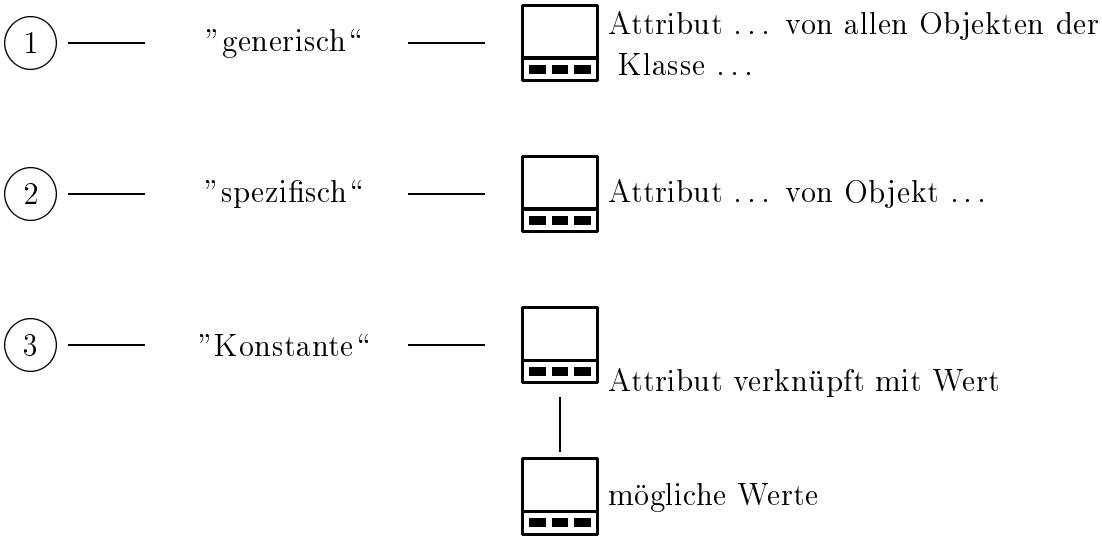


Abbildung A.8: Hierarchie der Menüs einer Regel

Die weiterführenden Aktionen bzw. Dialoge der Schaltflächen (1, 2, 3) und der Regelblöcke (Wenn: 4, 5, 6 und Dann: 7, 8, 9) werden auf den folgenden Seiten dargestellt.



Bedingungsteil-Bausteine



Aktionsteil-Bausteine

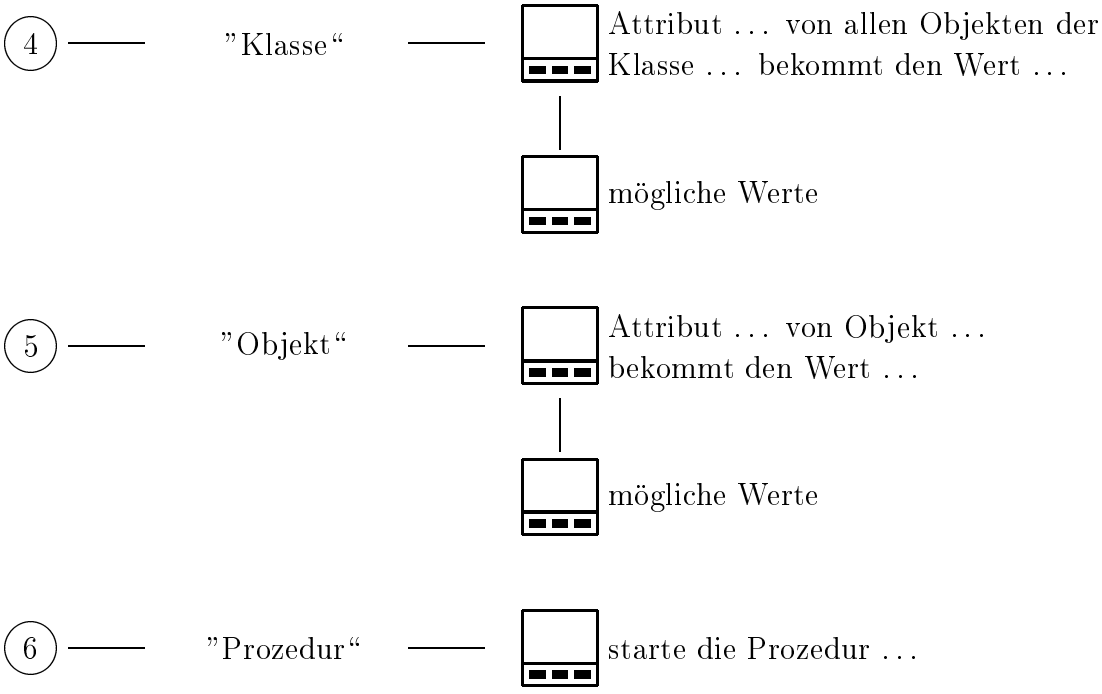
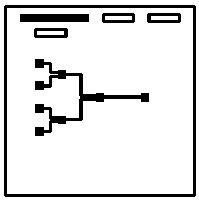


Abbildung A.9: Dialoge der Regel-Bausteine



(7) —— "Regel erzeugen" —— Die Grafik wird überprüft und in die interne Syntax übersetzt.
Die Arbeitsblätter "Regel-Palette" und "Regel-Werkstatt" werden geschlossen.

(8) —— "Abbrechen" —— Änderungen werden gelöscht.
Die alte Regel bleibt erhalten

(9) —— "Verwerfen" —— Alle Regelbausteine werden gelöscht.

Abbildung A.10: Aktionen der Schaltflächen in der Regel-Werkstatt

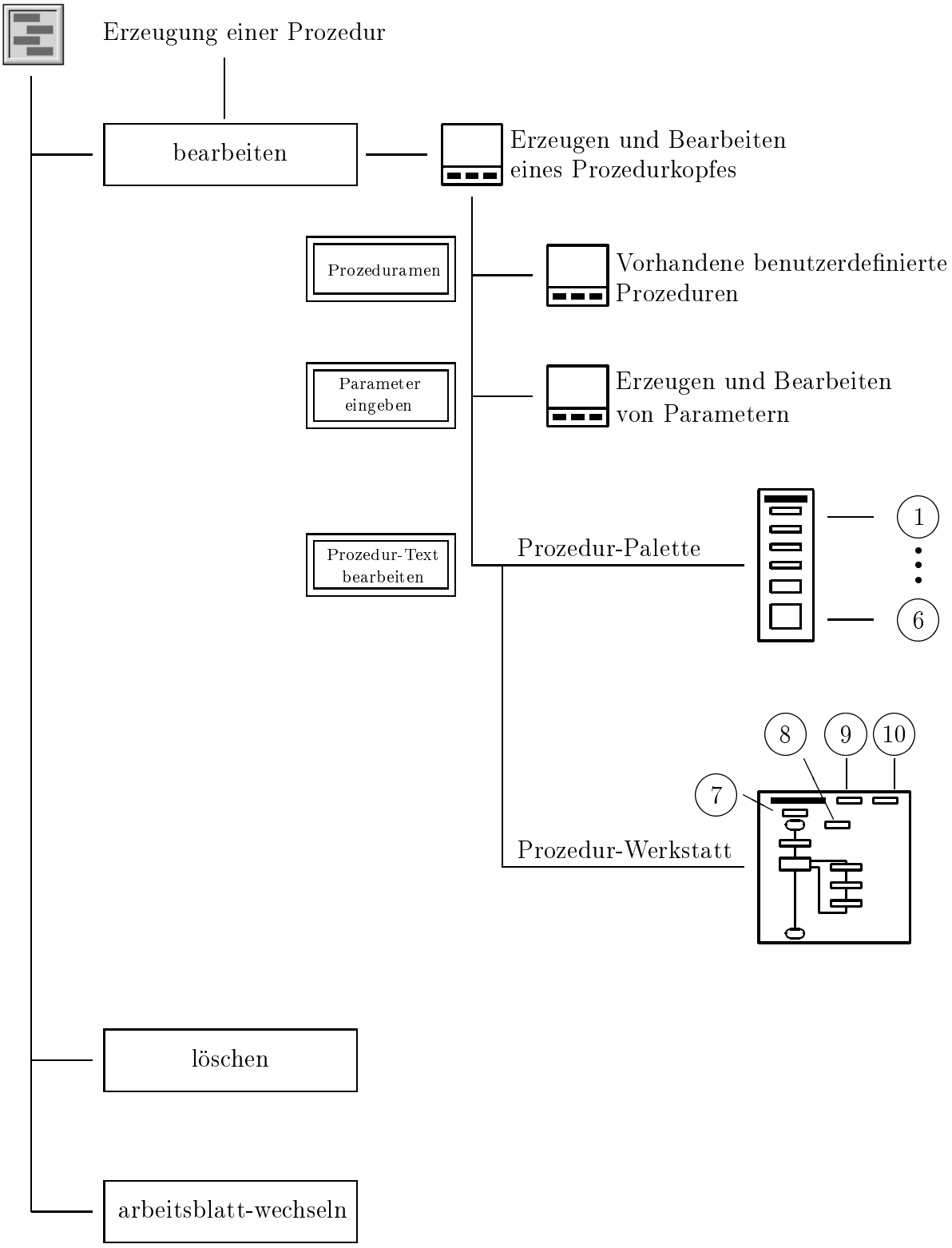


Abbildung A.11: Hierarchie der Menüs und Dialoge einer Prozedur



Einfache Bausteine

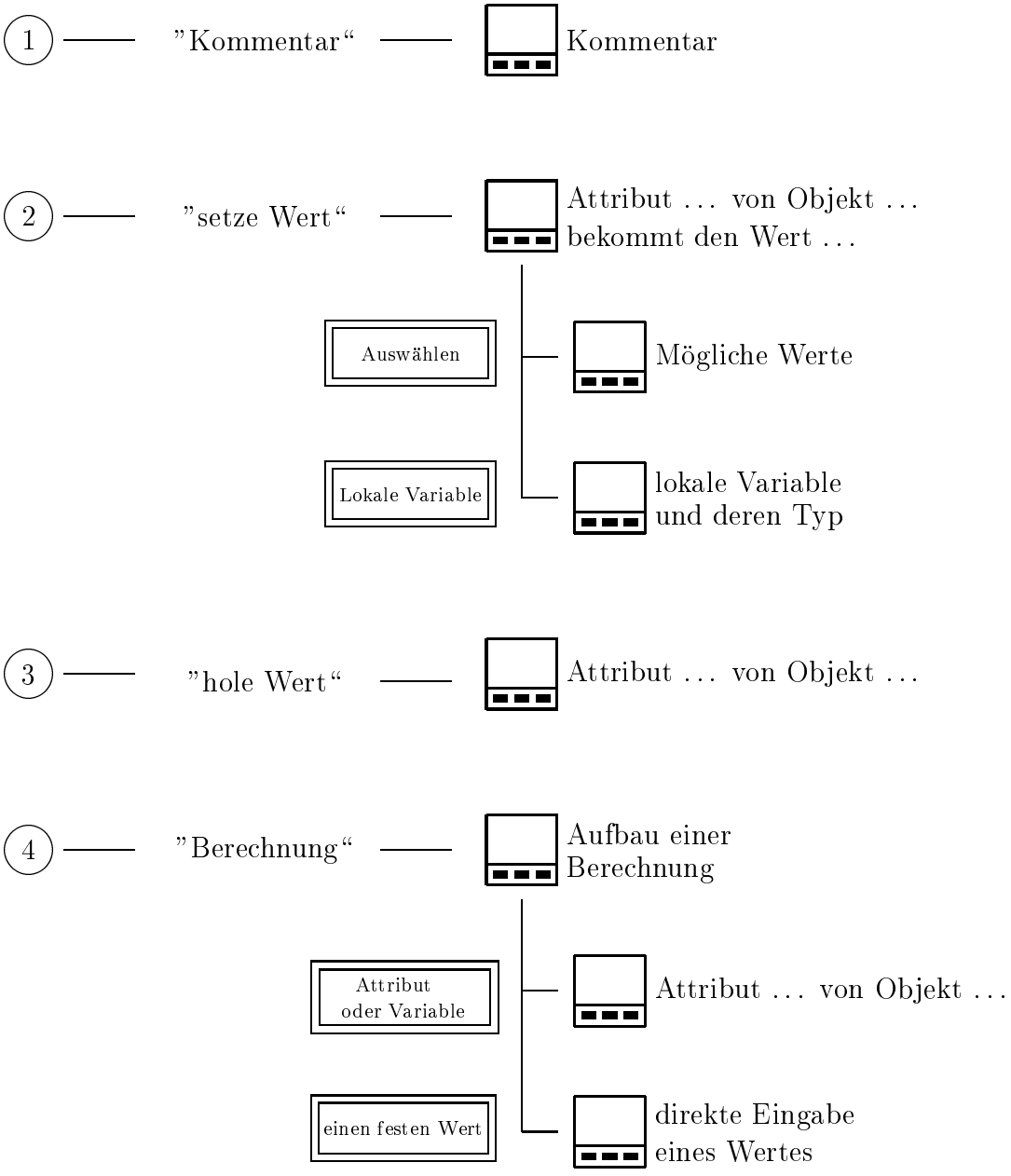


Abbildung A.12: Dialoge der Prozedur-Bausteine I

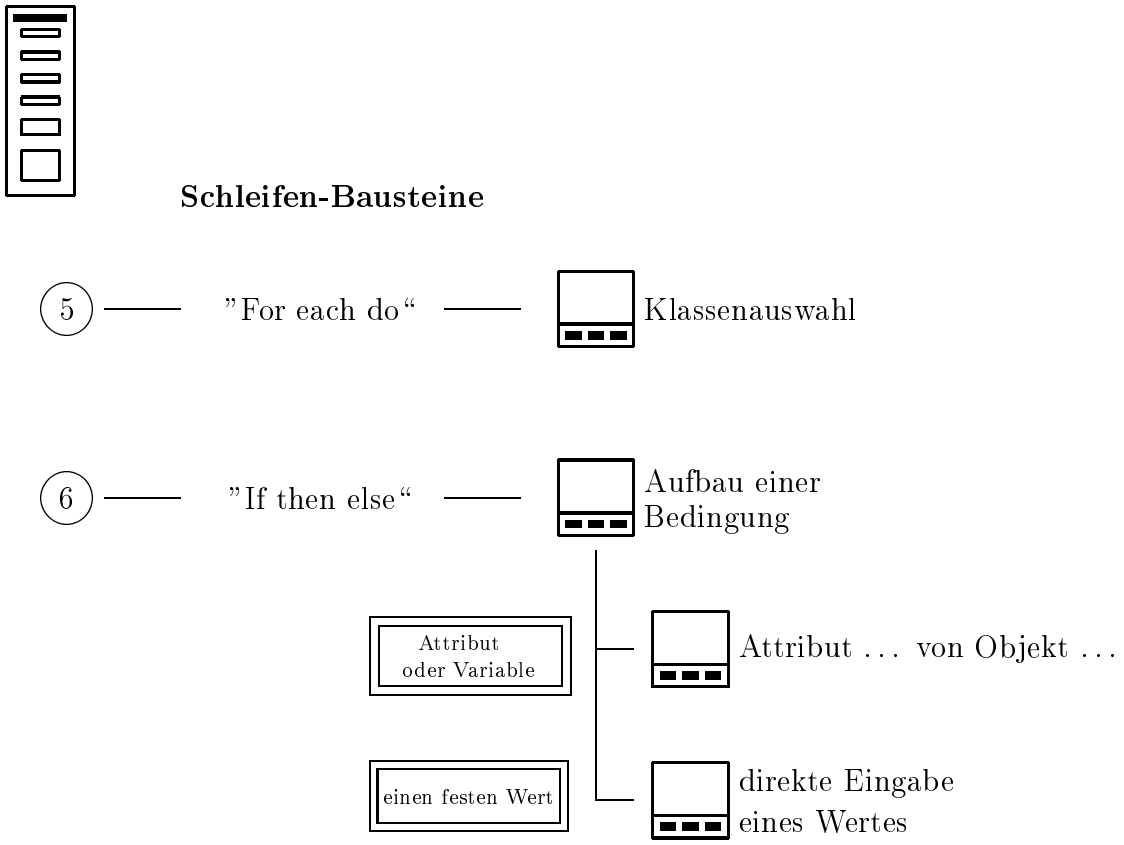
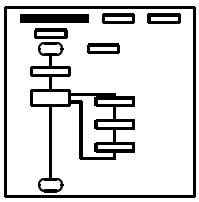


Abbildung A.13: Dialoge der Prozedur-Bausteine II



(7) —— "Prozedur erzeugen" —— Die Grafik wird überprüft und in die interne Syntax übersetzt.
Die Arbeitsblätter "Prozedur-Palette" und "Prozedur-Werkstatt" werden geschlossen.

(8) —— "Parameter" —— Erzeugen und Bearbeiten von Parametern

(9) —— "Abbrechen" —— Änderungen werden gelöscht. Die alte Prozedur bleibt erhalten

(10) —— "Verwerfen" —— Alle Prozedurbausteine werden gelöscht.

Abbildung A.14: Aktionen der Schaltflächen in der Prozedur-Werkstatt

Erzeugung eines Arbeitsblattes

Eingabe eines Namens

Abbildung A.15: Dialog eines Arbeitsblattes

A.5 Das Modul "funktionaler Test“

In diesem Modul ist alles zusammengefaßt, was der Benutzer an zusätzlicher Funktionalität für den schrittweisen Konsistenztest benötigt.
Zwei Schaltflächen dieses Moduls liegen auf dem Arbeitsblatt "USER-ROOT“ des Grundmoduls "User-Root“:

- die Schaltfläche "funktionaler Test“ und
- die Navigationsschaltfläche "Lösche / lade funktionalen Test“.

Mit der Betätigung der Schaltfläche "funktionaler Test“ wird der Testzyklus gestartet. Wurde ein kompletter Inferenzzyklus beendet, wurden entweder keine Inkonsistenzen gefunden, d.h. die Wissensbasis ist konsistent, oder ein Kontrolldurchlauf muß gestartet werden, um sicher zu gehen, daß die vorgenommenen Änderungen nicht Inkonsistenzen an anderen Stellen im Inferenzzyklus verursachen.

Ist die Entwicklung der gesamten Wissensbasis beendet, kann nach dem funktionalen Test das Konsistenzmodul mit seinen Untermodulen aus dem endgültigen wissensbasierten System entfernt werden. Damit für Wartungsarbeiten das Modul wieder geladen werden kann, verbleibt ein Arbeitsblatt des Moduls Konsistenz mit einer Schaltfläche und den zugehörigen Prozeduren im Modul User-Root.

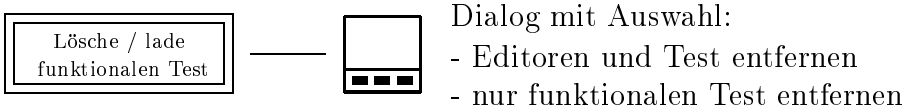


Abbildung A.16: Lösche / lade funktionalen Test

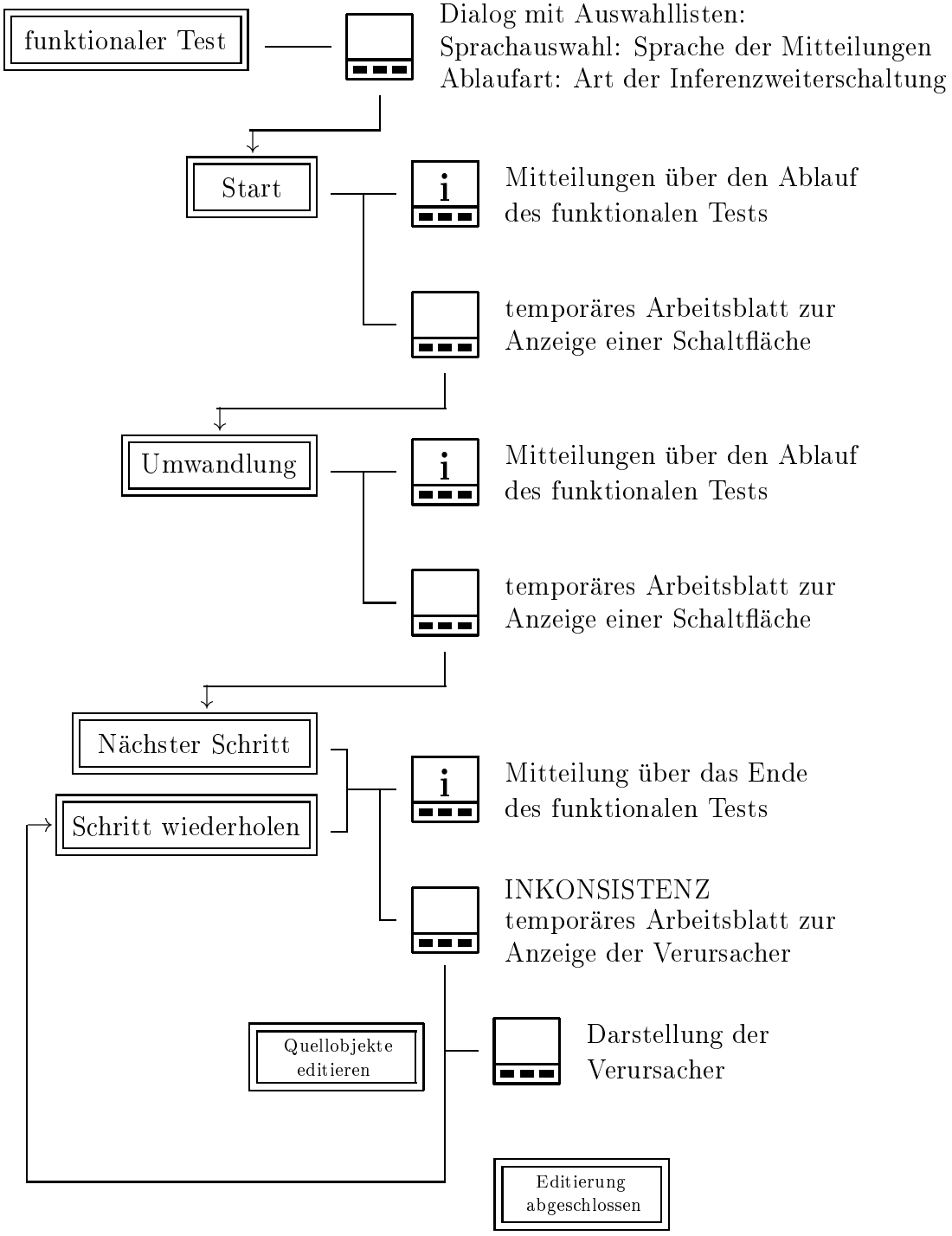


Abbildung A.17: Ablauf des funktionalen Tests