

Diplomarbeit-Nr. 1384

**Spezifikation und Umsetzung einer Modellierungssprache
für ein Aktives Semantisches Netz**

Izidor Jager

1996

Universität Stuttgart
Fakultät Informatik

Aufgabenstellung

Im Rahmen des Sonderforschungsbereichs 374 „Rapid Prototyping“ wird ein Aktives Semantisches Netz (ASN) entwickelt, in dem alle bei einer Produktentwicklung beteiligten Wissensgebiete abgebildet werden. Das ASN ist ein Netz, dessen Knoten Objekte der realen Welt und dessen Kanten die semantischen Zusammenhänge zwischen diesen Objekten repräsentieren. Durch den aktiven Teil des ASN können Änderungen an einer Stelle im Netz automatisch durch das Netz propagiert und auch Aktionen (z.B. an der graph. Oberfläche) ausgelöst werden.

In bisherigen Arbeiten wurde bereits eine Ablagestruktur für ein semantische Netz auf der Basis eines objektorientierten Datenbanksystems, sowie einer darauf aufbauenden Programmierschnittstelle entwickelt. Das Ziel dieser Diplomarbeit ist es, eine Modellierungssprache für diese Wissensbasis zu definieren und Werkzeuge zu entwickeln, die ein Modell in der definierten Modellierungssprache in die Wissensbasis eintragen und umgekehrt Inhalte der Wissensbasis in dieser Sprache darstellen. Im Rahmen dieser Arbeit muß zunächst eine formale Sprache definiert werden, in der alle Wissensinhalte des ASN beschrieben werden können. Für diese Sprache ist ein Werkzeug zu entwickeln, das die textuelle Sprachbeschreibung parst und als ein Modell in der Datenbasis einträgt bzw. bestehende Modelle modifiziert. Für diese Entwicklung können Werkzeuge aus dem Bereich Compilerbau (z.B. Lex und Yacc) verwendet werden. Zusätzlich ist ein Werkzeug zu entwickeln, das ein Modell in der Datenbank in der definierten Sprache ausgibt. Diese Werkzeuge werden in einer graphischen Umgebung integriert.

Inhaltsverzeichnis

AUFGABENSTELLUNG	1
INHALTSVERZEICHNIS	3
ABBILDUNGSVERZEICHNIS	7
TABELLENVERZEICHNIS	9
1 EINLEITUNG	11
2 WISSENSREPRÄSENTATION	17
2.1 Wissen	17
2.1.1 Elemente des Wissens	19
2.2 Arten der Wissensrepräsentation	22
2.3 Prädikatenlogik	22
2.4 Produktionssysteme	24
2.5 Semantische Netze	24
2.6 Objektorientierte Wissensrepräsentation	27
2.7 KL-ONE	31
2.8 BACK	33
3 DAS AKTIVE SEMANTISCHE NETZ	37
3.1 Anforderungen an die Wissensbasis	37
3.2 Die Wissensbasis	39
3.2.1 Konzepte	40
3.2.2 Slots	42
3.2.3 Instanzen	46
3.2.4 Slotwerte	48

3.2.5 Die aktive Komponente	50
3.3 Realisierung des ASN	54
4 KML	57
4.1 Anforderungen an KML	57
4.2 Sprachkonstrukte von KML	58
4.2.1 Aufbau der Sprache	59
4.2.2 Konzepte	60
4.2.3 Typkonstruktoren und komplexe Objekte	62
4.2.4 Konzepte als Konzeptkomponenten	67
4.2.5 Die Konzeptvererbung	70
4.2.6 Objekterzeugung	74
4.2.7 Kommentare in KML	78
4.2.8 Die Realisierung der aktiven Komponente in KML	81
4.3 Äquivalenz von KML und ASN	86
5 UPDATE UND SELEKTIERUNGSOPERATIONEN	91
5.1 Updates	91
5.2 Anfragen	96
6 GRAPHISCHE BENUTZEROBERFLÄCHE	101
6.1 Das Control panel	101
6.1.1 Menüs und Schaltflächen des Control panels	102
6.1.2 Dialog Widgets des Control panels	104
6.2 Das Show-Window	107
6.2.1 Menü und Schaltflächen des Show-Windows	108
6.2.2 Dialog Widgets des Show-Windows	109
7 IMPLEMENTATION	111
7.1 Werkzeuge	111
7.1.1 X-Windows	111
7.1.2 OSF/MOTIF	114
7.1.3 Lex	117
7.1.4 Yacc	119
7.1.5 C++	120

7.2 Datenstrukturen	120
7.3 Architektur und Module	123
8 FAZIT	129
8.1 Zusammenfassung	129
8.2 Schlußwort	130
ANHANG A: SYNTAX VON KML	131
LITERATURVERZEICHNIS	137

Abbildungsverzeichnis

ABBILDUNG 2.1: GRAPHISCHE DARSTELLUNG DES MUSTERS EINES OBJEKTES.	21
ABBILDUNG 2.2: DAS WORTKONZEPT FÜR EINE PFLANZE NACH QUILLIAN.	25
ABBILDUNG 2.3: BEISPIEL FÜR EIN TYPISCHES SEMANTISCHES NETZ.	26
ABBILDUNG 2.4: AUFBAU EINES FRAMES.	29
ABBILDUNG 2.5: BEISPIEL FÜR EINE FRAME-STRUKTUR.	30
ABBILDUNG 3.1: UNTERTEILUNG IN ASSERTIONALE UND TERMINOLOGISCHE KNOTEN	40
ABBILDUNG 3.2: BEISPIEL FÜR KONZEPTKNOTEN.	42
ABBILDUNG 3.3: BEISPIEL FÜR SLOTKNOTEN.	45
ABBILDUNG 3.4: : BEISPIEL FÜR INSTANZKNOTEN.	47
ABBILDUNG 3.5: EIN BEISPIEL FÜR SLOTWERTKNOTEN.	50
ABBILDUNG 3.6: READ-EVENT IM ASN REALISIEREN.	53
ABBILDUNG 3.7: WEITERES BEISPIEL FÜR EVENTS IM ASN.	53
ABBILDUNG 3.8: EINBETTUNG DER DIPLOMARBEIT IN DAS GESAMTPROJEKT.	56
ABBILDUNG 4.1: PRINZIPIELLER AUFBAU EINER WISSENSMODELLIERUNG IN EINER KML-DATEI	60
ABBILDUNG 4.2: WICHTIGE KONZEPTE FÜR DAS VIDEOKARTENPROJEKT DEFINIEREN	62
ABBILDUNG 4.3: EIGENSCHAFTEN DER KONZEPTE BESCHREIBEN.	66
ABBILDUNG 4.4: DIE SPEZIFIZIERUNG DER EIGENSCHAFTSTYPEN DER KONZEPTE	69
ABBILDUNG 4.5: VERERBUNGSBEZIEHUNG EINFÜHREN.	73
ABBILDUNG 4.6: OBJEKTE EINFÜHREN	77
ABBILDUNG 4.7: BEISPIEL UM KOMMENTARE FÜR KONZEPTE, SLOTS UND INSTANZEN ERWEITERT.	80
ABBILDUNG 4.8: REGELN IN BEISPIEL EINFÜHREN.	85
ABBILDUNG 4.9: STRUKTUR UND AUFBAU DES KONZEPTMODULS	87
ABBILDUNG 4.10: STRUKTUR UND AUFBAU DES OBJEKTMODULS	88
ABBILDUNG 5.1: VORGEHENSWEISE BEIM UPDATEN DER WISSENSBASIS.	93
ABBILDUNG 5.2: UPDATE-OPERATIONEN DURCHFÜHREN.	95
ABBILDUNG 6.1: DAS HAUPTFENSTER DER KML-GUI.	102
ABBILDUNG 6.2: MENÜS UND SCHALTFLÄCHEN DES STEUERUNGSFENSTER.	102
ABBILDUNG 6.3: DAS DATEIAUSWAHLFENSTER, NACH AKTIVIERUNG VON NEW ODER OPEN.	105
ABBILDUNG 6.4: MESSAGE BOX, ZUR ANZEIGE, DAß WISSENSBASIS BEREITS EXISTIERT.	105
ABBILDUNG 6.5: MESSAGE BOX, ZUR ANZEIGE, DAß WISSENSBASIS NICHT EXISTIERT.	106
ABBILDUNG 6.6: ANZEIGE, DAß KEINE WISSENSBASIS SELEKTIERT WURDE.	106
ABBILDUNG 6.7: MÖGLICHE FEHLERMELDUNG BEIM ÜBERSETZEN EINER KML-DATEI.	106
ABBILDUNG 6.8: DAS SHOW-WINDOW IN AKTION.	107
ABBILDUNG 6.9: MENÜ UND SCHALTFLÄCHEN DES SHOW-WINDOWS.	108

ABBILDUNG 6.10: PROMPT-WIDGET ZUR EINGABE EINES OBJEKTNAMENS.	110
ABBILDUNG 7.1: ZUSAMMENHANG ZWISCHEN XLIB, INTRINSICS, WIDGET SET UND X CLIENT	113
ABBILDUNG 7.2: HIERARCHIE DER CONTAINER WIDGETS	115
ABBILDUNG 7.3: HIERARCHIE DER DISPLAY WIDGETS	116
ABBILDUNG 7.4: HIERARCHIE DER DIALOG WIDGETS	117
ABBILDUNG 7.5: HIERARCHIE DER SHELL WIDGETS	117
ABBILDUNG 7.6: DATENFLUß ZWISCHEN ASN-WISSENSBASIS UND KML-DATEI.	121
ABBILDUNG 7.7: KML-WISSEN IN MEHRDIMENSIONALEN BINÄRBAUM ABLEGEN.	122
ABBILDUNG 7.8: DAS ZUSAMMENWIRKEN DER MODULE IM GESAMTSYSTEM.	124
ABBILDUNG 7.9: DIE WIDGET-HIERARCHIE DER KML-GUI.	126

Tabellenverzeichnis

TABELLE 3-1: DIE STRUKTUR UND DIE BEZIEHUNGEN EINES KONZEPTKNOTENS.	41
TABELLE 3-2: DIE STRUKTUR UND DIE BEZIEHUNGEN EINES SLOTKNOTENS.	45
TABELLE 3-3: DIE STRUKTUR UND DIE BEZIEHUNGEN EINES INSTANZKNOTENS.	47
TABELLE 3-4: DIE STRUKTUR UND DIE BEZIEHUNGEN EINES SLOTWERTKNOTENS.	49
TABELLE 3-5: DIE RELATIONENARTEN DES ASN.	54
TABELLE 4-1: ELEMENTARE KONZEPTE AUS DEM ASN UND DIE DAZUGEHÖRIGEN TYPEN AUS KML.	63

1 Einleitung

Die Konkurrenz zwischen Unternehmen nimmt national wie international immer weiter zu. Nicht nur im Dienstleistungssektor, auch im Bereich der Produktentwicklung sind zunehmend dynamische Innovationen für den Erfolg eines oder mehrerer Produkte und damit eines Unternehmens ausschlaggebend. Die Zeit, die für das Marketing zur Verfügung steht wird zunehmend kürzer, die Produktlebenszeit ist bereits in vielen Fällen kürzer als die Entwicklungszeit. Um ihre Stellung im Markt zu halten oder gar auszubauen, erhöhen Unternehmen die Zahl ihrer neuen Entwicklungen und versuchen die Prototypfertigung zu beschleunigen, um frühzeitig die Produkte auf den Markt zu bringen.

Die Vorgehensweise nach dem Taylorsystem¹ führte zu einer hochspezialisierten Arbeitsteilung und dadurch zu einer ausgeprägt hierarchischen Form der Organisation. Diese hierarchischen Strukturen erschweren die schnelle Anpassung an Verbraucherbedürfnisse wegen der aufwendigen über Umwege gehenden Kommunikation zwischen einzelnen Knoten im Hierarchiebaum und behindern innovative Kräfte aufgrund der strengen Reglementierung. Die Konsequenz ist die mangelnde Flexibilität eines nach diesem System wirtschaftenden Unternehmens, sich auf die Erfordernisse des Marktes einzustellen.

Zwar werden seit einigen Jahren zunehmend computerbasierte Werkzeuge wie CAD-Programme für die Produktentwicklung verwendet, diese sind aber in ihrer Mehrzahl zu einem Zeitpunkt nur von einer Person benutzbar und bieten auch sonst wenig Möglichkeiten für eine koordinierte Kooperation mehrerer Personen, obwohl die technischen Voraussetzungen dafür geschaffen sind. Gerade die Kooperation und Kommunikation sind aber entscheidende Elemente für ein erfolgreiches Arbeiten.

Die Produktentwicklungszeit hängt in starkem Maße von der Entwicklung von Prototypen ab. Diese stellen eine Grundlage für die Kommunikation zwischen Entwicklern untereinander und Entwicklern und Kunden dar. Obwohl sie noch nicht ausgereift sind, ermöglichen sie dennoch eine frühzeitige Bewertung des Produktes.

Um die schon geschilderten Probleme der Unternehmen (lange Produktentwicklungszeiten, steigender Konkurrenzdruck) zu bewältigen, müssen die Iterationszyklen im Entwicklungsprozeß radikal beschleunigt und der Wissensgewinn in den einzelnen Phasen transparent sowie der Wissensaustausch zwischen den Mitarbeitern deutlich gesteigert werden.

¹ F.W. Taylor *1856, †1915: System der Betriebsführung zur bestmöglichen Ausnutzung der menschl. Arbeitskraft.

Methoden wie das *Simultaneous Engineering* und *Lean Production* wurden entwickelt um die Produktionszeiten und Kosten zu senken. Dafür wurden eine Vielzahl von computerbasierten Werkzeugen entwickelt. Beispiele hierfür sind Workflow Systeme und Engineering Data Management. Ein Überblick verschiedener Werkzeuge für die Produktentwicklung findet sich in [Roller95].

Viele dieser Tools unterstützen nur einzelne Aspekte des Entwicklungsprozesses. So ist auch hier ein Problem, das viele Entwickler nur über Umwege und mit Zeitaufwand an benötigte Informationen gelangt.

Das führte zu einer neuen Organisationsform in der Produktentwicklung: dem *Rapid Prototyping*. Dieser Begriff beinhaltet alle technischen, methodischen und organisatorischen Maßnahmen, die ein Produkt von seiner Entstehungsidee bis zum marktfähigen Produkt begleiten.

Merkmale des Rapid Prototypings sind:

- eine dezentrale Organisation von Arbeitsgruppen, die aus Experten und Personen bestehen, die Brücken zu anderen Gruppen schlagen können,
- eine schnelle Erzeugung virtueller und physikalischer Prototypen, z. B. mit Hilfe der Stereolithographie, einem Verfahren zur schnellen Fertigung von Prototypen aus Kunststoff (mit Hilfe eines durch Computertdaten gesteuerten Lasers).
- eine vollständige Transparenz von Wissen während des gesamten Produktentwicklung, vor allem aber während der Prototyperstellung,
- dem Einsatz von Netzwerktechnologien, um eine Kommunikation und Kooperation der Mitarbeiter und Arbeitsgruppen zu ermöglichen,
- Werkzeuge, um schnelle Entscheidungen treffen zu können und die Hilfen bei der Wahl aus mehreren Alternativen geben (zum Beispiel Expertensysteme).

Aus dieser Aufzählung wird schon ersichtlich, welche immens wichtige Bedeutung der Informationstechnologie zukommt. Da das Wissen (der Experten, Mitarbeiter und in Informationssystemen) als die wertvollste Ressource eines Unternehmens angesehen werden kann, ist der Einsatz einer Wissensbasis entscheidend, die das gesamte Wissen das in allen Phasen der Produktentwicklung anfällt und das für die weitere Arbeit notwendig ist, enthält. Die Wissensbasis selbst ist in ein wissensbasiertes System eingebettet. Das Wissen wird in diesem Kontext auch als *rapid prototyping knowledge* bezeichnet. Durch die Integration von Informations-, Kommunikations- und Kooperationssystemen mit dieser Wissensbasis kann dieses Wissen auf wirksame Weise genutzt werden. Die Inhalte der Wissensbasis stellen die Basis für die Kooperation zwischen verschiedenen Entwicklern dar.

Diese Wissensbasis beinhaltet

- Wissen über den Entwurfsprozeß,

- Expertenwissen,
- Wissen über frühere Entwicklungsprozesse,
- Wissen über organisatorische Strukturen des Unternehmens und über Mitarbeiter,
- Entwurfspläne, Notizen, Zeichnungen,
- Projektpläne, Spezifikationen,
- ...

Die Informationen, die in der Wissensbasis abgelegt werden, sind breit gefächert und sehr vielfältig. Ferner findet eine permanente Erweiterung und Modifikation des Wissens statt.

Eine wichtige Eigenschaft dieser Wissensbasis ist ihre aktive Komponente. Diese veranlaßt die Wissensbasis angemessen auf bestimmte Ereignisse zu reagieren. Zum Beispiel sollte sie auf Inkonsistenzen innerhalb des Systems in angemessener Form reagieren, oder automatische Schlußfolgerungen ziehen können.

Die Art der Informationen, die die Wissensbasis speichern und verwalten sollte, übersteigt die Fähigkeiten einer herkömmlichen Datenbank [Eck96]. Es müssen Konzepte der Wissensrepräsentation, einem Forschungsbereich der Künstlichen Intelligenz, eingesetzt werden.

Ein System, das den oben geschilderten Anforderungen genügen soll, basiert auf dem *Aktiven Semantische Netz (ASN)*. Das ASN ist ein Forschungsprojekt am Lehrstuhl Grundlagen der Informatik an der Universität Stuttgart. Die Entwicklung des ASN ist wiederum Teil des Forschungsprojektes SFB 374 „Entwicklung und Erprobung innovativer Produkte - Rapid Prototyping“ der Deutschen Forschungsgesellschaft (DFG), der Universität Stuttgart, der TU Dresden und der Forschungsabteilung der Daimler Benz AG.

Das Aktive Semantische Netz stellt die Basis für ein wissensbasiertes System dar, welches Entwickler unterstützen soll, schnell Prototypen für Produkte zu realisieren, indem es Wissen für alle Phasen des Entwicklungsprozeß aufnimmt, speziell die Informationen, die während des Rapid Prototyping anfallen, verarbeitet und zur Verfügung stellt. Das im Aktiven Semantischen Netz abgebildete Wissen beinhaltet neben statischen Produkt-, Technologie- und Prozeßdaten auch dynamisches Wissen, um die Abhängigkeiten zwischen technischen Produktfunktionen, Kosten, Qualitätsmerkmalen, Kooperationsformen etc. darzustellen. Weiterhin sollen verschiedene Benutzermodelle berücksichtigt werden, um verschiedene Sichtweisen auf die Wissensbasis bereitzustellen. Da es gleichzeitig von mehreren Anwendern von unterschiedlichen Plattformen aus benutzt werden kann, muß weiterhin ein Transaktionskonzept für verteilte Systeme entwickelt werden.

Durch den aktiven Teil des ASN können Änderungen an einer Stelle in der Wissensbasis durch das gesamte Netz propagiert und Aktionen, wie zum Beispiel das Benachrichtigen eines Mitarbeiters, ausgelöst werden.

Hauptaufgabe dieser Diplomarbeit war es, eine formale Sprache zu entwerfen und zu spezifizieren, die das Wissen, welches eine ASN-Wissensbasis repräsentieren kann, darstellt. Diese Sprache dient als Schnittstelle zwischen Mensch und ASN, ebenso als Diskussionsgrundlage für Modellierer, um Wissen das beim Rapid Prototyping anfällt, mit Hilfe einer Sprache beschreiben zu können. Ferner wurde ein Übersetzer implementiert, der Wissen, das in der entworfenen Sprache vorliegt, in die ASN-eigene Darstellung überführt und ein Programm, das Wissen aus einer ASN-Wissensbasis in die Sprache übersetzt und damit diese Wissensbasis für die Anwender transparent macht.

Dabei wurde zunächst geklärt, welche Ausdrucksmöglichkeiten das ASN bietet und welche Konzepte der Wissensrepräsentation im ASN realisiert sind. Dabei ist zu beachten, daß das ASN kein abgeschlossenes System ist, sondern sich vielmehr in der Entwicklung befindet. Während diese Diplomarbeit lief wurde der erste Prototyp des ASN erstellt. Kern dieses Prototypen ist eine Wissensbasis mit einer noch später vorgestellten Ausdrucksmächtigkeit (siehe Kapitel 3).

Weiterhin wurde in dieser Arbeit untersucht, auf welche Weise eine vorhandene Wissensbasis nachträglich erweitert und modifiziert und wie gezielt auf bestimmtes Wissen zugegriffen werden kann. Dabei sind verschiedene Möglichkeiten zu beschreiben und eine geeignete Auswahl zu treffen. Zur Realisierung der Manipulationen und Selektionen wurden Tools implementiert.

Da von einem Benutzermodell ausgegangen wird, das von Computerexperten bis Computerlaien reicht, wird eine kommandoorientierte Benutzung der Programme vermieden, vielmehr werden die einzelnen Programme unter einer Benutzeroberfläche zusammengefaßt und dem Anwender damit eine komfortable Arbeitsweise ermöglicht.

Nach der Beschreibung der Aufgaben soll hier nun ein genauerer Überblick über den Aufbau dieser Dokumentation erfolgen.

In diesem ersten Kapitel wurde eine Motivation für das Thema der Diplomarbeit geschaffen und die Einordnung der Arbeit in laufende Projekte aufgezeigt.

Im zweiten Kapitel wird auf den Themenbereich der Wissensrepräsentation eingegangen, um eine Grundlage für das Verständnis des Aktiven Semantischen Netzes zu ermöglichen. Dabei soll zunächst, unabhängig von einer Darstellungsform, Wissen charakterisiert werden. Danach werden deklarative und prozedurale Repräsentationsansätze vorgestellt. Im Mittelpunkt stehen dabei Semantische Netze und Konzepte anderer Wissensdarstellungen, die für das Aktive Semantische Netz von Bedeutung sind, beispielsweise die Integration prozeduralen und deklarativen Wissens in Frames. Weiterhin wird eine Wissensrepräsentationsprache beispielhaft (BACK) vorgestellt.

Im darauffolgenden dritten Kapitel werden die Anforderungen, die sich durch das Rapid Prototyping ergeben, an das Aktive Semantische Netz näher vorgestellt. Danach wird der Aufbau, die Struktur, die Semantik und die aktive Komponente des Aktiven

1 Einleitung

Semantischen Netzes beschrieben. Mit Hilfe von Beispielen werden die Repräsentationskonstrukte veranschaulicht.

Im viertem Kapitel wird die in dieser Arbeit entworfene und spezifizierte Wissensrepräsentationsprache KML vorgestellt. Dabei werden zunächst die Anforderungen an diese Sprache hervorgehoben, danach werden die einzelnen Sprachkonstrukte, ihr Bezug zum ASN und deren Syntax vorgestellt. Am Ende des Kapitels wird die Äquivalenz der Ausdrucksmächtigkeit von KML und ASN beschrieben.

In Kapitel fünf werden verschiedene Möglichkeiten vorgestellt, um Update- und Selektionoperationen auf einer ASN-Wissensbasis durchzuführen. Es werden Vor- und Nachteile diskutiert und eine Auswahl getroffen.

Im sechsten Kapitel wird die graphische Benutzeroberfläche, ihre Aufgaben, ihre Bedienung und ihre Funktionalität behandelt.

In Kapitel sieben werden die Werkzeuge, die für die Implementation verwendet wurden, die zentrale Datenstruktur wie die Architektur der gesamten Applikation und deren Module vorgestellt.

Ein kurzer Ausblick auf mögliche Verbesserungen und Erweiterungen, ein Schlußwort, ein Anhang sowie ein Literaturverzeichnis schließen diese Arbeit ab.

2 Wissensrepräsentation

In diesem Kapitel sollen die Grundlagen für das Verständnis des Aktiven Semantischen Netzes geschaffen werden.

Der Themenbereich Wissensrepräsentation ist ein zentrales Teilgebiet der künstlichen Intelligenz (KI). In diesem Forschungsgebiet wird versucht, Problembereiche der realen Welt in ein formales Modell zu überführen, indem das Wissen über diesen Problembereich in adäquater Form dargestellt und strukturiert wird. Adäquat bedeutet, daß es eine Interpretationsvorschrift gibt, die mindestens die Formulierung und Auswertung von Anfragen auf diese Strukturen zuläßt, besser aber auch noch Änderungsoperationen gestattet [Reimer91]. Typischerweise sind die zu modellierenden Weltausschnitte derart komplex, sowohl in quantitativer wie qualitativer Hinsicht, das keine vollständige Repräsentation erreicht werden kann. Nach [Woods87] stellen Modelle der Problembereiche lediglich Approximationen davon dar.

Die Repräsentation von Wissen spielt eine zentrale Rolle bei der Entwicklung wissensbasierter Systeme, die dadurch charakterisiert sind, daß ihr Ziel die Lösung von Aufgaben in einem komplexen Weltausschnitt ist. Die wesentliche Stärke wissensbasierter Systeme (im Gegensatz zu Datenbanken) liegt in ihrer Fähigkeit, über dem internen Modell Schlüsse durchzuführen und hierdurch Probleme zu lösen.

Nach [Kurbel92] ist ein wissensbasiertes System ein Softwaresystem, bei dem das Fachwissen über ein Anwendungsgebiet explizit und unabhängig vom allgemeinen Problemlösungswissen dargestellt wird. Den Kern eines wissensbasiertes Systems bildet die Wissensbasis, in der das Anwendungswissen in einer geeigneten Repräsentationsform vorliegt.

Im Mittelpunkt jeder Wissensrepräsentation steht das Wissen. Im folgenden soll das Wissen charakterisiert werden.

2.1 Wissen

Es soll hier nicht versucht werden, Wissen formal zu definieren. Vielmehr soll hier, unabhängig von einem speziellen Wissensrepräsentationsformalismus, ein intuitives Modell über das Wissen vorgestellt werden, daß sich an der Darstellung in [PaChKhWo89] orientiert.

Über Wissen kann man immer auf verschiedenen Ebenen diskutieren. Wenn man Wissen darstellen will, muß man sich Gedanken darüber, welche Abstraktionsebene und welche Sicht auf das Wissen angemessen ist. So sieht ein Arzt den menschlichen Körper aus einer anderen Perspektive als ein Bildhauer. Im allgemeinen baut das Wissen des

Menschen auf elementaren und grundlegenden Konzepten, den Objekten auf (eine sehr ausführliche Beschreibung über die Natur der Objekte findet sich in [Booch94]). Für manche Problembereiche ist es relativ einfach, diese elementaren Konzepte oder Objekte zu erkennen und darzustellen, in anderen dagegen gehört eine Menge Erfahrung und eine Portion Fingerspitzengefühl dazu, die richtige Zerlegung des Problems in Objekte zu erreichen (auch dazu finden sich in [Booch94] geeignete Methoden).

Nach [Booch89] lassen sich Objekte folgendermassen definieren: „Objekte sind individuelle und identifizierbare Elemente oder ebensolche Einheiten, die entweder real oder abstrakt sein können und eine wohldefinierte Rolle im jeweiligen Problembereich haben.“ Allgemeiner läßt sich ein Objekt definieren als etwas mit einer klar definierten Grenze [Shankar84]. So kann ein chemischer Prozeß in einer Fabrik als Objekt behandelt werden, weil er eine klar definierte konzeptuelle Beschreibung besitzt, mit anderen Objekten in einer Menge von Operationen interagiert und daraus ein wohldefiniertes Verhalten entsteht. Dinge, die man nicht als Objekte beschreiben kann sind z.B. Zeit, Liebe, Schönheit. Auf der anderen Seite sind diese Dinge mögliche Eigenschaften von Objekten

Hat man erst einmal die elementaren Objekte für die Wissensrepräsentation erkannt, muß man sie selbst und ihre gegenseitigen Beziehungen definieren. Diese Definitionen basieren auf den elementaren Elementen des Wissens:

- Bezeichnung der Objekte
- Beschreibung der Objekte
- Organisation von Objekte
- Beziehungen zwischen den Objekten
- Bedingungen für ein Objekt

Diese Komponenten sind im allgemeinen in allen Wissensrepräsentationssystemen vorhanden, werden jedoch auf unterschiedliche Weise dargestellt und behandelt.

Diesen Komponenten kann man analoge Elemente der natürlichen Sprache zuordnen. So wird die Funktion der *Bezeichnung* gewöhnlich durch Substantive wahrgenommen. Die Funktion der *Beschreibung* übernehmen die Adjektive, wie zum Beispiel groß und glücklich. In formaler Notation wird man neben Adjektiven auch Werte verwenden, um ein Objekt zu beschreiben.

Beziehungen werden durch transitive Verben oder spezielle Hauptwörter, die Beziehungen beschreiben, wahrgenommen. Aus der Sicht der Wissensrepräsentation lassen sich mehrere Arten von Beziehungen unterscheiden, beispielsweise

die aktive – Strom fließt durch den Draht,

oder

die strukturelle – die Röhre ist mit dem Ventil verbunden.

Die Aufgabe der *Organisation* nehmen oft Hilfsverben in Kombination mit Artikeln wie *ist ein (is a)* oder *hat ein (has a)* wahr:

Das Auto hat einen Motor oder Sokrates ist ein Mensch.

Bedingungen werden typischerweise durch „Wenn, dann“-Sätze realisiert, die angeben welche Objektbeschreibungen und Beziehungsmuster zwischen Objekten überhaupt zulässig sind.

2.1.1 Elemente des Wissens

Objekte werden wie weiter oben schon gezeigt durch fünf grundlegende Elemente charakterisiert. Diese Komponenten sollen hier nun näher vorgestellt werden:

Bezeichnung eines Objektes:

Typischerweise werden Objekte durch Namen identifiziert. So ist „IBM“ der Name eines weltweit agierenden Computerunternehmens, „Schwefel“ der Name eines chemischen Elements. Damit es keine Verwirrungen gibt, sollte jedem Objekt ein eindeutiger Name, der ihn identifiziert, zugewiesen werden.

Beschreibung eines Objektes:

Abstraktion ist eine der fundamentalen Methoden, die wir Menschen zur Bewältigung von Komplexität benötigen. Eine Abstraktion gibt die wesentlichen Charakteristika eines Objektes an, die es von allen anderen Arten von Objekten unterscheiden. Eine gute Abstraktion konzentriert sich auf die wichtigen Details und vernachlässigt die Details, die, zumindest im Augenblick, unerheblich sind oder vom Wesentlichen ablenken würden [Shaw84]. So hat ein komplexes Objekt wie eine Person eine breite Palette von Eigenschaften: physiologische, anatomische, kognitive, handwerkliche Fähigkeiten, Für ein medizinisches Diagnosesystem sind von den eben genannten aber nur die ersten beiden Gruppen von Eigenschaften relevant. Die Beschreibung eines Objektes setzt sich aus einer Menge von Attributen und ihrer Ausprägungen (auch Werte genannt) zusammen. Für die Attributnamen gilt das gleiche wie für die Objektbezeichner, sie müssen eindeutig sein. Wobei sich diese Eindeutigkeit auf die Attribute innerhalb des Objektes bezieht. Zwei unterschiedliche Objekte dürfen ohne weiteres gleichnamige Attribute verwenden. Diese Attribute beschreiben die nach außen sichtbaren Eigenschaften des Objektes.

Obwohl der grundlegende Prozeß der Beschreibung relativ einfach ist, werden in den Wissensrepräsentationssystemen unterschiedliche Wege eingeschlagen. In logikorientierten Darstellungsmethoden wird die Beschreibung mit Prädikaten und Argumenten realisiert, in der Art:

Jochen ist-alt 24

In objektorientierten Systemen wird das Alter (um bei dem Beispiel zu bleiben) dem korrespondierenden Attribut des Objektes, das Jochen symbolisiert, zugewiesen, in der Art

alter von Jochen := 24

Es gibt auch Situationen in denen zu einem Attribut (vorläufig) kein Wert angegeben werden kann. So hat sicher jede Stadt einen Vorsteher (zum Beispiel den Bürgermeister), aber nicht immer ist er bekannt. Man kann demnach die Attribute eines Objektes generell in zwei Gruppen einteilen: Die eine, in der alle Attribute sind, von denen die Werte bekannt sind, und die andere, die alle Attribute enthält, dessen Werte (noch) nicht bekannt sind.

Organisation von Objekten:

Zusätzlich zur Beschreibung der Objekte, werden Objekte in konzeptuellen Kategorien organisiert. Eine Möglichkeit dies zu erreichen ist, bestimmte Objekte als Instanzen allgemeinerer Objekte zu beschreiben. Hier werden jedem Objekt ein oder mehrere Eltern zugewiesen, außer den Top-level Objekten in der Hierarchie. Dadurch wird eine *Vererbungshierarchie* realisiert. Hierbei werden die Eigenschaften der allgemeineren Objekte an die spezielleren Konzepte vererbt. Diese Art von Organisation hilft die Komplexität von Problembereichen zu mildern, indem neues Wissen auf vorhandenem (bekanntem) Wissen aufgebaut wird und nicht von Grund auf neu eingeführt werden muß. Semantisch gesehen, legt die Vererbung eine „is a“-Beziehung fest. Ein Krokodil ist eine Art Reptil und ein Reptil ist ein Wirbeltier.

Beziehungen zwischen den Objekten:

Neben den „is a“-Hierarchien, die man auch als Verallgemeinerungs- und Spezialisierungs-Beziehungen bezeichnen kann, beschreiben „part of“-Hierarchien sogenannte Verbund-Beziehungen. Dabei enthält ein Objekt ein oder mehrere Eigenschaften, die selbst wiederum durch Objekte beschrieben werden. Zum kann ein Objekt Garten existieren, in dem Blumen wachsen. Diese Blumen sind nun selbst Objekte, aber sie beschreiben auch das Objekt Garten und sind damit Eigenschaften des Gartens. Man kann sagen, daß sich der Garten auf einer höheren Abstraktionsebene befindet als die Blumen, da dieser durch diese mitbeschrieben wird.

Verbunde lassen sich noch weiter unterteilen, indem zum Beispiel danach gefragt wird, ob ein Objekt Eigentum des anderen ist oder vielleicht in einer Verwendungsbeziehung steht. Nähere Informationen finden sich wiederum in [Booch94].

Beziehungen zwischen Objekten können dadurch realisiert werden, indem Werte der Objektattribute andere Objekte sind oder aber auch über die weiter oben beschriebene Vererbungsbeziehung. Bei einer graphischen Darstellung (siehe semantisches Netz, Kap. 2.5) von Wissen lassen sich Beziehungen immer als Verbindungen zwischen Objekten darstellen, während die Knoten die Objekte sind.

Auch hier drücken logikbasierte Wissenrepräsentationssysteme diese Komponente des Wissens anders aus als objektbasierte. Im ersten Fall verwendet man Klauseln und Regeln, im zweiten dagegen über Attributwerte und Vererbung.

Bedingungen für ein Objekt:

Zu jeder Eigenschaft beziehungsweise Attribut eines Objektes lassen sich Prädikate anhängen, die immer dann in Aktion treten, wenn auf das Attribut zugegriffen. Man kann dabei zwischen Prädikaten unterscheiden, die in Aktion treten, wenn ein Attribut gelesen wird, also eine Informationsanfrage besteht (*if-Needed* Prädikat), oder solchen, die darauf warten, ob ein Attribut in seiner Ausprägung geändert wird, also neue Information abgelegt wird (*if-Added* Prädikat). Diese Prädikate stellen Boolesche Bedingungen dar, deren Zutreffen nicht verletzt werden darf. Erfolgt solch eine Verletzung, wird verhindert, daß auf das jeweilige Attribut zugegriffen werden kann (*if-Needed* Prädikat) oder ein Attribut modifiziert wird (*if-Added* Prädikat).

Diese Regeln steuern das Verhalten des Objektes. So kann man sie einsetzen, um den Wertebereich von Attributen zu begrenzen. Zum Beispiel kann für einen Menschen angegeben werden, daß sein Puls 300 Schläge in der Minute nicht überschreiten kann. Weiterhin können Regeln dazu dienen, bestimmte Integritätsforderungen zu wahren, beispielsweise, daß ein Mann nicht schwanger werden kann.

In Abbildung 2.1 ist ein allgemeines Muster für ein Objekt dargestellt:

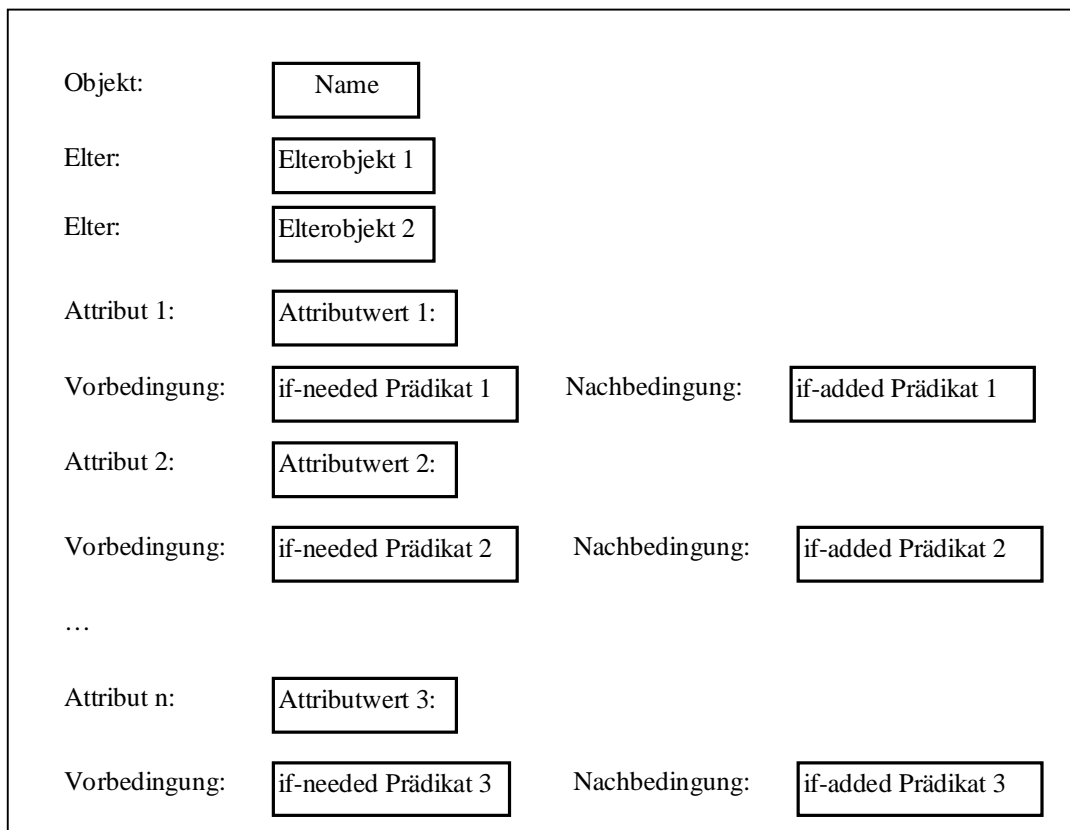


Abbildung 2.1: graphische Darstellung des Musters eines Objektes.

Nach dem in diesem Abschnitt entworfenen Modell von Wissen sollen im nächsten Kapitel konkrete Wissensrepräsentationsformalismen vorgestellt werden.

2.2 Arten der Wissensrepräsentation

Bei den in der KI-Forschung entwickelten Methoden unterscheidet man prinzipiell zwischen *deklarativer* und *prozeduraler* Repräsentation. Am Anfang der Forschung auf dem Gebiet der Wissensrepräsentation war die deklarative Repräsentation auf Grundlage der Logik die Standardmethode. Später entstand die prozedurale Repräsentation als Antwort auf die Probleme der deklarativen.

Als deklarativ bezeichnet man die Repräsentation von Wissen, wenn sie auf eine reine Beschreibung von Sachverhalten beschränkt ist und unabhängig von den Verfahren zur Anwendung dieses Wissens ausgerichtet ist. Wissen wird somit als eine Menge von Fakten angesehen. Diese beschreiben Elemente der Welt wie Objekte und Ereignisse, Zustände dieser Elemente, sowie Beziehungen zwischen den Elementen. Auf dieser statischen Wissensbasis operieren eine kleine Zahl von allgemeinen und von der Anwendung unabhängige Verfahren zur Bearbeitung dieser Fakten. Die Vorteile der deklarativen Darstellung liegen darin, daß jedes Wissensselement nur einmal gespeichert werden braucht und Wissen relativ einfach hinzugefügt, modifiziert oder entfernt werden kann. Der entscheidende Nachteil ist der, daß das Bearbeiten dieser Wissensbasen oft zu einer kombinatorischen Explosion führte.

Prozedural sind Wissensrepräsentationen, wenn das Wissen über die Anwendung von Wissen in der Wissensbasis im Vordergrund steht. Wissen manifestiert sich hier prozedural. Dies beruht auf der Annahme, daß spezielles Wissen dazu notwendig ist, um Wissen aus einem bestimmten Bereich sinnvoll anwenden zu können.

Die konsequenteste Realisierung des prozeduralen Schemas sind die ACTOR-Sprachen [Retti84]. Alle Elemente der Wissensbasis werden als Akteure aufgefaßt; es sind aktive Agenten, die ihre Rolle nach einem Skript spielen. Die Akteure können miteinander kommunizieren, indem sie Meldungen verschicken und empfangen. Diese Nachrichten sind wiederum selbst Akteure. In diesen Systemen besitzen die aktiven Akteure ihre eigene Steuerstruktur, werden also nicht von einer zentralen Instanz gesteuert.

Die beiden Aspekte, deklarative und prozedurale Wissensrepräsentation, schließen sich gegenseitig nicht aus, vielmehr ergänzen sie sich. In vielen Wissensrepräsentationssystemen sind beide Richtungen enthalten, wobei der eine oder andere Aspekt stärker dominiert.

Im folgenden sollen einige Darstellungsformen für die Wissensrepräsentation näher betrachtet werden.

2.3 Prädikatenlogik

Das ausgeprägteste Beispiel für die deklarative Repräsentation stellt die Prädikatenlogik erster Stufe dar. Sie ist die bekannteste, am meisten studierte und genutzte Logik. Elemente dieser Sprache sind Variable, Konstante, Funktionen und Prädikate, die mit Hilfe von logischen Verknüpfungsoperatoren („und“, „oder“, „nicht“, „wenn-so“ und „genau dann wenn“) und Quantoren (dem Allquantor und dem Existenzquantor) zu Ausdrücken verbunden werden. Prädikate sind Aussagen über Objekte und haben einen Wahrheitswert. Der Wahrheitsbegriff ist in der klassischen Logik durch zwei Motive gekennzeichnet: Der Dichotomie (der Wahrheitswert kann nur „wahr“ oder „falsch“ sein) und der Extensionalität (der Wahrheitswert einer zusammengesetzten Aussage hängt nur von den Wahrheitswerten der Teilaussagen ab). Die Prädikatenlogik ist nicht zuletzt deshalb so interessant, weil sie ein geschlossenes System darstellt, in dem formale Schlußmechanismen, unabhängig von der inhaltlichen Bedeutung der Aussagen zur Verfügung stehen. Eine ausführliche Vorstellung der Prädikatenlogik findet sich in [Richter89].

Aussagen über den zu darstellenden Anwendungsbereich werden in logische Formeln übersetzt. Diese Formeln werden als Axiome in das System aufgenommen.

Zum Beispiel könnte der Sachverhalt, daß alle Vögel Tiere sind, prädikatenlogisch wie folgt formuliert werden:

$$(\forall x)(VOGEL(x) \Rightarrow TIER(x))$$

Dabei stellt die Variable x ein bestimmtes Individuum der Vögel dar.

Die Wissensrepräsentation auf der Basis der Prädikatenlogik ist heute aber eher selten. Ihre Semantik ist gegenüber einer natürlichsprachlichen Ausdrucksweise sehr verarmt. Das hat dazu geführt, daß die Prädikatenlogik vielfältige Erweiterungen und Modifikationen erfahren hat:

- Quantoren, Funktionen und Relationen können sich auch auf Funktionen und Relationen beziehen (Prädikatenlogik zweiter Stufe).
- Die Dichotomie wird aufgegeben, zugunsten einer Menge von diskreten Wahrheitswerten (mehrwertige Logik), oder vollkommen durch einen stetigen Bereich dargestellt (Fuzzy Logik). Dadurch läßt sich vages Wissen leichter darstellen.
- Erweiterung um modallogischen Operatoren wie „möglicherweise“ und „notwendig“, um ein ganzes System denkbarer Situationen erfassen zu können (Modallogik).
- Einfügung zeitlicher Aspekte (temporale Logiken).
- Berücksichtigung, daß abgeleitetes Wissen beim Hinzufügen neuen Wissens seine Gültigkeit verlieren kann (nichtmonotone Logiken)
- ...

Es gibt eine Vielzahl von Wissensrepräsentationsprachen, die auf der Logik basieren. Unter diesen Sprachen hat vor allem Prolog Verbreitung gefunden. Prolog verwendet eine bestimmte Form von Prädikaten, die Hornklauseln. Sie lassen sich wie folgt darstellen:

$$L_1 \wedge L_2 \wedge \dots \wedge L_n \rightarrow L$$

wobei die L_i negierte oder unnegierte atomare Formeln sind. Eine Hornklausel hat die Eigenschaft, daß sie höchstens ein positives Literal besitzt.

Ein wesentliches Merkmal von Prolog ist, daß nicht ein Lösungsalgorithmus angegeben wird, sondern eine Problembeschreibung. Diese besteht aus einer Menge von Fakten und Regeln. Der Prolog-Interpreter bearbeitet die Wissensbasis, indem er versucht bestimmte Sachverhalte zu beweisen. Der Vorteil dieser deklarativen Sprache besteht darin, daß der Anwender dem Rechner nicht angeben braucht, „wie“ ein Problem zu lösen ist (wenigstens im Idealfall).

2.4 Produktionssysteme

Ein Wissensbasis eines Produktionssystems besteht aus zwei Komponenten[Behrendt90]:

- Einer Regelbasis, welche die Menge der Produktionsregeln enthält,
- sowie einer Faktenbasis, in der permanente oder temporäre Fakten gespeichert sind.

Besonders interessant ist in diesen Systemen die Regelbasis. In vielen Anwendungsgebieten liegt das Wissen schon fertig in Regeln oder regelähnlichen Formulierungen vor.

Die allgemeine Form einer Regel lautet:

$$\text{Wenn } P \text{ dann } Q$$

P und Q werden im allgemeinen als *Prämisse* und *Konklusion* bezeichnet. Eine Analogie zur Implikation in der Logik ist offensichtlich. In regelbasierten Systemen steht Q jedoch oft nicht für eine logische Aussage, sondern für eine Aktion. In diesen Fällen spricht man auch von *Produktionsregeln*. Die Regeln kann man folgendermaßen interpretieren: Wenn die Prämisse erfüllt ist, dann wird die Aktion ausgeführt. Sowohl der Bedingungs- als auch der Aktionsteil können beliebig komplex sein und beziehen sich letzten Endes auf das Wissen in der Faktenbasis. Da die Regeln angeben *wie* Lösungen ermittelt werden, sind Produktionssysteme ein Beispiel für die prozedurale Wissensrepräsentation.

Regeln sind eine verbreitete Repräsentationsform, die für viele Situationen angemessen und ausreichend ist. Sie lassen sich einfach handhaben und relativ einfach erweitern. Auf der anderen Seite haben die Regeln den Nachteil, daß sie sehr ähnlich aussehen und gleichförmig nebeneinanderstehen, aber unter Umständen Wissensstrukturen auf völlig unterschiedlichen Ebenen ausdrücken.

Ein bekannte Sprache für Produktionssysteme ist OPS5.

2.5 Semantische Netze

Eine der bekanntesten KI-Konzepte zur strukturierten Darstellung des Wissens ist das Semantische Netz. Es verfügt über zwei formale Konzepte, nämlich über Knoten und gerichtete Kanten. Damit ist es von der Struktur her, ein gerichteter Graph. Ein Knoten repräsentiert eine semantische Wissenseinheit. Das kann ein Objekt, ein Ereignis, ein abstrakter Begriff, ein Konzept u.a. sein. Die Kanten stellen eine binäre Relation zwischen zwei Knoten her. Im Grunde können das beliebige Beziehungen sein, verbreitet sind aber vor allem hierarchische wie weiter unten noch näher erläutert werden wird.

Das erste Semantische Netz wurde 1968 von M.R.Quillian entwickelt. Die Knoten dieses Netzes repräsentieren Wortkonzepte. Ein Beispiel für diese Art von Semantische Netz ist in Abbildung 2.2 zu sehen.

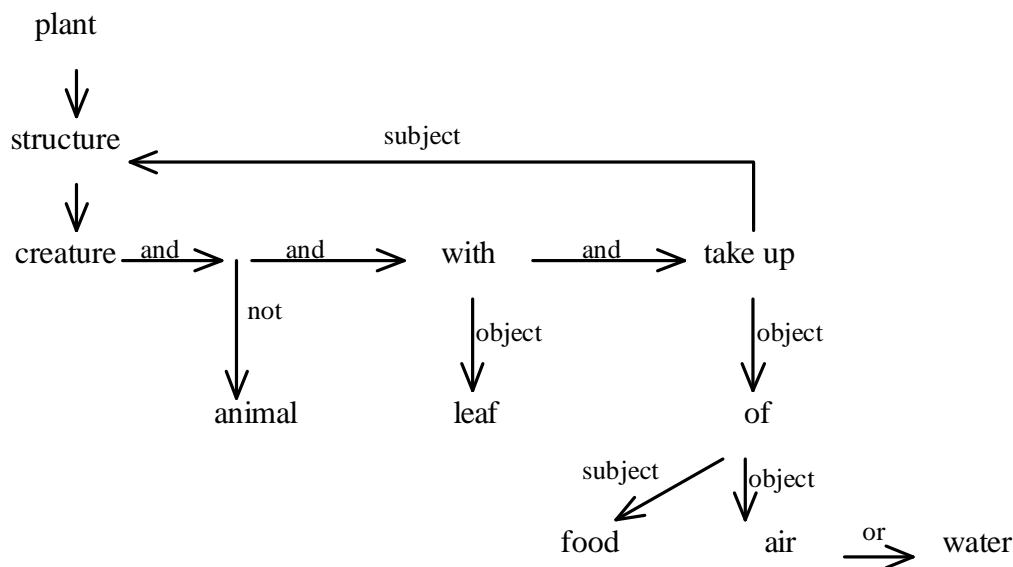


Abbildung 2.2: Das Wortkonzept für eine Pflanze nach Quillian.

Wie aus der Abbildung zu entnehmen ist, spielen logische Operatoren und Satzbauteile wie Subjekt und Objekt als Beziehungstypen eine wichtige Rolle. Die Knoten und Kanten, die der Beschreibung eines Wortkonzeptes dienen, sind zu einer Ebene zusammengefaßt. Die in einer Ebene referenzierten Konzepte sind durch Kanten ihrer eigenen Ebene verbunden. Dadurch muß jedes Wortkonzept nur einmal definiert werden (wie man es von der deklarativen Wissensrepräsentation auch erwartet). Semantische Netze wurden aufgrund dieser Konzeption vorwiegend im Bereich sprachverarbeitender Systeme eingesetzt.

Mit den von Quillian eingeführten Semantischen Netz konnte man das Umweltwissen beschreiben, um aber reale Objekte und Vorgänge in der Umwelt darzustellen, mußte man das Modell erweitern. Dazu wurden Individuen eingeführt. Diese stellen Instanzen bestimmter Konzepte dar. Eine weitere Erweiterung wurde durch den Knotentyp

Manifestation erreicht. Dadurch lassen sich mehrere zeitabhängige Ausprägungen eines Individuums ermöglichen. Man spricht in diesem Zusammenhang auch von episodischem Wissen, da das Auftreten eines Individuums in einem bestimmten Ereignis oder Episode dargestellt werden kann.

Je nach Anwendungs- beziehungsweise Aufgabengebiet erfuhren Semantische Netze immer neue Erweiterungen, gleichzeitig wurden die Netze immer unübersichtlicher. Außerdem benötigte man immer kompliziertere Methoden, um mit den Netzen sinnvoll arbeiten zu können.

Um diesen Nachteilen zu begegnen, wurde es nötig, den Aufbau Semantischer Netze neu zu überdenken. Verschiedene Ideen sind in [Retti84] aufgeführt. Einen interessanten Ansatz stellen die von Hendrix entwickelten Partionen dar. Vereinfacht gesagt geht es darum, zusammengehörige Bereiche des Netzes zu separieren. Die resultierenden Partitionen werden wiederum als neue abstraktere Knoten definiert, die mit anderen Knoten dieser Abstraktionsstufe über Kanten verbunden werden können. Im Grunde stellt es ein Modularisierungskonzept dar. Motiviert wurde diese Vorgehensweise auch durch neurophysiologische Erkenntnisse, die im menschlichen Gehirn ähnliche Strukturen entdeckten.

Wie weiter oben erwähnt gibt es in Semantischen Netzen Relationen, die besonders oft verwendet werden:

- Die *is a* Relation: Hierbei handelt es sich um eine Teilmengenbeziehung. Man spricht hier auch von einem Generalisierung- und Spezialisierungskonzept.
- Die Elementbeziehung (In Semantischen Netzen verwirrenderweise oft auch mit *is a* bezeichnet; besser ist die Verwendung von *instance of*, siehe dazu auch [Habel90]).
- Die Aggregation (*part-of*). Ein Element wird mit seinen Komponenten verbunden.
- Die durch die *connected-to* ausgedrückte allgemeine Beziehung zwischen zwei Knoten.

Siehe dazu auch die in Kapitel 2.1.1 vorgestellten Beziehungstypen.

Ein kleines Beispiel für ein typisches Semantisches Netz ist Abbildung 2.3 zu sehen.

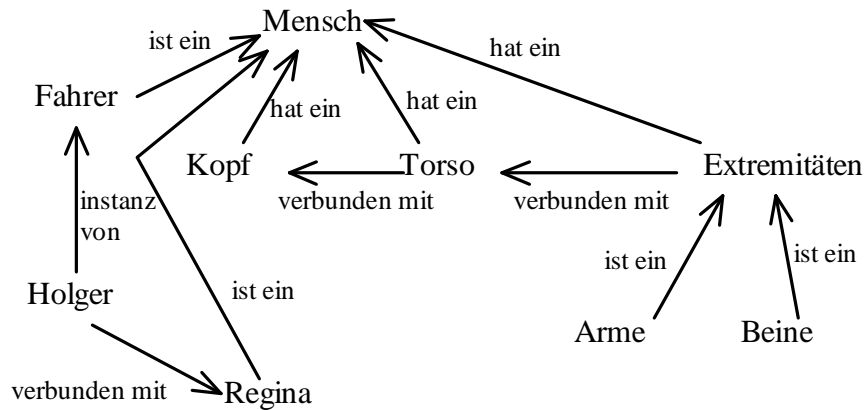


Abbildung 2.3: Beispiel für ein typisches Semantisches Netz.

Schon in diesem sehr einfachen Beispiel ist zu sehen, wie schnell Semantische Netze unübersichtlich werden. Neben der Unübersichtlichkeit haben Semantische Netze den Nachteil, daß ihre Ausdruckskraft eingeschränkt ist. Die Probleme beruhen dabei auf der Darstellung von Quantoren, logischer Operatoren wie der Negation und anderer als binärer Prädikate. Möglichkeiten, diese Nachteile zu umgehen, finden sich in [Lusti90] und [Richter89]. So können drei- und mehrstellige Beziehungen durch Einführung neuer Knoten und Reduktion auf binäre Relationen erreicht werden.

Ferner eignen sich Semantische Netze vor allem für Wissensgebiete mit einer stabilen Taxonomie. Wenn bereits Klassifikationsschemata und begriffliche Zuordnungen existieren, können diese relativ einfach abgebildet werden. Das es aber auch anders geht, zeigt das Aktive Semantische Netz in Kapitel (s. Kap. 3).

Die Semantischen Netze sind in erster Linie zur Repräsentation von Wissen gedacht, ohne die Möglichkeit wenigstens einige Manipulationen durchzuführen, wären sie wenig nutzbar. Das Manipulieren in semantischen Netzen geschieht mit Hilfe von Repräsentationssprachen wie der in Kapitel 4 vorgestellten KML.

2.6 Objektorientierte Wissensrepräsentation

Die objektorientierte Form der Wissensrepräsentation folgt einem Paradigma, welches unabhängig von den Forschungen in der Künstlichen Intelligenz entwickelt wurde. Dieses Paradigma beruht auf vier Hauptelementen [Booch94]:

- der Abstraktion
- der Kapselung
- der Modularität
- der Hierarchie

Die *Abstraktion* gibt die wesentlichen Charakteristika eines Objektes (sowohl statische als auch dynamische) an, die es von allen anderen Arten von Objekten unterscheidet und somit klare Abgrenzungen innerhalb der Problemstellung bietet. Der Prozeß der Abstraktion hängt wesentlich von der Perspektive des Betrachters ab. So sieht ein Tierarzt das Objekt Katze unter einem anderen Blickwinkel als ein Kind, das mit dieser Katze spielen möchte.

Die *Kapselung* ist ein Vorgang, in dem die während der Abstraktion ermittelten Elemente beziehungsweise Charakteristika nach Struktur und Verhalten getrennt werden. Sinn dieses Prozesses ist, die Schnittstelle eines Objektes und dessen Implementierung voneinander zu trennen. Abstraktion und Kapselung sind einander ergänzende Konzepte: die Abstraktion konzentriert sich auf das nach außen wahrnehmbare Verhalten eines Objektes, wohingegen die Kapselung die Realisierung des Verhaltens in den Vordergrund stellt.

Die *Modularität* ermöglicht das Zusammenfassen von den Eigenschaften eines Objektes. Im Gegensatz dazu werden die Informationen zu einer Abstraktion in einem logikbasierten System in einzelne Prädikate aufgegliedert und verringern das Verständnis für das in der Wissensbasis dargestellte Wissen. Eine objektorientierte Wissensbasis stellt eine Menge von in sich geschlossenen und lose gekoppelten Modulen dar. Das Konzept der Modularität hat auch zur Bildung von *Klassen* geführt, die ein abstraktes Grundmuster, nach dem Objekte erzeugt werden, darstellen. Durch die Instanziierung werden Objekte mit gleichen Eigenschaften erzeugt. Enthält ein System sehr viele Klassen können Klassen, die sich semantisch nahestehen, wiederum in Klassenkategorien zusammengefaßt werden. Die Bedeutung der Modularität liegt vor allem darin, die inhärente Komplexität vieler Problemstellungen in gewissen Maße zu reduzieren.

Das vierte Element, die *Hierarchie*, ist dasjenige daß Objekte von Variablen abstrakten Datentypen unterscheidet. Die Vererbung stellt eine Beziehung zwischen Klassen dar, in der eine Klasse die Struktur und das Verhalten einer (Einfachvererbung) oder mehrerer (Mehrfachvererbung) anderer Klassen übernimmt. Semantisch gesehen, legt die Vererbung also eine „is a“-Beziehung fest. Eine andere Art von Hierarchien stellen die Verbund-Beziehungen (auch part-of Beziehungen genannt) dar. Dadurch können Objekte höherer Abstraktionsstufen durch andere Objekte (mit)beschrieben werden.

Die objektorientierte Wissensrepräsentation gewinnt nicht zuletzt in modernen, nach diesem Paradigma ausgerichteten Datenbanken an Bedeutung [Heuer92]. Dadurch wird die Kluft, die zwischen Datenbanken und wissenbasierten Systemen wie Expertensysteme seit jeher existiert überbrückt [PetDel95].

Ein anderer aus der Künstlichen Intelligenz stammender Ansatz, der der objektorientierten Wissensdarstellung sehr ähnlich ist und in [Kurbel92] auch als objektorientiert bezeichnet wird, sind die von M. Minsky eingeführten *Frames*. Diese wurde ursprünglich zur Analyse von stereotypen Vorgängen („Szenen“) und Beschreibung visueller Informationen („Bilder“) verwendet. Ein Frame enthält die wesentlichen Elemente einer Szene sowie ihre Bedeutung zueinander. Minsky

charakterisiert Frames wie folgt: Ein Frame ist eine Darstellung stereotyper Situationen wie einer Szene in einem Zimmer. Mit jedem Frame sind verschiedene Arten von Informationen verbunden, z.B. Angaben darüber, wie der Frame benutzt werden kann, was als nächstes passieren wird usw.

In der objektorientierten Terminologie ausgedrückt, entspricht einem Frame ein Objekt oder eine Klasse. Die Charakteristika einer Abstraktion werden in Fächern, den *Slots*, festgehalten. Durch Instanzerzeugung aus einem Klassenframe erhält man einen weiteren Frames mit denselben Slots, wobei diese mit konkreten Einträgen („fillers“) gefüllt werden, sofern sie nicht mit *Defaultwerten* vorbesetzt sind. Der Slot akzeptiert dabei nur Elemente eines gewissen Wertebereichs. Ein Slot selbst kann wiederum strukturiert sein, seine Komponenten heißen *Fazetten*. Defaultwerte eines Slots sind vorläufige Werte („Arbeitshypothesen“), die revidiert werden können, falls nötig. *Generische Werte* eines Slots sind dagegen starre, unveränderliche Werte, die allen Instanzen gemein sind. *Slotbedingungen* schränken die Wertebereiche des Slots zusätzlich ein, zum Beispiel kann man festlegen, daß ein Slot nur Primzahlen enthalten soll.

In einem Slot können auch Prozeduren stehen, etwa Berechnungsvorschriften für einen Slot. *Dämonen* haben die Aufgabe, die Bedingungen zu überwachen, unter denen die Prozedur ausgeführt wird. Solche prozeduralen Zusätze können in mehreren Varianten auftreten. Die beiden wichtigsten sind:

die „if needed“-Prozedur: Hier werden Slotwerte nicht automatisch, sondern nur auf Anforderung hin berechnet.

die „if added“-Prozedur: die zur Ausführung gelangt, wenn eine neue Information in den Slot gelangt.

Die Verknüpfung eines Dämonen mit einem Slot wird so durchgeführt, daß die Bezeichnung des Dämonen in einer Fazette mit einem reservierten Namen gespeichert wird.

Durch diese prozeduralen Komponenten stellen die Frames eine Verbindung zwischen deklarativer und prozeduraler Wissensrepräsentation her(s. Kap. 2.2).

Die Struktur für einen Frame ist in folgender Abbildung zu sehen:

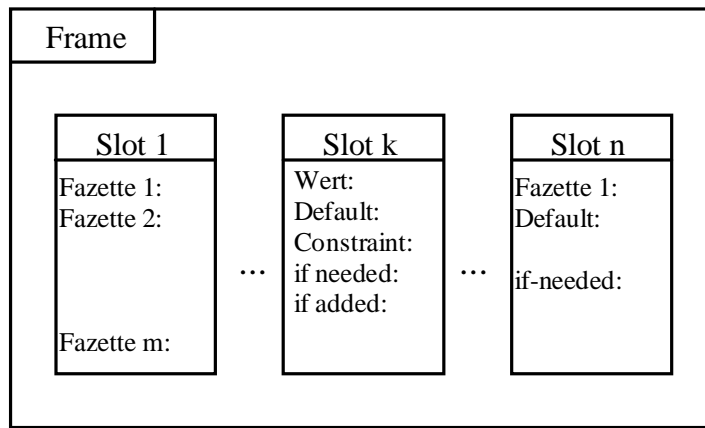


Abbildung 2.4: Aufbau eines Frames.

Komplexere und ausdrucksstärkere Darstellungen erreicht man, wenn ein Slot mit einem anderen Frame gefüllt wird (geschachtelte Frames) oder Frames über Kanten miteinander verbunden werden. Auf diese Weise erhält man ein *Frame-System*. Ein Frame-System ähnelt einem Semantischen Netz: Die Frames stellen die Knoten dar, die Verbindungen zwischen den Frames lassen sich in der Regel durch die in Kapitel 2.5 aufgeführten, häufigsten Relationentypen darstellen. Der wesentliche Unterschied zwischen Frame-Systemen und Semantischen Netzen ist der, daß die Knoten keine innere Struktur besitzen und die Netze einen überwiegend statischen Charakter haben. In Frames ist die Anzahl der Slots variabel, neue Werte und Beziehungen lassen sich mit Hilfe der Prozeduren berechnen. Im Prinzip lassen sich Frames unter Semantische Netze subsumieren [Richter89].

Ein Beispiel für ein Frame-System, welches Hotelzimmer für ein Hotel beschreibt, ist in Abbildung 2.5 aufgeführt. Der Frame „Hotelzimmer“ stellt einen Klassenframe dar, der zwei Unterklassen, nämlich „Einbettzimmer“ und „Zweibettzimmer“ besitzt. Die spezielleren Frames unterscheiden sich vom Wurzelframe, indem sie für den Slot „Preis“ einen Wertebereich vorgeben und die Prozeduren „Teste_Qualität“ und „Benachrichtige_Chef“ besitzen (nur „Einbettzimmer“), die von Dämonen unter bestimmten Bedingungen aktiviert werden. Der Frame „Zweibettzimmer“ setzt für die Kategorie als Defaultwert zwei Sterne fest.

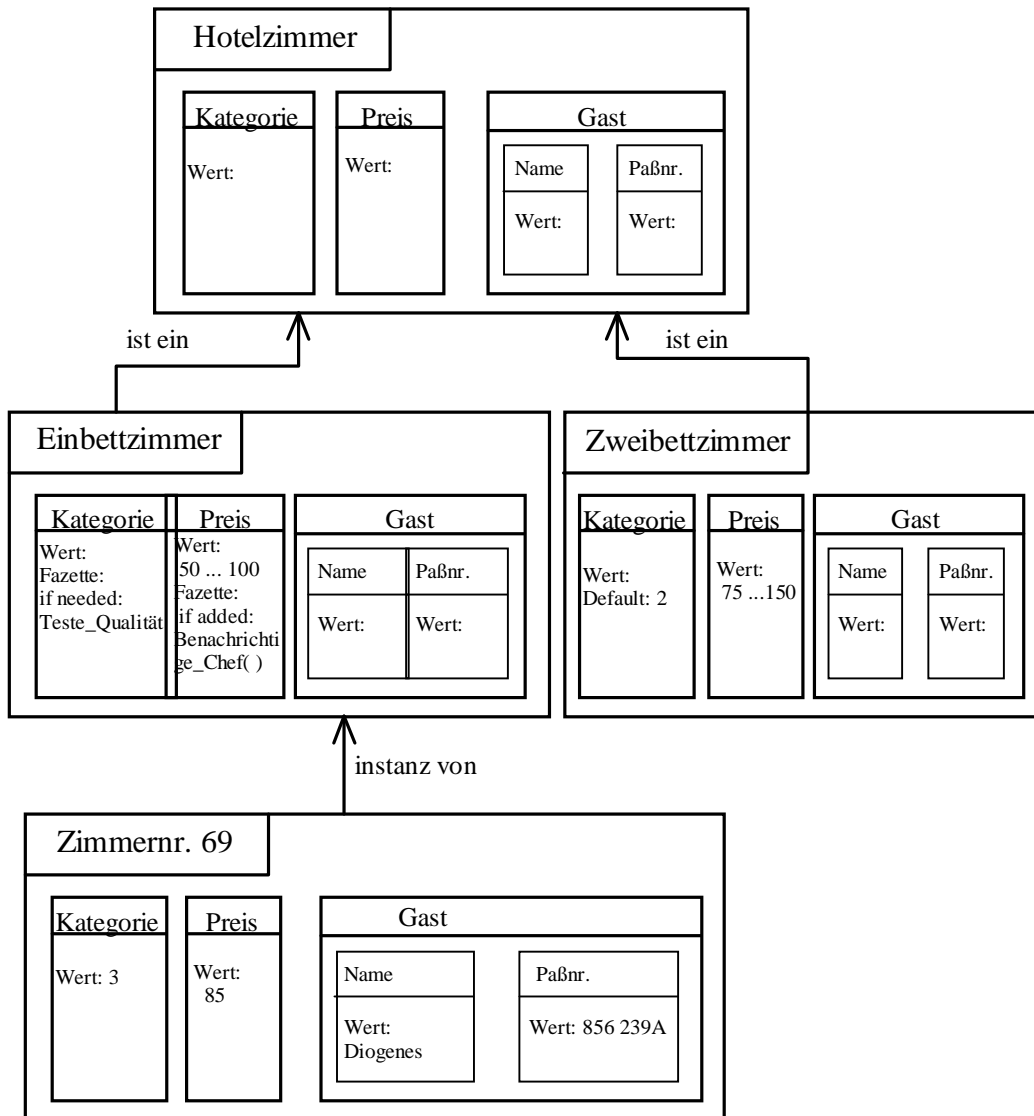


Abbildung 2.5: Beispiel für eine Frame-Struktur.

Der Frame „Hotelzimmer“ enthält den Slot „Gast“, der selbst einen Frame verkörpert (verschachtelter Frame). Eine Instanz eines Einbettzimmer ist das Zimmer mit der Nummer 69. Hier sind die Slots gefüllt

Frames bilden die Grundlage einer Reihe von Wissensrepräsentationsprachen. Die bekanntesten sind FRL und KRL. Eine Sprache, die eine Synthese von Frames und Semantischen Netzen darstellt, ist KL-ONE.

2.7 KL-ONE

Der große Nachteil von Frames und Semantischen Netzen besteht in der nicht formal spezifizierten Semantik und der daraus resultierenden Mannigfaltigkeit der verschiedenen

Typen dieser Wissensrepräsentationskonzepte. Die stark deklarative Sprache KL-ONE dagegen zeichnet sich durch seine klare Semantik aus und hat die Entwicklung zahlreicher auf KL-ONE basierender Termbeschreibungssprachen wie BACK (s. dazu Kap. 2.8), LOOM, KANDOR und KRYPTON zur Folge gehabt. Sprachen der KL-ONE Familie haben die wesentlichen Konzepte, die im weiteren vorgestellt werden, von KL-ONE übernommen.

KL-ONE wurde von Brachman mit der Intension entwickelt, allgemeines Wissen formal darstellen zu können. So verfügt KL-ONE über reichhaltige und umfassende Mechanismen für die Beschreibung und Definition von Wissen. Typisch für KL-ONE ist die Trennung von sogenannter *terminologischer* und *assertionaler* Komponente (TBox versus ABox), dabei steht die terminologische Komponente meist im Vordergrund. In der TBox werden Begriffe mit Hilfe anderer Begriffe beschrieben, wofür eine beschränkte Anzahl primitiver Operatoren zur Verfügung steht. Die Bildung solcher Begriffe sagt nichts über die Existenz entsprechender Elemente in der Welt aus. Die Verbindung zu konkreten Objekten wird im assertionalen Teil hergestellt, zu dem Funktionen des Zusicherns (Tell, Forget) und Suchens (Ask) stehen. Wegen dieser strikten Trennung von der Struktur der Sprache und den darzustellenden Inhalten wird KL-ONE auch als *hybrides* System bezeichnet.

KL-ONE wird in der Literatur häufig auch als objektorientiert [Retti84] bezeichnet, was aber irreführend ist, da es sich nicht um dieselbe Art der Objektabstraktion handelt wie in Kapitel 2.6 vorgestellt. [Scheffe91] bevorzugt die Bezeichnungen *termorientiert* oder *konzeptorientiert*.

Grundlegende Elemente von KL-ONE sind *Konzepte*. Konzepte sind im Grunde formale Objekte, welche inhaltlich gesehen Beschreibungen von Dingen der realen Welt darstellen. Ein Konzept soll einerseits die interne Struktur eines realen Objektes beschreiben und andererseits Relationen zu anderen Konzepten ermöglichen. Aus diesem Grunde hat ein Konzept Slots beziehungsweise Attribute, welche in der KL-ONE Terminologie *Rollen* (roles) heißen.

Es werden zwei Arten von Konzepten unterschieden:

- *Generische* Konzepte repräsentieren Klassen von Individuen und beschreiben charakteristische Eigenschaften eines prototypischen Elements dieser Klasse (siehe auch Defaults bei Frames).
- *Individuelle* Konzepte verkörpern spezielle Instanzen. Jedes individuelle Konzept wird genau einem generischen zugeordnet.

Konzepte sind in einer definatorischen Taxonomie zusammengefaßt, die eine hierarchische Ordnung der Konzepte darstellt. Es gibt nur ein Konzept (genannt „Thing“), welches keine übergeordnete Konzepte besitzt und damit die Wurzel der gesamten Hierarchie darstellt. Die anderen Konzepte verfügen über mindestens ein, in der Regel aber mehrere Superkonzepte. Dadurch wird eine Vererbungshierarchie

geschaffen, da alle untergeordneten Konzepte die Eigenschaften ihrer Superkonzepte erben und über diese definiert werden.

Da viele Begriffe der realen Welt, die nicht vollständig definiert werden können, wohl aber beschreibbar sind (vgl. die Begriffe „Vierfüßer“ und „Hund“), werden in KL-ONE neben den *definierten* noch *primitive* Konzepte verwendet. Die primitiven Konzepte werden besonders gekennzeichnet und es können ihnen keine Objekte eindeutig zugeordnet werden. Die Beschreibung der primitiven Konzepte gibt nur die notwendigen Eigenschaften von Instanzen dieses Konzeptes und nicht zusätzlich die hinreichenden wie bei den definierten Konzepten. [Scheffe91] spricht in diesem Zusammenhang auch von *formal-nominaler* und *natürlich-realer* Begriffsbildung. Formal-nominale Begriffsbildungen sind vor allem in Wissenschaften oder Disziplinen vorhanden, in denen formale Methoden verwendet werden.

Rollen beschreiben potentielle Beziehungen zwischen den Instanzen eines Konzeptes und anderen eng assoziierten Konzepten (z.B. Eigenschaften und Teile).

Eine Rolle hat nach [Richter89] folgenden Aufbau:

- eindeutiger Name der Rolle,
- Typ oder Beschreibung der Wertes, welche die Rolle ausfüllen können (Value Restriction),
- Anzahl derjenigen Dinge, welche eine Rolle füllen können (Number Restriction),
- Informationen darüber, ob ein Wert vorhanden ist.

Diesen Rollen entsprechen bei den individuellen Konzepten Wertknoten (genannt Iroles), die an die Rollen des generischen Konzeptes (RoleSet) gebunden werden.

Bei der Vererbung können die Rollen mit Hilfe der Inter-Role Relations auf unterschiedliche Weise abgeändert werden, etwa restriktiv, dadurch wird der RoleSet bei der Vererbung eingeschränkt oder differenziert, wobei eine RoleSet in mehrere aufgespalten wird (siehe dazu auch [Richter89]).

Zusätzlich gibt es strukturelle Beschreibungen, die Auskunft darüber geben, in welcher Beziehung die Füller der Rollen untereinander (etwa über RoleValueMaps) und wie sie zum Kontext als ganzes stehen (Verwendung von Parakonzepten).

Zu erwähnen ist noch, daß es Weiterentwicklungen gab, die neben der Trennung von Struktur (TBox) und Inhalt (ABox), Boxen für Inferenzen (IBox), für Wahrscheinlichkeiten (PBox), für Defaults (DBox), etc. vorsahen. Diese Trennung ist in der Künstlichen Intelligenz nicht unumstritten [Richter89].

Zusammenfassend läßt sich aber sagen, daß KL-ONE einen Standard in der Wissensrepräsentation gesetzt, an dem sich die meisten neueren Wissensrepräsentationsprachen ausgerichtet haben.

Als Beispiel für eine Termbeschreibungssprache, die auf KL-ONE basiert, wird hier *BACK* vorgestellt.

2.8 BACK

Das Entwicklungsprojekt BACK („Berlin Advanced Computational Knowledge representation system“) begann 1985 an der Technischen Universität Berlin. Ziel dieses Projektes war die Entwicklung einer zentrale Komponente für ein wissensbasiertes Systems, welche auf der Tradition und dem Paradigma von KL-ONE basiert. Hier wird auf die Version 3, die in [PeScKiQu89] beschrieben wird, eingegangen. Dabei wird auf Syntax- und Semantikangaben für die Sprachen BTL und BAL verzichtet. Lediglich Beispiele werden in der jeweiligen Sprache aufgeführt.

Aussagen mit definitorischem Charakter werden in der terminologischen Komponente (TBox) des Systems mit den Sprachmitteln von *BTL* (**BACK Term Language**) beschrieben. Wie in anderen Termbeschreibungssprachen wird auch in BTL ein Konzept durch die Angabe seiner Superkonzepte und einer Reihe von Restriktionen der Beziehungen zu anderen Konzepten dargestellt. Weiterhin werden in BTL Attributtypen, nämlich Attributmengen, Strings und Zahlen, verwendet, um Konzepte näher zu beschreiben. Die Ausprägungen dieser Typen werden als atomare, unstrukturierte Werte betrachtet. Instanzen der Attributtypen können ausschließlich als Rollenfüller verwendet werden. Das bedeutet, Rollen können Attributtypen für den Wertebereich nutzen, nicht aber für den Definitionsbereich. Attributmengen können beliebige atomare Elemente enthalten. Unter Verwendung des Vereinigungs-, Durchschnitts- und Ausschlußoperators können aus vorhandenen Mengen neue Mengen konstruiert werden.

In BACK wird zwischen definierten (**:=**) und primitiven (**:<**) Konzepten unterschieden (Schlüsselworte der Sprache sind fett geschrieben):

```
human :< animal
      and all (child, human).
```

```
person_with_fever := human
      and all (body_temperature, > 38).
```

Bei Rollen gibt es diese Unterscheidung nicht, diese gelten immer als primitiv. BTL unterscheidet zwischen drei Rollenrestriktionen:

- all-Restriktion: Dies stellt eine value-restriction dar. Als Füller für die dazugehörige Rolle sind nur Instanzen eines Konzeptes oder die Werte eines Attributtyps erlaubt.
- atmost-Restriktion: Dies stellt eine number-restriction dar: Die Anzahl der Beziehungen eines Konzeptes über diese Rolle zu anderen Konzepten wird nach oben beschränkt.
- atleast-Restriktion: wie atmost, nur daß hier eine untere Schranke angegeben wird.

Mehrere Beschränkungen einer Rolle können durch Konjunktion verbunden werden. So bedeutet die Angabe einer all-Restriktion und einer atleast-Restriktion für eine Rolle, daß insgesamt wenigstens die bei number-Restriktion angegebene Zahl von Instanzen, des durch die value-Restriktion angegebenen Konzeptes, als Rollenfüller verwendet werden dürfen.

In BTL kann der Sachverhalt, daß ein Vater, genau drei berühmte Chirurgeninnen als Töchter hat, von denen jede wenigstens ein Kind hat, folgendermaßen dargestellt werden:

male
and parent
and atleast (3, child)
and atmost (3,child)
and all (child, famous
 and female
 and surgeon
 and atleast (1, child))

Wie man aus diesem Beispiel erkennt, werden Konzepte durch Konjunktion anderer Konzepte, von denen einige Rollenrestiktoren, andere allgemeinere Konzepte sind, dargestellt. Genauso können Rollen durch Konjunktion einfacherer Rollen zusammengesetzt werden.

Das allgemeinste Konzept, welches alle anderen Konzepte subsumiert, ist *anything*. *Nothing* dagegen wird von allen anderen Konzepten subsumiert.

Die TBox von BACK wird aufgebaut, indem sukzessive Konzepte definiert werden. Dabei ist zu beachten, daß ein Konzept erst definiert werden muß, bevor es verwendet wird (zum Beispiel als Wertebereich für Rollen). Weiterhin ist es nicht möglich, vorhandenes Wissen nachträglich zu löschen oder zu revidieren.

Die ABox für das assertionales Wissen wird mit Hilfe von *BAL* beschrieben. *BAL* steht für The **BACK ABox Language**. Die Sprachkonstrukte gliedern sich in drei Teile:

- Erzeugung neuer Instanzen
- Angabe von Rollenfüller oder Attributwerte
- Anfragen (mit der ABox Query Language, AQL)

Bei der Konzeption von *BAL* wurde darauf Wert gelegt, eine syntaktische Ähnlichkeit (besonders bei AQL) mit SQL zu erreichen. Dadurch soll ein Vergleich zwischen *BAL* und SQL erleichtert werden. Die Ergebnisse von Anfragen (mit **getall**) werden in Listen aus Objekten der ABox angegeben. Darauf lassen sich ohne größeren Aufwand Mengenoperatoren anwenden, um speziellere Ergebnisse zu erhalten. Eine Anfrage besteht im wesentlichen aus einer generischen Konzeptbeschreibung der Objekte auf die zugegriffen werden soll sowie einer Folge von Restriktionen auf Rollenfüllern. Um zum

2 Wissensrepräsentation

Beispiel alle Institute zu ermitteln, die hans beschäftigen, wird folgende Anfrage angegeben (Es wird hier davon ausgegangen, daß „hans“ ein gültiger Objektidentifizierer ist) :

getall institute **with** employs: hans

Anfragen können überdies geschachtelt werden. Anwender können festlegen wie ausführlich die Ausgabe sein soll, ob beispielsweise nur der Name eines Objektes angegeben wird oder etwa die gesamte Beschreibung des Objektes aus der ABox.

Instanzen können jederzeit in die ABox eingefügt, gelöscht oder verändert werden.

3 Das Aktive Semantische Netz

In diesem Kapitel soll nun konkret die Konzeption und Realisierung der für das Rapid Prototyping (dieser Begriff wurde in Kap. 1 erklärt) entwickelten Wissensbasis vorgestellt werden. Dabei ist zu beachten, daß das Projekt „Rapid Prototyping – Entwicklung und Erprobung innovativer Produkte“ (siehe Kapitel 1) nicht abgeschlossen ist und sich auch die Wissensbasis noch in Entwicklung befindet. Diese Diplomarbeit konzentriert sich im wesentlichen auf den ersten Prototyp, der im Frühjahr 1996 fertig entwickelt wurde, wobei aber zukünftige Entwicklungen nicht unerwähnt bleiben sollen.

Bevor näher auf die Wissensbasis eingegangen wird, sollen die Anforderungen an diese konkretisiert werden.

3.1 Anforderungen an die Wissensbasis

Die Rapid Prototyping Wissensbasis beinhaltet alle für den Rapid Prototyping relevanten Informationen. Dazu gehört neben den (statischen) Produkt-, Technologie- und Prozeßdaten auch das (dynamische) Wissen in Form von Wirkketten, die die Abhängigkeiten zwischen technischen Produktfunktionen, Kosten, Qualitätsmerkmalen, Bearbeitungszeiten, Prototyp-Technologie und Kooperationsformen abbilden.

Konkrete Anforderungen an die Wissensbasis sind in [Eck96] und [Friedrich96] enthalten:

- **Online-Dialogfähigkeit:** Anfragen an die Wissensbasis müssen umgehend beantwortet werden.
- **Erweiterbarkeit:** Das Aktive Semantische Netz muß einfach erweiterbar und flexibel anpaßbar, bzgl. Veränderungen während des Entwicklungsprozesses, gestaltet werden. Diese Anforderung wird von heutigen Produktmodellen nicht berücksichtigt.
- **Ausdrucksfähigkeit:** Das Modell, und damit die gewählte Repräsentationsform, muß ein breites Spektrum verschiedener semantischer Beziehungen und kausaler Abhängigkeiten darstellen und verwalten können.
- **Kooperatives Bearbeiten:** Bei Auftreten von Problemen, muß Informationsaustausch zwischen Entwicklern und anderen Fachexperten ermöglicht werden, um so eine geeignete Lösung finden zu können. Dazu muß eine Kommunikationsmöglichkeit von der Wissensbasis zur Verfügung gestellt werden.
- **Dynamik:** Rapid-Prototyping Prozesse zeichnen sich durch eine rasche Änderung bzw. Revision von Entscheidungen aus. Dies erfordert den Umgang mit unsicherem, unvollständigem, widersprüchlichem ungenauem sowie defaultmäßigem Wissen.

- **Robustheit:** Das zu erstellende Modell muß sich als robust gegenüber kleinen Veränderungen innerhalb der Wissensbasis erweisen. Inkrementelle Veränderungen dürfen nur in Ausnahmefällen zu extremen Auswirkungen führen. Dabei ist zu berücksichtigen, daß die Kooperationsmodelle vom Zustand der Wissensbasis beeinflußt werden können.
- **Versionsverwaltung:** Bedingt durch die Dynamik des Gesamtprozesses ist die Modellierung temporaler Abhängigkeiten in zweifacher Hinsicht notwendig: zur Dokumentation des aktuellen Entwicklungsstandes sowie der Erfahrung früherer (auch fehlgeschlagener) Versuchsreihen. Damit übernimmt die Versionsverwaltung das Management der Projekthistorie.
- **Transparenz:** Eine Modellierung des Wissens und der semantischen Beziehungen (z.B. Kausalbeziehungen) muß derart erfolgen, daß für alle Mitglieder eines Entwicklungsteams der aktuelle Entwicklungs- und Wissensstand einfach abrufbar ist.
- **Darstellung unvollständigen Wissens:** Die Wissensbasis muß Konstrukte enthalten, aus denen klar ersichtlich ist, ob eine gespeicherte Information vollständig ist.
- **Zugriffskontrollmöglichkeiten:** Die Verwaltung verschiedener Zugriffsrechte ist zu berücksichtigen.

Die Erweiterbarkeit nimmt innerhalb eines Produktmodells eine besondere Rolle ein. Da die Modellinhalte im Laufe des Produktentwicklungsprozesses entstehen, muß die Wissensbasis inkrementellen Wissenszuwachs verarbeiten können. Weiterhin treten innerhalb eines Rapid-Prototyping Prozesses häufig unvorhergesehene Ereignisse auf, die es erforderlich machen, daß Veränderungen am Produkt durchgeführt werden müssen. Ein solches Ereignis könnte z.B. die Erkenntnis sein, daß der Prototyp eines Produktes die erwarteten Kosten überschreitet und aus diesem Grunde kostengünstigere Teile verwendet werden müssen. Vorhandenes Wissen in der Wissensbasis muß demnach auch wieder geändert und gegebenenfalls gelöscht werden können. Dabei ist zu achten, daß keine Inkonsistenzen innerhalb der Wissensbasis auftreten.

Hier stößt man auch an die Grenzen von Datenbanksystemen. Diese sind nur bedingt, wenn überhaupt, in der Lage nachträglich in einer Anwendung Schemaänderungen durchzuführen. Deswegen muß der Problembereich intensiv studiert und der Aufbau des Datenmodells einer Datenbank sehr sorgfältig geplant werden, bevor mit der Implementierung der Datenbank begonnen werden kann. Dies widerspricht aber dem Grundgedanken des Rapid Prototyping schnell lauffähige Prototypen erstellen zu können, was andererseits sehr flexible und dynamische Systeme erfordert, die diese Organisationsform unterstützen.

Bei näherer Betrachtung der Anforderungen wird ersichtlich, daß die reine deklarative Modellierung des Wissens nicht ausreicht, um diese zu erfüllen. Es wird zunehmend eine aktive Komponente benötigt, die das prozedurale Wissen abdeckt. Die aktive Komponente hat u.a. die Aufgaben, Inferenzen auf der Wissensbasis zu ziehen oder

Mitarbeiter zu benachrichtigen, wenn bestimmte Ereignisse eingetreten sind. In vielen wissensbasierten Systemen wird dies so realisiert, daß neben der Wissensbasis eine eigenständige Inferenzkomponente existiert, eventuell auch ein Managementsystem für die Verwaltung und Koordination der einzelnen Komponenten. Im Aktiven Semantischen Netz wurde aber ein anderer Weg eingeschlagen wie im nächsten Unterkapitel gezeigt wird.

3.2 Die Wissensbasis

Die Rapid Prototyping Wissensbasis wird in Form eines Aktiven Semantischen Netz, kurz dem ASN, realisiert. Wie in Semantischen Netzen üblich, sind im ASN die grundlegenden Konstrukte Knoten und Kanten. Um die Überschaubarkeit des Semantischen Netzes zu gewährleisten, wurde zum einen die Anzahl der verschiedenen Knoten- und Kantentypen niedrig gehalten, eine Trennung in terminologisches und assertionales Wissen durchgeführt sowie bestimmte Informationen in die Knoten selbst eingebettet. Wie ersichtlich wird, wurde das Modell des Semantischen Netzes um Konzepte aus KL-ONE (terminologisches und assertionales Wissen) und Frames erweitert. In Framesystemen stellt bekanntlich jeder Knoten einen Frame dar, der Wissen im Knoten eingekapselt hat. Eine weitere Gemeinsamkeit mit Frames ist in der Vermischung von prozeduralem und deklarativen Wissen (siehe Kap. 2.2) wie in Kapitel 3.2.5 noch näher erläutert wird. Das das ASN trotzdem als Semantisches Netz bezeichnet wird, liegt daran, daß der Hauptanteil des Wissens über Beziehungen zwischen den Knoten verkörpert wird.

Es gibt vier verschiedene Typen von Knoten, die eine der folgenden Bedeutungen verkörpern:

- Konzept
- Instanz
- Slot
- Slotwert

Die Kantentypen werden bei der Vorstellung der einzelnen Knotentypen sukzessive eingeführt (eine Zusammenfassung aller Relationentypen findet sich in Tabelle 3-5).

Die Knoten lassen sich in solche unterteilen, die terminologisches und solche die assertionales Wissen repräsentieren. Damit wurde die Unterteilung aus KL-ONE (Kap. 2.7) übernommen. Terminologisches Wissen, also solches, das der Beschreibung und Definition beliebiger Begriffe dient, stellen die Konzepte und Slots dar, assertionales, welches reale Objekte und ihre Beziehungen darstellt, dagegen die Instanzen und Slotwerte. Dieser Sachverhalt ist in Abbildung 3.1 graphisch beschrieben.

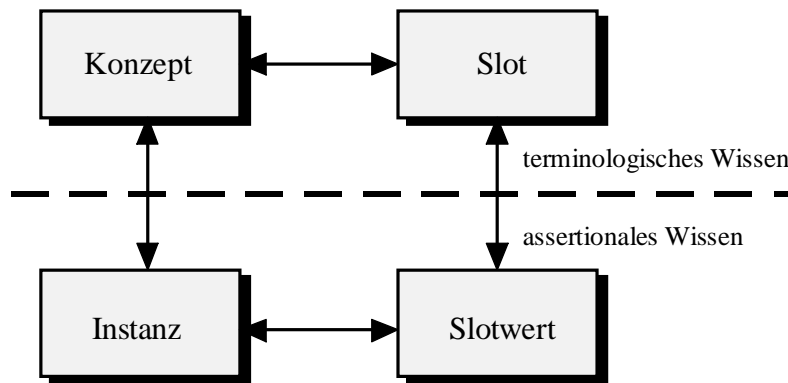


Abbildung 3.1: Unterteilung in assertionale und terminologische Knoten

In den nachfolgenden Kapiteln werden die einzelnen Knotentypen des ASN, ihre Bedeutung und deren mögliche Beziehungen zu anderen Knoten verdeutlicht.

3.2.1 Konzepte

Ein Konzept repräsentiert einen Begriff aus einem Anwendungs- oder Problembereich, das durch das ASN modelliert werden soll. Jedes Konzept wird eindeutig durch den ihm zugewiesenen Namen bestimmt (unique name assumption), das heißt es können nicht zwei verschiedene Konzepte mit demselben Namen in der Wissensbasis auftauchen.

Die Fähigkeit, Dinge durch Abstraktionen sinnvoll zu beschreiben, und damit die inhärente Komplexität vieler Problembereiche zu bewältigen, führt zu zwei Arten der Abstrahierung:

- Der Teile/Ganzes-Beziehung (has-a) und
- die Spezialisierungsbeziehung (is a)

Zu einem Konzept kann es speziellere Konzepte geben, die Unterbegriffe dieses Konzeptes sind. So ist beispielsweise ein Konzept Student ein Unterbegriff oder Unterkonzept des Konzeptes Mensch (Oberkonzept). Eine Konsequenz der Spezialisierungsbeziehung ist die Vererbung aller Eigenschaften der Oberkonzepte an die Unterkonzepte: Ein Mensch hat Eltern, also hat auch ein Student Eltern.

Die has-a Beziehung faßt mehrere Einzelkonzepte zu einem neuen Konzept zusammen. Dadurch braucht ein neues Konzept nicht vollständig neu definiert werden, sondern kann mit Hilfe anderer bereits existierender Konzepte beschrieben werden.

Im ASN sind einige Konzepte vordefiniert (*elementare* Konzepte genannt). Diese Konzepte können nicht erweitert werden oder verändert werden. Es ist auch nicht möglich sie zu spezialisieren. Sie können lediglich in der Teile/Ganzes Beziehung verwendet werden und zwar als Teil. Ihre Bedeutung liegt darin, als Grundbausteine zu fungieren, um komplexere Konzepte zu generieren (Es gibt kein System, welches ohne vorgegebene Bausteine auskommt).

Die elementaren Konzepte sind:

- *Text*: Dieses Konzept stellt eine beliebige Zeichenkette dar.
- *Integer*: Damit sollen ganze Zahlen beschrieben werden.
- *Float*: Erweitert das Integerkonzept um rationale Zahlen
- *Boolean*: Boolean drückt einen der beiden Wahrheitwerte aus: Wahr oder Falsch.
- *Opaque*: Kann eine beliebige Datei sein (z.B. eine Bild- oder CAD-Datei), die in der Wissensbasis abgelegt oder auf die referenziert wird.

Die andere Art von Konzepten sind die *nicht elementaren* Konzepte. Sie werden vom Anwender erzeugt und dienen der Modellierung der Anwendung.

Neben dem Namen kann ein Konzeptknoten einen Kommentar über sich erhalten. Dieser Kommentar enthält natürlichsprachliche Informationen zu dem Konzept, die vom Benutzer eingegeben werden. Kommentare können zu Konzepten, Instanzen und Slots angegeben werden. Das Ziel ist, allen Benutzern das gleiche Verständnis für den jeweiligen Knoten zu vermitteln.

Das oben geschilderte noch einmal zusammengefaßt: ein Konzeptknoten trägt einen eindeutigen Namen, der einen Begriff aus dem Anwendungsbereich beschreibt, einen optionalen Kommentar zu dem Konzept. Von diesem Knoten gehen zwei Arten von Beziehungen aus, nämlich Spezialisierungs- und Teile/Ganzes-Beziehungen. In der Tabelle 3-1 wird die innere Struktur (die Elemente der Struktur werden *Knotenelemente* genannt) und die Beziehungen, die von diesem Knoten ausgehen können aufgezeigt.

Knotenart	Knotenelemente	semantische Beziehungen
Konzeptknoten	eindeutiger Name	is-a
	Kommentar (optional)	has-a

Tabelle 3-1: Die Struktur und die Beziehungen eines Konzeptknotens.

Wie später noch zu sehen ist, kann es zu einem Konzeptknoten noch andere Beziehungen geben, diese gehen aber nicht von einem Konzeptknoten aus, sondern verweisen lediglich auf ihn. Wie in Kapitel 2.5 erwähnt wurde, sind die Kanten in einem Semantischen Netz gerichtet.

Graphisch wird im Rahmen der Diplomarbeit ein Konzeptknoten durch ein Viereck dargestellt. In den Knoten steht der Konzeptname, auf Kommentare wird in den Beispielen nicht näher eingegangen.

An einem einfachen Beispiel soll das in diesem Unterkapitel geschilderte erläutert werden. Das Beispiel wird nach der Einführung der anderen Knotentypen des ASN schrittweise ausgebaut.

3 Das Aktive Semantische Netz

Das fiktive Beispiel beschreibt ein Projekt an einer Hochschule, indem eine leistungsfähige Videokarte für einen neuen Multimediacomputer entwickelt werden soll. Die am Projekt beteiligten Personen (Mitarbeiter) sollen im ASN ebenfalls modelliert werden. Die Personen können Angestellte sein oder Studenten oder Hiwi. Es bietet sich an das Projekt, die Videokarte, ihre Bauteile wie Bildspeicher und Graphik-Controller, sowie die Mitarbeiter wie Angestellte, Studenten oder Hiwis als Konzepte zu modellieren.

Die Modellierung erhebt keinen Anspruch auf Vollständigkeit, es gibt sicher auch bessere Möglichkeiten das Wissen zu strukturieren. Die Intension des Beispiels ist, die diversen Konstrukte des ASN vorzustellen.

In Abbildung 3.2 wurde die is-a Beziehungen, die zwischen einigen Konzepten stehen aufgeführt, nicht aber die has-a Beziehungen, die erst nach Einführung der Slotknoten verwendet werden.

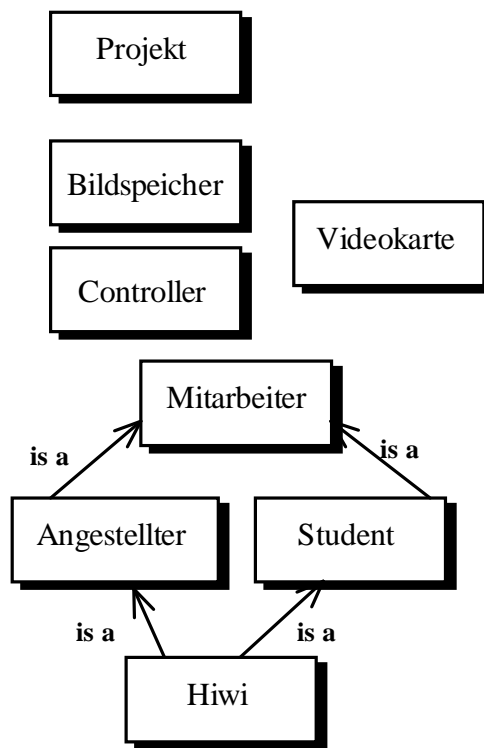


Abbildung 3.2: Beispiel für Konzeptknoten.

Die Konzepte Projekt, Videokarte, Bildspeicher, Controller und Mitarbeiter besitzen keine Oberkonzepte. Anders als in KL-ONE gibt es kein Konzept (*thing* genannt), von dem alle anderen Konzepte abgeleitet werden. Das Aktive Semantische Netz kann demnach eine beliebige Zahl von Vererbungshierarchien enthalten.

3.2.2 Slots

Wie oben erwähnt, gibt es für Konzepte zwei Arten der Abstrahierung. Neben der Spezialisierungsbeziehung existiert noch die Teile/Ganzes Beziehung. Die Teile eines Konzeptes sind selbst elementare oder nicht elementare Konzepte. Die Verbindung zwischen zwei Konzepten wird im ASN aber nicht direkt durch eine Kante zwischen den beteiligten Konzepten erreicht, sondern über einen weiteren Knoten, den Slotknoten. Das mag auf den ersten Blick umständlich erscheinen, verhindert aber eine gewaltige Zunahme an Relationentypen und das ASN wird dadurch flexibler und leichter zu erweitern. Würde man nämlich zwei Konzeptknoten wie Videokarte und Controller über eine has-a Kante verbinden, dann kann der Knoten, der das Teil verkörpert, nämlich der Controller, kein eingekapseltes Wissen (d.h. im Knoten abgelegtes Wissen), das die has-a Beziehung näher beschreibt, beinhalten, denn dieser Knoten kann gleichzeitig auch Teil eines ganz anderen Konzeptes sein: Einen Graphik-Controller gibt es auch in Graphik- und Kinokarten. Man müßte demzufolge die Semantik der has-a Beziehung mit weiteren Kanten beschreiben, zum Beispiel die Angabe, ob das „Ganzes“-Konzept mehrere dieser Teile besitzt oder nur eines, ob das „Teile“-Konzept physikalisch im „Ganze“-Konzept enthalten ist oder nicht.

Mit Hilfe eines Slotknoten wird eine has-a Beziehung ganz individuell zwischen zwei Konzepten beschrieben und kann damit einen Namen erhalten. Die Slotknoten entsprechen den Slots in Frames (Kap. 2.6). Der Unterschied zwischen einem Frame-System und dem ASN wird hier ganz deutlich. In einem Frame-System sind die Slots Elemente des Frameknotens, im ASN dagegen wird ein Slot durch einen eigenen Knoten realisiert.

Der Slotknoten enthält Wissen über die mögliche Kardinalität einer has-a Beziehung. Damit kann ausgedrückt werden, ob ein Konzept einen Slot genau einmal oder aber viele dieser Slots besitzt. Beispielsweise kann es in einer Fakultät viele Studenten geben, aber es gibt nur einen Dekan. Der Sinn das Wissen über die Kardinalität des Slots im Slotknoten zu bewahren ist, daß es so nahezu keinen Aufwand macht eine Vielzahl neuer Kardinalitätstypen einzuführen (außer den momentan vorhandenen *single* und *multiple* Typen).

Durch das Vorhandensein von Slotknoten kann jeder has-a Beziehung ein Namen zugewiesen werden, der innerhalb einer Konzeptvererbungshierarchie eindeutig sein muß. Durch diese Festlegung wird gewährleistet, daß jede Instanz eines Unterkonzeptes auch als Instanz jedes Oberkonzeptes betrachtet werden kann und die Semantik der is-a Beziehung nicht untergraben wird. Ansonsten würde durch das Neudefinieren (Overriding) eines Slots in einem Unterkonzept, den gleichnamige Slot in einem Oberkonzept außer Kraft setzen.

Wie bei den Konzeptknoten kann auch jedem Slotknoten ein Kommentar seitens der Anwender zugewiesen werden.

Weiterhin beinhalten Slots, die elementaren Konzepten angehören, den Typ des Konzeptes innerhalb des Knotens. Das widerspricht zwar der Intention Konzepte durch Konzeptknoten darzustellen, erhöht aber die Übersicht im Aktiven Semantischen Netz ungemein. Diese elementaren Konzepte stehen nämlich sooft in Kontakt mit anderen Konzepten, das ein großer Teil der Kanten im Semantischen Netz zu diesen Konzepten führen würde. Diese elementaren Konzepte haben außerdem eine so niedrige Aussagekraft was die Modellierung komplexerer Anwendungen betrifft, daß es ganz sinnvoll erscheint sie in Slotknoten zu kapseln und auch ihre Instanzen nicht explizit als Instanzknoten (s. Kap. 3.2.3) aufzuführen.

Da beim Rapid Prototyping eine Vielzahl unterschiedlichster und großer Datenmengen anfallen können (Bilder, Dokumente, Audiodateien, CAD-Dateien, ...) und auf diese in der Wissensbasis Bezug genommen werden muß, kann ein Slot auch eine Verbindung zu solch einer Datei herstellen. Dazu wurde das Opaquekonzept (dt. undurchsichtig) eingeführt. Im Slotknoten wird angeführt, ob die Datei innerhalb der Wissensbasis vorhanden ist, oder ob nur eine Referenz in Form einer URL (siehe Kap. 4.2) vorliegt. Im Gegensatz zu den anderen elementaren Konzepten, werden Instanzen dieses Konzeptes durch Instanzknoten (die in diesem Fall letztendlich Dateien sind) im ASN dargestellt, sofern die Dateien auch wirklich innerhalb der Wissensbasis gespeichert werden.

Nachdem die innere Struktur eines Slotknotens beschrieben wurde, sollen nun die semantischen Beziehungen, die von den Slotknoten ausgehen, erklärt werden.

Wie bereits erwähnt, stellen die Slots die Verbindung zwischen zwei Konzepten einer Teile/Ganzes Abstraktion fest. Dabei zeigt die has-a Kante von dem „Ganze“-Konzept zum Slot. Vom Slot wird die Verbindung zum „Teile“-Konzept über zwei mögliche Relationentypen festgelegt. Zwei deswegen, weil dadurch geklärt wird, ob ein „Teil“-Konzept auch physikalisch in dem „Ganzen“-Konzept enthalten ist (*is-part* Kante) oder nicht (*relation* Kante). So befindet sich der Bildspeicher physikalisch in der Videokarte, ein Projekt hat ein Produkt als Ziel, dieses Produkt kann aber nicht in einem Produkt physikalisch enthalten sein (ein Projekt ist schließlich ein Begriff, nach Platons Ideenlehre sogar real existierend, aber nicht materiell). Gerade im Bereich der Produktentwicklung ist eine Unterscheidung der beiden Fälle von Bedeutung. Der wichtigste Unterschied zwischen den beiden Beziehungstypen ist der, daß bei part-of Beziehungen, die Lebensdauer eines Teilobjektes von der Lebensdauer des dieses Teilobjekt beinhaltenden Objektes abhängt. Bei der relation-Beziehung ist das nicht der Fall. Wenn eine Firma geschlossen wird, existieren die ehemaligen Mitarbeiter trotzdem weiter, wird dagegen ein Auto verschrottet, werden auch der Motor, die Sitze und andere Teile des Autos zerstört, sofern sie vorher nicht explizit aus dem Auto entfernt wurden beziehungsweise diese Art der Beziehung im ASN nicht gelöst wurde. Ein weiterer Unterschied ist, daß ein bestimmtes Teil, das in einer part of Beziehung zu einem anderen Objekt steht, eben nur mit diesem einen Objekt verbunden sein kann und sonst mit keinem.

In nachfolgender Tabelle 3-2 wird das eben Geschilderte noch einmal zusammengefaßt.

Knotenart	Knotenelemente	semantische Beziehungen
Slotknoten	eindeutiger Name Kommentar (optional) ggf. elementares Konzept Kardinalität	relation part-of

Tabelle 3-2: Die Struktur und die Beziehungen eines Slotknotens.

Das Beispiel aus Abbildung 3.2 kann nun erweitert werden. An einem Projekt sind mehrere Mitarbeiter beteiligt, die Mitarbeiter haben jeweils einen Namen. Das Produkt, das in dem Projekt realisiert wird, ist eine Videokarte, aus Bauteilen wie einem Controller und einem Bildspeicher besteht. Von jedem Angestellten liegt ein Photo in digitaler Form vor, damit sich die Mitarbeiter von den Angestellten ein Bild machen können. Die Photodateien befinden sich aber nicht innerhalb der Wissensbasis. Dieses zusätzliche Wissen wird im ASN wie in Abbildung 3.3 gezeigt.

Die Slotknoten werden durch gestrichelte Rechtecke symbolisiert. In den Knoten steht der Slotname, die Kardinalität und falls nötig, der Typ des elementaren Konzeptes (auch hier soll dem Element Kommentar keine weitere Bedeutung zugemessen werden).

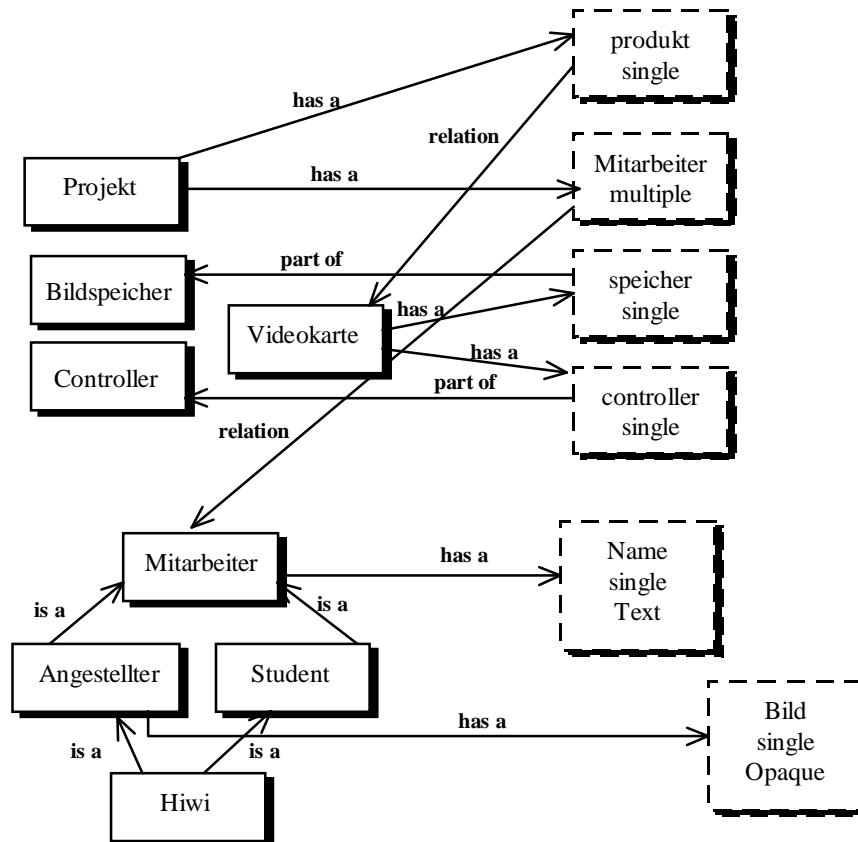


Abbildung 3.3: Beispiel für Slotknoten.

Die Konstrukte Konzept und Slot repräsentieren die abstrakten Begriffe, das terminologische Wissen. Um die im terminologischen Wissen eingeführten Konzepte und Slots mit konkreten Individuen zu belegen, werden die Konstrukte Instanz und Slotwert benötigt:

3.2.3 Instanzen

Eine Instanz verkörpert eine konkrete Verwirklichung eines Konzeptes, ein reales Objekt des Anwendungsbereich. *Objekt* und *Instanz* werden hier synonym verwendet. Zu einem Konzept kann es beliebig viele Instanzen geben. Wird von einem Konzept kein Objekt instantiiert, wird das Konzept als *abstrakt* bezeichnet. Im Beispiel aus Abbildung 3.2 und Abbildung 3.3 wurde der Begriff Hiwi beschrieben ohne auf einen real existierenden Hiwi Bezug zu nehmen. Durch Instanziierung eines Konzeptes kann ein bestimmter Hiwi der zu modellierenden Welt erzeugt werden.

Im ASN kann ein Objekt (vorerst) nur einem Konzept angehören. Dies ist auch in den meisten objektbasierten und objektorientierten Wissensrepräsentationssystemen der Fall. Diese Festlegung ist aber nicht zwingend. Es gibt nach [Booch94] drei Ansätze für die Klassifizierung, nämlich die klassische (die schon auf Plato zurückgeht), die konzeptuelle und prototypische. Die Einteilung der Objekte in disjunkte Gruppen ist zwar populär, jedoch hat schon Wittgenstein gezeigt, daß es meistens keine gemeinsamen Eigenschaften gibt, die für alle Objekte einer Gruppe oder eines Konzeptes zutreffen. Nicht alle Vögel können fliegen, andererseits legen alle Vögel Eier, aber das tun auch alle Reptilien. Man wird keine Eigenschaften finden, die nur für Vögel gelten.

Die konzeptuelle Zuordnung berücksichtigt diesen Sachverhalt und ermöglicht, daß ein Objekt zu mehreren Gruppen gehört. Dieses Verfahren ist eng verwandt mit der unscharfen Mengentheorie, wobei Elemente zu einer oder mehreren Gruppen gehören können. Die konzeptuelle Zuordnung stellt im Grunde eine Wahrscheinlichkeitsklassifizierung dar. Ein Tier wäre demnach zu einem bestimmten Grad ein Vogel.

Könnte eine Instanz im ASN mehreren Konzepten angehören wie es auch geplant ist, wäre die konzeptuelle Klassifizierung möglich.

Jede Instanz wird durch einen Namen eindeutig identifiziert. Der Name muß nicht nur innerhalb des Kontextes eines Konzeptes unmißverständlich sein, sondern innerhalb der gesamten Wissensbasis wie das auch bei Konzeptnamen der Fall ist.

Ein Instanzknoten enthält den Identifikator des Objekts und optional einen Kommentar zu dem Objekt.

Da eine Instanz immer einem Konzept zugeordnet ist, gibt es die *instance-of* Kante, die Instanzknoten mit einem (später auch mehreren) Konzeptknoten verbindet. Der zweite Relationentyp, der von einem Instanzknoten ausgeht, ist die *value* Kante. Diese zeigt auf

einen Slotwertknoten, einer Instanziierung eines Slots. Über die value Kanten wird eine Verbindung zu den tatsächlichen Eigenschaften gezogen.

In Tabelle 3-3 sind die Struktur und die semantischen Beziehungen eines Instanzknoten dargelegt.

Knotenart	Knotenelemente	semantische Beziehungen
Instanzknoten	eindeutiger Name	instance-of
	Kommentar (optional)	value

Tabelle 3-3: Die Struktur und die Beziehungen eines Instanzknotens.

Das Beispiel über das Videokartenprojekt wird nun erweitert, um Objekte des Anwendungsbereichs. Es soll für das Beispiel genügen, es um das konkrete Projekt („Videokartenprojekt“), einen Angestellten („Abaelard“) und einen Hiwi („Pontifar“) zu erweitern (Abbildung 3.4). Der Knotentyp value wird erst nach Einführung der Slotwertknoten vorgestellt.

Instanzknoten werden mit Hilfe von Instanzknoten dargestellt.

3 Das Aktive Semantische Netz

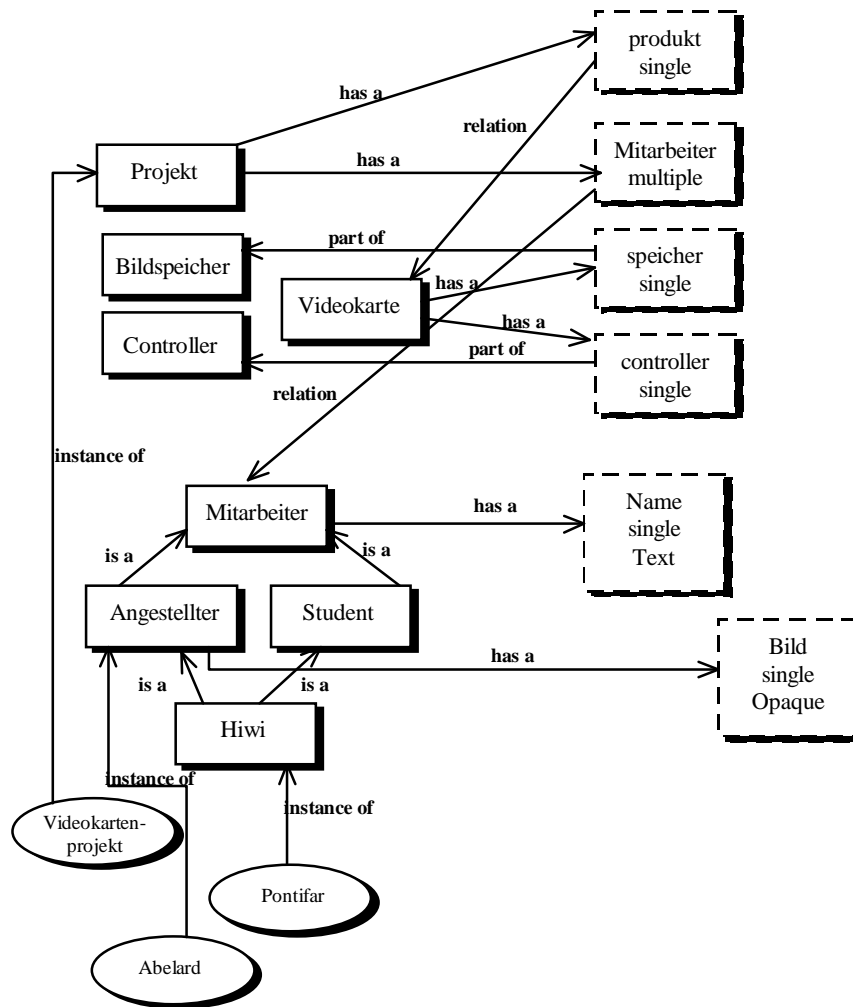


Abbildung 3.4: : Beispiel für Instanzknoten.

Die eindeutige Identifikation von Instanzen ist ein kreativer Prozeß. Zum einen gibt es die Bestimmung, daß jeder Namen höchstens einmal verwendet wird, zum anderen erhöht es das Verständnis des Semantischen Netzes, wenn aussagekräftige Identifikatoren verwendet werden und nicht kryptische Codierungen in der Art „XY23 a4“. Besonders wenn viele Objekte eines Konzeptes modelliert werden gibt es leicht Probleme der Namensfindung. Bei Menschen würde es sich anbieten Vor- und Nachname zu verwenden. Problematisch wird es, wenn zwei Personen denselben Namen besitzen. In solchen Fällen kann der Identifikator um weitere Angaben über die Person erweitert werden (z.B. Spitzname, Wohnort u.a.). Jeder Knoten im ASN kann deshalb mit einem beliebig langen Namen bezeichnet werden.

3.2.4 Slotwerte

Ein Slotwert ist eine Instanz eines Slots. Jedes Objekt hat seine individuellen Merkmale und Eigenschaften, deswegen erhält jedes seine eigene Ausprägung der Slots der

Konzepte und Superkonzepte. Diese Ausprägungen werden durch die Slotwertknoten symbolisiert.

Im Gegensatz zu den anderen drei Arten von Knoten im ASN verfügen Slotwertknoten über keinen Namen. Diese Knoten werden identifiziert, durch eine Kante, die vom Instanzknoten auf sie zeigt und eine Kante, die vom Slotwertknoten zum zugehörigen Slotknoten zeigt.

Es ist aber trotzdem einiges an Wissen in einem Slotwertknoten eingebettet.

Abhängig davon, ob der Typ eines Slotwertes eine Instanziierung eines elementaren oder nicht elementaren Konzeptes ist, wird unterschiedlich vorgegangen. Gehört der Slot zu einem elementaren Konzept (s. Kap. 3.2.1) wird der Wert innerhalb des Slotwertknotens abgelegt. Im anderen Fall ist der Slot über eine Kante mit einer Instanz (symbolisiert durch einen Instanzknoten) verbunden, die den Wert des Slots darstellt. Diese unterschiedliche Vorgehensweise ist sinnvoll, weil sonst für jede Instanz eines elementaren Konzeptes wie Integer ein eigener Instanzknoten erzeugt werden muß.

Ein weiteres Knotenelement eines Slotwertknotens macht eine Aussage über den Wissensstatus eines Slots. Wissen kann unvollständig, unsicher, widersprüchlich oder ungenau sein. Im ASN wird bisher das unvollständige Wissen über Eigenschaften eines Objektes repräsentiert. Zu jeder Eigenschaft einer Instanz kann angegeben werden, ob die Ausprägung der Eigenschaft zu einem Zeitpunkt bekannt ist oder nicht. Gerade bei der inkrementellen und iterativen Vorgehensweise beim Rapid Prototyping ist es wichtig, angeben zu können, daß ein Objekt zwar bestimmte Merkmale besitzt, diese aber (noch) nicht bekannt sind. Das Ausdrücken unbekanntes Wissens kann nicht durch das Weglassen des Slotwertknotens erreicht werden, da das auch bedeuten könnte, daß ein Objekt dieses Merkmal einfach nicht besitzt und im ASN hat es auch genau diese Bedeutung. Gibt es zum Beispiel das Konzept Mensch und ein Merkmal eines Menschen ist, daß er Freunde hat. Wenn von einem bestimmten Menschen nicht bekannt ist, ob er Freunde hat, dann wird ein Slotwertknoten erzeugt, der diese Unwissenheit als Wissen in sich trägt. Hat dagegen ein anderer konkreter Mensch keine Freunde, dann wird dies dadurch angezeigt, daß der Instanzknoten auf keinen Slotwertknoten verweist, der eine Instanziierung des Slots für Freunde ist. Wird ein Objekt erzeugt, werden dessen Slots a priori auf Unknown gesetzt.

Neben der Aussage über den Wissenstatus, dem nur bei Slots von elementaren Konzepten vorhanden Wertes des Slots, kann zu jedem Slotwertknoten auch ein Kommentar angegeben werden.

Von Slotwertknoten können zwei Arten von Relationen ausgehen. Die *of* Kante verbindet jeden Slotwertknoten mit dem dazugehörigen Slotknoten. Die *value* Kante zeigt auf den Wert eines Slots, falls dieses eine Instanz eines nicht elementaren Konzeptes ist.

In Tabelle 3-4 sind die Struktur und die semantischen Beziehungen eines Instanzknoten dargelegt.

3 Das Aktive Semantische Netz

Knotenart	Knotenelemente	semantische Beziehungen
Slotwertknoten	Wert unbekannt(Default) Wert (bei elementaren Konzepten) Kommentar (optional)	of value

Tabelle 3-4: Die Struktur und die Beziehungen eines Slotwertknotens.

Das Beispiel wird nun noch um Slotwerte erweitert. Slotwertknoten werden durch gestrichelte Ellipsen dargestellt. Der Anschaulichkeit halber werden die unbekannt Slotwerte in der Abbildung 3.5 nicht explizit gezeichnet. Zum Hiwi „Pontifar“ gibt es eine Bilddatei, die nicht innerhalb der Wissensbasis abgelegt ist. Im Slotwertknoten steht aber die Adresse an der sich diese Ressource befindet. Da Hiwi unter anderem ein Unterkonzept von Angestellter ist, erbt es den Slot Bild von dem Konzept Angestellter. Ferner wird dargestellt, daß „Abaelard“ (mit vollständigem Namen Peter Abaelard) und „Pontifar“ Mitarbeiter beim Projekt „Videokartenprojekt“ sind.

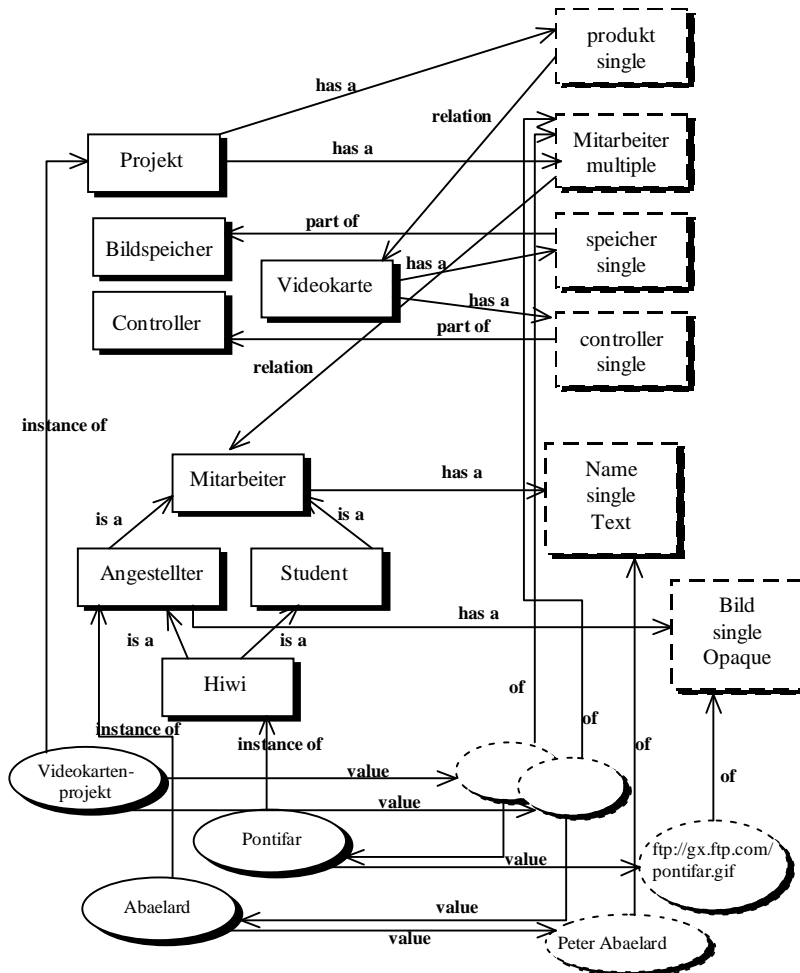


Abbildung 3.5: Ein Beispiel für Slotwertknoten.

Das Beispiel zeigt, dass selbst sehr einfache Beispiele ein Semantisches Netz schnell unübersichtlich werden lassen.

3.2.5 Die aktive Komponente

Da regelhafte Zusammenhänge vom Aktiven Semantischen Netz selbständig und automatisch auf eine aktive Weise bearbeitet werden sollen, wird diese Wissensart auch als *aktives* Wissen bezeichnet.

In den meisten Datenbanken und Expertensystemen werden prozedurales und deklaratives Wissen voneinander getrennt. Nach [Eck96] kann der konsistente Zustand der Daten jedoch nur gewährleistet werden, wenn Daten und Konsistenzbedingungen nicht getrennt werden. Im ASN werden wie bei Frames prozedurales und deklaratives Wissen vereint.

Durch die aktive Komponente im ASN können Änderungen an einer Stelle im Semantischen Netz automatisch durch das gesamte Netz propagiert werden und Einfluß auf die Struktur des Semantischen Netzes nehmen. Überdies können auch vom ASN unabhängige Operationen ausgeführt werden. Zum Beispiel könnte Mitwirkenden am Rapid Prototyping Prozeß eine Nachricht übermittelt werden oder externe Applikationen aufgerufen werden (das ASN stellt bekanntlich nur den Kern eines Projektes dar, in dem eine Reihe weiterer Tools wie CSCW-Werkzeuge zum Einsatz kommen) oder einem Experten als Assistent zur Seite stehen, indem es diesen Routineaufgaben abnimmt und ihn auf eventuelle Probleme hinweist. Weiterhin dienen sie der Konsistenzsicherung der Wissensbasis.

Die aktive Komponente setzt sich aus Regeln zusammen, die eine dreigeteilte Struktur besitzen:

- einem Eventteil,
- einem Conditionteil und
- einem Aktionsteil.

Diese Regeln werden auch als *ECA-Regeln* (*Event Condition Action*) bezeichnet.

Regeln im ASN entsprechen den Produktionsregeln in regelbasierten Wissensrepräsentationsystemen (siehe Kapitel 2.4). Sie enthalten zwei essentielle Komponenten: die Prämisse (Vorbedingung) und die Konklusion (Aktion).

Wenn die Vorbedingung einer Aktion erfüllt ist, kann die entsprechende Aktion ausgeführt werden, die wie weiter oben erwähnt, das Semantisch Netz manipulieren kann oder davon unabhängig ist. Die Vorbedingungen beziehen sich meist auf die Wissensbasis, in der Aktionen Änderungen durchführen können und dadurch bewirken, daß eventuell die Vorbedingungen anderer Regeln ausgeführt werden.

Der Conditionteil einer ASN-Regel nimmt Bezug auf einen Zustand im Semantischen Netz. Ein Zustand ist durch eine bestimmte Wertekombination definiert. Zum Beispiel

könnte eine einfache Vorbedingung lauten, „Haltbarkeitsdatum kleiner als aktuelles Datum“. Ist dies erfüllt, könnte eine Aktion gestartet werden, die veranlaßt, daß das dazugehörige Produkt entfernt wird.

Der Eventteil ähnelt dem Conditionteil, weil auch er Bedingungen überprüft und je nachdem die Regelarbeitung fortsetzt oder abbricht. Der Unterschied zum Conditionteil ist der, daß die Bedingungen unabhängig vom Zustand des Semantischen Netzes sind und nur auf Ereignisse reagieren, die von außen auf das Netz oder Teile davon einwirken.

Typische Ereignisse sind das Lesen oder Verändern von Konzepten, Slots, Instanzen oder Slotwerten. Im ASN werden die Regeln sozusagen an die Knoten gehängt, wie weiter unten im Text in Abbildung 3.6 und Abbildung 3.7 gezeigt wird.

Mit dem Eventteil kann explizit angegeben werden, bei welchen dieser Ereignisse, die Regelarbeitung aktiviert wird. Dadurch wird auch vermieden, daß bei jedem Eintreten eines Ereignisses eine Regel abgearbeitet wird (Eventteil als Filter).

In der momentanen Implementierung ist das ASN in der Lage auf zwei Arten von Ereignissen zu reagieren:

- Lesen eines Knotens und
- Ändern eines Knotens.

Tritt bei einem Knoten das spezifizierte Ereignis zu, wird der Conditionteil abgeprüft und falls die Bedingung wahr ist, wird der Aktionsteil ausgeführt.

Anzumerken ist noch, daß der Eventteil in einer Regel obligatorisch ist, nicht jedoch der Conditionteil. Das hat zur Folge, daß schon bei Eintreten eines Ereignisses eine Aktion gestartet wird. Man könnte damit zum Beispiel realisieren, daß ein Benutzer (ein Konzept im ASN) sofort unterrichtet wird, falls sich jemand Informationen über ihn abrufen.

Der Condition- als auch der Aktionsteil liegen als Tcl-Skripte vor. Tcl ist in kurzer Zeit zu einer der wichtigsten Programmiersprachen unter Unix geworden, nicht zuletzt deswegen, weil die Entwicklungszeit für Tcl-Programme oft sehr viel kleiner als beispielsweise für äquivalente C-Programme ist. Was Tcl für das ASN so interessant macht, ist, daß es eine interpretierende Programmiersprache ist. So können zur Laufzeit jederzeit Tcl-Skripte eingefügt oder verändert werden, ohne daß dabei das gesamte System neu kompiliert werden muß.

Innerhalb des ASN werden die Tcl-Skripte als *Tcl-Knoten* (symbolisiert durch ein Dreieck) und die Regelteile über semantische Kanten dargestellt. Es gibt demnach eine *read-event* (für das lesende Zugreifen auf einen Knoten) eine *modify-event* (für das Ändern eines Knotens), und eine *action* Kante, die jeweils auf einen Tcl-Knoten verweisen. Die anderen Knotentypen symbolisieren die elementaren Konstrukte der Wissensrepräsentation: Konzepte, Slots, Instanzen und Slotwerte.

Soll beispielsweise dargestellt werden, daß beim Lesen eines Konzeptes A eine Condition überprüft werden soll, die gegebenenfalls (wenn Bedingung wahr ist) eine Aktion startet, dann wird dies wie in Abbildung 3.6 gezeigt, realisiert.

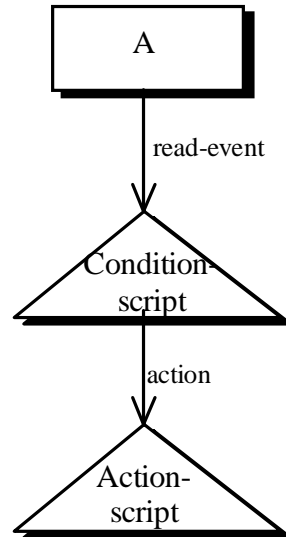


Abbildung 3.6: Read-Event im ASN realisieren.

Soll dagegen beim Verändern eines Konzeptes A eine Aktion unmittelbar gestartet werden (ohne Überprüfung einer Condition), dann kann dies im ASN wie in Abbildung 3.7 realisiert werden. Dadurch wird der Anforderung nach Online-Dialogfähigkeit der Wissensbasis Rechnung getragen, denn Anfragen an die Wissensbasis können umgehend beantwortet werden. In derselben Abbildung ist auch dargestellt, wie am Slot a von A, eine Regel „hängt“, die beim Lesen des Slots, aktiviert wird.

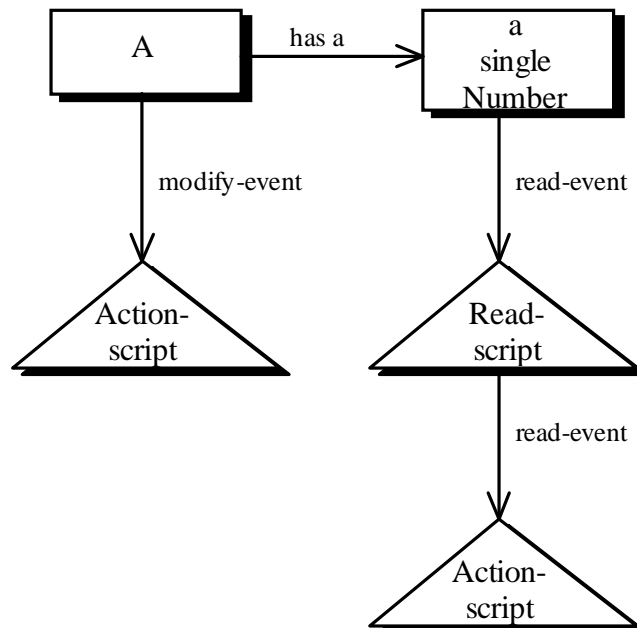


Abbildung 3.7: Weiteres Beispiel für Events im ASN.

Nachdem nun die Knotenarten und die Relationentypen des ASN vorgestellt worden sind, werden die Relationentypen in zusammengefaßt. Dabei wird gezeigt, welche Knotenarten über die einzelnen Kanten verbunden werden können.

Relationentyp	Ursprungsknoten	Zielknoten
is a	Konzept	Konzept
has a	Konzept	Slot
part of	Slot	Konzept
relation	Slot	Konzept
instance of	Instanz	Konzept
value	Instanz	Slotwert
of	Slotwert	Slot
value	Slotwert	Instanz
read-event	Konzept, Slot, Instanz, Slotwert.	Tcl-Skript
modify-event	Konzept, Slot, Instanz, Slotwert.	Tcl-Skript
action	Konzept, Slot, Instanz, Slotwert Tcl-Skript	Tcl-Skript

Tabelle 3-5: Die Relationenarten des ASN.

Das ASN erfüllt ein breites Spektrum verschiedener semantischer Beziehungen und kausaler Abhängigkeiten und hat eine große Ausdrucksfähigkeit ohne gleichzeitig mit unzähligen Konstrukten überladen zu sein.

Damit wären die Repräsentationskonstrukte des Aktiven Semantischen Netzes vorgestellt. Es soll hier noch kurz darauf eingegangen werden wie das Aktive Semantische Netz realisiert wird.

3.3 Realisierung des ASN

Das ASN wird mit Hilfe der objektorientierten Programmiersprache ObjectStore implementiert. ObjectStore stellt unter anderem eine eigene Erweiterung von C++ mit generischen Klassen, Anfrageausdrücken und Ausnahmebehandlungen zur Verfügung.

Für jedes grundlegende Konstrukt im ASN (Konzept, Slot, Instanz, Slotwert und Regel) wird in ObjectStore eine eigene Klasse zur Verfügung gestellt. Objekte von Klassen

können zur Laufzeit jederzeit konstruiert, manipuliert und gelöscht werden. Da auch die Konzepte und Slots (die terminologischen Entitäten) auf der Implementierungsebene nur Objekte von Klassen sind, kann im ASN die Struktur der terminologischen Komponente verändert werden. Dieser Sachverhalt macht das ASN immens flexibel. Es gibt keine starren Strukturen, die einmal festgelegt werden und die danach nicht oder nur sehr mühselig änderbar sind. Dies unterscheidet das ASN grundlegend von Datenbanken auch den objektorientierten, bei denen der Strukturteil einmal, nach einer sorgfältigen Analyse des Problembereichs, festgelegt wird.

Das ASN erfüllt damit wesentliche Anforderungen an die Wissensbasis (siehe Kapitel 3.1) in Bezug auf Erweiterbarkeit und Dynamik. Das Aktive Semantische Netz ist einfach erweiterbar, anpaßbar und flexibel und unterstützt damit den iterativen, inkrementellen und dynamischen Entwicklungsprozeß beim Rapid Prototyping.

Von Nachteil ist, daß wegen der umfangreichen Konsistenzüberprüfungen, eine hohe Zeitkomplexität beim Arbeiten mit der Wissensbasis in Kauf genommen werden. Man denke nur daran, was für Auswirkungen das Löschen eines Konzeptes aus einer Konzepthierarchie auf die Wissensbasis hat: Instanzen dieses Konzeptes müssen gelöscht, alle Pfade zu diesen Objekten umgeleitet oder gekappt, Unterkonzepte an die Oberkonzepte des gelöschten Konzeptes gelöscht werden, Slotwerte von Objekten der Unterkonzepten entfernt werden, wenn die dazugehörigen Slots im gelöschten Konzept definiert wurden, usw.

Die Erweiterung des Aktiven Semantischen Netzes (zum Beispiel um die Darstellung von unsicheren Wissens) hängt nicht zuletzt davon ab, wie effizient mit der Wissensbasis gearbeitet werden.

Die Kanten im ASN werden mit Hilfe von Zeigern und Referenzen in den Klassen realisiert. Dadurch daß eine gerichtete Kante im ASN auf der Implementierungsebene durch zwei Zeiger (einer vom Ursprungsobjekt zum Zielobjekt, der andere vom Zielobjekt zum Ursprungsobjekt) realisiert wird, ist eine effizientere Bearbeitung des ASN möglich.

Eine Programmierschnittstelle zum ASN wurde in der Studienarbeit von [Friedrich96] realisiert.

3 Das Aktive Semantische Netz

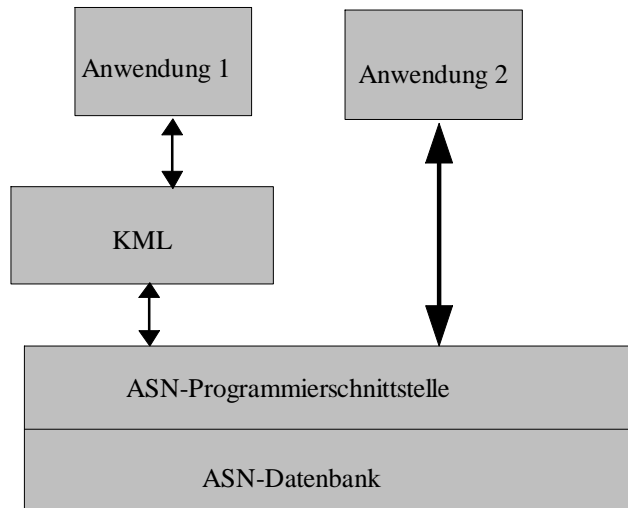


Abbildung 3.8: Einbettung der Diplomarbeit in das Gesamtprojekt.

Dort finden sich weitere Informationen zur ASN-Datenbank und der datenbankunabhängigen C-Schnittstelle, mit der auf die Wissensbasis zugegriffen werden kann.

Abbildung 3.8 verdeutlicht, an welcher Stelle im Rahmen des Gesamtprojektes des Entwicklung der Rapid Prototyping Wissensbasis, die Sprache, die in dieser Arbeit konzipiert und implementiert wurde, steht.

4 KML

In diesem Kapitel wird die, in dieser Arbeit entwickelte, kontextfreie Wissensbeschreibungssprache *KML* (*Knowledge Modelling Language*) vorgestellt. Zunächst wird geklärt, welche Anforderungen an die Sprache gestellt werden, danach werden die einzelnen Konstrukte, die die Sprache zur Verfügung stellt, ihre Bedeutung, ihre Syntax und jeweils ein kurzes Beispiel dazu, ausführlich beschrieben. Danach erfolgt ein Vergleich der Ausdrucksmächtigkeit von KML und ASN.

4.1 Anforderungen an KML

Die erste und wichtigste Forderung an die Wissensbeschreibungssprache KML ist, daß sie genau das Wissen, welches durch das ASN repräsentiert werden kann, darstellen kann. Dazu müssen die Repräsentationsstrukturen und -konzepte aus dem ASN in die KML übertragen werden. Das bedeutet, daß die verschiedenen Arten von Knoten, Relationen und die innere Struktur der Knoten in die Sprachkonstrukte der KML abgebildet werden können. Umgekehrt darf die KML das ASN nicht in ihrer Ausdrucksmächtigkeit überbieten. Dieser Anspruch wird verständlich, wenn man sich vor Augen hält, daß das gesamte Wissen, das mit Hilfe des ASN dargestellt werden kann, auch in die KML überführt werden können muß. Andererseits soll das Wissen einer Anwendung, die mit KML modelliert wird, in das ASN übertragen werden können. Letztlich bedeutet das nichts anderes, als das das ASN und die KML in ihrer Darstellungsmächtigkeit von Wissen äquivalent sind. Dieser Zusammenhang wurde beim Sprachentwurf stets berücksichtigt werden.

Während diese Forderung an die KML keine Freiheit bei der Wahl der Ausdrucksmächtigkeit läßt, kann bei der nächsten Forderung, eine „hohe“ Benutzerschnittstelle zu schaffen, ein beliebiger Formalismus gewählt werden.

Diese Forderung ergibt sich aus der Zielgruppe der Anwender. Da das ASN als Wissensbasis für das Rapid Prototyping konzipiert wurde, sind die Benutzer, Personen, die an der Produktentwicklung beteiligt sind. Das können zwar auch Computerfachleute sein, in der Regel sind es aber Ingenieure, Techniker, kaufmännisch ausgebildete Personen u.a. Um dem Rechnung zu tragen, sollte eine Sprache entworfen werden, die keine Programmierkenntnisse voraussetzt und die intuitiv verständlich ist. Der Anwender soll sich ganz auf das Problem konzentrieren können und sich bei der Produkt- oder Wissensmodellierung nicht mit Bedienungsanleitungen herumschlagen müssen.

Dies führte zu einer Sprache, die objektbasiert ist, da sie durch ihre modulare Objektbezogenheit anschaulicher ist als zum Beispiel eine relationsbezogene Darstellung, in der Beziehungen zwischen Objekten im Mittelpunkt stehen (wie in logikorientierten

Sprachen). Im Gegensatz dazu stehen bei der objektbezogenen Darstellung komplexe Objekte im Zentrum der Modellierung. Diese Modularisierung durch Objekte ist nach [Booch89] ein wesentliches Prinzip, der Komplexität vieler (besonders großer) Anwendungen Herr zu werden. Die Darstellung von Wissen in Objekten, in der Wissen über das Objekt gekapselt ist, erleichtert das Verständnis für Anwendungen erheblich.

Außerdem werden möglichst wenige Sprachkonzepte zur Verfügung gestellt (was durch die Modularisierung erleichtert wird), denn eine Sprache kommt der Denkökonomie des Menschen entgegen, wenn sie möglichst viele Probleme zielförderlich mit wenig Grundkonzepten lösen kann [Lusti90]. Dadurch wird ein „Programmieren“ des Wissens verhindert, das Wissen wird vielmehr in einer formularartigen und dem Menschen vertrauten Darstellung beschrieben. Bei der Wahl der Schlüsselworte wurde darauf geachtet, daß diese auch in der natürlichen Sprache vorkommen und mit einer möglichst ähnlichen Semantik verwendet werden.

Viele Elemente der objektbezogenen textuellen Wissensdarstellung lassen sich relativ einfach in eine objektbezogene graphische Darstellung überführen. Beispielsweise kann die Vererbungshierarchie zwischen Konzepten (s. Kap. 4.2.2 und Kap. 4.2.5) leicht mit einem Klassen- beziehungsweise Konzeptbrowser dargestellt werden. In diesem Kapitel soll jedoch nur auf die Sprache selbst eingegangen werden.

Im folgenden werden die Konzepte der Sprache einzelnen vorgestellt.

4.2 Sprachkonstrukte von KML

Bei der Vorstellung der einzelnen Sprachkonstrukte werden in jedem Unterabschnitt die Bedeutung eines jeden Konzeptes beschrieben, dessen Syntax in Backus-Naur-Form und die Semantik in natürlicher Sprache schrittweise beschrieben und dazu ein einfaches Beispiel vorgestellt. Die Backus-Naur-Form ist eine *Metasprache*, durch die andere (kontextfreie) Sprachen definiert oder beschrieben werden können. Die Nichtterminale der Sprachgrammatik werden durch spitze Klammern gekennzeichnet. Die linke und rechte Seite einer Produktionsregel werden durch das Symbol $::=$ getrennt. Existieren mehrere rechte Seiten für ein Nichtterminal, so werden diese durch senkrechte Striche getrennt hintereinander geschrieben.

Die Syntax, die in den einzelnen Unterabschnitten vorgestellt wird, ist nicht immer gleichlautend mit der Syntax im Anhang A, wo die gesamte Syntax der implementierten KML aufgeführt ist. Vielmehr wird bei der Vorstellung der einzelnen Konzepte die Syntax angezeigt wie sie in den einzelnen Phasen der Sprachentwicklung gültig war. Bei Hinzufügen von Sprachkonstrukten mußten einzelne Sprachregeln umgeschrieben werden. Die Änderungen werden an gegebener Stelle im Text angezeigt. Die Beschreibung der KML erfolgt in einer Art und Weise wie es in Tutorials häufig der Fall ist.

Die Schlüsselworte der Sprache, Beispiele und Syntaxangaben werden in dem Zeichensatz `Courier` vorgestellt (Dies gilt auch in den nachfolgenden Kapiteln).

4.2.1 Aufbau der Sprache

Wissensbasierte Systeme enthalten meist zwei Arten von Wissen [Lusti90]. Auch das Aktive Semantische Netz unterteilt Wissen in zwei Kategorien: in solches, das der Beschreibung und Definition beliebiger Begriffe dient (terminologisches Wissen) und (assertionales) Wissen zur Beschreibung realer Objekte und deren Beziehungen zueinander (siehe dazu auch Kap. 3.2).

So wird das Wissen über eine Anwendung in einer KML-Datei in zwei Schritten eingeführt: Im Definitionsteil werden die Konzepte und deren Slots beschrieben. Im Objektteil sind die Instanzen bzw. Objekte der Konzepte aufgeführt. Ein Konzept kann man sich als eine abstrakte Menge von gleichartig aufgebauten Objekten vorstellen, die sich in ihren konkreten Eigenschaften aber erheblich unterscheiden können. Da KML eine objektbasierte Sprache ist, soll neben den im vorherigen Kapitel beschriebene Begriff „Instanz“ synonym dazu der Terminus *Objekt* verwendet werden. Der Begriff Objekt ist weit verbreitet und begünstigt Mißverständnisse. Deswegen soll hier noch einmal die Bedeutung von Objekt (beziehungsweise Instanz) im Kontext dieser Diplomarbeit erklärt werden: Ein Objekt verkörpert eine konkrete Verwirklichung eines Konzeptes und ist ein reales Objekt des Anwendungsbereichs. Ein Objekt ist nicht nur ein körperlicher Gegenstand, sondern kann auch eine Person, eine Situation, ein Ereignis oder eine abstrakte Idee sein.

Der Definitionsteil wird mit `DEFINITIONS:` eingeleitet, der Objektteil mit `OBJECTS:`. Groß- oder Kleinschreibung spielt bei den Schlüsselwörtern im übrigen keine Rolle. Es ist demnach auch erlaubt `definiTIONs` oder `ObjectS` zu schreiben. Genauso können zwischen den einzelnen Elementen der Sprache beliebig viele Whitespacezeichen (Blanks, Tabulatoren oder Zeilenvorschübe) stehen.

Der Definitions- bzw. Strukturteil und Objektteil können innerhalb einer KML-Datei einander abwechseln, so das dem Anwender die Möglichkeit bleibt Konzepte und Instanzen davon an den Stellen einzuführen, an denen er es für angebracht hält und nicht etwa alle Konzepte und alle Objekte jeweils aufeinanderfolgend beschreiben zu müssen (siehe mögliches Schema einer KML-Datei in Abbildung 4.1)

Zu beachten ist allerdings, daß zum einem in jedem Definitionsabschnitt wenigstens ein Konzept und in jedem Objektteil wenigstens ein Objekt beschrieben werden muß und zum anderen kann ein Objekt erst verwendet werden, wenn das Konzept von dem es eine Instanz ist, davor in einem Definitionsteil definiert wurde. In einer späteren Version der Sprache soll diese Einschränkung aufgehoben werden.

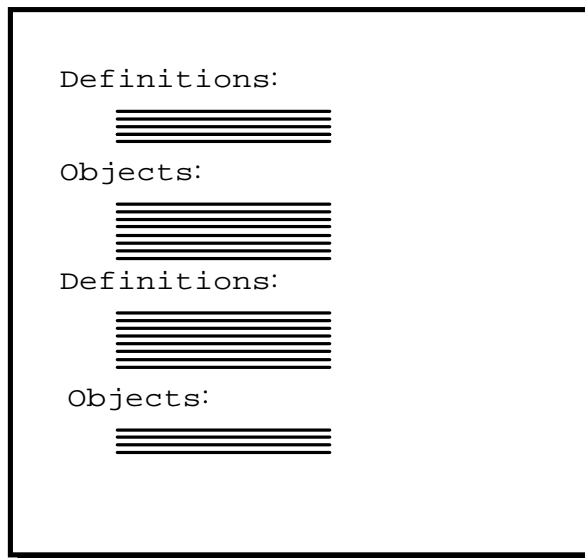


Abbildung 4.1: prinzipieller Aufbau einer Wissensmodellierung in einer KML-Datei

Die Syntax des Aufbaus einer Wissensbeschreibung in KML lautet:

```
<kml_sprache> ::= <kml_sprache> <aufbau>
                | <aufbau>
<aufbau> ::= DEFINITIONS : <strukturteil>
              | OBJECTS : <objektteil>
```

Auf den Definitionsteil soll nun zuerst näher eingegangen werden.

4.2.2 Konzepte

Im ASN wird ein Konzept durch einen Knoten repräsentiert (s. Kap. 3.2.1). Dessen Bedeutung ergibt sich durch Beziehungen zu anderen Konzeptknoten. In KML werden Konzepte im Definitionsteil aufgeführt. Der Definitionsteil einer Modellierung in KML besteht ausschließlich aus einer Ansammlung von Konzeptbeschreibungen. Diese Modularisierung des Konzeptwissen ist eine Konsequenz der Forderung, eine Sprache zu entwerfen, die der Denkökonomie von Menschen nahe kommt. Um ein Konzept zu definieren, muß diesem ein eindeutiger Name zugewiesen werden. Eindeutig heißt, daß es kein anderes Konzept in der Anwendung geben darf, daß denselben Namen trägt (unique name assumption). Ein Name muß mit einem Buchstaben anfangen, danach sind eine beliebige Folge von Buchstaben, Ziffern und den Zeichen '!', '_', '-', '@' und '?' erlaubt.

Um einen Namen als Konzept zu kennzeichnen, folgt diesem ein Doppelpunkt und das Schlüsselwort CONCEPT. Danach werden die Eigenschaften (in der Terminologie des ASN Slots genannt) des Konzeptes vorgestellt (siehe dazu folgendes Unterkapitel). Abgeschlossen wird jede Konzeptbeschreibung durch einem Punkt '.' (genauso wird das

Ende einer Objektbeschreibung mit einem Punkt angezeigt). Im Definitionsabschnitt können beliebig viele Konzepte definiert werden, wenigstens aber eines.

In KML wird ein Konzept mit seinen Eigenschaften (später kommen noch Regeln hinzu, siehe Kapitel 4.2.8) als *Konzeptmodul* (dieser Begriff wird in Kapitel 4.3 veranschaulicht) bezeichnet, da das Wissen über ein Konzept an einer Stelle (eben diesem Modul) zusammengefaßt wird. Diese Modularisierung von Wissen erleichtert das Verständnis für eine Anwendung und ist ein wesentliches Prinzip des Sprachentwurfs von KML gewesen. Analog wird das Wissen für eine Instanz der Anwendung modularisiert (siehe Kap. 4.2.6).

Die Syntax zu dem eben gesagten lautet:

```

<strukturteil> ::= <konzept>
                | <strukturteil> <konzept>

<konzept> ::= <konzeptname> : CONCEPT <eigenschaften> .

<konzeptname> ::= <bezeichner>

<bezeichner> ::= <buchstabe> <rest_name>

<rest_name> ::= <buchstabe> | <ziffer>
                | ! | _ | - | @ | ?

```

Dies ist eine vorläufige Syntaxangabe, da noch nicht berücksichtigt wurde, daß Konzepte in einer Vererbungsbeziehung zueinander stehen können und die später noch ergänzt wird.

Die Nichtterminale *buchstabe* und *ziffer* werden an dieser Stelle nicht näher erläutert, da ihre Bedeutung offensichtlich sein dürfte (Im Anhang A ist die Syntax vollständig angegeben).

Um einen guten Vergleich zur Darstellung von Wissen im ASN zu haben, wird das Beispiel aus Kapitel 3 über das Hochschulprojekt zur Entwicklung einer leistungsfähigen Videokarte für einen Multimediacomputer erneut aufgegriffen.

Konzepte aus diesem Beispiel sind unter anderem Projekt, Bildspeicher, Controller, Videokarte und Mitarbeiter (siehe Beispiel aus Abbildung 3.2). In KML kann das wie in Abbildung 4.2 dargestellt werden.

```

Definitions:

Projekt: CONCEPT
.
Bildspeicher: CONCEPT
.
Controller: CONCEPT
.
Videokarte: CONCEPT
.
Mitarbeiter: CONCEPT
.

```

Abbildung 4.2: Wichtige Konzepte für das Videokartenprojekt definieren

Wie der Aufbau von Konzepten beschrieben werden können, wird im folgenden Unterabschnitt gezeigt.

4.2.3 Typkonstruktoren und komplexe Objekte

Die Objekte oder Instanzen eines Konzeptes sind dadurch gekennzeichnet, daß sie dieselben Merkmale besitzen. Die Merkmale der einzelnen Objekte können unterschiedliche Ausprägungen haben. So besitzt jedes Videokarte einen Bildspeicher (ein Merkmal einer Videokarte), welcher spezielle Speicher vorliegt, ist dann eine Frage der Ausprägung. Die Merkmale oder Eigenschaften werden mit Hilfe von Slots (oft wird auch der Begriff Attribut verwendet) dargestellt.

Das Merkmal wird mit einem Namen identifiziert, dazu wird angegeben, welche Art von Ausprägungen dieses Merkmals annehmen kann. Im ASN werden die elementaren Konzepte Text, Integer, Float, Boolean und Opaque Object zur Verfügung gestellt (siehe Kap. 3.2.1) In der KML stellen die elementaren Konzepte Datentypen dar (beim Compilieren findet eine Typüberprüfung sowie ein Überprüfen der verwendeten Operationen statt), die zum Teil andere Bezeichnungen besitzen als im ASN. Der Grund liegt darin, daß Schlüsselworte in KML intuitiv verständlich sein sollen. Ein Informatiker oder Programmierer assoziiert mit dem Begriff „Float“ sofort gewisse Vorstellungen, andere Personen hätten damit aber möglicherweise gewisse Schwierigkeiten, deswegen die Wahl von Bezeichnern, die sofort verständlich sind oder sein sollen. Eine Ausnahme bildet dabei das elementare Konzept Opaque Object. Opaque Object umfaßt beliebige Dateitypen, die in die Wissensbasis mit aufgenommen werden können (z.B. Vektordateien, Sounddateien u.a.) oder auf die aus der Wissensbasis heraus referenziert wird und die sich damit auch auf anderen Rechnern befinden können, deren innere Struktur für das ASN aber keine Bedeutung hat. In diesem Fall geht es um computerspezifisches Wissen, für das es in der Umgangssprache kein

allgemeinverständliches Wort gibt. In KML wurde auch die Bezeichnung *Opaque Object* dafür gewählt. Diese Slotwerte werden mit Hilfe der URL-Notation spezifiziert werden. Mit der URL können beliebige Ressourcen (v.a. Dateien) im Internet spezifiziert werden. Weiter unten wird der Begriff näher erläutert. In Tabelle 4-1 werden die elementaren Konzepte des ASN und die entsprechenden Datentypen aus KML gegenübergestellt.

ASN	KML
Text	TEXT
Integer	INTEGER
Float	REAL (NUMBER)
Boolean	{YES, NO}
Opaque	Opaque (Object)

Tabelle 4-1: Elementare Konzepte aus dem ASN und die dazugehörigen Typen aus KML.

Jedem Datentyp und jedem Konzept wird eine *Domäne* zugeordnet. Diese stellen die Menge der erlaubten Attribut- oder Slotwerte dar, die auch Instanzen dieses Typs genannt werden. Als Domäne wird auch die Menge der Instanzen bezeichnet, die zu einem Konzept gehört.

Number soll für die Menge der ganzen Zahlen stehen. Natürlich kann ein Computer nur einen endlichen Teilbereich davon darstellen. Wie groß dieser ist, ist plattformspezifisch. In der Regel dürfte dieser aber für die allermeisten Anwendungen genügen.

Um größere oder kleinere Zahlen zu nutzen oder rationale Zahlen zu verwenden, steht der Datentyp Real bzw. Real Number zur Verfügung. Auch hier wird ein nur ein endlicher Ausschnitt der rationalen Zahlen zur Verfügung gestellt.

Ein weiterer wichtiger, ja essentieller Datentyp ist Text. Sehr viele Merkmale lassen sich am besten textuell beschreiben.

Neben den drei erwähnten Standard-Datentypen, die auch in nahezu allen typ-basierten Datenbankmodellier- und Wissensrepräsentationssprachen vorkommen, bietet das ASN auch die Möglichkeit boolesche Werte zu repräsentieren. In der Sprache wird das mit der einer Menge und den zwei Elementen Yes und No dargestellt. Gewählt wurde diese Darstellung weil ein {Yes, No} intuitiv verständlicher ist, als die im ASN und in Programmiersprachen häufig verwendete Bezeichnung *Boolean*.

Der letzte der vorhandenen Datentypen ist Opaque bzw. Opaque Object. Die Slotwerte werden durch eine URL dargestellt. URL steht für *Uniform Resource Locator*, ein Begriff der eng mit dem World Wide Web verknüpft ist. Die URL bietet nun eine Möglichkeit, Ressourcen im Internet auf eine einheitliche Art und Weise, zu identifizieren und direkt zu benennen. Jede Instanz dieses Typs stellt sozusagen eine Adresse dar, an der sich eine Datei, die für die Anwendung von Bedeutung ist, befindet.

Da URLs nicht nur von Rechnern sondern auch von Menschen interpretiert werden, waren die Voraussetzungen bei deren Einführung, kurz und verständlich zu sein. Dazu sollten sie aus druckbaren Zeichen, ohne Leerzeichen bestehen. Im wesentlichen benötigt man drei Informationen, um eine Ressource im Internet zu beschreiben. Dies ist die *Zugriffsmethode*, ein *Rechnername* und ein *Dateiname* mit Pfadangabe.

Eine URL hat demnach den Aufbau `URL=Zugriffsart/Rechnername/Dateiname`.

Will man auf einen FTP-Server zugreifen, so lautet die Zugriffsart `ftp://`, soll dagegen ein HTML-Dokument angesprochen werden, würde man ein `http://` angeben, bei Zugriff auf eine Datei `file://`. Weitere Informationen zu diesem Thema entnehme man [ScBoGeKa94].

Durch diese URL wird dem Wunsch nach multimedialer Informationsdarstellung Rechnung getragen. Mit der URL können alle Arten von Daten, wie Bild-, Sound- und Videodateien, ins ASN übertragen werden. Soll nur der Verweis auf eine Ressource im ASN gespeichert werden, dann kann man das Schlüsselwort `LINK` benutzen. Ansonsten wird die spezifizierte Datei über das Internet in den Rechner übertragen auf dem sich die Wissensbasis befindet und dort gespeichert.

So bedeutet beispielsweise

```
Bild: ftp://ftp.ask.uni-
karlsruhe.de/pub/graphics/elephant.gif
```

daß das spezifizierte Bild in das ASN kopiert werden würde.

Im Falle von

```
Bild: LINK ftp://ftp.ask.uni-
karlsruhe.de/pub/graphics/elephant.gif
```

würde dagegen nur die URL im ASN gespeichert werden. Beim Zugriff auf dieses Bild, müßte es jedesmal vom entsprechenden FTP-Server in Karlsruhe geladen werden. Der Vorteil besteht aber darin, daß nicht alle Information, die in einer Anwendung auftauchen, auch direkt im ASN gespeichert werden müssen und die Datenbank unnötig aufblähen, dafür müssen aber längere Wartezeiten beim Zugriff auf die Daten in Kauf genommen werden. Der Anwender muß einen Kompromiß zwischen diesen beiden Größen finden.

Um komplexe Objekte modellieren zu können, genügen diese Datentypen nicht. Viele Eigenschaften eines Objektes sind damit nicht zu beschreiben. Aus diesem Grunde kann ein Slot auch einen Wertebereich haben, der aus Objekten besteht. Diese Objekte sind Instanzen eines anderen Konzeptes (Im ASN sind dies die nichtelementaren Konzepte).

Die Grammatik für Slots hat folgenden Aufbau:

```
<eigenschaft> ::= <slotname> : <wertebereich>
```

```

<wertebereich> ::= <rangekonzeptname>
                | NUMBER
                | REAL
                | REAL NUMBER
                | TEXT
                | {YES,NO}
                | Opaque
                | Opaque Object

<slotname> ::= <bezeichner>

<rangekonzeptname> ::= <bezeichner>

```

Aus diesen Datentypen können durch rekursiv anwendbare Typkonstruktoren kompliziertere Datentypen aufgebaut werden. Die beiden wichtigsten Typkonstruktoren [Heuer92], die *Aggregation* und die *Gruppierung*, werden in KML verwirklicht. Eine Stärke des KML ist die beliebig häufige Kombination dieser beiden Konstruktoren, das heißt, die beiden Konzepte sind *orthogonal* zueinander. In der KML wird die Orthogonalität dadurch ermöglicht, daß Slots auch Konzepte als Domäne besitzen können.

Mit der Aggregation wird ein neuer Typ erzeugt, der die in die Aggregation eingehenden Typen und Konzepte als Komponenten beinhaltet. Eine Instanz der Aggregation ist dann eine Menge aus Instanzen der Komponententypen und -konzepte.

In der KML ist man die Aggregation verwirklicht, indem die Merkmale (Slots), die ein Objekt auszeichnen, hintereinander aufgezählt werden. Es sind dadurch keine speziellen Sprachkonstrukte notwendig, um die Slots zu realisieren. Außerdem wird damit einem modularen und damit verständlicheren Aufbau Rechnung getragen. Zur Erinnerung im ASN gibt es spezielle Knoten, die Slotknoten, die die Merkmale repräsentieren. Diese Slotknoten sind über die has-a Kante mit dem Konzept verbunden, das sie beschreiben (siehe Kapitel 3.2.1). In KML tauchen Slots nicht mehr als eigenständige beziehungsweise unabhängige Elemente auf, da sie nur im Zusammenhang mit dem Konzept, das sie beschreiben, eine Bedeutung besitzen. Die Modularisierung der Konzeptbeschreibungen in den Definitionsabschnitten einer KML-Datei hat den Vorteil das Wissen nicht nur irgendwo auftaucht, sondern zusammenhängend auftritt und damit die Übersichtlichkeit erhöht.

Die Syntax dazu lautet (vorläufig):

```

<eigenschaften> ::= <eigenschaft>
                  | <eigenschaften> <eigenschaft>

```

Siehe dazu auch Kapitel 4.2.2, wo die Syntax der Konzeptdefintion aufgeführt ist.

Mit der Gruppierung wird ein neuer Typ erzeugt, der aus mehreren Instanzen des in die Gruppierung eingehenden Typs eine Instanz des neuen Typs formt. Eine Instanz der Gruppierung ist dann eine Menge von Instanzen des zugrundeliegenden Typs oder

Konzeptes. Bei der Gruppierung lassen sich prinzipiell zwei Typen unterscheiden: Die *Menge* und die *Liste*. Eine Menge kann Elemente nicht mehrfach beinhalten und diese haben keine Reihenfolge. In der Liste dagegen können Elemente mehrfach vorkommen und besitzen eine Ordnung. Im ASN wird für die Gruppierung eine Menge verwendet. Streng genommen ist es eine Multi-Menge (in der Informatik wird eine Multi-Menge üblicherweise als *bag* bezeichnet), da Elemente mehrfach enthalten sein können. Was die Kardinalität der Slots betrifft, kennt das ASN nur zwei Arten: Ein Konzept kann einen bestimmten Slot genau einmal oder beliebig oft besitzen (So ist die Anzahl der Freunde eines Menschen nicht von vornherein festgelegt, wohl aber steht fest, daß er genau ein Herz besitzt) (siehe Kap. 3.2.2).

In der KML wird die Gruppierung mehrerer Elemente durch SET OF angezeigt. Ist ein Merkmal nur einmal vorhanden wird nur der Typ oder das passende Konzept angegeben. Die Syntax für das Nichtterminal eigenschaft wird um eine Regel erweitert:

```
<eigenschaft> ::= <slotname> : <wertebereich>
                | SET OF <slotname> : <wertebereich>
```

Mit den nun vorhanden syntaktischen Möglichkeiten lassen sich die Merkmale der Konzepte aus dem Beispiel aus Abbildung 4.2 besser beschreiben (siehe auch nachfolgende Abbildung 4.3)

```
Definitions:

Projekt: CONCEPT
  produkt: Videokarte
  mitarbeiter:
    SET OF Mitarbeiter
  .
Bildspeicher: CONCEPT
  .
Controller: CONCEPT
  .
Videokarte: CONCEPT
  speicher: Bildspeicher
  controller: Controller
  .
Mitarbeiter: CONCEPT
  name: TEXT
  .
```

Abbildung 4.3: Eigenschaften der Konzepte beschreiben.

Natürlich lassen sich auch andere und weitere Eigenschaften bei dem und anderen Beispiel angeben. Der Zweck der Beispiele ist aber nicht eine wohldefinierte

Modellierung von Wissen, sondern das Vorstellen der Konzepte der entworfenen Wissenmodellierungssprache.

Zu beachten ist, daß wie schon erwähnt, die Groß- und Kleinschreibung bei Schlüsselwörtern keine Rollen spielt. Dies gilt aber nicht bei benutzerdefinierten Namen, wie Konzept- und Slotnamen. In Beispiel würde die Anweisung `Speicher: Bildspeicher` bei der Übersetzung des Beispiels ins ASN zu einem Fehler führen, da dem Übersetzungsprogramm zwar `Bildspeicher` bekannt ist, nicht aber `Speicher`.

Bei der Definition von Konzepten muß keine Reihenfolge beachtet werden. Ein Konzept kann als Domäne für einen Slot verwendet werden, bevor das Konzept definiert wurde. Das Konzept `Videokarte` kann demnach ohne Probleme wie in Abbildung 4.3 zu sehen ist dem Slot `produkt` zugewiesen werden, obwohl es erst danach definiert wird.

4.2.4 Konzepte als Konzeptkomponenten

Im ASN gibt es zwei grundlegende Typen von Beziehungen zwischen Konzepten (siehe Kap. 3.2.1):

- die Konzept—Komponentenkonzept-Beziehung, mit der Objekte und ihre Komponentenobjekte verbunden werden (im ASN über die `has-a` Kante realisiert).
- die Konzept—Unterkonzept-Beziehung, durch die Objekte in eine Vererbungshierarchie eingereiht werden können (im ASN über die `is-a` Kante realisiert).

Der erste Typ von Beziehungen wurde bereits im vorherigen Unterabschnitt eingeführt, als gezeigt wurde wie komplexe Objekte erzeugt werden können. Es soll hier darum gehen, wie verschiedene Arten von Konzept—Komponentenkonzept-Beziehungen im ASN beziehungsweise der KML verwirklicht werden.

Eine ausführlichere Übersicht über die Semantik solcher Beziehungen ist in [Heuer92] aufgeführt.

Im ASN gibt es zwei verschiedene Arten dieser Beziehung. Ein Objekt kann Teilobjekt eines anderen sein, das heißt physisch in diesem enthalten sein. Für die Produktmodellierung ist das eine wichtige Eigenschaft, da sich ein Produkt oft aus vielen Teilen (Objekten) zusammensetzt. So hat eine Videokarte einen Bildspeicher, der ein fester Bestandteil dieser Karte ist. Auf der anderen Seite kann ein Objekt mit anderen in Beziehung stehen, ohne daß diese Teil des Objektes sind. Dieselbe Videokarte ist Teil eines Projektes, ohne aber physikalisch ein Teil davon zu sein. Im Gegensatz zu den physischen Teilobjekten sind die anderen Komponentenkonzepte unabhängige Komponentenobjekte. Sie können auch Komponentenobjekte anderer Objekte sein. Das ist bei den Teilobjekten nicht möglich; ein bestimmter Bildspeicher kann zu einer Zeit nur in einer Videokarte sein. Andererseits ist die Lebensdauer nicht an das umgebende Objekte gebunden. Eine Videokarte kann verschrottet (aus dem ASN gelöscht werden)

werden, der Bildspeicher aber weiterhin verwendet werden, zum Beispiel auf einem anderen Board.

In der KML werden die Schlüsselworte RELATIONS und PARTS zur Kennzeichnung der beiden Arten von Konzept—Komponentenkonzept-Beziehungen verwendet. Im ASN werden die beiden Typen von Konzept—Komponentenkonzept-Beziehungen durch die relation- beziehungsweise part of Kante dargestellt.

Um die anderen Slots, dessen Domäne einem der im KML vorhandenen Datentypen angehört, davon abzugrenzen, werden diese unter der Schlüsselwort ATTRIBUTES aufgeführt (Im ASN werden elementare Konzepte innerhalb des Slotknotens angegeben).

Die Syntax des Nichtterminals eigenschaften (siehe Kapitel 4.2.3) wird umgeändert in:

```

<eigenschaften> ::= <teile> <relationen> <attribute>
<teile> ::= PARTS : <teile_eigenschaften>
           | epsilon
<relationen> ::= RELATIONS : <relationen_eigenschaften>
                | epsilon
<attribute> ::= ATTRIBUTES : <attribut_eigenschaften>
                | epsilon
<teile_eigenschaften> ::= <teile_eigenschaft>
                           | <teile_eigenschaften>
                           | <teile_eigenschaft>
<relationen_eigenschaften> ::= <relationen_eigenschaft>
                                | <relationen_eigenschaften>
                                | <relationen_eigenschaft>
<attribut_eigenschaften> ::= <attribut_eigenschaft>
                               | <attribut_eigenschaften>
                               | <attribut_eigenschaft>
<teile_eigenschaft> ::= <slotname> : <konzeptname>
                       | SET OF <slotname> : <konzeptname>
<relationen_eigenschaft> ::= <slotname> : <konzeptname>
                              | SET OF <slotname> :
                              | <konzeptname>
<attribut_eigenschaft> ::= <slotname> : <wertebereich>
                           | SET OF <slotname> :
                           | <wertebereich>
<wertebereich> ::= NUMBER
                  | REAL

```

4 KML

```
| REAL NUMBER
| TEXT
| {YES,NO}
| OPAQUE
| OPAQUE OBJECT
```

Wenn auf der rechten Seite einer Regel ein `epsilon` steht, bedeutet daß das Nichtterminal auf der rechten Seite ohne Ersetzung aus dem Ableitungsbaum entfernt werden kann.

Die Eigenschaften eines Konzeptes werden demnach in drei Schritten angegeben. Zuerst die Komponentenobjekte, die Teil des Objektes sind, danach die Komponentenobjekte, die in einer nicht-physikalischen Beziehung zum Objekt stehen und zuletzt die Slots, die als Domäne einen Datentyp haben. Jeder der Schritte ist optional, ein Konzept kann bei Bedarf auch ohne Slots übergeben werden. Diese Konzepte sind aber keine *abstrakten* Konzepte (in Anlehnung an abstrakte Klassen in objektorientierten Sprachen). Denn auch von diesen Konzepten können ohne weiteres Instanzen generiert werden.

Das Beispiel aus Abbildung 4.3 wird in der nachfolgenden Abbildung korrekt angegeben:

Definitions:

```
Projekt: CONCEPT
RELATIONS:
  produkt: Videokarte
  mitarbeiter:
    SET OF Mitarbeiter
ATTRIBUTES:
  budget: REAL NUMBER
.
Bildspeicher: CONCEPT
ATTRIBUTES:
  typ: TEXT
.
Controller: CONCEPT
.
Videokarte: CONCEPT
PARTS:
  speicher: Bildspeicher
  controller: Controller
ATTRIBUTES:
  bezeichnung: TEXT
.
Mitarbeiter: CONCEPT
ATTRIBUTES:
  name: TEXT
.
```

Abbildung 4.4: Die Spezifizierung der Eigenschaftstypen der Konzepte

In obigen Beispiel ist das Konzept `Controller` ein Konzept ohne eigene Slots.

Der zweite, am Anfang dieses Unterabschnitts aufgeführte, grundlegende Typ von Beziehungen zwischen Konzepten, die Konzept—Unterkonzept-Beziehung, soll im nachfolgenden Unterkapitel vorgestellt werden.

4.2.5 Die Konzeptvererbung

Mit den bisher besprochenen Sprachkonstrukten von KML kann man schon einige Elemente des terminologischen Wissens von Anwendungen adäquat modellieren. Gewisse Schwächen zeigen sich aber immer noch. So sind die Zusammenhänge von Konzepten bisher nur auf Konzept—Komponentenkonzept-Beziehungen beschränkt. Die Konzept—Unterkonzept-Beziehung zur Vererbung von Slots von allgemeineren zu spezielleren Konzepten wird nun in diesem Unterabschnitt beschrieben.

Die Konzept—Unterkonzept-Beziehung wird im ASN durch den `is a` Relationentyp repräsentiert (siehe Kap. 3.2.1), der ein Konzept mit seinen Oberkonzepten verknüpft.

Diese Konzept—Unterkonzept-Beziehungen (in semantischen Datenbankmodellen und Wissensrepräsentationmethoden oft auch als *Is-a*-Beziehungen bezeichnet [Heuer92]) kann unter zwei Gesichtspunkten betrachtet werden:

Ein Konzept A ist Unter- bzw. Subkonzept eines Konzeptes B (oder B ist Ober- bzw. Superkonzept von A) wenn

- die Domäne von A (nicht unbedingt echte) Teilmenge der Domäne von B ist,
- A mindestens die Slots von B besitzt.

Da ein Unterkonzept mehr Merkmale (Slots) als sein Oberkonzept besitzt, wird das Unterkonzept auch *Erweiterung* des Oberkonzeptes genannt. Andererseits stellt jedes Unterkonzept gleichzeitig eine *Einschränkung* des Oberkonzeptes dar, da zu dem Unterkonzept nur eine Teilmenge der Objekte der Superkonzeptes gehören. Man verwendet den einen oder den anderen Begriff je nachdem, ob man mehr Wert auf die Merkmale oder auf die Domänen der Konzepte legt.

Der Mechanismus der Weitergabe von Slots von Superkonzepten an Subkonzepte wird *Vererbung* genannt. Der Terminus *Vererbung* ist angebracht, weil bei einem Unterkonzept die Slots der Oberkonzepte nicht erneut definiert werden müssen, sondern automatisch verwendet werden können. *Vererbung* bezieht sich aber nicht nur auf Slots, auch auf Regeln (siehe Kap. 4.2.8), die für einen Slots definiert wurden. Ein Unterkonzept erbt also nicht nur die Slots seiner Oberkonzepte, sondern auch die Regeln, die sich auf den Slot beziehen.

Konzepte können demnach in Vererbungshierarchien angeordnet werden. Da Konzeptbeziehungen aber nicht immer durch Baumstrukturen beschrieben werden

können, wohl aber durch gerichtete, azyklische Graphen wäre der Begriff „Netzwerk“ angebracht. Wie [Heuer92] feststellt, wird der Ausdruck „Hierarchie“ üblicherweise in der Literatur verwendet, deswegen soll auch hier von Vererbungshierarchien und nicht von Vererbungsnetzwerken gesprochen werden.

Es gibt Is-a Hierarchien in zwei Varianten: *Spezialisierungen* und *Generalisierungen*. Beide Hierarchien fixieren die Domänen der Konzepte auf unterschiedliche Art und Weise. Spezialisierungen ermitteln dabei von gegebenen Domänen der Oberkonzepte die Domäne für ein Unterkonzept, Generalisierungen von gegebenen Domänen der Unterkonzepte die eines Oberkonzeptes. Bei der Spezialisierung ist die Domäne des Unterkonzeptes die Schnittmenge der Domänen der Oberkonzepte. Bei der Generalisierung ist die Domäne des Oberkonzeptes die Vereinigung der Objektmengen der Unterkonzepte.

In den frühen Phasen des KML-Sprachentwurfs wurden beide Ansätze realisiert. Dies führte jedoch zu Verwirrungen: So wurde durch die Angabe, daß B eine Generalisierung von A ist (A:CONCEPT GENERALIZES B), nicht direkt ersichtlich, von welchen Konzepten, wenn überhaupt, A selbst erbt. Es mußte die gesamte KML-Datei oder die gesamte Datenbank nach Generalisierungangaben für A durchsucht werden. Bei der Spezialisierung tritt dieses Problem nicht auf, da direkt bei der Konzeptdefinition angegeben ist, von welchen Konzepten geerbt wird. Dadurch wird vermieden, daß zusammenhängendes Wissen zusammenhangslos verstreut wird, vielmehr wird das Wissen eines Konzeptes in KML modularisiert. Selbst wenn nicht alle Informationen zu einem Konzept an einer Stelle auftauchen, so gibt es wenigstens Referenzen darauf, wo sich weiteres Wissen befindet.

Durch Erweiterung der Syntax für die Konzeptdefinition (siehe Kapitel 4.2.2) wird die Spezialisierung eingeführt:

```

<konzept> ::= <konzeptname> : CONCEPT <eigenschaften> .
           | <konzeptname> : CONCEPT
             IS A <superkonzeptname>
               <mehrfachvererbung> <eigenschaften> .

<mehrfachvererbung> ::= <mehrfachvererbung> AND
                       <superkonzeptname>
                       | epsilon

<superkonzeptname> ::= <bezeichner>

```

Die Spezialisierung wird sehr schön durch die natürlichsprachliche Schlüsselwortkombination IS A angegeben. Die Formulierung Fahrzeughalter IS A Mensch hört sich wesentlich natürlicher und verständlicher an als: Fahrzeughalter IS SUBCONCEPT OF Mensch, wenn auch beide Sätze dieselbe Semantik besitzen.

Es stellt sich nun die Frage, was passiert, wenn Slotnamen in einer Vererbungsbeziehung mehrfach vorkommen. Existiert nur die Einfachvererbung läßt sich durch *Overriding* der von einem Oberkonzepten geerbte Slot durch den neu definierten Slot ersetzen. Bei der Mehrfachvererbung wird die Sache erschwert, weil ein Slot von verschiedenen Konzepten geerbt werden kann. Welcher dieser Slots hat dann den Vorrang? Es gibt zwar verschiedene aus objektorientierten Programmiersprachen bekannte Kollisionsstrategien, doch, werden diese vornehmlich zum Auflösen von Methodenkonflikten eingesetzt.

Ein weiteres Problem beim Overriding von Slots ist, daß beim Überschreiben von Slots diese einen anderen Wertebereich erhalten, als in einem Oberkonzept definiert. Dann gilt aber nicht mehr die Integritätsbedingung, daß alle Objekte eines spezielleren Konzeptes auch Instanzen des allgemeineren Konzeptes sind. Der neudefinierte Slot hätte im Superkonzept eine andere Bedeutung und somit wäre dieses Merkmal eines Objektes des Unterkonzeptes im Oberkonzept nicht darstellbar. Das Objekt selbst könnte damit nicht mehr ein Objekt des Oberkonzeptes sein.

Um diese Probleme zu umgehen, wurde bei der Konzeption des ASNs ein Überschreiben von Slots bei der Spezialisierung verboten. Ein Slotname darf in einer Vererbungshierarchie nur an einer Stelle definiert werden. Diese Festlegung gilt damit auch die KML.

Das Beispiel aus Abbildung 4.4 kann nun um die Konzepte `Angestellter`, `Student` und `Hiwi` (siehe Abbildung 3.4) erweitert werden:

```

DEFINITIONS:

Projekt: CONCEPT
RELATIONS:
  produkt: Videokarte
  mitarbeiter:
    SET OF Mitarbeiter
ATTRIBUTES:
  budget: REAL NUMBER
.
Bildspeicher: CONCEPT
Attributes:
  typ: TEXT
.
Controller: CONCEPT
.
Videokarte: CONCEPT
PARTS:
  speicher: Bildspeicher
  controller: Controller
ATTRIBUTES:
  bezeichnung: TEXT
.
Mitarbeiter: CONCEPT
ATTRIBUTES:
  name: TEXT
.
Angestellter: CONCEPT IS A Mitarbeiter
ATTRIBUTES:
  Bild: OPAQUE OBJECT
.
Student: CONCEPT IS A Mitarbeiter
.
Hiwi: CONCEPT IS A
      Angestellter AND Student
.

```

Abbildung 4.5: Vererbungsbeziehung einführen.

Im obigen Beispiel wurde drei Vererbungsbeziehungen eingeführt. Zum einen werden die Konzepte Angestellter und Student als Spezialisierungen von dem Konzept Mitarbeiter angegeben, zum anderen wird der Hiwi als Unterkonzept von Angestellter und Student eingeführt. Dabei stellt die Domäne von Hiwi die Schnittmenge der Domänen von Angestellter und Student dar. Alle Konzepte erben die Slots ihrer Superkonzepte. Hiwi in diesem Fall von Angestellter, Student und Mitarbeiter (einem indirektem Superkonzept von Hiwi). Zu beachten ist auch, daß Angestellter eine Erweiterung von Mitarbeiter ist, da im Unterkonzept ein weiterer Slot (nämlich Bild) eingeführt wurde. Das trifft nicht

beim Konzept `Student` zu. Dieses Konzept wird in diesem Beispiel auf dieselbe Weise beschrieben wie sein Oberkonzept, beide besitzen demnach dieselbe Domäne. Trotzdem ist `Student` kein Alias für `Mitarbeiter`. So kann einem Slot dessen Wertebereich als `Mitarbeiter` definiert wurde, eine Instanz von `Student` zugewiesen werden. Umgekehrt geht das aber nicht! Einem Slot dessen Range `Student` ist, kann kein Objekt, das als Instanz von `Mitarbeiter` erzeugt wurde, zugewiesen werden.

Allgemein gesagt, kann einem Slot immer eine Instanz eines Unterkonzeptes des Wertebereiches von diesem Slot zugewiesen werden. Dies leuchtet auch ein, wenn man sich vor Augen hält, daß die Instanzen der Unterkonzepte auch gleichzeitig Instanzen aller Oberkonzepte sind. Unterkonzepte sind Spezialisierungen der Oberkonzepte. Dies wird auch dadurch gewährleistet, daß es im ASN und damit in KML kein Overriding von Slots gibt.

Nachdem die Sprachkonstrukte von KML vorgestellt worden sind, die das terminologischen Wissen einer Anwendung beschreiben, sollen nun noch Sprachelemente, die dem assertionalen Wissen dienen, beschrieben werden.

4.2.6 Objekterzeugung

Im Objektteil oder den Objektabschnitten einer KML-Datei, die mit `OBJECT :` eingeleitet werden, werden die Instanzen der Anwendung definiert. In einem Objektabschnitt können beliebig viele Objekte vorkommen, wenigstens aber eines:

```
<objektteil> ::= <objektteil> <objekt>
                | <objekt>
```

Ein Objekt wird ähnlich definiert wie ein Konzept, jedoch wird anstelle des Schlüsselwortes `CONCEPT` das Konzept, von dem das Objekt Instanz ist, angegeben:

Wie bei Konzepten werden Objekte mit ihren konkreten Eigenschaften modularisiert. Ein Objekt mit seinen Werten (später kommen noch Regeln hinzu, siehe Kapitel 4.2.8) wird in KML als *Objektmodul* bezeichnet (dieser Begriff wird in 4.3 verdeutlicht), da das Wissen über ein Objekt an einer Stelle (eben diesem Modul) zusammengefaßt wird.

```
<objekt> ::= <objektname> : <objektkonzeptname> .
```

```
<objektname> ::= <bezeichner>
```

```
<objektkonzeptname> ::= <bezeichner>
```

Der Objektname muß eindeutig in der Anwendung sein (unique name assumption). Es kann nicht zwei Objekte desselben oder unterschiedlicher Konzepte geben, die den gleichen Namen besitzen. Durch den Objektnamen wird die Objektidentität gewahrt. Auch bei der Objektbeschreibung dient der Punkt wie bei Konzeptbeschreibungen als Terminator.

Ein Objekt kann nicht nur erzeugt, sondern auch in seinen Eigenschaften beschrieben werden. Die Slots werden wie bei der Konzeptdefinition sukzessive angegeben. Der

Unterschied ist aber, daß bei den Konzepten die Slots mit einem Wertebereich versehen werden, bei den Objekten dagegen konkrete Werte von Domänen dieser Wertebereiche stehen. Ein Objekt ist in all seinen Eigenschaften variabel, da all Slots veränderbar sind, obwohl es in einigen Fällen sicher nützlich wäre, bei der Konzeptbeschreibung einigen Slots feste und unveränderliche Werte zuzuweisen (etwas was in künftigen Erweiterungen des ASN eventuell realisiert werden wird).

Wie im Definitionsteil alle Informationen zu einem Konzept zusammengefaßt (modularisiert) wurden, werden auch im Objektteil die Slots mit ihren Slotvalues bei der Definition eines Objektes mit aufgeführt. Das gesamte assertionale Wissen einer Instanz wird dadurch modularisiert und das Wissen wird klar strukturiert: Das terminologische Wissen wird im Definitionsbereich, das assertionale Wissen im Objektteil aufgeführt. Das Konzeptwissen wird bis auf die ererbten Slots in einem Konzeptmodul beschrieben. Auf die ererbten Slots wird über die Angabe der Superkonzepte eines Konzeptes referenziert. Beim Objektwissen findet sich das gesamte Wissen im Objektmodul, selbst die geerbten Slots werden hier mit Werten versehen, falls diese einen anderen als den Defaultwert (siehe weiter unten) besitzen. Durch diese klare Strukturierung von Wissen in KML können Informationen kompakt vermittelt werden.

Die obige Syntaxangabe wird erweitert zu:

```
<objekt> ::= <objektname> : <konzeptname> <slotwerte>.
```

Slots mit ihren Werten werden folgendermaßen angegeben:

```
<slotwerte> ::= <slotwerte> <slotnamewert> : <wert>
                | epsilon
<slotnamewert> ::= <bezeichner>
<wert> ::= <objektwert> | <stringwert>
           | <numberwert> | realwert
           | <booleanwert> | <urlwert> | NONE | UNKNOWN
<objektwert> ::= <objektnamewert>
                | <objektnamewert> AND <objektwert>
<objektnamewert> ::= <bezeichner>
<stringwert> ::= " <string> "
                | " <string> " AND <stringwert>
<numberwert> ::= <number>
                | <number> AND <numberwert>
<realwert> ::= <real>
                | <real> AND <realwert>
```

```

<booleanwert> ::= YES | NO
                | YES AND <booleanwert>
                | NO AND <booleanwert>

<urlwert> ::= <url> | LINK <url>
             | <url> AND <urlwert>
             | LINK <url> AND <urlwert>

<objektname> ::= <bezeichner>

```

Die Nichtterminale `string`, `number`, `real` und `url` werden im Anhang aufgelöst. Aus Kapitel 4.2.3 sollte aber ersichtlich werden, wie diese Nichtterminale zu interpretieren sind. Zu beachten ist, daß Zeichenketten in doppelten Anführungszeichen angegeben werden (dadurch können Zeichenketten sich aus beliebigen Symbolen des ASCII-Zeichensatzes zusammensetzen).

Die Reihenfolge in der die Slots angegeben werden spielt keine Rolle, auch wenn die Wertebereiche der Slots Datentypen oder Konzepte sind oder die Slots physikalisch in den Konzepten enthalten sind oder nicht (bei der Konzeptbeschreibung gibt es diese Unterscheidung wie in Kap. 4.2.4 geschildert). Ebenso müssen nicht alle Slots des Konzeptes des zu beschreibenden Objektes oder die Slots der Superkonzepte aufgeführt werden. Den nicht angegeben, aber zum Objekt gehörenden, Slots wird automatisch der Defaultwert `UNKNOWN` zugewiesen, was soviel bedeutet wie: der entsprechende Wert des Slots ist für das Objekt (momentan) nicht bekannt. So wäre eine Zuweisung von `UNKNOWN` an den Slot `name` einer Instanz von `Mitarbeiter` (siehe Beispiel aus Abbildung 4.6) so zu verstehen, daß der Name des Mitarbeiters (noch) nicht bekannt ist. Nicht zu verwechseln mit `UNKNOWN` ist der Wert `NONE`. Hat ein Slot diesen Wert, dann bedeutet daß, das der Slot beim Objekt zwar eine Bedeutung hat, jedoch zum gegenwärtigen Zeitpunkt keinen Wert besitzt. So wie etwa für eine bestimmte Person Zeiten existieren können, in denen sie keine Arbeit besitzt. Es macht also durchaus Sinn zwischen diesen beiden elementaren Werten zu unterscheiden und nicht einen „Nullwert“ für beide Fälle einzuführen wie es in Datenbankmodellen oft der Fall ist [Heuer92].

Wurde einer Domäne eines Slot der Mengenkonstruktor (`SET OF`) vorangestellt, so werden die einzelne Werte des Slots einzelnen aufgezählt und mit `AND` verknüpft (siehe Abbildung 4.6, wo dem Slot `mitarbeiter` die beiden Objekte `Abaelard` und `Pontifar` zugewiesen werden). Dies gilt auch für Instanzen der Datentypen.

Nun kann das letzte Beispiel aus Abbildung 4.6 um einen Objektteil erweitert werden wie er schon in Abbildung 3.5 im ASN dargestellt wurde.

Die Wahl des Objektname ist lediglich zur Identifizierung des Objektes gedacht, es ist sinnvoll einen Namen zu wählen, mit dem der Benutzer in Gedanken leicht eine Verbindung zum Objekt herstellen kann. So hat der Objektname `Abaelard` sicher eine größere Aussagekraft als zum Beispiel `Ax4`. Trotzdem sollte nicht auf Slots in der Konzeptbeschreibung verzichtet werden, um den tatsächlichen Namen des Objektes zu beschreiben (Beispielsweise `vorname` im Konzept `Mitarbeiter`).

```

DEFINITIONS:
Projekt: CONCEPT
  RELATIONS:
    produkt: Videokarte
    mitarbeiter:
      SET OF Mitarbeiter
  ATTRIBUTES:
    budget: REAL NUMBER
.
Bildspeicher: CONCEPT
  Attributes:
    typ: TEXT
.
Controller: CONCEPT
.
Videokarte: CONCEPT
  PARTS:
    speicher: Bildspeicher
    controller: Controller
  ATTRIBUTES:
    bezeichnung: TEXT
.
Mitarbeiter: CONCEPT
  ATTRIBUTES:
    name: TEXT
.
Angestellter: CONCEPT IS A Mitarbeiter
  ATTRIBUTES:
    Bild: OPAQUE OBJECT
.
Student: CONCEPT IS A Mitarbeiter
.
Hiwi: CONCEPT IS A Angestellter AND Student
.
OBJECTS:
  Pontifar: Hiwi
    Bild: ftp://gx.ftp.com/pontifar.gif
.
  Videokartenprojekt: Projekt
    mitarbeiter: Pontifar AND Abaelard
.
  Abaelard: Angestellter
    name: "Peter Abaelard"
.

```

Abbildung 4.6: Objekte einführen

Wie aus dem Beispiel ersichtlich wird, können Objekte schon verwendet werden, bevor sie definiert wurden (Objekt Abaelard wird dem Slot mitarbeiter zugewiesen

und erst danach definiert). Was andererseits in KML (noch) nicht möglich ist, Instanzen zu definieren, deren Konzepte davor nicht definiert wurden.

Im ASN können Konzepte, Slots und Instanzen kommentiert werden. Wie dies in KML bewerkstelligt wird, wird im nächsten Unterabschnitt erläutert.

4.2.7 Kommentare in KML

Im ASN kann jeder Konzept-, Slot- und Instanzknoten einen Kommentar enthalten (siehe Kap. 3.2.1). Diese Kommentare bieten Anwendern die Möglichkeit natürlichsprachliche Informationen zu dem jeweiligen Repräsentationskonstrukt anzugeben beziehungsweise abzurufen. Dies soll das Verständnis für das dargestellte Wissen erleichtern. Besonders wenn mehrere Personen an einer Anwendung beteiligt sind, ist es wichtig, daß diese (möglichst) das gleiche Verständnis davon besitzen. Die Kommentare werden in der Wissensbasis gespeichert und können jederzeit gezielt daraus abgerufen werden. Andererseits wird auf den Inhalt der Kommentare von der aktiven Komponente aus nicht zugegriffen. Da zwischen Kommentaren für Konzepte, Slots und Instanzen ein Unterschied gemacht wird, lassen sich Kommentare nicht einfach durch Slots simulieren, da dann nicht mehr ersichtlich wäre, auf welchem dieser drei Repräsentationskonstrukte sich der Kommentar bezieht.

In KML wird ein Kommentar durch das Schlüsselwort `COMMENT` gefolgt von einem Doppelpunkt eingeleitet. Durch die Position an der Kommentar beginnt, wird deutlich, ob sich der Kommentar auf ein Konzept, ein Objekt oder einen Slot bezieht. Soll das Konzept kommentiert werden, folgt der Kommentar unmittelbar nach Angabe des Konzeptnamens, ein Slotkommentar folgt nach Angabe des Slots und ein Objektkommentar nach Angabe des Objektnamens.

Dies wird durch folgende Erweiterung einiger Grammatikregeln erreicht:

Für die Konzeptkommentare:

```
<konzept> ::= <konzeptname> : CONCEPT
           <konzept_kommentar> <eigenschaften> .
           | <konzeptname> : CONCEPT
             IS A <superkonzeptname> <mehrfachvererbung>
               <konzept_kommentar> <eigenschaften> .

<konzept_kommentar> ::= COMMENT : " <string> "
                    | epsilon
```

Für die Kommentare von Slots betrifft es folgende Nichtterminale:

```
<teile_eigenschaften> ::= <teile_eigenschaft>
                        <slot_kommentar>
                        | <teile_eigenschaften>
                          <teile_eigenschaft>
                          <slot_kommentar>
```

```

<relationen_eigenschaften> ::= <relationen_eigenschaft>
                               <slot_kommentar>
                               | <relationen_eigenschaften>
                                 <relationen_eigenschaft>
                                 <slot_kommentar>

<attribut_eigenschaften> ::= <attribut_eigenschaft>
                               <slot_kommentar>
                               | <attribut_eigenschaften>
                                 <attribut_eigenschaft>
                                 <slot_kommentar>

<slot_kommentar> ::= COMMENT : " <string> "
                    | epsilon

```

Und schließlich noch die Kommentare für die Instanzen:

```

<objekt> ::= <objektname> : <konzeptname>
            <objekt_kommentar> <slotwerte>.

<objekt_kommentar> ::= COMMENT : " <string> "
                    | epsilon

```

Nun lassen sich in dem Videokartenbeispiel auch natürlichsprachliche Kommentare einfügen (Abbildung 4.7). So wird festgehalten, daß Abaelard nicht nur Angestellter, sondern auch der Projektleiter ist und daß seine Sprechzeit nur Montags ab 14 Uhr ist. Diese Informationen ließen sich auch formal mit den im ASN und damit auch in KML vorhanden Möglichkeiten modellieren, aber nicht so einfach. Andererseits kann nun zum Beispiel durch das ASN nicht ermittelt werden, wer das Projekt leitet, da Kommentare eine Black Box für das ASN darstellen. Bei der Modellierung einer Anwendung muß man sich Gedanken darüber machen, wann es sinnvoll ist, Wissen in Form von Kommentaren darzustellen und wann nicht.

Ein weiterer Kommentar bezieht sich auf den Slot `speicher` des Konzeptes `Videokarte`. Es wird hier die Anregung gemacht, nur VRAM-Module für den Bildspeicher zu verwenden. Ratschläge oder Empfehlungen lassen sich ausschließlich in Kommentaren darstellen.

Im Beispiel findet sich auch ein Kommentar für ein Konzept. Zu dem Konzept `Controller` wird angegeben, daß dieser alle Videofunktionen enthält. Hier wird auch gleich der Nachteil der natürlichsprachlichen Formulierung sichtbar. Natürliche Sprache ist oft nicht eindeutig. So kann der Kommentar so verstanden werden, daß im allgemeinen ein Controller für Videokarten alle Videofunktionen enthält oder aber, daß nur die Controller, die für das Videokartenprojekt eingesetzt werden, diese Anforderung erfüllen sollen.

```

DEFINITIONS:
Projekt: CONCEPT
RELATIONS:
  produkt: Videokarte
  mitarbeiter:
    SET OF Mitarbeiter
ATTRIBUTES:
  budget: REAL NUMBER
.
Bildspeicher: CONCEPT
Attributes:
  typ: TEXT
.
Controller: CONCEPT
COMMENT: "integriert alle Videofunktionen
(Scaler, YUV-Konverter, ...)"
.
Videokarte: CONCEPT
PARTS:
  speicher: Bildspeicher
  COMMENT: "Es sollten möglichst nur VRAM-
Module verwendet werden"
  controller: Controller
ATTRIBUTES:
  bezeichnung: TEXT
.
Mitarbeiter: CONCEPT
ATTRIBUTES:
  name: TEXT.
Angestellter: CONCEPT IS A Mitarbeiter
ATTRIBUTES:
  Bild: OPAQUE OBJECT.
Student: CONCEPT IS A Mitarbeiter.
Hiwi: CONCEPT IS A Angestellter AND Student
.
OBJECTS:
Pontifar: Hiwi
  Bild: ftp://gx.ftp.com/pontifar.gif.
Videokartenprojekt: Projekt
  mitarbeiter: Pontifar AND Abaelard.
Abaelard: Angestellter
  COMMENT: "Ist Projektleiter. Sprechzeiten
nur MO ab 14 Uhr."
  name: "Peter Abaelard"
.

```

Abbildung 4.7: Beispiel um Kommentare für Konzepte, Slots und Instanzen erweitert.

Als letztes soll nun gezeigt werden wie die aktive Komponente des ASN in der Knowledge Modelling Language verwirklicht wird.

4.2.8 Die Realisierung der aktiven Komponente in KML

Der Zweck und die Aufgaben der aktiven Komponente sowie ihre Realisierung im ASN wurde bereits in Kapitel 3.2.5 vorgestellt. Hier soll noch einmal kurz das Wesentlichste erwähnt werden.

Die aktive Komponente des ASN ist in der Lage Veränderungen, die an einer Stelle im Semantischen Netz auftreten, durch das Netz zu propagieren und Aktionen auszulösen. So könnte beispielsweise ein Projektleiter vom ASN informiert werden, falls das Budget für das Projekt überschritten wird oder andere Probleme auftreten, ferner könnte das ASN Routineaufgaben automatisch erledigen, z.B. Mitarbeiter erinnern, vereinbarte Termine wahrzunehmen und nimmt damit gewissermaßen die Rolle eines Assistenten ein.

Die aktive Komponente kann u.a. folgende Aktionen in Gang setzen [Eck96]:

- Durchführung von Inferenzen auf der Wissensbasis
- Mitteilung von Nachrichten an Benutzer
- Herstellung von Kommunikationsverbindungen
- Anstoßen beliebiger Anwendungen

Durch das Durchführen von Inferenzen auf der Wissensbasis sind unter anderem regelhafte Zusammenhänge wie das automatische Berechnen von Formeln möglich. Durch das Benachrichtigen von Benutzern und die Herstellung von Kommunikationsverbindungen ist das System in der Lage Probleme, die nicht innerhalb des ASN gelöst werden können, an fachkundige Personen zu delegieren. Das ASN stellt dabei die gemeinsame Diskussions- und Arbeitsgrundlage für die Kommunikation bzw. Kooperation für die am Projekt beteiligten Personen dar.

Die aktive Komponente des ASN besteht aus Regeln, die in der Wissensbasis integriert sind. Die Regeln bestehen aus drei Komponenten, dem Event-, dem Condition- und dem Aktionsteil und werden deswegen auch als ECA-Regeln bezeichnet (siehe dazu auch Kapitel 3.2.5).

Regeln werden in KML durch `RULES:` eingeleitet. In dem anschließenden Regelabschnitt können beliebig viele, mindestens aber eine Regel stehen. Regeln können an jeden der vier Knotentypen im ASN „gehängt“ werden. Das bedeutet, es ist möglich Regeln für Konzepte, Slots, Instanzen und Slotwerte zu definieren. Es ist ohne weiteres möglich mehrere Regeln für einen Slot, ein Objekt oder ein Konzept festzulegen

Der Eventteil einer Regel, der zuständig ist, auf bestimmte Ereignisse zu reagieren wird in KML mit `ON` eingeleitet. Danach steht der Eventtyp. Im ASN werden im vorerst zwei Ereignisse unterschieden: Das Lesen eines Knotens und das Ändern eines Knotens oder der mit diesem verbundenen Relationentypen. Das Lesen wird in KML umgangssprachlich mit `READ`, das Ändern mit `MODIFY` bezeichnet. Danach muß angegeben werden auf welches Repräsentationskonstrukt sich das Ereignis bezieht. In KML werden wie schon erwähnt Konzepte und Slots in einem Konzeptmodul, Instanzen

und ihre Werte in einem Objektmodul zusammengefaßt. Um dem Grundsatz, zusammengehörige Informationen zu bündeln, zu wahren, wurden die Regeln für ein Konzept und dessen Slots in dem dazugehörigen Konzeptmodul, die Regeln für ein Objekt und dessen Slots im dazugehörigen Objektmodul integriert.

In einem Konzeptmodul wird das Schlüsselwort `CONCEPT` verwendet, wenn sich eine Regel auf das gesamte Konzept beziehen soll, sonst wird der Slotname (der ja innerhalb des Konzeptes und der Konzepthierarchie eindeutig ist) angegeben. Ähnlich sieht es in einem Objektmodul aus (diese Ähnlichkeit, im Aufbau und Struktur von Objekt- und Konzeptmodul hat den Vorteil, daß KML leichter erlernbar ist und Anwendungen, die in KML modelliert werden, schneller verstanden werden können). Bezieht sich eine Regel auf das gesamte Objekt wird dies mit dem Schlüsselwort `OBJECT` gekennzeichnet, sonst mit dem Slotnamen.

Der Eventteil folgen der Condition- und der Aktionsteil, wobei der Conditionteil optional ist. Die Condition überprüft Bedingungen, z.B. den Wert eines Slots und stellt damit einen booleschen Ausdruck dar. Trifft die Bedingung zu, wird die Aktion ausgeführt, wobei die Aktion Einfluß auf das ASN nehmen kann oder davon unabhängige Aktionen starten kann, trifft die Bedingung nicht zu, wird die Aktion nicht gestartet. Im ASN werden diese beiden Komponenten der ECA-Regeln durch Tcl-Skripte beschrieben. Da Tcl eine eigenständige Programmiersprache ist und im ASN die Tcl-Skripte nicht auf bestimmte Aufgaben beschränkt werden, müßte KML selbst zu einer Programmiersprache ausgebaut werden (wobei hier unter einer Programmiersprache eine Sprache verstanden wird, die alle berechenbaren Funktionen implementieren kann) um Tcl-Programme in KML darzustellen.

Deswegen werden Conditions und Aktionen auch in KML in Tcl realisiert. Die Tcl-Skripte für Conditions werden mit `CONDITIONS:`, die für die Aktionen mit `ACTIONS:` eingeleitet. Die Skripte werden durch den Ausdruck `END OF TCL` terminiert.

Die Grammatik wird nun erweitert, um die Syntax der Regeln zu überprüfen. Dabei werden einige Grammatikregeln vervollständigt:

```

<eigenschaften> ::= <teile> <relationen>
                  <attribute> <konzeptregeln>

<konzeptregeln> ::= RULES : <konzeptregelteil>
                  | epsilon

<konzeptregelteil> ::= <konzeptregelstruktur>
                    | <konzeptregelteil>
                    <konzeptregelstruktur>

<konzeptregelstruktur> ::= ON READ CONCEPT
                        <conditon> <action>
                        | ON READ CONCEPT <action>

```

4 KML

```
| ON MODIFY CONCEPT  
  <condition> <action>  
| ON MODIFY CONCEPT <action>  
| ON READ <slotnamerule>  
  <conditon> <action>  
| ON READ <slotnamerule>  
  <action>  
| ON MODIFY <slotnamerule>  
  <condition> <action>  
| ON MODIFY <slotnamerule>  
  <action>
```

```
<condition> ::= CONDITION : <tcl_script> END OF TCL  
  | epsilon
```

```
<action> ::= ACTION : <tcl_script> END OF TCL
```

```
<slotnamerule> ::= <bezeichner>
```

Und noch die Regeln für das Objektmodul:

```
<objekt> ::= <objektname> : <konzeptname>
```

```
<objekt_kommentar>
```

```
  <slotwerte> <objektregeln> .
```

```
<objektregeln> ::= RULES : <objektregelteil>
```

```
  | epsilon
```

```
<objektregelteil> ::= <objektregelstruktur
```

```
  | <objektregelteil>
```

```
  <objektregelstruktur>
```

```
<objektregelstruktur> ::= ON READ OBJECT
```

```
  <conditon> <action>
```

```
  | ON READ OBJECT <action>
```

```
  | ON MODIFY OBJECT
```

```
    <condition> <action>
```

```
  | ON MODIFY OBJECT <action>
```

```
  | ON READ <slotnamevaluerule>
```

```
    <conditon> <action>
```

```
  | ON READ <slotnamevaluerule>
```

```
    <action>
```

```
  | ON MODIFY <slotnamevaluerule>
```

```
    <condition> <action>
```

```
  | ON MODIFY <slotnamevaluerule>
```

```
    <action>
```

```
<slotnamvaluerule> ::= <bezeichner>
```

Das Nichtterminal `tcl_script` wird nicht weiter aufgelöst und bildet die Wurzel für den Ableitungsbaum eines Tcl-Programmes. In den Tcl-Programmen muß lediglich darauf geachtet werden, daß nie der Ausdruck „END OF TCL“ vorkommt, da der KML-Parser darin vorzeitig das Ende des Tcl-Skriptes erkennen würde.

Auf welche Weise aus Tcl-Skripten Einfluß auf das ASN genommen wird, kann hier nicht geschildert werden, da zum Zeitpunkt der Abfassung dieser Dokumentation dies nicht bekannt war. Im folgenden Beispiel werden die Tcl-Skripte deswegen nicht ausformuliert. Durch Kommentare soll aber gesagt werden was die jeweiligen Tcl-Skripte für eine Aufgabe haben. Zeilen, die mit einem #-Zeichen beginnen, gelten in Tcl als Kommentare. Im Beispiel soll vor allem gezeigt werden wie Regeln strukturiert sind und was man prinzipiell damit machen kann.

Im Beispiel aus Abbildung 4.8 wurden nur die Konzepte und Objekte aus dem letzten Beispiel (Abbildung 4.7) beibehalten, die einen Regelteil besitzen. Im Konzeptmodul über den Angestellten sind zwei Regeln angeführt, in den beiden Objektmodulen jeweils eine Regel.

```

DEFINITIONS:
...
Angestellter: CONCEPT IS A Mitarbeiter
  ATTRIBUTES:
    Bild: OPAQUE OBJECT
  RULES:
    ON MODIFY CONCEPT
      ACTION: #Benachrichtige Mitarbeiter, daß
terminologisches Wissen geändert wurde
        ... END OF TCL
    ON MODIFY Bild
      CONDITION: #Teste, ob Domäne ein Opaque
Object ist
        ... END OF TCL
      ACTION: #Meldung, daß Änderung nicht den
Typ der Domäne betreffen darf und Änderung
rückgängig machen
        ... END OF TCL
.
...
OBJECTS:
Videokartenprojekt: Projekt
  mitarbeiter: Pontifar AND Abaelard
  RULES:
    ON MODIFY mitarbeiter
      CONDITION: #Wird Mitarbeiter aus Liste
entfernt
        ... END OF TCL
      ACTION: #Lösche Instanz aus dem ASN
        ... END OF TCL
.
Abaelard: Angestellter
  COMMENT: "Ist Projektleiter. Sprechzeiten
nur MO ab 14 Uhr."
  RULES:
    ON READ OBJECT
      CONDITION: #Ist zugreifende Person nicht
Abaelard
        ... END OF TCL
      ACTION: #Melde Abaelard, daß jemand
Informationen über ihn abrufen
        ... END OF TCL
.
...

```

Abbildung 4.8: Regeln in Beispiel einführen.

Die erste Regel im Konzeptmodul hat keinen Conditionabschnitt. Wird die Konzeptbeschreibung geändert wird sofort eine Meldung an die Anwender geschickt, daß hier eine Änderung vollzogen wurde. Dies kann sinnvoll sein, denn Modifikationen im Definitionsteil können große Wirkungen auf die gesamte Wissensbasis haben (zum

Beispiel wenn ein Konzept aus einer Konzepthierarchie gelöscht wird). Die zweite Regel im Konzeptmodul verbietet Änderungen am Datentyp des Slots. Im Objektmodul Videokartenbeschreibung wird beim Zugriff auf den Slot `mitarbeiter` getestet, ob eine Instanz der Werteliste gelöscht wird. Falls dies zutrifft, wird die Instanz vollständig aus der Wissensbasis entfernt (es können in dieser Anwendung demnach nur Personen modelliert werden, die am Projekt beteiligt sind). Diese Regel ist ein Beispiel für eine Wirkkette: Eine Aktion startet weitere Aktionen im ASN. Die Aktionen der anderen Regeln haben keinen Einfluß auf das ASN und dienen mehr der Benachrichtigung der Mitarbeiter. Die letzte Regel bewirkt, daß der Projektleiter Abaelard informiert wird, wenn eine andere Person Informationen über ihn abrufen.

Damit wurden alle Sprachkonstrukte und Schlüsselworte von KML vorgestellt. Im nächsten Unterkapitel wird die Ausdrucksmächtigkeit von KML mit dem Aktiven Semantischen Netz ASN verglichen und gezeigt werden, daß diese in beiden äquivalent ist.

4.3 Äquivalenz von KML und ASN

Die Hauptaufgabe der Diplomarbeit war eine Sprache zu entwerfen (und zu implementieren), die genau das Wissen, das im ASN abgelegt ist, ausdrücken kann. Damit ist es möglich Programme zu schreiben, die Wissen aus dem ASN in die Knowledge Modelling Language und umgekehrt überführen.

Der Vergleich der Ausdrucksmächtigkeiten von ASN und KML soll umgangssprachlich erfolgen ohne formale Hilfsmittel. Zuvor muß aber etwas Vorarbeit geleistet werden, damit der Vergleich leichter zu führen ist. Dazu werden die in Kap. 4.2 eingeführten Begriffe Konzept- und Objektmodul näher beschrieben. Im wesentlichen soll der Aufbau und die Struktur, die diesen Modulen zugrunde liegt, nähergebracht werden.

Der Definitionsteil von KML besteht aus einer beliebigen Zusammenreihung von Konzeptmodulen, die über Vererbungsbeziehungen miteinander in Kontakt stehen. Diese Konzeptmodule bündeln jeweils das gesamte Wissen über ein Konzept direkt und indirekt. Indirekt in den Fällen, in denen ein Konzept die Slots und deren Regeln von den allgemeineren Konzepten erbt. In diesen Fällen werden nur die Namen der Oberkonzepte angegeben, die Beschreibung der Slots dieser Konzepte findet sich in den Konzepten, in denen diese Slots definiert wurden. Ein Konzept wäre vielleicht anschaulicher, wenn alle Slots, die das Konzept beschreiben, in dessen Konzeptmodul vorkommen würden, aber die Redundanz würde dadurch besonders bei großen Anwendungen über Gebühr zunehmen.

Ein Konzeptmodul besteht aus zwei Teilen: einem Modulkopf und einem Modulrumpf. Im Modulkopf wird der Name des Konzeptes und falls existent, seine Oberkonzepte angegeben. Im Modulrumpf stehen die Slots und die Regeln. Die Slots werden unterteilt, in solche, die eine part-of Beziehung eines Merkmals zum Konzept darstellen, solche die eine (nicht physikalischen) Beziehung zum Konzept herstellen und Slots, die einen

Datentyp als Wertebereich besitzen. Für jeden Slot wird angegeben, welcher Domäne seine Instanzen angehören können und was für eine Kardinalität das Merkmal besitzt (vorerst wird ja nur zwischen `single` und `multiple` im ASN unterschieden, siehe Kap. 3.2.2). Die Regeln für das Konzept und dessen Slots können in einer beliebigen Reihenfolge angegeben werden. Diese Schilderung ist in nachstehender Abbildung 4.9 zu sehen.

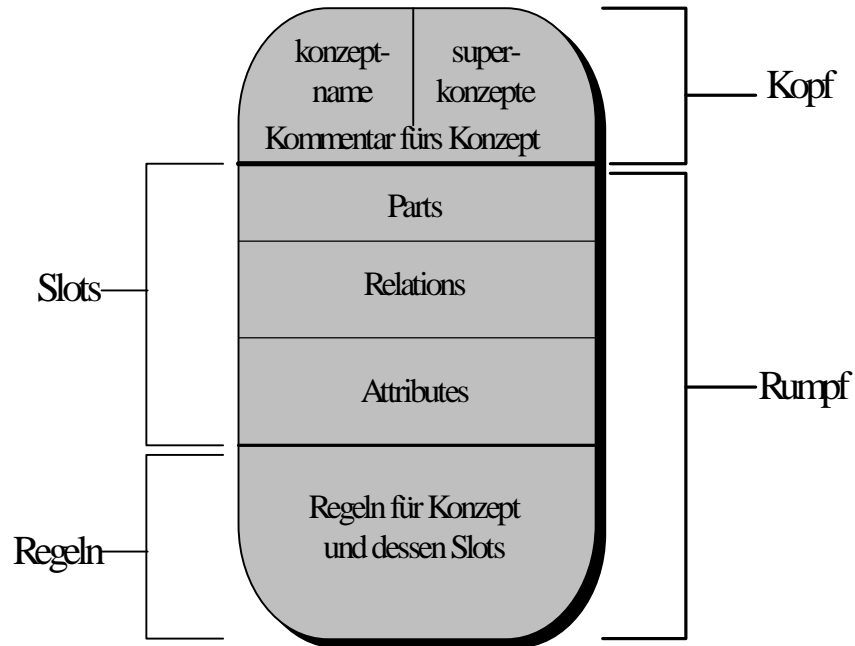


Abbildung 4.9: Struktur und Aufbau des Konzeptmoduls

Im Kopf des Konzeptmoduls kann das Konzept kommentiert werden. Genauso kann jeder Slot einen Kommentar enthalten, der der Slotbeschreibung folgt.

Das Objektmodul, welches das gesamte Wissen für eine Instanz einer Anwendung enthält (und hier wirklich alles direkt), ist ähnlich strukturiert wie das Konzeptmodul. Es besteht aus einem Kopf und einem Rumpf. Im Objektkopf steht der eindeutige Name des Objektes und das Konzept von dem es instanziiert wurde. Optional ist der Kommentar, mit dem Benutzer Zweck und Sinn des Objektes natürlichsprachlich beschreiben können. Der Objektrumpf wird gegliedert in einen Slot- und einen Regelabschnitt. Im Slotabschnitt werden die Slots und ihre Werte (wenn mehrere, dann durch AND verknüpft) im Regelabschnitt die Regeln für das Objekt und dessen Slots aufgeführt. Dabei spielt hier die Reihenfolge der Komponenten auch im Slotabschnitt keine Rolle. In welcher Beziehung diese zum Objekt stehen, wurde bereits im Konzeptmodul für das Konzept des Objektes erläutert. Das Objektmodul wird in der Abbildung 4.10 graphisch dargestellt.

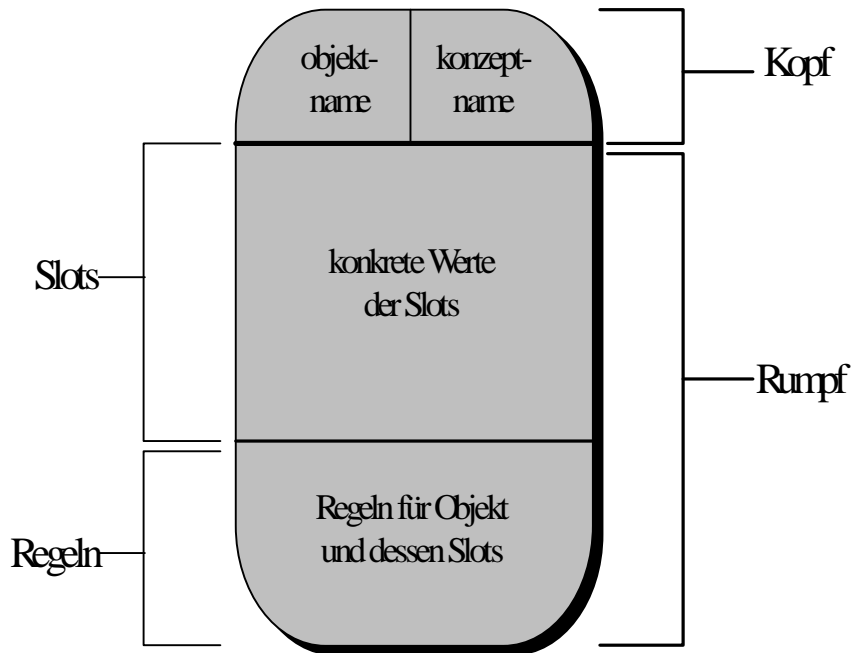


Abbildung 4.10: Struktur und Aufbau des Objektmoduls

Anhand der Beschreibung von Objekt- und Konzeptmodul wird nun gezeigt, daß die Ausdrucksmächtigkeit von KML der des Aktiven Semantischen Netzes entspricht. Dabei werden zunächst die Repräsentationsstrukturen des ASNs herangezogen (siehe dazu Kap. 3.2) und es wird beschrieben wie diese in KML realisiert werden.

Konzept- und Slotknoten werden in KML durch das Konzeptmodul dargestellt. Dadurch erübrigt sich auch die Einführung eines Sprachkonstrukts für die *has a* Relation, die Konzeptknoten mit Slotknoten verbindet. Die *is a* Relation wird im Kopf der Konzeptmoduls durch Angabe der Superkonzepte verwirklicht. Die Elemente eines Konzeptknotens (s. Tabelle 3-1), der Konzeptname und der Konzeptkommentar finden sich ebenso im Konzeptkopf.

Die Elemente eines Slotknotens und seine semantischen Beziehungen (beides in Tabelle 3-2 aufgeführt) werden im Rumpf des Konzeptmoduls realisiert. Dazu gehören bei den Knotenelementen der optionale Kommentar, der eindeutige Name innerhalb der Konzepthierarchie, die Kardinalität und gegebenenfalls der Datentyp (im ASN als elementares Konzept bezeichnet), bei den semantischen Beziehungen, die *part of* und *relation* Verbindung. In KML steht der Name eines Slots im Rumpf des Konzeptmoduls. Ist die Kardinalität *multiple*, wird die Schlüsselwortkombination *SET OF* angegeben, im anderen Fall (*single*) erfolgt keine spezielle Anweisung. Hat der Slot ein elementares Konzept als Wertebereich, wird der passende Datentyp aus KML angegeben (siehe Kap. 4.2.3). Ansonsten wird das Konzept, zu dem im ASN über die *relation* oder *part of* Kante eine Verbindung hergestellt wird, als Wertebereich angegeben. Die Semantik der Beziehung (s. Kap. 4.2.4) wird über die Einteilung der Slots in einen „Parts“- oder „Relations“-Abschnitt erreicht (s. Abbildung

4.9). Um von diesen, die Slots zu unterscheiden, die einem Typ angehören, werden diese im „Attributes“-Abschnitt zusammengefaßt. Der Kommentar zu einem Slot wird nach Definition des Slots angegeben (eingeleitet durch COMMENT:)

Instanz- und Slotwertknoten (siehe auch Tab. 3-3 und Tab. 3-4) werden durch das Objektmodul repräsentiert. Hierdurch entfällt eine eigene Darstellung der `value` Kante im ASN, die Instanzknoten und Slotwertknoten verbindet. Die `instance of Relation` wird in KML dadurch erzielt, daß im Kopf des Objektmoduls der Name des Konzeptes angegeben wird, aus dessen Domäne das Objekt stammt. Im selben Modulkopf findet sich auch noch der eindeutige Name des Objektes (Die Eindeutigkeit von Namen wird natürlich nicht durch die Sprache selbst dargestellt, vielmehr gehört diese Forderung in das Aufgabengebiet der semantischen Analyse beim Übersetzen der Sprache ins ASN). Falls zu einer Instanz ein Kommentar gegeben ist, wird dieser auch im Kopf der Objektmoduls eingefügt.

Zum Slotwertknoten gibt es zwei Relationentypen. Die `of` Kante wird in KML durch die Angabe des Slotnamens zum Slotwert im Rumpf des Objektmodul erreicht. Die Angabe des Slotnamens genügt, da dadurch ein eindeutiger Bezug zur dazugehörigen Slotbeschreibung im Konzeptmodul hergestellt wird (was im ASN die Aufgabe dieses Relationentyps ist). Mit der `value` Kante wird im ASN die Verbindung des Slotwertknotens erreicht, falls dieser Instanz eines Konzeptes ist. In KML wird der Wert eines Slots direkt nach Angabe des Slotnamens (und einem Doppelpunkt) angegeben. Dies trifft auch in dem Fall zu, wenn der Wertebereich ein Datentyp ist (in der ASN Beschreibung aus Kap. 3.2.4 steht ein elementarer Wert innerhalb des Slotwertknotens).

Nun bleibt noch zu klären, wie die aktive Komponente des ASNs in KML abgebildet wird. Regeln gibt es im ASN für Konzepte, Instanzen, Slots und Slotwerte. In KML stehen die Regeln für Konzepte und deren Slots im Regelabschnitt des Rumpfes des Konzeptmoduls, die Regeln für Instanzen und deren Slots (diese werden im ASN durch die Slotwertknoten symbolisiert) im Regelabschnitt des Rumpfes des Objektmoduls. Im ASN gibt es drei Arten von Relationen, die mit Regeln zu tun haben (s. Tabelle 3-5): die `read-event`, die `modify-event` und die `action-Relation`. Die `read-event` und die `modify-event` Kante symbolisieren ein bestimmtes Ereignis das die Abarbeitung einer Condition in Gang setzen kann. In KML wird dies durch den ersten Teil der dreiteiligen Regel (ECA-Regel) erzielt. Für die `read-event` Kante wird in KML ein `ON READ`, für die `modify` Kante ein `ON MODIFY` angegeben, gefolgt von dem Repräsentationskonstrukt auf das sich die Regel beziehen soll. Den Ereigniskanten folgt im ASN ein Tcl-Knoten in dem die Bedingung (Condition) steht, die abgeprüft werden soll. In KML steht dies im zweiten Teil der Regel (eingeleitet durch `CONDITION:`, danach das Tcl-Skript aus dem Tcl-Knoten abgeschlossen durch ein `END OF TCL`). Dieser zweite Teil ist optional, da im ASN ein Ereignis unmittelbar eine Aktion in Gang setzen kann. Falls dies nicht der Fall ist, folgt dem Tcl-Knoten für die Condition, der Tcl-Knoten mit der Aktion. Verbunden sind diese beiden Knoten durch die `action` Kante. In KML steht die Aktion im dritten Teil der Regel (eingeleitet durch

ACTION:, gefolgt vom Tcl-Script, das die Aktionen enthält und einem END OF TCL). Da der Tcl-Knoten mit der Condition im ASN optional ist, kann der zweite Teil der Regel in KML auch weggelassen werden.

Damit wurde gezeigt, daß alle Repräsentationsstrukturen und -konzepte in KML abgebildet werden können. Es stellt sich nun die Frage, ob KML vielleicht nicht ausdrucksstärker ist als das Aktive Semantische Netz.

So gibt es im ASN keine direkte Trennung von terminologischem und assertionalem Wissen (Konzeptknoten sind etwa mit Instanzknoten über eine Kante verbunden) wie in KML die Aufteilung des Wissens in einen oder mehrere Definitionsabschnitte und einen oder mehrere Objektabschnitte. Diese Trennung erhöht aber nicht die Ausdrucksfähigkeit von KML, sondern dient lediglich dem leichteren Verständnis der Anwendung. Genauso gut könnte man darauf verzichten und Konzepte und Objekte vermischen, wenn nicht ein Ziel des Sprachentwurfs gewesen wäre, eine Sprache zu konzipieren, die Wissen möglichst überschaubar und damit verständlich darstellt). Da es keine weiteren noch nicht erwähnten Sprachkonstrukte gibt (auf die es eine Abbildung vom ASN aus gibt), kann KML nicht mehr darstellen als das ASN. Damit wäre die Äquivalenz der Ausdrucksmächtigkeit von KML und ASN gezeigt.

Zusammenfassend läßt sich sagen, daß mit KML eine Sprache existiert, in der Wissen, daß durch das ASN repräsentiert wird oder werden soll, mit Hilfe von KML dargestellt werden kann. Dadurch wird die auf dem Aktive Semantische Netz basierende Wissensbasis für Anwender transparent und es existiert eine Sprache in der sich Personen, über das Wissen aus der Wissensbasis austauschen können. Die Sprache wurde so konzipiert, daß sie Wissen in einer möglichst verständlichen und überschaubaren Art darstellt. Wesentliches Konzepte des Sprachentwurfs war die Modularisierung des Wissens in Module (Konzept und Objektmodul), das Auftrennen in Definitions- und Objektabschnitte, Verwendung weniger Sprachkonstrukte und ausdrucksstärker, weil allgemeinverständlicher Schlüsselworte.

5 Update und Selektierungsoperationen

Im vorangegangenen Kapitel wurden gezeigt wie das Wissen einer Anwendung (v.a. beim Entwurfsprozeß des Rapid Prototypings) mit Hilfe von KML modelliert werden kann. In diesem Kapitel wird besprochen, wie Operationen auf einer existierenden ASN-Wissensbasis zu bewerkstelligen sind. Dabei lassen sich zwei Gruppen unterscheiden: die Anfrage- und die Update-Operationen. Bei den Anfrageoperationen geht es darum, spezielles, für den Anwender interessantes Wissen aus der Wissensbasis zu selektieren und in einer angemessenen Form darzustellen. Bei den Updateoperationen soll nachträglich der Inhalt der Wissensbasis modifiziert werden. Es soll in diesem Kapitel eine Konzeption entwickelt werden, wie sich diese Operationen (möglichst benutzerfreundlich) realisieren lassen.

5.1 Updates

Dem Rapid Prototyping liegt ein iterativer, inkrementeller und dynamischer Entwicklungsprozeß zugrunde. Es liegt in der Natur der Sache, daß die Wissensbasis für das Rapid Prototyping Stück für Stück aufgebaut wird. Es ist dabei auch nicht auszuschließen, daß vorhandene Konzepte und Ideen beiseite geschoben werden und dafür andere in den Entwicklungsprozeß eines Produktes aufgenommen werden. Um dem Rechnung zu tragen, wurde die Wissensbasis so konzipiert und realisiert, daß jedwedes Repräsentationskonstrukt des ASN (Konzept, Slot, Instanz und Slotwert), daß in irgendeiner Form in der Wissensbasis gespeichert ist, auch wieder entfernt werden kann oder aber neue in die Wissensbasis hinzugefügt werden können, falls das notwendig ist. Die ASN-Wissensbasis hat nicht den Nachteil, daß nach einmaliger Festlegung des Strukturteils oder der terminologischen Beschreibung der Anwendung diese starr und unveränderlich sind und nur noch auf konkrete Objekte und ihre Eigenschaften Einfluß genommen werden kann (wie es in Datenbanken im allgemeinen der Fall ist). Das gesamte Wissen kann jederzeit erweitert, modifiziert und gelöscht werden.

Man kann prinzipiell zwischen drei Typen von Update-Operationen unterscheiden:

- Einfügeoperationen erweitern die Wissensbasis um neues Wissen,
- Löschoptionen entfernen Wissens aus der Wissensbasis
- Änderungsoperationen ändert Ausprägungen von Wissensseinheiten (z.B. Slotwerte)

Der dritte Punkt der Aufzählung läßt sich durch eine Hintereinanderausführung einer Löscho- und einer Einfügeoperationen ersetzen, wenn diese ohne Unterbrechung

aufeinander folgen (im Sinne einer Transaktion), da eine einzelne dieser Ersatzoperationen die Wissensbasis in einen inkonsistenten Zustand überführen kann. Eine Verletzung der Integrität kann jede der Update-Operationen bewirken, wenn sie nicht im Hinblick auf das Gesamtsystem realisiert werden. So hat das Löschen eines Konzeptes Auswirkungen auf alle Instanzen des Konzeptes und alle Slots, die eine oder mehrere dieser Instanzen als Wert besitzen und alle Unterkonzepte, die von dem zu löschenden Konzept Eigenschaften übernommen haben. Wie die Konsistenz und Integrität einer ASN-Wissensbasis auch bei Anwendung von Update-Operationen gewahrt bleibt, ist aber nicht Thema dieses Kapitels (nähere Informationen dazu entnehme man [Friedrich96]).

In der ASN-API Beschreibung aus [Friedrich96] finden sich unter anderem jene Funktionen, die diese Update-Operationen durchführen. Diese Funktionen liegen allerdings als C-Funktionen vor. Damit lassen sich Updates auf der Wissensbasis nur über Programmieren in einer Programmiersprache erreichen.

Hier soll es darum gehen, zu klären wie auf einer „höheren“, benutzerfreundlicheren Ebene die Update-Operationen zu bewerkstelligen sind. Im Vordergrund stehen dabei nicht minimale Änderungen der Wissensbasis (z.B. Ändern eines Slotwertes), sondern umfassendere Modifikationen der Wissensbasis wie sie in den verschiedenen Phasen der Produktentwicklung beim Rapid Prototyping entstehen. Prinzipiell gibt es zwei Ansätze, dies zu bewerkstelligen, wie im weiteren gezeigt werden wird.

Nachdem mit KML eine Sprache konzipiert wurde, die Wissen, aus der ASN-Wissensbasis anwenderfreundlich darstellt, liegt es nahe eine Sprache zu entwerfen, die ausschließlich für Update-Operationen auf der Wissensbasis zuständig ist. So könnte der Umstand, daß ein Konzept aus der Wissensbasis gelöscht werden soll, anschaulich folgendermaßen ausgedrückt werden:

```
DELETE CONCEPT <konzeptname>
```

oder das Einfügen eines Objektes:

```
ADD OBJECT <objectname> OF CONCEPT <konzeptname>
```

Nicht so leicht ausdrücken läßt sich beispielsweise der Wunsch, die Beschreibung eines Slots zu ändern. Dabei muß nicht nur der Name des Slots und des Konzeptes angegeben werden, zu dem das durch den Slot beschriebene Merkmal gehört, sondern auch, ob die Domäne des Slots oder seine Kardinalität oder gar beides geändert werden soll.

Mit

```
MODIFY SLOT <slotname> OF CONCEPT:  
SET RANGE <konzept> AND SET CARDINALITY MULTIPLE
```

könnte ausgedrückt werden, daß ein bestimmter Slot mehrere Werte tragen kann (Kardinalität: multiple) und diese einer bestimmten Domäne angehören sollen.

Wie man hierbei schon erkennen kann, muß für jeden möglichen Update-Vorgang in der Wissensbasis ein Sprachkonstrukt existieren, mit dem sich dieser ausdrücken läßt. Genauer gesagt, muß für jeden Knoten, jeden Relationentyp und jedes Knotenelement (siehe Kap. 3.2), daß in irgendeiner Form geändert wird, ein Sprachkonstrukt zur Verfügung gestellt werden, um diese Änderung zu realisieren. Dies ist zwar immer noch benutzerfreundlicher, als das Programmieren in einer Programmiersprache, da die Sprache nur auf dieses Aufgabengebiet beschränkt ist und damit zum einen leichter zu erlernen ist und zum anderen leichter auf einer höheren, dem Menschen zugänglicheren Ebene, gestaltet werden kann. Trotzdem ist diese Vorgehensweise immer noch umständlich. Der Anwender muß nämlich explizit ausdrücken, was sich ändern soll.

Ein anderer, besserer Ansatz ist, wenn der Anwender nur angeben muß, wie das Wissen für die Anwendung nach dem neusten Stand modelliert wird beziehungsweise aussieht und sich keine Gedanken darüber machen muß, wie er den Inhalt aus der vorhandenen Wissensbasis verändern muß, daß dieser dem neusten Stand entspricht. Das setzt aber voraus, daß ein Programm existiert (hier Update-Generator genannt), daß die Veränderungen zur „alten“ Wissensmodellierung erkennt und die Wissensbasis selbständig auf den neusten Stand bringt. Soll beispielsweise ein Slot in einem Konzept nicht mehr vorhanden sein, dann könnte das so realisiert werden, daß die Konzeptbeschreibung nach neuem Stand angegeben wird (also ohne diesen Slot) und ein Update-Generator erkennt, daß darin ein Slot nicht mehr auftaucht, der in der alten Konzeptbeschreibung vorhanden war und daraufhin diesen Slot löscht.

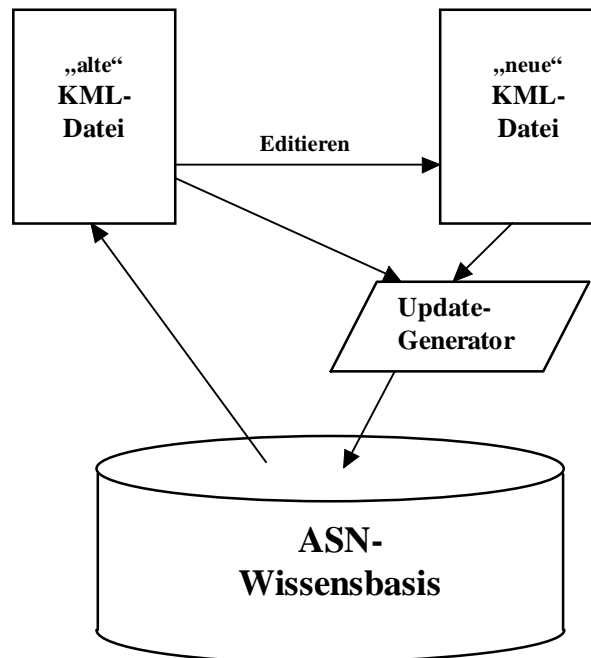


Abbildung 5.1: Vorgehensweise beim Update der Wissensbasis.

Ein Vorteil dieser Vorgehensweise ist, daß weiterhin KML verwendet werden kann, um die Update-Operationen zu realisieren. Der Anwender braucht damit keine neue Sprache erlernen oder gar in einer Programmiersprache die Änderungen durchführen. Es genügt,

wenn er sich den Inhalt der ASN-Wissensbasis in einer KML-Datei anzeigen läßt, diese Datei mit einem Editor (etwa durch Anwendung der in allen Editoren vorhandenen Kopier-, Lösch-, Einfügefunktionen) auf den neusten Stand bringt und diese dem Update-Generator übergibt. Der Generator sorgt dafür, daß in der Wissensbasis die Update-Operationen durchgeführt werden, die notwendig sind, um den Inhalt der Wissensbasis auf den neusten Stand zu bringen. In Abbildung 5.1 ist diese Vorgehensweise graphisch illustriert.

Es soll nun noch genauer spezifiziert werden, was für konkrete Operationen auf der Wissensbasis der Update-Generator erkennen muß und wie er diese im Prinzip realisieren kann (Es geht dabei nicht darum, zu zeigen wie der Generator implementiert wird, sondern daß sich dieser überhaupt realisieren läßt). Änderungsoperationen lassen sich wie schon weiter oben im Text erwähnt, durch die Hintereinanderausführung einer Lösch- und Einfügeoperation darstellen. Statt zu sagen, der Wert eines Slots ändert sich, könnte auch gesagt werden, der aktuelle Wert des Slots wird gelöscht und ein neuer eingefügt. Deswegen genügt es für den Update-Generator nur nach den Lösch- und Einfügeoperationen in der neuen KML-Datei zu suchen.

Updates können sich auf

- Konzepte,
- Slots,
- Slotbeschreibungen: Kardinalität, Wertebereich, Art der Beziehung zum Konzept (Part-of, Relation, Datentyp),
- Objekte,
- Vererbungsbeziehungen,
- Slotwerte,
- Kommentare für Konzepte, Slots, Objekte und
- Regeln für Konzepte, Slots von Konzepten, Objekte, Slots von Objekten

beziehen.

Für jedes dieser Elemente gibt es in der ASN-API Funktionen, die diese löschen als auch einfügen können.

Der Update-Generator benötigt die zwei KML-Dateien (siehe auch Abbildung 5.1), eine die den aktuellen Inhalt der Wissensbasis repräsentiert („alte“ KML-Datei) und eine die den neusten Stand der Anwendung modelliert („neue“ KML-Datei). Durch Vergleich dieser beiden Dateien muß der Generator erkennen, welche Update-Operationen auf der Wissensbasis durchzuführen sind.

Prinzipiell könnte der Generator in zwei Schritten vorgehen. Im ersten könnte er ermitteln, welche der in der obigen Aufzählung genannten Elemente zwar in der „alten“ Datei vorkommen nicht aber in der „neuen“. Diese werden daraufhin aus der

Wissensbasis gelöscht. Im nächsten Schritt könnten die Elemente gesucht werden, die in der „neuen“ Datei auftauchen, nicht aber in der „alten“. Diese Elemente müßten in die Wissensbasis aufgenommen werden.

Ein simples Beispiel soll dies verdeutlichen:

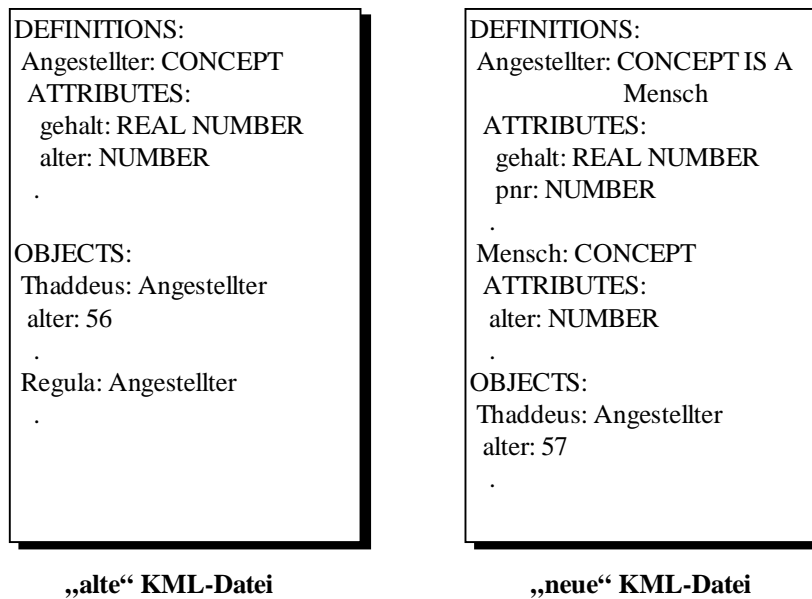


Abbildung 5.2: Update-Operationen durchführen.

Auf der linken Seite in der Abbildung 5.2 steht die KML-Datei, die den aktuellen Inhalt einer ASN-Wissensbasis anzeigt, rechts ist angegeben wie das Wissen über eine Anwendung nach dem neusten Stand aussieht. So wurde ein allgemeineres Konzept zu Angestellter eingeführt, nämlich Mensch. Da jeder Mensch ein Alter hat wurde auch der Slot von alter in das übergeordnete Konzept überführt. Ferner spielt die Angestellte Regula in der Anwendung keine Rolle mehr und ist in der neuen KML-Datei nicht mehr als Instanz aufgeführt. Das Alter von Thaddeus wurde auf den neusten Stand gebracht.

Der Update-Generator erkennt durch Vergleich der beiden Dateien, daß das Objekt Regula nicht mehr in der neuen Version vorkommt und entfernt es aus der Wissensbasis. Das gleiche gilt für den Slotwert 56, auch dieser wird in der Wissensbasis gelöscht. Genauso wie der Slot alter im Konzept Angestellter, da er in der neuen Konzeptbeschreibung dieses Konzeptes nicht mehr auftaucht.

Einfügen in die Wissensbasis muß er dagegen das Konzept Mensch mit all seinen Slots, da es in der „alten“ Datei nicht vorkommt. Ebenso muß die Vererbungsbeziehung zwischen den Konzepten Angestellter und Mensch im ASN für diese Anwendung repräsentiert werden. Der Slot pnr (für Personalnummer) von Angestellter ist neu und wird eingefügt. Beim Objekt Thaddeus gibt es noch den Slotwert 57, der nicht in

der „alten“ Datei vorkommt und deswegen noch als Wert des Slots `alter` der Instanz `Thaddeus` in die Wissensbasis eingefügt werden muß.

Der Einsatz eines Update-Generators ist in den Fällen gerechtfertigt, in denen größere Änderungen und Erweiterungen sowie komplexere Operationen wie Abwandlungen in den Konzepthierarchien in der Anwendungsmodellierung aufgetreten sind. Bei kleineren Modifikationen ist von Nachteil, daß ein Anwender immer das gesamte Wissen aus der Wissensbasis in Form einer KML-Datei erhält. In dieser muß er sich zunächst zurechtfinden und die Stellen suchen, die verändert werden sollen. Besser wäre es, wenn es speziell für das Einfügen neuer Konzepte und Objekte eine Möglichkeit geben würde, dies leichter zu arrangieren. Denkbar wäre, in einer graphischen Oberfläche für das ASN, Buttons einzufügen, mit denen der Wunsch nach Einfügen von Konzepten oder Objekten ausgedrückt werden kann. Nach Betätigung dieser Schalter könnte ein Editor oder eine Maske erscheinen, in denen die Beschreibung (z.B. in KML) des einzufügenden Konzeptes oder Objektes angegeben wird. Auch für das Löschen von Objekten und Konzepten könnte komfortable Wege gefunden werden, diese zu entfernen. Dies sind aber nur Realisierungen von Teilaufgaben, die bei Update-Operationen anfallen. Der Update-Generator ist von Vorteil, weil sich damit jede Modifikation auf einheitliche Weise ausdrücken läßt.

Nach Besprechung von Update-Operationen auf der Wissensbasis soll im nächsten Unterkapitel auf den zweiten Typ von Operationen auf der Wissensbasis, den Selektierungs- oder Anfrageoperationen, eingegangen werden.

5.2 Anfragen

Die Wissensbasen von Anwendungen für die Produktentwicklung enthalten oft große Mengen von Wissen. Den Inhalt solcher Wissensbasen in KML darzustellen und anzuzeigen, hat wenig Sinn, denn die Informationsmenge ist zu groß, als daß eine Person die Übersicht darüber behalten könnte. Meist interessiert den Anwender auch nicht was alles in der Wissensbasis abgelegt wurde, vielmehr hat er Interesse an Teilwissen. So möchte er beispielsweise nur wissen, welche Objekte eine bestimmte Eigenschaft erfüllen und nicht etwa wie die Konzepthierarchie aussieht. Es muß daher eine Möglichkeit geben, damit Anwender ganz gezielt auf spezielle, sie interessierende Informationen aus der Wissensbasis zugreifen können. Das geschieht über Anfrageoperationen, die nur das gesuchte Wissen aus der Wissensbasis ermitteln und anzeigen.

Es soll hier geklärt werden,

1. was für Selektionsoperationen benötigt werden,
2. wie die Ergebnisse der Anfragen dargestellt werden,
3. wie diese Operationen realisiert werden
4. und welchen Anforderungen sie genügen müssen.

Die elementaren Selektions- oder Anfrageoperationen auf einer Wissensbasis sind im folgenden aufgelistet [Reimer89]:

- Die Konzepte in der Wissensbasis anzeigen
- Die Objekte in der Wissensbasis anzeigen
- Die Slots eines Konzeptes anzeigen
- Die Menge aller Einträge eines Slots eines Objektes anzeigen.

Hierbei muß dem Benutzer die Möglichkeit gegeben werden, über den Namen hinaus genauere Informationen über Konzepte, Objekte und Slots zu erhalten, falls er dies wünscht (wie das in der Anfragesprache von BACK der Fall ist, siehe Kap. 2.8. und [PeScKiQu89]), zum Beispiel, welche Superkonzepte ein Konzept besitzt, welche Aktionen beim Zugriff auf ein Objekt gestartet werden (können) (siehe Kap. 3.2.5). Optimal wäre es, wenn alle Repräsentationsstrukturen und -konzepte, die das ASN enthält, gezielt abgefragt werden könnten [Reimer89].

Überdies sollte es die Möglichkeit geben, Eigenschaften anzugeben, die die Repräsentationsstrukturen erfüllen sollten, damit sie angezeigt werden (Ähnlich der Projektion in der Relationenalgebra, siehe [Vossen87]). Beispielsweise alle Objekte eines bestimmten Konzeptes anzeigen, bei denen ein spezifizierter Slot einen vorgegebenen Wert enthält. Dabei muß man beachten, daß zu jedem Datentyp und Konzept bestimmte Funktionen vordefiniert sind, wie zum Beispiel die booleschen Funktionen „gleich“, „kleiner“, „größer gleich“... . Bei Konzepten gibt es nur den Namensvergleich. Es macht nämlich wenig Sinn darüber zu reden, daß ein Konzept „kleiner“ als ein anderes ist. Nicht so bei Zahlen, bei denen eine größere Zahl von booleschen Funktionen vordefiniert ist. Diese Menge von definierten Funktionen und Operatoren sollten in den Selektionsoperatoren ausgedrückt werden können.

Neben Anfragen, die bestimmtes Wissen aus der ASN-Wissensbasis selektieren, gibt es noch eine andere Art von Anfrageoperationen, die *Existenzoperationen*. Diese überprüfen die Wissensbasis nach bestimmten Eigenschaften und geben den Wahrheitswert „True“ zurück, falls die Suche erfolgreich verlief, sonst „False“. Eine typische Anfrage dieser Kategorie könnte lauten „Gibt es Instanzen zu einem bestimmten Konzept?“ .

Diese Existenzoperationen sind aber nicht notwendig, denn sie können auf die anderen allgemeineren Anfragen zurückgeführt werden. Die Anfrage „Gibt es Instanzen zu einem bestimmten Konzept?“ könnte ersetzt durch „Zeig mir alle Instanzen eines bestimmten Konzeptes“ werden. Nur die Art der Rückgabe wäre dabei anderes. Einem „True“ würde entsprechen, daß (in diesem konkreten Fall) Instanzen als Ergebnis aufgeführt werden, während ein „False“ dem Fall entspricht, daß keine Instanzen zurückgegeben werden. Es muß dabei nur sichergestellt werden, daß der Benutzer merkt, wann die Anfrage abgearbeitet wurde.

Nachdem geklärt wurde, was für Selektionsoperationen benötigt werden, soll nachfolgend geschildert werden, wie die Rückgabe der Operationen dargestellt wird.

Will ein Benutzer wissen, welche Konzepte in der Anwendungsmodellierung auftreten, könnte ihm eine entsprechende Anfrageoperation alle Konzeptnamen in der Wissensbasis suchen und diese anzeigen (beispielsweise in runden Klammern, getrennt durch Kommata), ebenso bei Objekten. Interessiert sich ein Anwender für einen Kommentar zu einem Objekt, könnte das Ergebnis der Anfrage eine String sein, bei bestimmten Slots ein oder mehrere Zahlen, Je nach Anfrage könnte das Ergebnis unterschiedlich dargestellt werden. Komplizierter wird es, wenn die Rückgabe der Selektionsoperationen keine atomare Werte liefert, sondern Elemente mit innerer Struktur, zum Beispiel ein Konzept mit seinen Slots, oder ein Objekt mit seinen konkreten Eigenschaften. Hier könnten für verschiedene Ergebnistypen eigene Darstellungen entwickelt werden. Besser ist es aber, eine einheitliche Darstellung für alle möglichen Ergebnistypen der Anfrageoperationen bereitzustellen. Dafür bietet sich die Verwendung der Wissensmodellierungssprache für das Aktive Semantische Netz KML an. Jedes Ergebnis einer Anfrageoperation wird in dieser Sprache ausgedrückt. Die Sprache kann alle Repräsentationsstrukturen -und konzepte des ASN darstellen und damit auch alle Arten von Rückgaben von Anfragen auf eine ASN-Wissensbasis ausdrücken. Ein Vorteil der Wahl von KML ist, daß alle Anfrageoperationen abgeschlossen wären, ein Ergebnis immer in KML vorliegt und dadurch sich Anfrageoperationen schachteln ließen, auf dem Ergebnis einer Anfrage also eine neue Anfrage gestartet werden könnte. Ein anderer Vorteil ist die intuitiv verständliche Darstellung von Wissen durch KML.

Nimmt man das Beispiel aus Abbildung 4.7 als Grundlage, dann würde eine Anfrage nach allen Objekten aus der Wissensbasis für das Videokartenprojekt folgendes Ergebnis liefern:

Objects:

Pontifar: Hiwi

.

Videokartenprojekt: Projekt

.

Abaelard: Angestellter

.

Auf den Ergebnis dieser Anfrage könnte nun jede andere Anfrageoperation gestartet werden, z.B. das Suchen aller Objekte die dem Konzept Angestellter angehören. Das Ergebnis wäre

Objects:

Pontifar: Hiwi

.

Abaelard: Angestellter

(Auch Hiwis sind Angestellte, Hiwi wurde im Beispiel als Unterkonzept von Angestellter definiert.). Da jede Anfrageoperation mit jeder anderen kombiniert werden kann, sind diese orthogonal zueinander.

Die Vorteile der Verwendung von KML als „Ergebnissprache“ der Selektionsoperationen ist offensichtlich, aus diesem Grund wird auch in der Realisierung der Anfrageoperationen, das Ergebnis immer in KML ausgedrückt.

Nun stellt sich die Frage wie die Anfrageoperationen realisiert werden. Eine Möglichkeit ist die Verwendung einer eigenen Anfragesprache. Der Wunsch alle Objekte eines Konzeptes zu sehen könnte etwa so formuliert werden (in eigener SQL-nahen Syntax):

```
SELECT ALL OBJECTS FROM <Conceptname>
```

Die Anfragesprache müßte die Möglichkeiten bieten, gezielt auf Konzepte, Objekte und Slots zuzugreifen. Dazu müßten Eigenschaften dieser Repräsentationskonstrukte spezifiziert werden können. Je gezielter bestimmtes Wissen selektiert werden soll, um so ausdrucksstärker muß die Sprache werden. Der Nachteil ist, daß Anwender die Sprache lernen müssen.

Ein anderer Ansatz ist, die möglichen Selektionsoperationen dem Anwender über ein Menü anzubieten. Wichtigen Anfragen werden durch einen Menüpunkt repräsentiert (z.B. das Anzeigen aller Objekte, oder die Vererbungshierarchie). Er wählt die Operation aus, die er benötigt. Falls die Operation Parameter benötigt, werden diese vom Benutzer interaktiv abgefragt (beispielsweise den Namen des Konzeptes, dessen Instanzen gezeigt werden sollen). Der Benutzer braucht sich dabei keine Gedanken zu machen, was für Anfragen überhaupt möglich, wie diese zu formulieren sind, welche Syntax- und Semantikregeln einzuhalten sind, usw. Er muß lediglich in einer Liste die Operation auswählen, die er benötigt und wird falls notwendig, um weitere Angaben gebeten (parametrisierte Anfragen). Dieser Ansatz ist benutzerfreundlicher und wurde in der Diplomarbeit gewählt (siehe das Show-Window in Kap. 6). Wie ausführlich die Ausgabe erfolgen soll, kann über ein weiteres Menü festgelegt werden. Sollen etwa nur der Modulkopf gezeigt werden oder auch die Slots, sollen die Kommentare (falls vorhanden) angezeigt werden, usw.

6 Graphische Benutzeroberfläche

Die graphische Benutzeroberfläche *KML-GUI* integriert die verschiedenen Programme, die in der Diplomarbeit entwickelt wurden, unter einer Oberfläche. Dem Anwender bleibt es erspart, sich mit diversen Tools auseinanderzusetzen, diese bleiben vielmehr transparent. Für den Anwender ist es nicht weiter interessant, daß zwischen ASN-Wissensbasis und KML-Datei Konvertierungsprogramme zwischengeschaltet sind (siehe Kap. 7.3). Für ihn ist entscheidend, daß er eine Wissensbasis auswählen und/oder erzeugen kann, daß er eine Sprache hat (hier KML) mit der er eine Anwendung modellieren kann und die verwendet wird, um den Inhalt der Wissensbasis anzuzeigen. Diese Aufgabe erfüllt die KML-GUI.

Im weiteren Verlauf dieses Kapitels sollen die Windows der KML-GUI vorgestellt, deren Aufgabe beschrieben und deren Bedienung erklärt werden. Zur Verdeutlichung werden die auftretenden Windows in dieser Dokumentation illustriert. Dabei ist zu erwähnen, daß die Begriffe „Window“ und „Fenster“ hier dasselbe bedeuten sollen (das ist in der Literatur nicht immer so, s. [GoKaKeZh92]) und zwar ein Bildelement, das mit dem Window-Manager hin und hergeschoben werden kann.

6.1 Das Control panel

Nach dem Start (Eingabe von KML-GUI) erscheint ein Fenster, das *Control panel*, mit Menüleiste und einer Zeichnung mit einem Behälter, der eine ASN-Wissensbasis symbolisieren soll. Wurde eine Wissensbasis ausgewählt (siehe weiter unten), steht in dem Behälter der Name der Wissensbasis (siehe Abbildung 6.1).

Das Fenster ist das Steuerungsfenster der KML-GUI. Über dieses können Wissensbasen selektiert oder erzeugt werden, ein Editor aufgerufen werden, der den Inhalt in KML-Notation anzeigt und mit dem Veränderungen und Erweiterungen des Anwendungswissen erreicht werden können. Außerdem kann ein Window gestartet werden, mit dem (eine begrenzte Menge von) Anfragen auf die Wissensbasis gestellt werden können und das gesamte Programm kontrolliert abgebrochen werden.

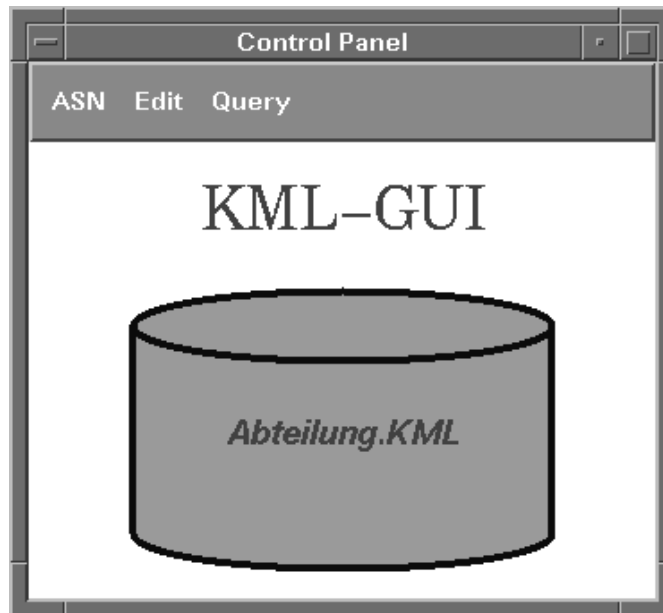


Abbildung 6.1: Das Hauptfenster der KML-GUI.

Die Funktionalität des Control panels ist über die Menüleiste abzurufen, die nun vorgestellt werden soll

6.1.1 Menüs und Schaltflächen des Control panels

In der nachstehenden Abbildung 6.2 werden die Menüs des Control panels und deren Schaltflächen angezeigt.

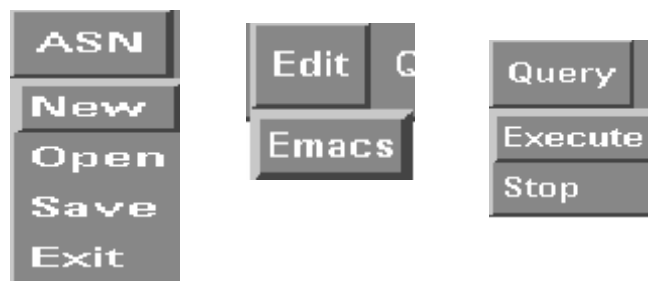


Abbildung 6.2: Menüs und Schaltflächen des Steuerungsfenster.

Die Namen der Schaltflächen und der Menüs werden im weiteren **fett** geschrieben.

Das Menü **ASN** koppelt die KML-GUI mit einer ASN-Wissensbasis, hier finden auch die wichtigen Umsetzungsprozesse von KML nach ASN und vice versa statt, die Benutzer in Gang setzen, ohne sich darüber bewußt sein zu müssen. Er sieht nur die in der nachfolgenden Aufzählung aufgeführten Menüpunkte:

- **New**
damit kann eine neue ASN-Wissensbasis erzeugt werden. Beim Aktivieren dieses Buttons erscheint ein Dateiauswahlfenster (siehe Abbildung 6.3). Dieses Fenster,

auch File Selection Box genannt, hat bei diesem Menüpunkt vor allem den Zweck, das Verzeichnis auszuwählen, in dem die Wissensbasis abgelegt werden soll. Wird eine vorhandene Datei selektiert, erscheint ein weiteres Dialog Widget (siehe Kap. 7.1.2), eine Message Box (siehe Abbildung 6.4), die dem Benutzer meldet, daß die Datei bereits existiert und die Eingabe wird zurückgewiesen. Dies hat den Zweck, das versehentliche Überschreiben einer Wissensbasis zu verhindern. Der Name der neuen Wissensbasis wird in den Behälter der Zeichnung (s. Abbildung 6.1) des Steuerungsfensters geschrieben.

- **Open**

mit Open kann eine bereits existierende Wissensbasis geöffnet werden. Auch hier erscheint eine File Selection Box (Abbildung 6.3) zur komfortablen Auswahl von Pfad und Datei. Wird ein Name eingegeben, der nicht existiert, wird keine neue Wissensbasis erzeugt, sondern die Eingabe zurückgewiesen, indem mit Hilfe einer Message Box (s. Abbildung 6.5), dem Anwender die fehlerhafte Eingabe mitgeteilt wird. Andernfalls erfolgt, versteckt für den Anwender, die Überführung des Wissens aus der ASN-Wissensbasis nach KML. Dazu wird eine temporäre Datei angelegt, die aber (noch) nicht angezeigt wird. Der Name der selektierten Wissensbasis wird in der Zeichnung im Behälter angezeigt.

- **Save**

dieser Menüpunkt überführt den Inhalt der temporären KML-Datei in die durch den aktuellen Namen (angezeigt im Behälter der Zeichnung) spezifizierte Wissensbasis. Existiert die Wissensbasis noch nicht, wird sie neu angelegt, im anderen Fall tritt der Update-Generator in Aktion (siehe Kap. 5.2) und bringt die Wissensbasis auf den neusten Stand. Wird der Menüpunkt aufgerufen, ohne daß eine Wissensbasis selektiert wurde, wird dies über eine Message Box mitgeteilt (s. Abbildung 6.6).

- **Exit**

mit Exit wird das Programm verlassen, dabei werden alle geöffneten Fenster geschlossen und temporäre Dateien gelöscht. Eine Ausnahme bildet der Editor (s. nächstes Menü). Dieser wird zwar über das Control panel gestartet, ist danach aber ein eigenständiges Programm, das lediglich über die temporäre Datei mit der KML-GUI verbunden ist und muß explizit gelöscht werden.

Durch Öffnen einer Wissensbasis wird diese in eine KML-Datei überführt, diese bleibt dem Anwender aber verborgen, solange er nicht einen Editor auswählt, um diese Datei anzuschauen. Das geschieht über das Menü **Edit**. Vorerst kann damit nur der *Emacs* aufgerufen werden, der allerdings auf allen Unix-Systemen vorhanden sein sollte. Im Prinzip lassen sich aber beliebige Editoren oder Textverarbeitungsprogramme ohne großen Aufwand in das Menü aufnehmen.

- **Emacs**

durch Betätigung dieses Schaltfläche, wird Emacs aufgerufen und zeigt die temporäre Datei an, die entweder leer ist (wenn eine neue Wissensbasis erzeugt wird)

oder aber das Wissen aus einer selektierten Wissensbasis enthält. Ist noch keine Wissensbasis angegeben worden, wird dies über das Fenster aus Abbildung 6.6 angezeigt. Mit dem Editor kann die schon oben erwähnte temporäre Datei editiert werden, beispielsweise können neue Objekte und Konzepte eingetragen werden, Slotwerte geändert werden, Regeln gelöscht werden, Soll der Inhalt der Datei in die Wissensbasis übertragen werden, muß zunächst die temporäre Datei innerhalb des Editors gespeichert werden und danach der entsprechende Menüpunkt im ASN Menü aktiviert werden. Tritt bei der Übersetzung der KML-Datei ein Fehler auf (etwa wegen falscher Syntax) so wird eine Fehlermeldung über die Art des Fehler und die Zeile in der es sich befindet, in einem eigenen Window angezeigt (s. Abbildung 6.7). Natürlich muß berücksichtigt werden, daß nicht jeder Fehler vom Übersetzer richtig interpretiert werden kann [AhSeUI92] und die Fehlermeldung oft nur Anhaltspunkte gibt. Die Übersetzung wird bei Auftreten eines Fehlers sofort unterbrochen und kann nach erst Behebung der Fehlerquelle wieder aufgenommen werden. Soll der Editor verlassen werden, geschieht das über die für den Editor übliche Weise.

Da der Umfang der Wissensbasis sehr groß werden kann und andererseits oft nur Teilwissen von Interesse ist, wurde ein eigenes Fenster geschaffen, daß die Informationen aus der Wissensbasis anzeigt, die für den Anwender von Bedeutung ist. Aktiviert werden kann das Fenster, über das Menü **Query**. Das Menü enthält zwei Menüpunkte:

- **Execute**
mit Execute wird das *Show*-Window aktiviert, über das der Benutzer Selektionen auf der Wissensmenge aufnehmen kann. Näher beschrieben wird das Fenster und dessen Funktionalität in Kap. 6.2. Es soll noch erwähnt werden, daß das Aktivieren dieses Buttons, ohne vorherige Selektion einer Wissensbasis, zum Erscheinen des Windows aus Abbildung 6.6 führt.
- **Stop**
mit diesem Button wird ein Show-Window deaktiviert. Das gleiche wird über den Menüpunkt exit aus dem ASN-Menü erreicht, jedoch wird in diesem Fall das gesamte Programm terminiert.

Im nächsten Unterabschnitt sind die Dialog Widgets (s. Kap. 7.1.2) aufgeführt, die beim Arbeiten mit dem Control panel erscheinen können.

6.1.2 Dialog Widgets des Control panels

Dialog Widgets haben den Zweck, auf Eingaben des Benutzers zu reagieren. Mit ihnen können Texte oder Kommandos eingegeben werden, oder Parameter eingestellt, Optionen ausgewählt oder sonstwie die Applikation gesteuert werden. Diese Dialogfenster sind immer dem Hauptprogramm untergeordnet (so werden bei der Ikonifizierung des Hauptfensters auch alle untergeordneten Dialogfenster unsichtbar

gemacht) und in der KML-GUI so eingestellt, daß sie, solange sie sichtbar sind, den Zugriff auf die anderen Fenster der Applikationen verweigern.

Beim Arbeiten mit dem Control panel können folgende Dialogboxen erscheinen:

Die File Selection Box (Abbildung 6.3) dient der Auswahl einer Datei aus einem beliebigen Verzeichnis. Der gesamte Leistungsumfang soll hier nicht beschrieben werden, dieser kann der Fachliteratur (beispielsweise [GoKaKeZh92]) entnommen werden.

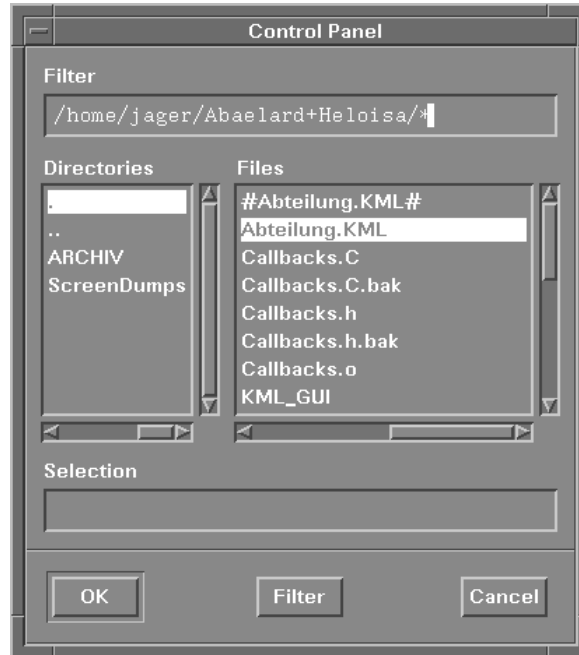


Abbildung 6.3: Das Dateiauswahlfenster, nach Aktivierung von New oder Open.

Message Boxen liefern, wie der Name schon sagt, Meldungen an den Benutzer. Der genaue Text wird in der Applikation festgelegt. Vier Message Boxen können beim Arbeiten mit der KML-GUI auftreten. Die aus Abbildung 6.4 wird angezeigt, wenn der Anwender eine neue Wissensbasis erzeugen will, aber (versehentlich !?) eine bereits existierende auswählt.

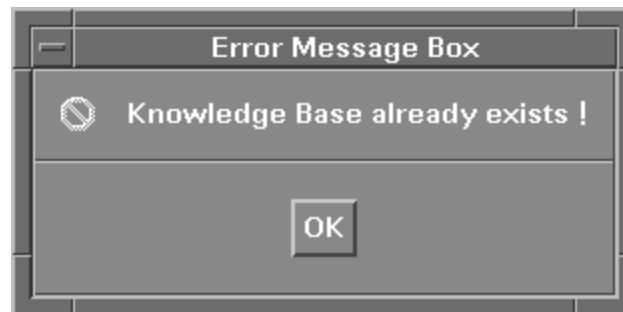


Abbildung 6.4: Message Box, zur Anzeige, daß Wissensbasis bereits existiert.

Im Falle, daß der Anwender eine Wissensbasis öffnen will, die noch gar nicht existiert (oder möglicherweise in einem andern Verzeichnis liegt) wird die Message Box aus Abbildung 6.5 aktiviert.



Abbildung 6.5: Message Box, zur Anzeige, daß Wissensbasis nicht existiert.

Schließlich kann die Situation eintreten, daß der Editor gestartet werden soll, oder die Save-Schaltfläche aus dem Menü ASN gedrückt wird (und damit die Übersetzung von KML nach ASN gestartet wird), obwohl noch gar keine Wissensbasis selektiert wurde. In dieser Situation wird dem Anwender die Message Box aus Abbildung 6.6 angezeigt.

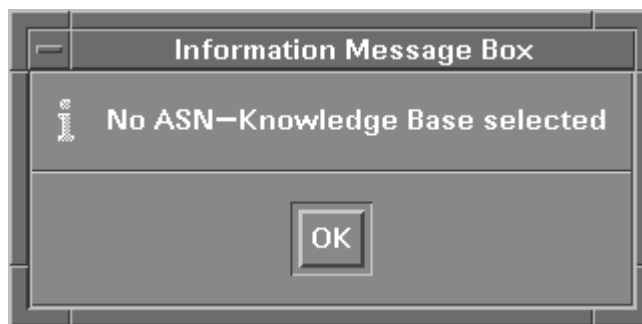


Abbildung 6.6: Anzeige, daß keine Wissensbasis selektiert wurde.

Tritt ein Fehler bei der Übersetzung auf, wird eine spezielle Message Box, ersichtlich am Logo vor dem Text, auf dem Bildschirm sichtbar, die dem Benutzer anzeigt, in welcher Zeile der temporären KML-Datei, welche Art von Fehler aufgetreten ist. Diese „Error“ Box ist in Abbildung 6.7 zu sehen.

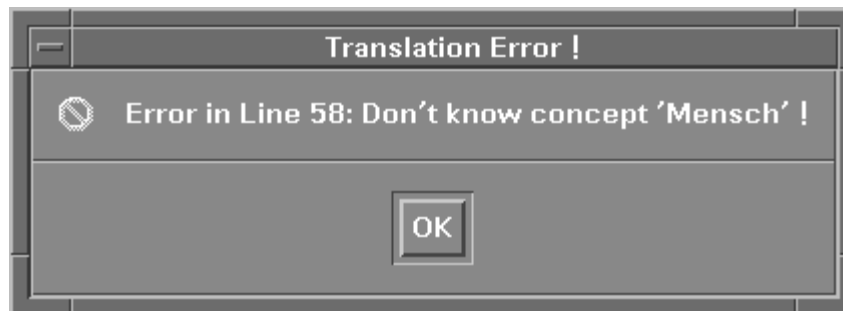


Abbildung 6.7: mögliche Fehlermeldung beim Übersetzen einer KML-Datei.

Solange die Message Boxen sichtbar sind, kann keine andere Funktion der KML-GUI gestartet werden.

6.2 Das Show-Window

Das *Show-Window* wird durch Betätigen der Execute-Schaltfläche im Query-Menü des Control panels gestartet, falls eine Wissensbasis selektiert wurde. Das Show-Window ist in Abbildung 6.8 dargestellt. Es enthält einen Ausschnitt aus einer Anwendung.

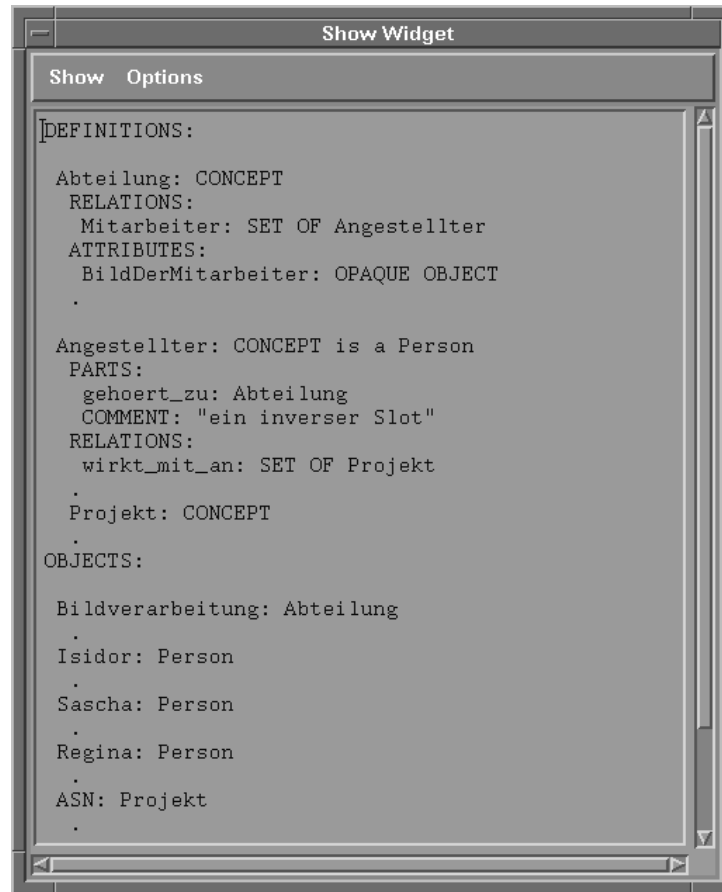


Abbildung 6.8: Das Show-Window in Aktion.

Das Fenster kann in zwei Bereiche unterteilt werden: Der Menüleiste und dem Darstellungsbereich. Im Darstellungsbereich kann nicht editiert werden, er dient lediglich der Anzeige von Wissen aus der aktuellen Wissensbasis. Da oftmals nur ein Teil der Datenmenge im Fenster sichtbar ist, können mit Hilfe der Scrollbars durch (vertikale und horizontale) Verschiebung nach und nach alle Daten sichtbar gemacht werden. Deaktiviert wird das Fenster durch Drücken des Stop-Buttons aus dem Query-Menü, oder durch Beenden des Programmes (in diesem Fall findet allerdings nicht nur eine Deaktivierung, sondern eine Zerstörung des Fensters statt).

Die Ausgabe im Darstellungsbereich des Show-Windows kann durch die Menüleiste gesteuert werden. Es stehen zwei Menüs zur Verfügung: Das **Show**- und das **Options**-Menü. Über das Show-Menü kann spezifiziert werden, was für Wissen angezeigt werden soll (zum Beispiel nur die Objekte der Anwendung oder ein spezielles Konzept), während das Options-Menü festlegt wie genau, die Ausgabe erfolgen soll. Beispielsweise kann es

genügen, nur die Namen oder nur die Regeln eines Konzeptes anzuzeigen). Die Ausgabe erfolgt immer in KML-Notation, so daß für den Anwender durchgängig eine Darstellungsweise sichtbar ist (in Kap. 5.2 wird diese Festlegung begründet). Weiterhin kann der Anwender wählen, ob eine Anfrage auf der gesamten Wissensbasis ausgeführt wird oder aber auf der Ergebnismenge der zuvor erfolgten Anfrage.

6.2.1 Menü und Schaltflächen des Show-Windows

In Abbildung 6.9 sind die Menüs des Show-Fensters dargestellt. Die Funktionalität des Show-Menüs soll zunächst geklärt werden.

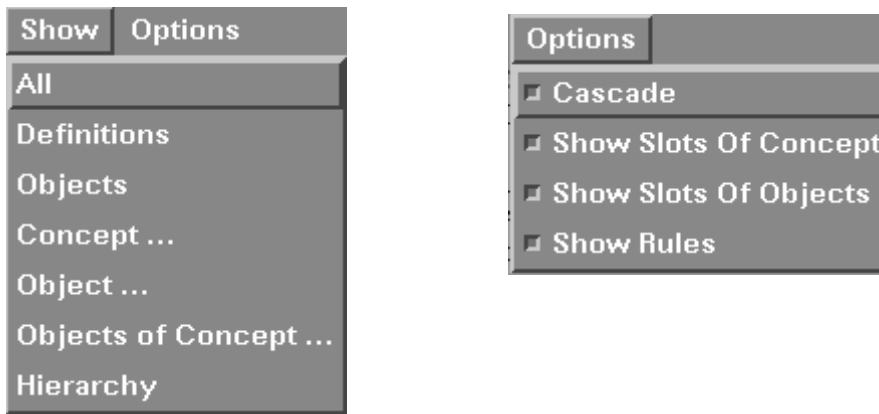


Abbildung 6.9: Menü und Schaltflächen des Show-Windows.

Es enthält folgende Auswahlmöglichkeiten, wobei berücksichtigt werden muß, daß eine Anfrage sich entweder auf die gesamte Wissensbasis oder aber auf das Ergebnis der vorherigen Selektion beziehen kann:

- **All**
das gesamte Wissen wird angezeigt.
- **Definitions**
mit dieser Schaltfläche kann bestimmt werden, daß nur der Definitionsteil (s. Kap. 4.2.1) der KML-Datei angezeigt wird.
- **Objects**
dieser Button zeigt den Objektteil (s. Kap. 4.2.1 oder 4.2.6).
- **Hierarchy**
nur die Konzeptionshierarchien werden selektiert.
- **Concept ...**
nach einem bestimmten, durch den Benutzer eingegebenen Konzeptnamen wird gesucht. Der Konzeptname wird über ein Prompt Widget (s. Abbildung 6.10) eingegeben. Wird ein Konzept dieses Namens nicht gefunden, wird nur das

DEFINITIONS Schlüsselwort aus KML und ein Doppelpunkt ausgegeben (s. Kap. 4.2.1). Eine Ausgabe ist nötig, damit die Termination der Selektionsoption erkennbar ist. Um die Konformität zu wahren, wurde diese Ausgabe gewählt und nicht etwa ein „No“ oder „False“.

- **Object ...**
mit Object ... wird nach einen, durch den Anwender bestimmbaren Objekt gesucht. Auch hier erfolgt die Eingabe durch ein Prompt Widget. Existiert das Objekt nicht, wird lediglich OBJECTS: ausgegeben. Dazu kann dasselbe geschrieben werden wie beim vorherigen Menüpunkt.
- **Objects of Concept ...**
letztendlich kann es von Interesse sein, alle Instanzen eines Konzeptes zu betrachten. Über das Prompt Widget wird der Name des Konzeptes eingegeben. Ansonsten gilt das gleiche wie bei Object... .

Es sind weitere, sinnvolle Arten von Selektionen denkbar, die in Erweiterungen der KML-GUI verwirklicht werden könnten. Hier wurden nur die (nach Meinung des Autors) wichtigsten Anfragen realisiert.

Über das zweite Menü, Options, kann festgelegt werden, wie ausführlich die Ausgabe sein soll und auf welche Datenmenge die Selektionen ausgeführt werden. Die Schaltflächen stellen binäre Schalter dar, die ein oder ausgeschaltet werden können. Ob ein Schalter eingeschaltet ist, erkennt man daran, daß die viereckige Markierung links vom Text in den Bildschirm versenkt ist. (im anderen Fall ragt sie heraus).

- **Cascade**
ist der Cascade Button eingeschaltet, wird jede Anfrage auf der Ergebnismenge der vorherigen Anfrage ausgeführt, anderenfalls auf der gesamten Wissensbasis.
- **Show Slots of Concept**
hiermit kann festgelegt werden, ob die Slots der Konzepte angezeigt werden oder nicht.
- **Show Slots of Objects**
damit wird bestimmt, ob die Slots (und ihre Werte) der Objekte angezeigt werden sollen oder nicht.
- **Show Rules**
ob die Regeln der Konzepte, Objekte oder Slots dargestellt werden sollen oder nicht, wird hier entschieden.

6.2.2 Dialog Widgets des Show-Windows

Durch das Show-Window können Prompt Widgets aktiviert werden, die die Aufgabe haben, vom Benutzer Texteingaben entgegenzunehmen. Das Prompt Widget kann in drei Situationen auftreten (siehe die letzten drei Menüpunkte im Show Menü), dabei kann der

Benutzer aufgefordert werden, entweder einen Konzeptnamen oder einen Objektnamen einzugeben. In Abbildung 6.10 ist der Fall aufgeführt, daß Informationen über das Objekt *Oliver* gesucht werden sollen.



Abbildung 6.10: Prompt-Widget zur Eingabe eines Objektnamens.

Falls dieses Objekt in der Wissensbasis enthalten ist, wird es im Show-Window angezeigt werden.

Zusammenfassend läßt sich sagen, daß mit der KML-GUI ein Werkzeug existiert, mit der sich komfortabel auf ASN-Wissensbasen arbeiten läßt. Diese können erzeugt, geöffnet und bearbeitet werden. Man kann gezielt nach Informationen darin suchen und die Darstellung des Wissen erfolgt stets in KML und ermöglicht damit eine einheitliche und zugleich benutzerfreundliche Sichtweise auf das Wissen, das mit Aktiven Semantischen Netzen modelliert werden kann.

7 Implementation

In diesem Kapitel werden zunächst die Programmierumgebung und die Werkzeuge beschrieben, mit der die Software der vorliegenden Diplomarbeit realisiert wurde, danach soll näher auf die wesentlichen Datenstrukturen, die Systemarchitektur und die Programmmodule eingegangen werden.

7.1 Werkzeuge

Als Werkzeuge wurden für die lexikalische Analyse der Scannergenerator *Lex*, für die syntaktische Analyse *Yacc* eingesetzt. Als Programmiersprache wurde C++ verwendet. Die Oberfläche ist mit dem X Windows System und dem Widget Set (s.u.) *Motif* von der Open Software Foundation gestaltet. Das X Windows System bedarf einer etwas ausführlicheren Erklärung, um die Wirkungsweise des Oberflächenmoduls *KML-GUI* und dessen Widget-Hierarchie besser verstehen zu können.

7.1.1 X-Windows

Das X Window System des Massachusetts Institute of Technology ist mittlerweile zu einem Standard auf Unix Systemen geworden.

Die Grundanforderungen bei der Konzeption des X Windows Systems (im folgenden auch nur X genannt) waren:

- Hardwareunabhängigkeit
- Netzwerktransparenz
- Multitaskingfähigkeit
- freie Gestaltbarkeit der Benutzeroberfläche

Durch Einfügen einer Zwischenschicht zwischen Applikationen und Graphikhardware wird ein direktes Zugreifen von Anwendungsprogrammen auf die Hardware verhindert. Diese Zwischenschicht wird X-Server genannt. Seine Aufgabe ist es, abstrakte Graphikbefehle auf die konkrete Hardware abzubilden. Nur für bestimmte Aufgabenstellungen wie die Farbausgabe lassen sich die Hardwareeigenschaften abfragen, um die Anwendung auf die Hardware einzustellen. Für jedes Ausgabegerät existiert ein X-Server. Die Anwendungsprogramme, auch Clients genannt, können auf jedes X Windows System portiert werden, weil der X-Server immer dieselben Leistungen zur Verfügung stellt. So stellt ein X-Server den Clients sogenannte Ressourcen zur Verfügung, die über Ids (eindeutige 32-Bit Zahlen) angesprochen werden.

Die Ressourcen werden vom X-Server gespeichert und können von mehreren Clients gleichzeitig benutzt werden.

Nachteilig bei der Hardwareunabhängigkeit ist, daß im Grunde der kleinste gemeinsame Nenner aller Graphiksysteme zur Definition von X diente, spezielle leistungsfähige Graphikoperationen mancher Graphikkarten gar nicht genutzt werden können.

Unter Netzwerktransparenz versteht man, daß ein X-Client keine Kenntnis darüber haben muß, wie die Kommunikation mit dem X-Server, der sich sehr oft auf einem anderen Rechner befindet, vonstatten geht. X benutzt das Client-Server Modell. Die Clients sind dabei die eigentlichen Applikationen, während die Server Ressourcen zur Verfügung stellen.

Da Client und Server auf demselben Rechner laufen können, muß das X Windows System mehrprozeßfähig sein, was für Unix Rechner, auf denen X normalerweise aufgesetzt ist, kein Problem darstellt, da es von Anfang an als Multitasking Betriebssystem konzipiert wurde.

Ferner wird von X kein einheitliches Aussehen für die Benutzeroberfläche festgelegt. Es liefert nur Funktionen, um Buttons, Menüs, ... zu erzeugen. Das Erscheinungsbild der Benutzeroberfläche wird durch die Widget Sets bestimmt. Die bedeutendsten Widget Sets auf X Windows Systemen sind OSF/Motif und OpenLook von Unix International (UI).

Ein X-Client verwendet in der Regel drei unterschiedliche Bibliotheken, die auch gleichzeitig drei Abstraktionsebenen darstellen (ersichtlich aus Abbildung 7.1).

Diese heißen:

- Xlib
- X Toolkit Intrinsics
- Widget Set

Die Xlib ist die Software Schnittstelle des X Windows Systems. Sie stellt die grundlegenden Graphikfunktionen zur Verfügung. Es werden keine komplexeren Strukturen wie z.B. Scrollbars unterstützt. Diese finden sich erst auf der nächsten Abstraktionsebene.

Immer wiederkehrende Graphikoperationen werden von den X Toolkit Intrinsics (kurz: Intrinsics) zu Widgets zusammengefaßt. Die Widgets sind Graphikobjekte, die nach dem Baukastenprinzip zu einer Benutzeroberfläche zusammengesetzt werden können. Dabei werden ganz im Sinne der objektorientierten Denkweise, existierende Graphikelemente miteinander kombiniert und mit dem gewünschten Verhalten, d.h. Attributen, ausgestattet.

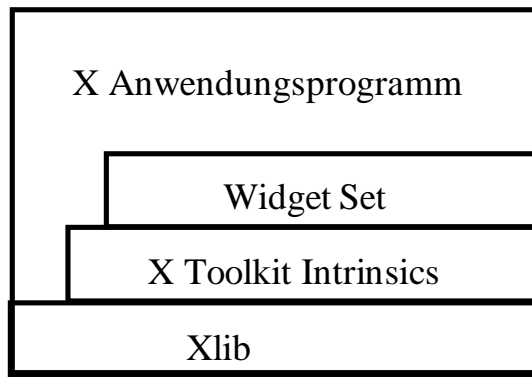


Abbildung 7.1: Zusammenhang zwischen Xlib, Intrinsics, Widget Set und X Client

Die nächsthöhere Abstraktionsebene bilden die Widget Sets. Unter Widget Sets versteht man „höhere“ Widgets, die schon aus einfacheren Elementen zusammengesetzt wurden und ein bestimmtes Verhalten und Aussehen erhielten. Ein Beispiel für solch ein Widget ist die File Selection Box.

Während die Xlib und die Intrinsics Bestandteile der X Windows Systems sind, sind die Widget Sets herstellerabhängig. Was zur Folge hat, das das Erscheinungsbild der Oberfläche in starkem Maße vom Hersteller festgelegt wird. Wie weiter oben erwähnt sind Motif und OpenLook die bekanntesten Widget Sets. Die Unterschiede zwischen den beiden sind vom Konzept her minimal. Daraus folgt, daß jemand, der sich mit dem einem Widget Set auskennt, in kurzer Zeit auch mit dem anderen zurecht kommt.

Es sollen nun einige Grundbegriffe von X erläutert werden:

Ein X Programm begibt sich nach seiner Initialisation in eine Endlosschleife. In dieser Schleife reagiert es nur noch auf *Events*. Unter Events versteht man Ereignisse, die durch den Anwender oder andere Programme hervorgerufen. Events sind z.B. das Drücken einer Maustaste, das Verschieben eines Fensters oder das Drücken einer Taste auf der Tastatur. Der wichtigste Event ist übrigens das Expose Event, das durch das Sichtbarwerden eines Fensters oder eines Ausschnitts davon ausgelöst wurde. Da der X Server verdeckte Fenster nicht zwischenspeichert, muß der Client das Neuzeichnen übernehmen. Auf das Expose Event reagiert jedes funktionierende X Programm. Da es nicht wünschenswert ist, das ein Programm auf jedes erdenkliche Event reagieren soll, kann jedem Fenster einer Applikation eine Eventmaske zugeordnet werden, die unnütze Ereignisse ausfiltert, die anderen werden in eine, jedem Fenster eigene, Warteschlange eingereiht. Das Programm braucht demnach nicht sofort auf ein Event reagieren und kann zunächst andere Aufgaben zu Ende führen.

Windows sind die wichtigsten Datenstrukturen in X. Es sind viereckige Bereiche des Bildschirms, in die gezeichnet werden kann. Die Windows werden in einer baumartigen Hierarchie angeordnet. Durch dieses Anordnung lassen sich ganz im Sinne der objektorientierten Programmierung Eigenschaften von Elternfenster wie Farbe und Eventmaske auf ihre Kindfenster übertragen.

Die *Widgets* sind ebenfalls Datenstrukturen. Die öffentlichen Variablen der Widgets bezeichnet man in X als Ressourcen. Auch hier werden Ressourcen von Widgets an ihre Kinder weitervererbt. Es gibt zwei Methoden, um auf die Ressourcen zuzugreifen: Zum einen kann das innerhalb des Programms stattfinden, zum anderen durch Resource-Dateien. Diese werden durch die X Programme bei Start eingelesen. Der große Vorteil bei der zweiten Methode ist der, daß auf das Aussehen und Verhalten der Benutzeroberfläche Einfluß genommen werden kann, ohne sich mit dem Programmcode auseinandersetzen zu müssen. Bekannt sein muß lediglich die Widgethierarchie sowie der Name der Widgets oder der Widgetklassen. Es ist aber nicht möglich, Ressourcen zu manipulieren, die explizit innerhalb des Programmes gesetzt wurden, weil diese Methode eine höhere Priorität besitzt.

Callback und *Eventhandler* sind Funktionen. Callbacks sind Bestandteile der Widgets (Methoden der Widgetklassen). Sie werden aufgerufen, wenn bestimmte Ereignisse im Widget eingetreten sind. So kann ein Fenster ein Callback besitzen, das aufgerufen wird, wenn der Balken im Scrollbar eines Fensters bei gedrückter Maustaste verschoben wird. Der Callback sorgt für das neue Erscheinungsbild des Fensters.

Eventhandler sind den Callbacks ähnlich, aber im Gegensatz zu ihnen nicht Bestandteile eines Widgets. Sie können vielmehr für jedes Widget eingesetzt werden. Mit einer Eventmaske werden Eventhandler an bestimmte Events gebunden.

Timer rufen nach einer frei wählbaren Zeit nach Programmbeginn eine Funktion auf. Es ist auch möglich einer Timerfunktion in periodischen Zeitabständen aufzurufen.

Work Procedures werden eingesetzt, um Wartezeiten in der Hauptschleife eines X Programms sinnvoll zu überbrücken. Diese Unterprogramme werden aufgerufen, wenn keine Events anliegen, haben aber den Nachteil, daß vom Programm auf Events erst wieder reagiert wird, wenn eine Work Procedure beendet wurde, was sich auf die Performance eines Programmes negativ auswirkt.

Da für die Diplomarbeit OSF/Motif vorgegeben wurde, soll im folgenden noch kurz auf dieses Widget Set eingegangen werden.

7.1.2 OSF/MOTIF

Motif wurde von der Open Software Foundation, in der viele wichtige Workstation-Hersteller wie Hewlett-Packard und DEC vertreten sind, entwickelt. Motif ist aus vier Komponenten zusammengesetzt:

7 Implementation

- **Motif-Toolkit:** Das Widget Set. Es ist eine Sammlung von Grundbausteinen, der Widgets, für Applikationen mit graphischer Benutzeroberfläche. Hierzu zählen unter anderem Dialogboxen, Menüs und Schalter.
- **User Interface Language (UIL):** Die UIL ist eine Beschreibungssprache zum schnellen Entwurf von Widgetbäumen. Mit ihr können Widgets unabhängig vom Programm entworfen und verändert werden. Da die Syntax aber kompliziert ist, wird sie in der Praxis selten eingesetzt.
- **Motif Window Manager (mwm):** Ist der X Client, mit der die Fenster anderer Clients manipuliert und verwaltet werden.
- **Motif Style Guide:** Eine Sammlung von Richtlinien, die das Erscheinungsbild von Motif Applikationen festlegen.

Interessant für den Programmierer ist das Motif Toolkit. Um dem objektorientierten Ansatz gerecht zu werden, sind die Widgets in einer Klassenhierarchie angeordnet. Diese Hierarchie kann man in vier Zweige unterteilen:

- Container Widgets
- Display Widgets
- Dialog Widgets
- Shell Widgets

Alle Unterklassen der Klasse Constraint nennt man Container Widgets (siehe Abbildung 7.2). Sie sind wichtig, um sichtbare Widgets wie die Display Widgets richtig anzuordnen. So können RowColumn Widgets ihre Kinder in Spalten und Zeilen anordnen.

Die Display Widgets sind auf dem Bildschirm direkt sichtbar. Eine Übersicht über die verschiedenen Klassen dieses Widgets Typs kann man Abbildung 7.3 entnehmen. Die Klassennamen vermitteln dem Lesern in einigen Fällen was für eine Art von Widget damit geschaffen werden kann. Genauere Angaben finden sich in [GoKaKeZh92].

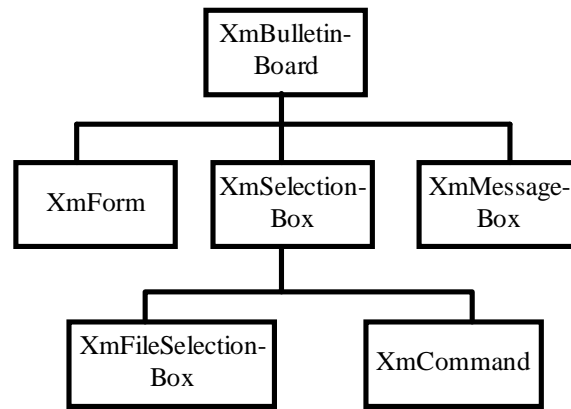


Abbildung 7.4: Hierarchie der Dialog Widgets

Die Shell Widgets schließlich stellen die Schnittstelle eines Motif Programms zur Außenwelt dar. Sie stehen in enger Verbindung mit dem Motif Window Manager. Die Hierarchie dieser Art von Widgets ist in Abbildung 7.5 dargestellt. Das wichtigste Shell Widgets für die Programmierung ist das Application Shell Widget. Es ist in jedem Programm genau einmal vorhanden.

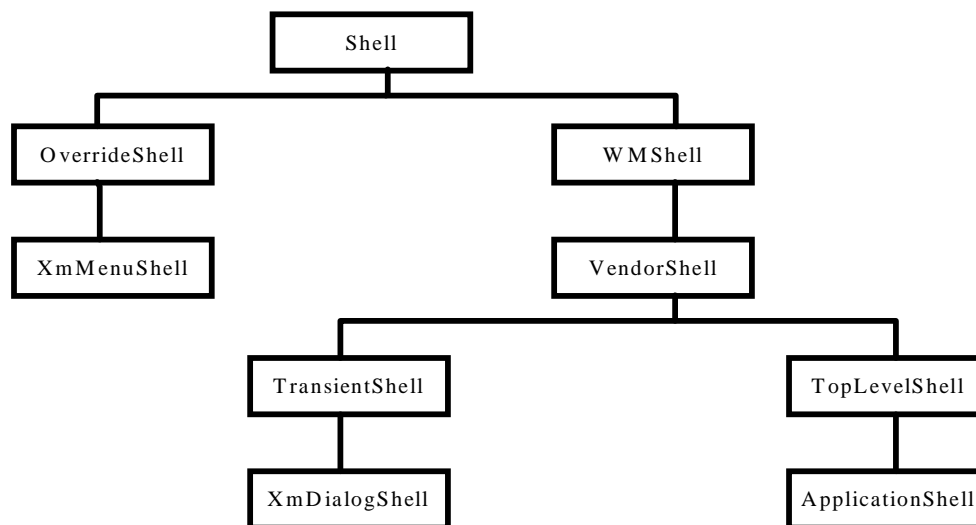


Abbildung 7.5: Hierarchie der Shell Widgets

In den folgenden Unterabschnitten soll Bezug auf die beiden Werkzeuge des Compilerbaus genommen werden

7.1.3 Lex

Lex wurde in den 70er Jahren von Lesk und Schmidt als Ergänzung zu Yacc (siehe Kap. 7.1.4) entwickelt und dient der lexikalischen Analyse, dabei wird die vorgelegte Quelldatei in sogenannte *Tokens* (das sind Bestandteile der Sprache, in der die Quelldatei vorliegt oder benutzerdefinierte Bezeichner) zerlegt. Tokens werden durch endliche

Automaten (deterministisch oder nichtdeterministisch) [HopUll94] erkannt. Die von einem endlichen Automaten akzeptierte Sprache läßt sich durch einen einfachen Ausdruck, der als *regulärer Ausdruck* bezeichnet wird, beschreiben und umgekehrt (Beweise dafür sind in [HopUll94] und [Schönig92] aufgeführt).

Lex ist selbst eine Art Compiler, der Lex-Programme in C-Programme (oder Ratfor-Programme, die aber heutzutage kaum noch eingesetzt werden) übersetzt. Dieses generierte Programm muß dann noch kompiliert und gelinkt werden, um ein ablauffähiges Programm zu erhalten.

Ein Lex-Programm setzt sich aus drei Teilen zusammen:

- Definitionen
- Lex-Regeln
- Benutzerdefinierte Routinen

Im Definitionsteil werden unter anderem die Wirtssprache (C oder Ratfor), Makrodefinitionen und Startbedingungen festgelegt (damit läßt sich zu einem Zeitpunkt nur ein bestimmter Teil der Lex-Regeln aktivieren).

Im Regelteil werden in der Lex-eigenen Weise reguläre Ausdrücke angegeben, die die möglichen Tokens der zu übersetzenden Sprache spezifizieren. Neben den aus [HopUll94] erwähnten Bildungsoperatoren Alternation, Konkatenation, Kleenscher Hülle zur Generierung regulärer Ausdrücke aus vorhandenen, bietet Lex noch eine Reihe weiterer Operatoren wie z.B. die Komplementbildung an, die sich aber auf die drei Basisoperatoren reduzieren lassen und nur zur einfacheren Beschreibung von Tokens dienen.

Die Lex-Regeln werden in Form einer Tabelle angegeben, die in der linken Spalte reguläre Ausdrücke und in der rechten die zugehörige Aktion enthält. Die Aktion ist ein Programmstück, welches ausgeführt wird, wenn in der Quelldatei eine Zeichenkette gefunden wird, die durch den zur Aktion gehörendem regulären Ausdruck abgedeckt wird.

Außerdem bietet Lex eine Reihe von Hilfsfunktionen an, um die Arbeit für Anwender zu erleichtern.

Nach dem Regelteil können noch benutzerdefinierte Routinen angegeben werden. Üblicherweise stehen in diesem Abschnitt Definitionen von Funktionen, die in den Aktionen der Lex-Regeln benötigt werden.

Lex ist nicht nur ein Werkzeug für den Compilerbau, es wird in vielen Bereichen der Softwareentwicklung eingesetzt, wie z.B. der Textverarbeitung, Chiffrierungen, Kommandoprozessoren, eben in allen Bereichen, in denen sich Teile der Daten durch reguläre Ausdrücke beschreiben lassen.

7.1.4 Yacc

Yacc wurde von Johnson bei den Bell Labs entwickelt. Der Name ist eine Kurzform für *yet another compiler-compiler*, was auf die Popularität von Parser-Generatoren in den 70er Jahren hinweist, in denen *Yacc* entstanden ist. *Yacc* ist ein LALR-Parser-Generator. Auf LALR-Grammatiken basierende Sprachen bilden eine Untermenge der auf LR-Grammatiken basierenden Sprache, die sich aber besonders effizient parsen lassen. LR-Parser gehen generell nach der Bottom-Up Syntaxanalysetechnik vor, dabei wird der Parsebaum an den Blättern beginnend aufgebaut (im Gegensatz zur Top-Down Methode). Die LR-Grammatiken bilden wiederum eine Untermenge der kontextfreien Grammatiken, jedoch können damit die meisten Programmiersprachen grammatikalisch beschrieben werden. Tiefergehende Aspekte zur Syntaxanalyse sind aus [HopU1194] und [AhSeU192] zu entnehmen.

Yacc generiert aus dem vorgelegten *Yacc*-Programm ein C-Programm (ist demnach wie *Lex* eine Art Compiler), dieses muß danach noch kompiliert und gelinkt werden, um einen ablauffähigen Parser-Generator zu erhalten.

Ein *Yacc*-Programm besteht aus drei Teilen:

- Definitionen
- *Yacc*-Regeln
- Benutzerdefinierte Routinen

Im Definitionsteil werden die Tokens angegeben, die beim Scannen auftreten können. Weiterhin wird das Startsymbol der Grammatik festgelegt sowie Prioritäts- und Assoziativitätsbedingungen beschrieben. Außerdem können Typen für die Rückgabewerte der lexikalischen Analyse bestimmt werden. So ist nämlich oft nicht nur das Token interessant, sondern der konkrete Wert der dahinter steckt.

Der Regelteil ist das Kernstück eines *Yacc*-Programmes. Es besteht aus einer Folge von Grammatikregeln der folgenden Form:

```
regelname : rechte_seite ;
```

Der *regelname* ist dabei ein Name für ein nichtterminales Symbol und *rechte_seite* ist eine Menge von Terminalen oder Nichtterminalen. Haben mehrere *Yacc*-Regeln die gleiche linke Seite, so können diese zu einer Regel zusammengesetzt werden. Beispielsweise kann statt

```
A: B C
```

```
A: D
```

auch

```
A: B C
```

```
  | D
```

geschrieben werden. Damit ähneln Yacc-Regeln sehr BNF-Regeln. Und in der Tat liegt der Unterschied nur darin, daß in Yacc-Regeln statt $::=$ ein $:$ gesetzt wird und die Regel durch einen Strichpunkt abgeschlossen wird.

Auf der rechten Seite der Regeln können zwischen den einzelnen Symbolen noch Aktionen angegeben werden. Die Aktionen enthalten C-Anweisungen und werden ausgeführt, wenn das links davon stehende Symbol erkannt wurde. In den Aktionen kann auf die Werte zugegriffen werden, die von den einzelnen Symbolen der Regel geliefert werden. Soll eine Regel einen Wert zurückliefern, so muß in einer der angegebenen Aktionen einer speziellen Variablen der entsprechende Wert zugewiesen werden. Die Aktionen können dazu genutzt werden, um semantische Überprüfungen der Sprache durchzuführen.

Nach dem Regelteil können noch beutzerdefinierte Routinen angegeben werden. Üblicherweise stehen in diesem Abschnitt Definitionen von Funktionen, die in den Aktionen der Yacc-Regeln benötigt werden.

Es ist möglich, Yacc und Lex gemeinsam zu verwenden. Dabei ruft der Parser, den von Lex generierten Scanner auf, wenn er ein neues Token benötigt.

Auch für Yacc ist das Einsatzgebiet heute nicht mehr auf den Compilerbau beschränkt. Es kann in vielen Anwendungsbereichen eingesetzt werden, in denen die Eingabe für die Software eine gewisse Struktur voraussetzt.

7.1.5 C++

C++ wurde von Stroustrup an AT&T Bell Labs (wie Yacc) entwickelt. C++ ist größtenteils eine Obermenge von C und beinhaltet neben intensiverer Typüberprüfung, überladenen Funktionen viele weitere Verbesserungen gegenüber C. Am wichtigsten ist jedoch, daß objektorientierte Programmierfeatures, wie Mehrfachvererbung, Kapselung, Abstraktion, hinzugefügt wurden.

Es soll und kann hier nicht diese Programmiersprache näher vorgestellt werden. Es sei aber auf das Standardwerk von [Stroustrup92] verwiesen, in dem die Sprache sehr ausführlich vorgestellt wird.

Nach Vorstellung der Werkzeuge, die in der Diplomarbeit verwendet wurden, sollen nun konkret wichtige Aspekte der Implementation besprochen werden.

7.2 Datenstrukturen

Beim Übersetzen einer KML-Datei in ein Aktives Semantisches Netz wird das Wissen aus der KML-Datei zunächst in eine *Zwischendatenstruktur* (deren Struktur folgt weiter unten im Text) überführt, ähnlich wie beim Compilieren von Programmiersprachen, wo gewöhnlich die Quellsprache nicht direkt in die Zielsprache übersetzt wird, sondern zuerst in einen Zwischencode (bei Pascal z. B. der p-Code). Prinzipiell wäre diese

Zwischendatenstruktur nicht nötig gewesen, sie hat aber den Vorteil, daß die Verwendung der ASN-API von der Syntaxanalyse getrennt werden konnte. Erst nachdem sichergestellt wurde, daß eine KML-Datei syntaktisch und semantisch korrekt ist, wird die Zwischendatenstruktur mit Hilfe der API-Funktionen in eine ASN-Wissensbasis überführt. So können Fehler schon dort abgefangen werden, wo sie auch auftreten und müssen nicht auf die nächste darunterliegende Schicht delegiert werden (ein wichtiges Prinzip beim Compilerbau, siehe [AhSeUI92]). Die Zwischendatenstruktur übernimmt auch die Funktionen einer Symboltabelle (Speichern aller Bezeichner und Informationen über die verschiedenen Attribute der Bezeichner zu sammeln), so daß auf eine eigenständige Symboltabelle verzichtet werden konnte.

Ein weiterer Vorteil dieser Zwischendatenstruktur ist, daß Anfragen auf einer Wissensbasis (über das Show-Window, s. Kap. 6.2) nur auf dieser Zwischendatenstruktur ausgeführt werden brauchen, da alles Wissen darin enthalten ist und damit keinen Datenbankzugriff erforderlich macht, was sich in einer wesentlich höheren Effizienz widerspiegelt.

Auch bei der Darstellung einer ASN-Wissensbasis in KML werden die Informationen aus der Wissensbasis zunächst in die Zwischendatenstruktur überführt und danach daraus das Wissen entnommen, daß nötig ist, um eine KML-Datei zu generieren.

Der Datenfluß zwischen ASN-Wissensbasis und einer KML-Datei ist in Abbildung 7.6 aufgezeichnet.

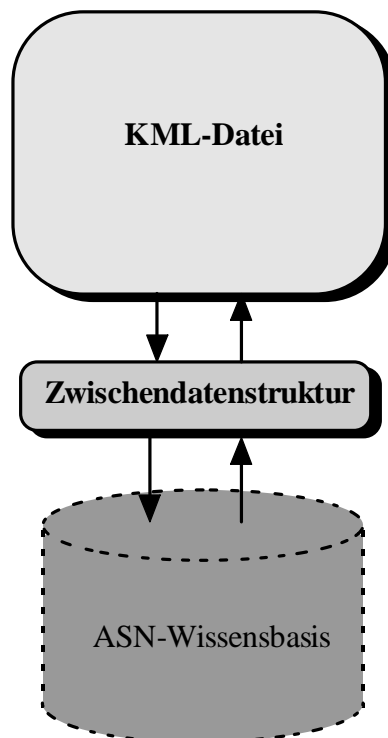


Abbildung 7.6: Datenfluß zwischen ASN-Wissensbasis und KML-Datei.

Die Zwischendatenstruktur wird durch eine Instanz der Klasse **KonzeptBaumKlasse** realisiert. Der Name rührt daher, daß alle Konzepte, die auftreten, in einem binären Suchbaum angeordnet werden. Die Wahl fiel auf diese Datenstruktur, weil sie einerseits einfach zu bearbeiten ist, andererseits im Mittel für die wichtigsten Operationen, Suchen, Einfügen und Löschen, logarithmischen Aufwand benötigt. Jedes Konzept wird durch einen Knoten in diesem Baum repräsentiert. Diese Knoten sind Instanzen der Klasse **KonzeptKlasse**. Jede Instanz dieser Klasse enthält alle Informationen, die zu dem entsprechenden Konzept gehören: Slots, Superkonzepte, dessen Objekte, Regeln, Kommentar, Jeder Slot und jedes Objekt werden durch eine eigene Klasse beschrieben: **SlotKlasse** und **ObjektKlasse**. Da zu einem Konzept mehrere Slots wie auch Objekte gehören können, werden diese ebenfalls in binären Suchbäumen angeordnet, die Instanzen der Klassen **SlotBaumKlasse** und **ObjektBaumKlasse** sind. Diese beiden Klassen sind Komponenten der KonzeptKlasse. Zwei weitere Klassen tauchen im Zusammenhang mit den Slotwerten auf. Jeder Slotwert wird durch eine Instanz einer eigenständigen Klasse, nämlich **SlotwertKlasse**, dargestellt. Auch hier gilt, daß ein Objekt viele Slotwerte besitzen kann, deswegen werden diese in einer baumartigen Struktur angeordnet und bilden eine Instanz der Klasse **SlotwertBaumKlasse**. Diese Klasse ist eine Klassenkomponente von ObjektKlasse. Damit existieren acht Klassen, die gemeinsam für den Aufbau der Zwischendatenstruktur von Nöten sind. Durch die Verzahnung der Klassen läßt sich die Zwischendatenstruktur durch einen dreidimensionalen binären Suchbaum darstellen. So werden alle Konzepte in einem Baum angeordnet. An den Knoten hängen eventuell zwei weitere Binärbäume, einer der alle Slots des Konzeptes enthält, der andere der all dessen Objekte beinhaltet. Da ein Objektbaum seine Slotwerte selbst in einer Baum organisiert, kommt man zu dieser dreidimensionalen Struktur. In Abbildung 7.7 wird dieser Sachverhalt graphisch erläutert.

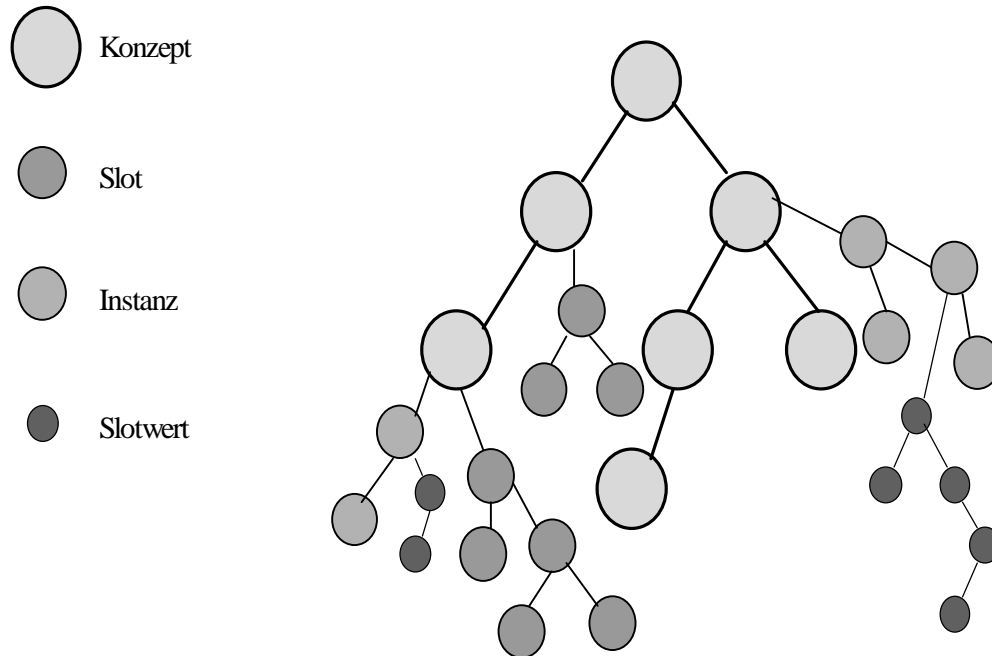


Abbildung 7.7: KML-Wissen in mehrdimensionalen Binärbaum ablegen.

Trotz der Komplexität der Struktur bleibt die Komplexität für das Einfügen, das Suchen und das Löschen im Mittel logarithmisch (genauer gesagt $O(\log_2 n) + O(\log_2 m) + O(\log_2 l)$, wenn n die Anzahl der Konzepte und m bzw. l die durchschnittliche Anzahl von Slots bzw. Objekten eines Konzeptes ausdrücken. Im Falle von degenerierten Bäumen ist mit linearem Aufwand zu rechnen, um dieses zu vermeiden, könnten statt Suchbäumen, ausgeglichene oder B-Bäume verwendet werden, die eine logarithmische Komplexität garantieren (siehe auch [Sedgewick92]).

Da wie weiter oben erwähnt, Abfragen auf der Zwischendatenstruktur ausgeführt werden können, wurden der Klasse `KonzeptBaumKlasse` Methoden hinzugefügt, die diese Abfragen auf der Zwischendatenstruktur ausführen.

Die Kopplung der Zwischendatenstruktur mit der ASN-API Funktionen, erfolgt in der von `KonzeptBaumKlasse` abgeleiteten Klassen `ASNKonzeptBaumKlasse`. Genauso wurden aus den anderen oben erwähnten Klassen, Klassen abgeleitet, die Methoden enthalten, in denen die Umsetzung von und zu einem ASN erfolgen. Die Klassennamen unterscheiden sich von den Oberklassen nur durch ein vorangestelltes `ASN`. An Attributen kam nichts neues hinzu, lediglich Methoden, die die ASN-API verwenden, sind hinzugefügt worden.

Nach Vorstellung der zentralen Datenstruktur der Implementierung sollen die Architektur und die Programmmodule vorgestellt und beschrieben werden.

7.3 Architektur und Module

Die gesamte Applikation besteht aus acht Programmmodulen, die im Rahmen der Diplomarbeit entstanden sind, sowie zweien, die importiert werden, nämlich der ASN-API und die X und Motif Bibliotheken. Ein weiteres Programm (*Emacs*) wird zum Editieren benötigt. Durch die Zerlegung in Module sinken die Software-Kosten, da einerseits durch das Aufteilen eines Programms in einzelne Komponenten seine Komplexität reduziert werden kann und andererseits Module unabhängig voneinander (bis auf Schnittstellen) entworfen und getestet werden können.

Die Architektur des Gesamtsystems ist der Abbildung 7.8 zu entnehmen. An der Spitze steht das Oberflächenmodul, daß die Schnittstelle zum Anwender bildet. Mit den anderen Modulen kommt er dagegen nicht direkt in Kontakt.

Im weiteren sollen die Aufgaben der einzelnen Module kurz beschrieben werden.

Das Oberflächenmodul **KML-GUI** erzeugt die graphische Benutzeroberfläche wie in Kapitel 6 vorgestellt und steuert die Arbeitsweise der Applikation, indem die eintreffenden Events abgefangen und entsprechende Callbacks gestartet werden (s. Kap. 7.1.1). Das Oberflächenmodul besteht selbst aus drei Untermodulen: Eines enthält die gesamten Callbacks, ein anderes zeichnet die Oberfläche und bindet die Callbacks an die Widgets und im dritten werden die Graphikressourcen erzeugt, also unter anderem die Graphikkontexte, Einträge in die Farbtabelle gemacht, Fonts ausgewählt, usw. Die Widget-Hierarchie für die Oberfläche ist Abbildung 7.4 zu entnehmen. Dabei wurde nicht jede Einzelheit aufgeführt. Vielmehr soll ein Eindruck von der Hierarchie gewonnen werden. Die Namen der Widgets in den Message-Boxen, die implizit festgelegt und nicht bekannt sind, sind mit drei Fragezeichen gekennzeichnet.

7 Implementation

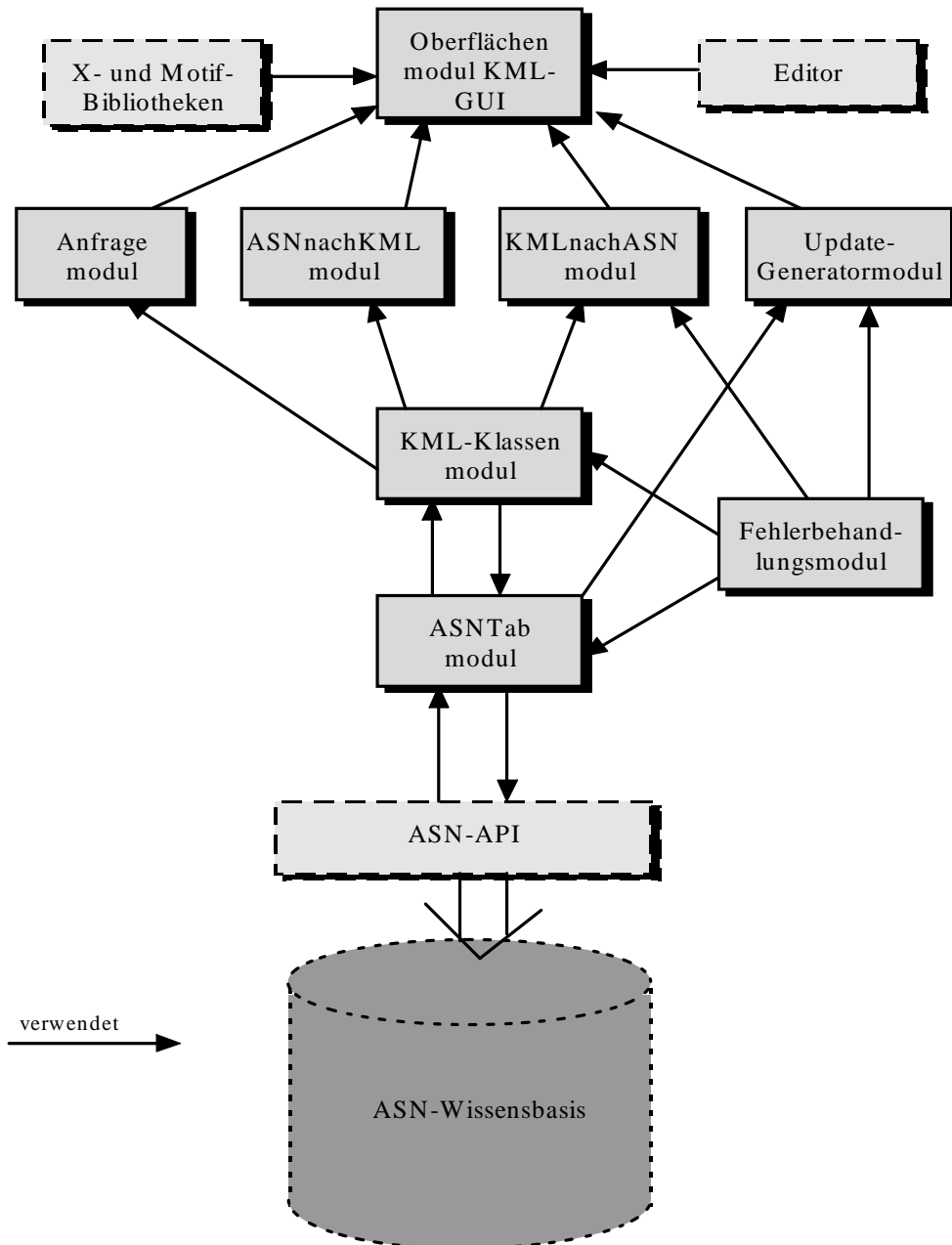


Abbildung 7.8: Das Zusammenwirken der Module im Gesamtsystem.

Das Modul **KMLnachASN** hat den Zweck KML-Code in die ASN eigene Darstellung zu übersetzen. Dabei wird der KML-Code zunächst in die Zwischendarstellung überführt und danach diese in das ASN. Der erste Schritt läuft in drei Phasen, die ineinander verzahnt sind: die lexikalische, syntaktische und semantische Analyse. Da in KML Bezeichner verwendet werden können, bevor sie definiert werden, muß dieser erste Schritt zweimal durchlaufen werden (zwei-Pass-Compiler), um die Analyse vollständig ausführen zu können und die Zwischendatenstruktur aufzubauen. Danach wird aus der Zwischendatenstruktur mit Hilfe der API-Funktionen des ASNs die Wissensbasis aufgebaut. KMLnachASN kann auch direkt ohne Verwendung der Oberfläche durch

Aufruf des Programmnamens gestartet werden. Als Parameter müssen die KML-Datei, die übersetzt werden soll und der Name der ASN-Wissensbasis angegeben werden.

Im Modul **ASNnachKML** wird wie der Name schon sagt, das Wissen einer ASN-Wissensbasis nach KML überführt. Auch hier ist die Zwischendatenstruktur zwischengeschaltet. Wie KMLnachASN kann ASNnachKML direkt aus der Shell gestartet werden. Als Parameter sind der Name der Wissensbasis und der Name der zu erzeugenden KML-Datei anzugeben. Existiert eine Datei mit demselben Namen bereits, wird diese überschrieben.

Das Modul **Anfrage** setzt die Selektionsoperatoren auf der Zwischendatenstruktur um, und erzeugt eine Datei in der das Ergebnis der Anfrage in KML abgelegt wird. Dabei wird auch dafür gesorgt wie ausführlich die Ausgabe sein soll, z.B. ob nur die Modulköpfe angezeigt werden oder auch die Regeln oder alles,

Der **UpdateGenerator** ist nur rudimentär realisiert worden. Es passiert nicht anderes, als das die vorhandene Wissensbasis gelöscht wird und aus dem modifizierten Code die Wissensbasis vollständig neu generiert wird. Das ist zwar keine schöne Lösung, aber sie erfüllt für das erste ihren Zweck.

Das **KML_Klassen** Modul beherbergt die Klassen, die für die Erzeugung der Zwischendatenstruktur benötigt werden (in Kap. 7.2 wurden diese vorgestellt).

Das Modul **ASNTab** beinhaltet die von den im KML_Klassen Modul vorhandenen Klassen abgeleiteten Klassen. Diesen wurden Methoden hinzugefügt, die die ASN-API verwenden. Es ist das einzige Modul, das mit der ASN-API in Verbindung steht und falls Änderungen in der ASN-API stattfinden, muß nur dieses Modul angepaßt werden.

Das Modul **Fehlerbehandlung** wird von anderen Modulen verwendet, wenn bei der Überführung einer ASN-Wissensbasis nach KML und umgekehrt, Fehler auftreten, z.B. Syntax- oder Semantikfehler. Jedem möglichen Fehler ist ein Nummer zugeordnet, die eine entsprechende Fehlermeldung zur Folge hat. Fehler in den Übersetzungsphasen führen grundsätzlich zum Programmabbruch.

7 Implementation

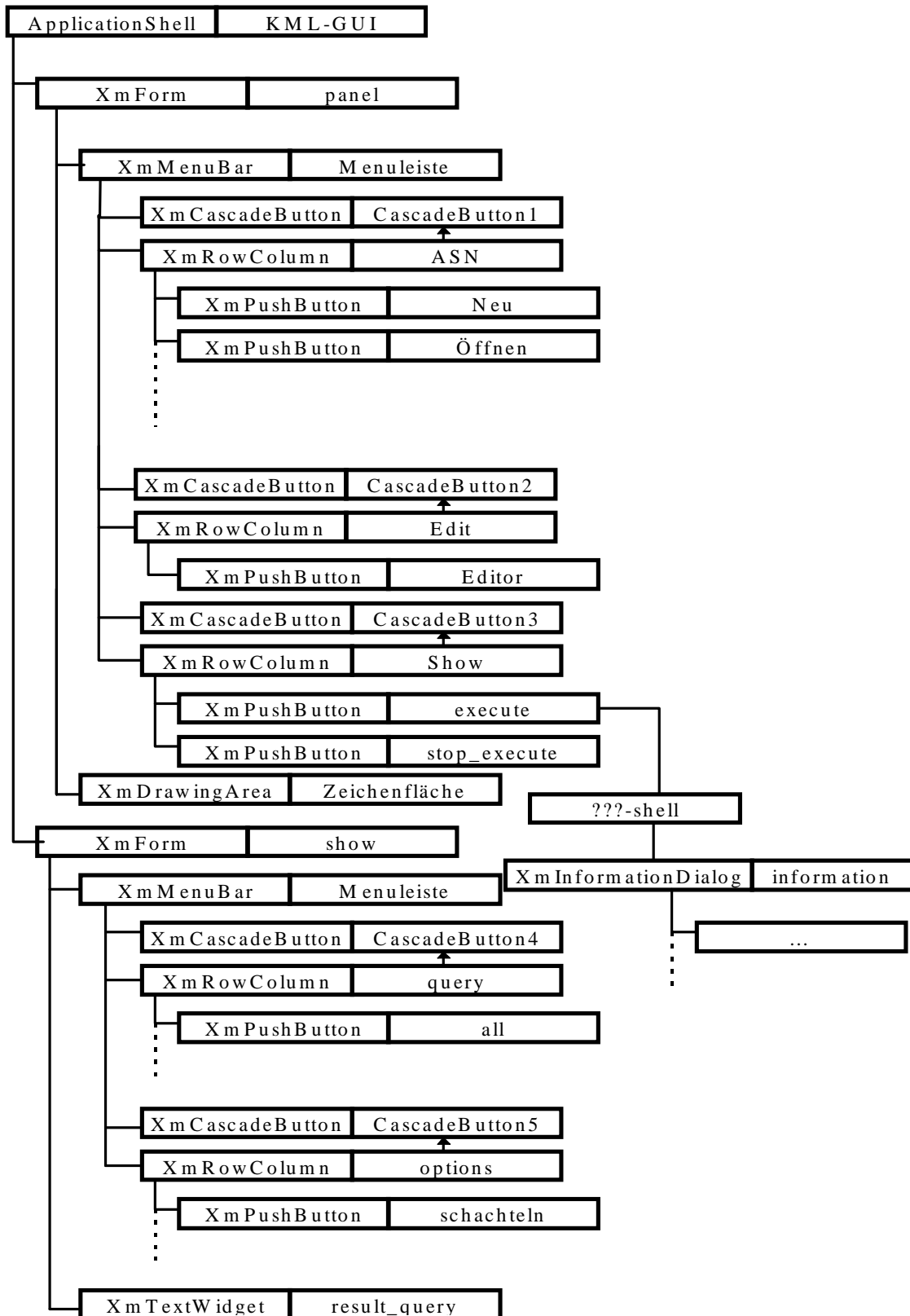


Abbildung 7.9: Die Widget-Hierarchie der KML-GUI.

7 Implementation

Damit wurde die Implementation des Gesamtsystems umrissen. Im letzten Kapitel sollen die Ergebnisse der Diplomarbeit zusammengefaßt, sowie mögliche Verbesserungen und Erweiterungen erwähnt werden.

8 Fazit

Zum Abschluß der Arbeit erfolgt noch eine Zusammenfassung der Ergebnisse. Es werden überdies Vorschläge für mögliche Verbesserungen und Erweiterungen der entwickelten Tools gemacht. Am Ende folgt noch ein Schlußwort.

8.1 Zusammenfassung

In dieser Arbeit wurde die Modellierungssprache *Knowledge Modelling Language* (kurz *KML*) für das Aktive Semantische Netz (ASN), in dem alle bei einer Produktentwicklung beteiligten Wissensgebiete abgebildet werden, spezifiziert und Werkzeuge entwickelt, die ein Modell in der definierten Modellierungssprache in die Wissensbasis eintragen und umgekehrt Inhalte der Wissensbasis in dieser Sprache darstellen. Dabei wurde die (kontextfreie) Grammatik der Sprache mit Hilfe der Backus-Naur-Form beschrieben und deren Semantik umgangssprachlich erklärt. Indem gezeigt wurde, welche Wissenskonstrukte im ASN dargestellt werden können, konnte die Modellierungssprache so konzipiert werden, daß das ASN und KML die gleiche Ausdrucksmächtigkeit besitzen.

Wesentlich bei der Konzeption der Sprache war, daß sie der Denkökonomie des Menschen entgegen kommt und ihm dabei einerseits die Möglichkeit gibt, sich auf das eigentliche Problem (der Beschreibung bzw. der Modellierung einer Anwendung im Bereiche der Produktentwicklung) zu konzentrieren und nicht viel Zeit für die Verwendung der Sprache investieren muß und andererseits die „Lücke“ zwischen der Denkweise von Menschen und der Beschreibung des Wissens in einer formalen Sprache möglichst klein hält. Dies führte zu einer Sprache, die Wissen modular (in Konzept- und Objektmodulen) organisiert, eine formularartige Beschreibung ermöglicht und mit wenigen Sprachkonstrukten auskommt. Auch die Schlüsselworte wurden so gewählt, daß sich deren Bedeutung durch ihre umgangssprachliche Semantik ergibt. So dürfte Wissen, das mit KML beschrieben wird, auch für Personen, die mit der Sprache noch nicht vertraut sind, einigermaßen nachvollziehbar sein.

Die weitere Entwicklung von KML hängt untrennbar mit der des ASN zusammen. Werden dem ASN wie geplant weitere Konzepte der Wissensrepräsentation (wie probabilistisches oder temporales Wissen) hinzugefügt, so müßte auch die KML um Sprachkonstrukte ergänzt werden, die diese Art von Wissen ausdrücken können. Durch den objektorientierten Programmentwurf (den Kern bildet die Klassenbibliothek der Zwischendatenstruktur, s. Kap. 7.2) wurde eine Grundlage für Erweiterungen geschaffen.

Weiterhin wurde erläutert, welche prinzipiellen Methoden es gibt, auf einer ASN-Wissensbasis Modifikationen durchzuführen, was für Vor- und Nachteile sie bieten und wie sie zu realisieren sind. Es wurde ein einfacher Update-Generator implementiert, der diese Aufgabe durchführt. Die Einfachheit wurde jedoch durch ein zeitaufwendiges Update der Wissensbasis erkauft. Durch eine intelligentere Vorgehensweise könnte sich der Zeitaufwand jedoch erheblich senken lassen.

Ebenso wurde Verfahren vorgestellt, wie sich Anfragen auf der Wissensbasis durchführen lassen, deren Vor- und Nachteile diskutiert und geklärt in welcher Form das Ergebnis einer Anfrage dargestellt werden soll und wie ausführlich es sein soll. Die Auswahl der wichtigsten Anfragen über ein Menü (dieser Ansatz wurde in der Applikation realisiert) hat zwar den Vorteil, daß Anwender sich keine weiteren Gedanken darüber machen müssen, welche Anfragen es überhaupt gibt und wie sich diese ausdrücken lassen, beschränkt andererseits die Anfragen aber nur auf die im Menü aufgeführten. Zwar kann durch geschachtelte Anfragen, Auswahl über die Ausführlichkeit des Ergebnisses und Parameterübergabe eine größere Flexibilität erreicht werden, letztendlich kann aber ein umfassender Ansatz nur über eine eigene Anfragesprache erzielt werden.

Um eine komfortable und benutzerfreundliche Arbeitsweise zu ermöglichen, wurden die vier implementierten Tools (der KML-Übersetzer, der ASN nach KML Konverter, der Update-Generator und das Anfragemodul) unter einer graphischen Oberfläche (*KML-GUI*) integriert. Dabei kommt der Anwender mit diesen Tools nicht direkt in Kontakt, vielmehr kann er eine Wissensbasis neu anlegen, öffnen, deren Inhalt editieren (und damit auch modifizieren), das Wissen in der selektierten Wissensbasis ablegen, über ein eigenes Fenster kann er Anfragen über der Wissensbasis starten. Über diese Aktion werden die Tools gestartet und bleiben für den Benutzer transparent.

In der graphischen Oberfläche wird stets KML verwendet, um das Anwendungswissen aus der Wissensbasis dem Benutzer sichtbar zu machen. Eine Verbesserung wäre es, wenn gewisse Informationen auch graphisch aufbereitet werden würden. Zum Beispiel könnten Vererbungsbeziehungen zwischen Konzepten graphisch besser illustriert werden, als in textueller Form, ferner würde ein Konzeptbrowser (in Analogie zu einem Klassenbrowser) das Kennenlernen der diversen Konzeptmodule erleichtern. Eine weitere Verbesserung wäre auch der Einsatz bestimmter Icons in der Oberfläche, deren Betätigung das Einfügen oder Löschen von Konzepten oder Objekten im Dialog mit dem Anwender bewirken würde. Das sind nämlich Operationen, die häufig stattfinden und für die der zeitaufwendige Weg über den Update-Generator (bezogen auf die Laufzeit des Tools) zu umständlich ist.

8.2 Schlußwort

Diese Arbeit hat mir ein tieferes Verständnis für die Themenbereiche Wissensrepräsentation, Erstellung graphischer Oberflächen, formale Sprachen und

8 Fazit

Compilerbau, sowie den objektorientierten Ansatz in der Softwareentwicklung gebracht. Ferner habe ich einiges, über die Methodik beim Bearbeiten einer wissenschaftlicher Arbeit, gelernt.

Bedanken möchte ich mich bei meinem Betreuer O. Eck, für seine Geduld, Hilfsbereitschaft und die wichtigen Anregungen, die mich zum Denken anregten.

Anhang A: Syntax von KML

Es folgt die vollständige Angabe der Syntax von KML, wie sie auch in der Implementation der Sprache verwendet wurde.

Zur besseren Übersicht sind die Produktionen durchnummeriert. Die Nummern sind auch bei Nichtterminalen angeführt und dienen als Verweise. Formal sind die Nummern aber bedeutungslos.

Die Darstellung stützt sich auf [Ludewig89]. Jedoch wird als Beschreibungsform einer kontextfreien Grammatik hier die Backus-Naur-Form verwendet und nicht Syntaxdiagramme.

```
<kml_sprache>1 ::= <kml_sprache>1 <aufbau>2
                | <aufbau>2

<aufbau>2 ::= DEFINITIONS : <strukturteil>3
                | OBJECTS : <objektteil>31

<strukturteil>3 ::= <konzept>4
                | <strukturteil>3 <konzept>4

<konzept>4 ::= <konzeptname>7 : CONCEPT
                <konzept_kommentar>6 <eigenschaften>11 .
                | <konzeptname>6 : CONCEPT IS A <superkonzeptname>8
                <mehrfachvererbung>5 <konzept_kommentar>6 <eigenschaften>11 .

<mehrfachvererbung>5 ::= <mehrfachvererbung>5 AND <konzeptname>7
                | epsilon

<konzept_kommentar>6 ::= COMMENT : " <string>50 "
                | epsilon

<konzeptname>7 ::= <bezeichner>9

<superkonzeptname>8 ::= <bezeichner>9

<bezeichner>9 ::= <buchstabe>59 <rest_name>10

<rest_name>10 ::= <buchstabe>59 | <ziffer>53 | ! | _ | - | @ | ?

<eigenschaften>11 ::= <teile>12 <relationen>13 <attribute>14 <konzeptregeln>22

<teile>12 ::= PARTS : <teile_eigenschaften>15
                | epsilon

<relationen>13 ::= RELATIONS : <relationen_eigenschaften>16
                | epsilon
```

<attribute>₁₄ ::= ATTRIBUTES : <attribut_eigenschaften>₁₇
 | epsilon

<teile_eigenschaften>₁₅ ::= <teile_eigenschaft>₁₅ <slot_kommentar>₁₈
 | <teile_eigenschaften>₁₅ <teile_eigenschaft>₁₉
 <slot_kommentar>₁₈

<relationen_eigenschaften>₁₆ ::= <relationen_eigenschaft>₂₀ <slot_kommentar>₁₈
 | <relationen_eigenschaften>₁₆
 <relationen_eigenschaft>₂₀ <slot_kommentar>₁₈

<attribut_eigenschaften>₁₇ ::= <attribut_eigenschaft>₁₇ <slot_kommentar>₁₈
 | <attribut_eigenschaften>₁₇ <attribut_eigenschaft>₂₁
 <slot_kommentar>₁₈

<slot_kommentar>₁₈ ::= COMMENT : " <string>₅₀ "
 | epsilon

<teile_eigenschaft>₁₉ ::= <slotname>₂₈ : <rangekonzeptname>₂₉
 | SET OF <slotname>₂₈ : <rangekonzeptname>₂₉

<relationen_eigenschaft>₂₀ ::= <slotname>₂₈ : <rangekonzeptname>₂₉
 | SET OF <slotname>₂₈ : <rangekonzeptname>₂₉

<attribut_eigenschaft>₂₁ ::= <slotname>₂₈ : <wertebereich>₃₀
 | SET OF <slotname>₂₈ : <wertebereich>₃₀

<konzeptregeln>₂₂ ::= RULES : <konzeptregelteil>₂₃
 | epsilon

<konzeptregelteil>₂₃ ::= <konzeptregelstruktur>₂₄
 | <konzeptregelteil>₂₃ <konzeptregelstruktur>₂₄

<konzeptregelstruktur>₂₄ ::= ON READ CONCEPT <condition>₂₅ <action>₂₆
 | ON READ CONCEPT <action>₂₆
 | ON MODIFY CONCEPT <condition>₂₅ <action>₂₆
 | ON MODIFY CONCEPT <action>₂₅
 | ON READ <slotnamerule>₂₇ <condition>₂₅ <action>₂₆
 | ON READ <slotnamerule>₂₇ <action>₂₆
 | ON MODIFY <slotnamerule>₂₇ <condition>₂₅ <action>₂₆
 | ON MODIFY <slotnamerule>₂₇ <action>₂₆

<condition>₂₅ ::= CONDITION : <tcl_script> END OF TCL
 | epsilon

<action>₂₆ ::= ACTION : <tcl_script> END OF TCL

<slotnamerule>₂₇ ::= <bezeichner>₉

<slotname>₂₈ ::= <bezeichner>₉

<rangekonzeptname>₂₉ ::= <bezeichner>₉

```

<wertebereich>30 ::= NUMBER
    | REAL
    | REAL NUMBER
    | TEXT
    | {YES,NO}
    | OPAQUE
    | OPAQUE NUMBER

<objektteil>31 ::= <objektteil>31 <objekt>32
    | <objekt>31

<objekt>32 ::= <objektname>36 : <objektkonzeptname>37 <objekt_kommentar>38
    <slotwerte>39 <objektregeln>33 .

<objektregeln>33 ::= RULES : <objekttregelteil>34
    | epsilon

<objekttregelteil>34 ::= <objekttregelstruktur>35
    | <objekttregelteil>34 <objekttregelstruktur>35

<objekttregelstruktur>35 ::= ON READ OBJECT <condition>25 <action>26
    | ON READ OBJECT <action>26
    | ON MODIFY OBJECT <condition>25 <action>26
    | ON MODIFY OBJECT <action>26
    | ON READ <slotnamevaluerule>40 <condition>25 <action>26
    | ON READ <slotnamevaluerule>40 <action>26
    | ON MODIFY <slotnamevaluerule>40 <condition>25 <action>26
    | ON MODIFY <slotnamevaluerule>40 <action>26

<objektname>36 ::= <bezeichner>9

<objektkonzeptname>37 ::= <bezeichner>9

<objekt_kommentar>38 ::= COMMENT : " <string>50 "
    | epsilon

<slotwerte>39 ::= <slotwerte>00 <slotnamewert>41: <wert>42
    | epsilon

<slotnamevaluerule>40 ::= <bezeichner>9

<slotnamewert>41 ::= <bezeichner>9

<wert>42 ::= <objektwert>43 | <stringwert>45 | <numberwert>46 | <realwert>47
    | <booleanwert>48 | <urlwert>49 | NONE | UNKNOWN

<objektwert>43 ::= <objektnamewert>44
    | <objektnamewert>44 AND <objektwert>43

<objektnamewert>44 ::= <bezeichner>9

```

```

<stringwert>45 ::= " <string>50 "
           | " <string>50 " AND <stringwert>45

<numberwert>46 ::= <number>51
           | <number>51 AND <numberwert>46

<realwert>47 ::= <real>54
           | <real>54 AND <realwert>47

<booleanwert>48 ::= YES | NO
           | YES AND <booleanwert>48
           | NO AND <booleanwert>48

<urlwert>49 ::= <url>55 | LINK <url>55
           | <url>55 AND <urlwert>49
           | LINK <url>55 AND <urlwert>49

<string>50 ::= <string>50 <ascii>
           | <ascii>
           | epsilon

<number>51 ::= - <zahl>52
           | + <zahl>52
           | <zahl>52

<zahl>52 ::= <zahl>52 <ziffer>53
           | <ziffer>53

<ziffer>53 ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<real>54 ::= - <zahl>52 . <zahl>52
           | + <zahl>52 . <zahl>52
           | <zahl>52 . <zahl>52
           | - . <zahl>52
           | + . <zahl>52
           | . <zahl>52

<url>55 ::= <zugriffsart>56://<rechnername>57/<>dateiname>58
           | <zugriffsart>56://<rechnername>57 /

<zugriffsart>56 ::= http | ftp | file

<rechnername>57 ::= <string>50
           | <rechnername>57 . <string>50

<dateiname>58 ::= <dateiname>58
           | <string>50 / <dateiname>58

<buchstabe>59 ::= A | ... | Z | a | ... | z

```

Zu erwähnen ist noch, daß das Nichtterminal *ascii* durch ein beliebiges Zeichen auf der Tastatur substituiert werden kann (auf ein explizites Aufführen aller Regeln wurde aus

naheliegenden Gründen verzichtet). Außerdem sollte beachtet werden, daß zwischen Terminalen und Nichtterminalen auf der linken Seite der *url*-Regel keine Whitespacezeichen erlaubt sind (wie sonst üblich)! Das Nichtterminal *tcl_script* wird nicht weiter aufgelöst und bildet die Wurzel für den Ableitungsbaum eines Tcl-Programmes. In diesen Programmen muß lediglich darauf geachtet werden, daß nicht der Ausdruck „END OF TCL“ auftaucht, da der KML-Parser darin vorzeitig das Ende des Tcl-Skriptes erkennen würde.

Es sollte auch klar sein, daß reservierte Wörter als Bezeichner nicht zulässig sind.

Literaturverzeichnis

- [AhSeUI92] **Aho, Alfrd V.; Sethi, Ravi; Ullman, Jeffrey D.:**
Compilerbau.
Addison Wesley, 1992.
- [Behrendt90] **Behrendt, Reinhard (Hrsg.):**
Angewandte Wissensverarbeitung: Die Expertensystemtechnologie erobert die Informationsverarbeitung.
Oldenbourg, 1990.
- [Booch89] **Booch, Grady:**
What Is and What Isn't Object-Oriented Design.
American Programmer vol.2(7-8), 1989.
- [Booch94] **Booch, Grady:**
Objektorientierte Analyse und Design.
Addison-Wesley, 1994.
- [Bullinger95] **Bullinger, H.-J.:**
Rapid Product Development: An Integrated Approach of Human, Technical and Organisational Resources.
Proceedings of the 28th ISATA, Stuttgart 1995.
- [Eck96] **Eck, Oliver:**
Integration von wissensbasierten Informations- und Kommunikationssystemen zur Unterstützung der Produktentwicklung.
„CAD '96 - Verteilte und Intelligente CAD-Systeme“, 1996.
- [Frank96] **Frank, Anett:**
Entwicklung einer aktiven Komponente für ein Aktives Semantisches Netz.
Diplomarbeit, Nr.: 1383, 1996.
- [Friedrich96] **Friedrich, Alexandar:**
Entwicklung einer Programmierschnittstelle für ein Aktives Semantisches Netz auf der Basis des objektorientierten Datenbanksystems ObjectStore.
Studienarbeit, Nr.: 1516, 1996.

- [GenFik92] **Genesereth, Michael R.; Fikes Richard E.:**
Knowledge Interchange Format.
 Reference Manual, Version 3.0, Computer Science Department,
 Stanford University, 1992
- [GetSch91] **Gettys, James; Scheiffler, R.W.:**
Xlib - C Language X Interface. MIT X Consortium Standard. X
Version 11, Release 5.
 MIT, 1991.
- [Gregor90] **Mac Gregor, R.:**
LOOM Users Manual.
 Draft 1990
- [GoKaKeZh92] **Gottheil, K.; Kaufmann, H.-J.; Kern, Th.; Zhao, R.:**
X und Motig: Einführung in die Programmierung des Motif-Toolkits
und des X-Windows-Systems.
 Springer-Verlag, 1992.
- [Habel90] **Habel, Christopher:**
Repräsentation von Wissen.
 Informatik-Spektrum 13, 1990, 126-136.
- [Herold92] **Herold, Helmut:**
lex und yacc: Lexikalische und syntaktische Analyse.
 Addison Wesley, 1992.
- [Heuer92] **Heuer, Andreas:**
Objektorientierte Datenbanken: Konzepte, Modelle, Systeme.
 Addison Wesley, 1992.
- [HopUll94] **Hopcroft, John E.; Ullman, Jeffrey D.:**
Einführung in die Automatentheorie, Formale Sprachen und
Komplexitätstheorie.
 3. Aufl., Addison-Wesley, 1994.
- [HulKin87] **Hull, Richard; King, Roger:**
Semantic Databases Modelling: Survey, Applications and Research
Issues.
 ACM Computing Surveys, Vol. 19, 1987.
- [Johannes90] **Johannes, P.:**
Expertensysteme: Entscheidungskriterien für Manager.
 Oldenbourg 1990.

- [Kim95] **Kim, W.:**
Modern Database Systems: The Object Model, Interoperability, and Beyond.
 Addison Wesley, New York 1995
- [Kurbel92] **Kurbel, Karl:**
Entwicklung und Einsatz von Expertensystemen: Eine anwendungsorientierte Einführung in wissensbasierte Systeme.
 Springer-Verlag, 1992.
- [Ludewig89] **Ludewig, J.:**
Einführung in die Informatik, Skriptum Informatik .
 Verlag der Fachvereine Zürich, 2. Auflage 1989
- [Lusti90] **Lusti, Markus:**
Wissensbasierte Systeme: Algorithmen, Datenstrukturen und Werkzeuge.
 BI-Wissenschaftsverlag, 1990.
- [OSF92] **OSF**
OSF:OSF/Motif Programmer's Guide.
 Open Software Foundation, Cambridge, MA, Revision 1.2, 1992.
- [PaChKhWo89] **Parsaye, K.; Chignell, M.; Khoshafian, S.; Wong, H.:**
Intelligent Databases, Object-Oriented, Deductive Hypermedia Technologies.
 John Wiley & Sons, Ney York 1989.
- [PeScKiQu89] **Peltason, C.; Schmiedel A.; Kindermann,C.; Quantz, J.:**
The Back System Revisited.
 Projekt KIT-BACK, TU Berlin 1989
- [PetDel95] **Petry, Frederick E.; Delcambre, Lois M. L.:**
ADVANCES IN DATABASES AND ARTIFICIAL INTELLIGENCE.
 JAI PRESS INC., 1995.
- [Prata93] **Prata, S.:**
C++: Einführung in die objektorientierte Programmierung.
 The Waite Group, 1993.
- [Reimer89] **Reimer, Ulrich:**
FRM: ein Frame-Repräsentationsmodell und seine formale Semantik.
 Informatik-Fachberichte 198, Springer-Verlag 1989.

- [Reimer91] **Reimer, Ulrich:**
Einführung in die Wissensrepräsentation.
Teubner 1991.
- [Retti84] **Retti, J. u.a.:**
Artificial Intelligence: Eine Einführung.
Teubner 1984.
- [Richter89] **Richter, Michael M.:**
Prinzipien der Künstlichen Intelligenz.
Teubner, 1989.
- [RoEcBiSt95] **Roller, Dieter; Eck, Oliver; Bihler, Monika; Stolpmann, Markus:**
An Active Semantic Network for supporting Rapid Prototyping.
Proceedings of the 28th ISATA, Stuttgart, September 1995.
- [Roller95] **Roller, Dieter:**
Werkzeuge für die Produktentwicklung.
CAD-CAM Report Nr. 2, 1995
- [ScBoGeKa94] **Scheller, Boden, Geenen, Kampermann:**
Internet: Werkzeuge und Dienste.
Springer-Verlag, 1994.
- [Scheffe91] **Scheffe, Peter:**
Künstliche Intelligenz — Überblick und Grundlagen.
BI-Wissenschaftsverlag, 1991.
- [Schönig92] **Schönig, Uwe:**
Theoretische Informatik kurz gefaßt.
BI-Wissenschaftsverlag, 1992.
- [Sedgewick92] **Sedgewick, Robert:**
Algorithms in C++.
Addison Wesley, 1992
- [Shankar84] **Shankar, K.:**
Data Design: Types, Structures, and Abstraction. Handbook of Software Engineering.
Van Nostrand Reinhold, 1984
- [Shaw84] **Shaw, M.:**
Abstraction Techniques in Modern Programming.
IEEE Software vol. 1(4), 1984.

- [Stroustrup92] **Stroustrup, Bjarne:**
Die C++ Programmiersprache.
2., überar. Auflage., Addison-Wesley, 1992.
- [Taylor92] **Taylor, David A.:**
Objektorientierte Technologien: Ein Leitfaden für Manager.
Addison Wesley, 1992.
- [Vossen87] **Vossen, G.:**
Datenmodelle, Datenbanksprachen und Datenbank-Management-Systeme.
Addison-Wesley, 1987.
- [Vossen91] **Vossen, G.:**
Data Models, Database Languages and Database Management Systems.
International Computer Science Series. Addison-Wesley, 1991.
- [Wittig92] **Wittig, U.:**
X Windows System und OSF/Motig.
IWT Verlag GmbH, 1992.
- [Woods87] **Woods, W.A.:**
Knowledge Representation: What´s important About it?
in: Cerone, N., McCalla, G. (eds.): *The Knowledge Frontier*, S. 44-79,
Springer 1987.

