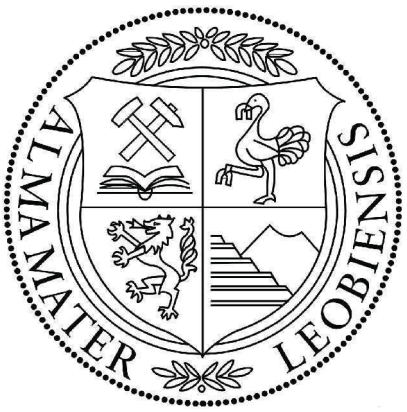
Montanuniversität Leoben
Studienrichtung industrieller Umweltschutz, Entsorgungstechnik und Recycling



DIPLOMARBEIT

am

Institut für Automation

Betreuer: o.Univ.Prof.Dr.tech.Dipl.-Ing. Paul O'Leary

zum Thema

ENTWURF EINES OBJEKTORIENTIERTEN UND WISSENSBASIERTEN PROGRAMM-MUSTERS MITTELS SCHICHTENARCHITEKTUR ANHAND EINER ERDÖLBOHRDATENINTERPRETATION

von

Engelbert Steiner

Hiermit versichere ich, daß ich die vorliegende Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Leoben, 3.September 2003

(Steiner Engelbert)

Mein Dank gilt

Herrn o.Univ.Prof.Dr.tech.Dipl.-Ing. Paul O'Leary

für die Betreuung und Unterstützung bei all meinen Anliegen,
sowie für die gewährten Möglichkeiten und Freiheiten,

dem Institut für Automation und all seiner Mitarbeiter/-innen und

meiner Mutter,

die mir das Studium ermöglichte.

Abstrakt

In Folge der zunehmenden Computerisierung werden immer mehr Programme von Technikern und Experten geschrieben; deren unterschiedliche Programmierfähigkeiten führen zu uneinheitlichen und teils unausgereiften Programmen, was aus Sicht des Softwareengineerings und des Wissensmanagements nicht zielführend ist.

Diese Arbeit beschäftigt sich mit der Erstellung eines Programm-Musters anhand einer Interpretation von Erdölbohrdaten. Um die geforderten Entwurfsprinzipien Modularität und Transparenz zu realisieren, wurden Konzepte der Wissensbasierung, Objektorientierung und Schichtenarchitektur angewandt.

Im Theorieteil werden zuerst die notwendigen Grundlagen und Konzepte der Künstlichen Intelligenz, Expertensysteme, Objektorientierung und Schichtenarchitektur erörtert.
Im Praxisteil wurde ein Programmsystem für die Interpretation von Sensordaten einer Erdölbohrung als Prototyp entwickelt und implementiert. Die generischen Programmaufgaben wurden objektorientiert in Java realisiert. In dieses objektorientierte Programmsystem wurde ein wissensbasiertes System, welches mittels dem regelbasierten Expertensystem Jess erstellt wurde, integriert. Bei der Bildung der Systemarchitektur wurde das Schichten-Entwurfsprinzip angewandt.
Das Ergebnis ist ein mehrteiliges hybrides Programmsystem für Auswertungen und Interpretationen, deren Wissen durch Objekte und Regeln repräsentiert werden kann.

Abstract

Due to the increasing computerization, more and more programs are written by technical experts; their different programming abilities lead to varying and partial imperfectly results, that don't serve software engineering and knowledge management.

This thesis works on the creation of a program pattern with the help of an interpretation of oil drilling data. Knowledgebased and objectorientated concepts and layer-architecture are applied to realize the requested design principles modularity and transparency.

The theoretical part contains necessary basics and concepts of artificial intelligence, expert systems, objectorientation and layer-architecture.
In the practical part, a program system for the interpretation of sensor data of an oil drilling has been developed and implemented as a prototype . The generic tasks of the program have been realized objectorientated in java. A knowledgebased system, made with the rulebased expertsystem jess, has been integrated into the objectorientated program system. The layer design principle has been used for the formation of the system architecture.
The result is a complex hybrid program system for evaluations and interpretations, of which knowledge can be represented with objects and rules.

Inhaltsverzeichnis

1 Einführung

1.1 Motivation

In vielen technischen Unternehmen und Instituten gibt es oft keine professionellen Informatiker (Entwickler), und so werden nur bei großen Programmprojekten externe Informatiker hinzugezogen. Deshalb müssen die Techniker und Experten die benötigten kleinen und mittleren Programme selbst schreiben oder sich mit Ferialpraktikanten behelfen. Aufgrund dessen wird mit vielen unterschiedlichen und zum Teil veralteten Programmiersprachen gearbeitet, wobei verschiedene oder mangelnde Programmkonzepte verwendet werden. Diese behelfsmäßigen Programme sind oft schlecht strukturiert und dadurch nicht weiter nutzbar. Die spätere Weiterverwendung bzw. Erweiterung ist oft bereits für den Programmautor schwer und erst recht für Andere. Dadurch ist das Expertenwissen des Autors an diesen gebunden und nicht weiter nutzbar, was unter dem Gesichtspunkt des Wissensmanagements nicht vertretbar ist.

Bei der konkreten Aufgabenstellung am Institut für Automation entwickeln Technik-Ingenieure und Technik-Studenten, die über Fachwissen und Programmierwissen verfügen, einzelne Programme. Aufgrund der unterschiedlichen Programmierfähigkeiten dieser Techniker entstehen Programme mit unterschiedlichen Programmkonzepten. Teils wird dabei bereits mehr oder weniger objektorientiert, aber unabhängig davon konventionell programmiert. Dabei sind die unterschiedlichen Programmaufgaben wie Ein-/Ausgabe, Problemlösung und Datenhaltung zum Teil miteinander verzahnt. Weiters ist durch die konventionelle Programmierung das Wissen im Programmcode verteilt und verzerrt und so für spätere Projekte schwer nutzbar. Dadurch ist die Weiterverwendung des Programms bzw. bestimmter Programmteile und speziell des Programmwissens für Andere schwierig und somit geht ein Teil des Wissens wieder verloren.

Probleme:

- Die Programme haben verschiedene oder fehlende Programmkonzepte, was die Weiterverwendung des Programms verunmöglicht.

- Die Programme und deren Aufgaben sind verschieden oder gering strukturiert, was die Weiterverwendung einzelner Programmteile behindert.
- Das Wissen ist im Programm verteilt und durch die prozedurale Logik der Programmierung oft verzerrt, wodurch das Wissensmanagement erschwert wird.

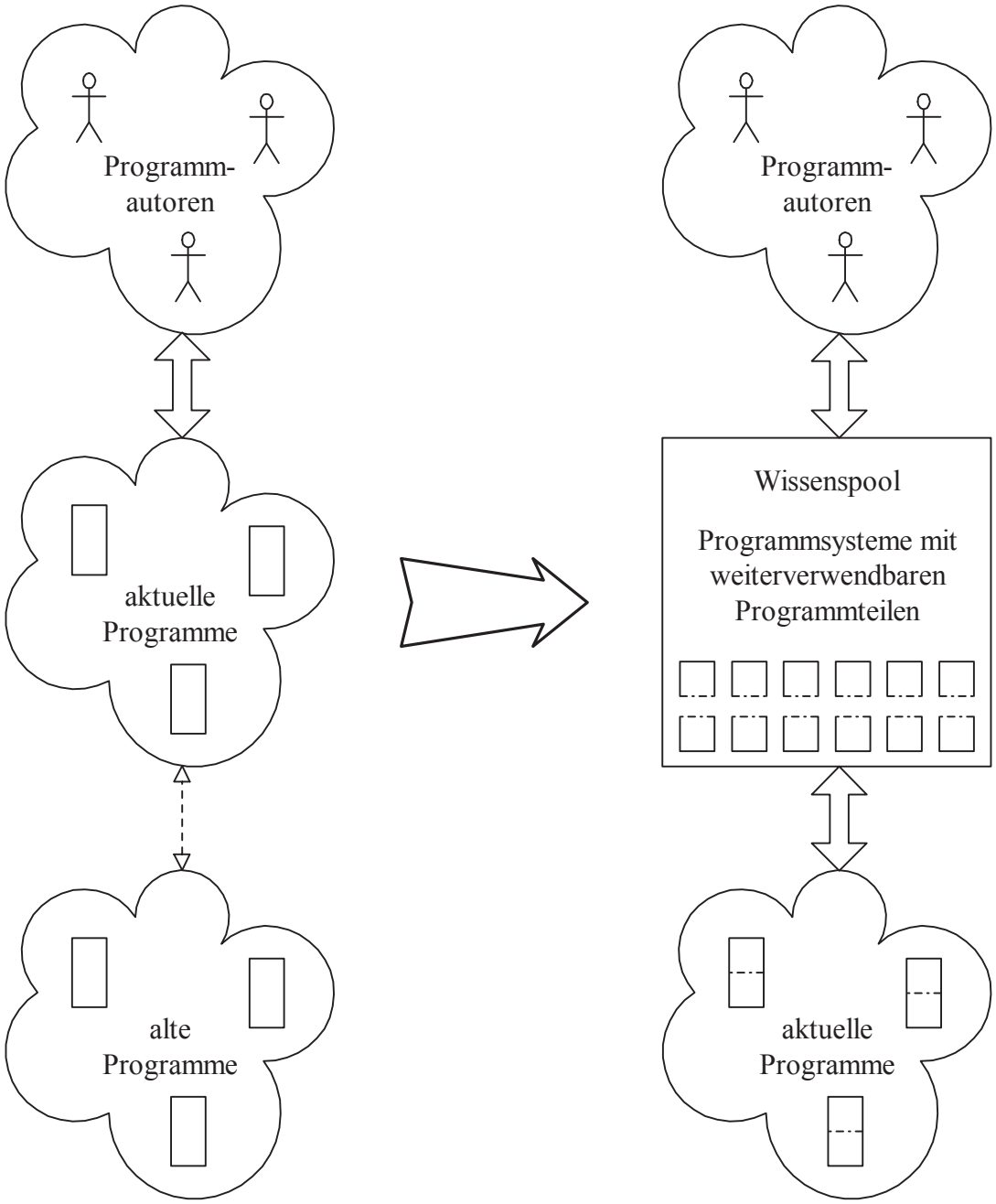


Abbildung 1.1: herkömmliche Programmentwicklung und effizientere Systementwicklung.

Deswegen wäre ein Programmkonzept wünschenswert, das auch nachfolgenden Entwicklern

ermöglicht, die bereits erstellten Programme und das darin enthaltene Wissen nachzuvollziehen, weiterzuverwenden und darauf aufzubauen (siehe Abb 1.1).

1.2 Aufgabenstellung

Die Aufgabe meiner Diplomarbeit ist das Erschliessen von wissensbasierten Konzepten, das Erstellen eines objektorientierten Programm-Entwurfsmusters und dessen exemplarische Implementation anhand einer Interpretation von Sensorendaten einer Erdölbohrung.

Dabei sind folgende **Entwurfsprinzipien** umzusetzen:

- Modularität und
- Transparenz

für Verständlichkeit, Wiederverwendbarkeit und Erweiterbarkeit.

Ziele:

- Programm-Muster, das auf andere Aufgaben desselben Anwendungsbereichs bzw. für konzeptionell ähnliche Probleme herangezogen werden kann;
- klare Programmstruktur, modulare Programmkomponenten, und
- geeignete Wissensdarstellung für Verständlichkeit und Weiterverwendbarkeit.

1.2.1 Beschreibung der Interpretation einer Erdölbohrung

Bei einer Erdölbohrung werden oft sehr viele Sensorenwerte aufgenommen. Die Aufzeichnung dutzender Sensoren-Kanäle, die jeweils einmal pro Sekunde einen Wert liefern, führt zu einer Datenflut, welche eine sinnvolle Auswertung erschwert. Wegen dieser Datenflut werden diese erhobenen Meßwerte nur stichweise und unsystematisch durch den Bohringenieur ausgewertet oder sogar nur ungesehen archiviert. Oder die unsystematische Auswertung ist für andere Anwender nicht mehr verwendbar, und somit geht das Expertenwissen verloren.

Aufgaben:

- Datenflut reduzieren:
  - wichtige Sensor-Kanäle behalten;

  - Sensor-Kanal-Werte auswerten.

- Aus Daten Informationen ableiten, die Aussagekraft besitzen (für Reporting):
  - basierend auf einzelne Sensor-Kanäle Aussagen treffen, wobei auch das Zeitverhalten interessant ist;
  - basierend auf mehrere Sensor-Kanäle Aussagen treffen.

Die Meßdaten werden entweder in Dateien (XML) zwischengespeichert und dann Offline oder direkt Online ausgewertet.

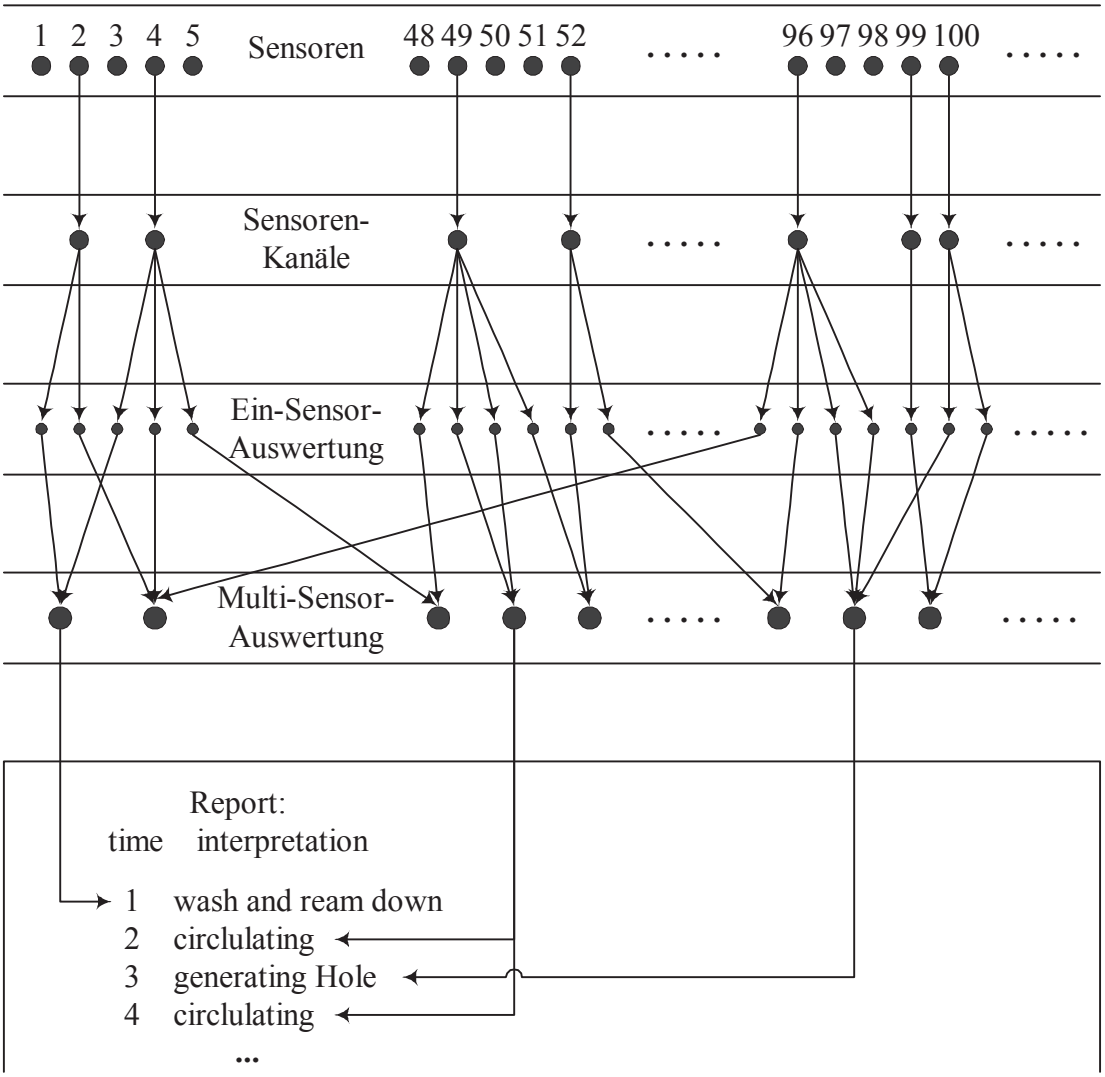


Abbildung 1.2: schematischer Ablauf einer Bohrdatenauswertung.

Gesucht ist ein objektorientiertes Programmsystem zur systematischen Interpretation der Erdöl-

bohrdaten, welches eine modulare und transparente Programmarchitektur aufweist und eine geeignete Wissensdarstellung realisiert.

1.2.2 Anforderungskatalog

- Konzept des Expertensystems erarbeiten und anwenden;
- Programm-Muster anhand der Erdölbohrdatenauswertung erstellen;
- Entwurfsprinzipien bei Umsetzung berücksichtigen;
- Verwenden der modernen objektorientierten Programmiersprache Java;
- Verwenden von XML für Daten- und Wissensaustausch.
- Didaktische Ziele:
  - Selbstständige Wissenserarbeitung:
    - Programmierung und Objektorientierung,
    - künstliche Intelligenz und Expertensysteme.
  - Informationssuche: Internet, Bücher, Artikel;
  - Problemlösung.

1.2.3 Lösungsansatz

Für das Problem der Bohrdateninterpretation bietet sich ein **konventionelles objektorientiertes System** an, da dieses deklaratives und prozedurales Wissen auf natürliche und mächtige Weise darstellen kann. Jedoch hat die Objektbasierung die Tendenz, das zur Interpretation benötigte Wissen auf verschiedene Objekte zu dezentralisieren, was die Transparenz und somit die Weiterverwendbarkeit beeinträchtigt. Da dies im Gegensatz zu den Entwurfsprinzipien steht, muß ein anderer Ansatz gefunden werden.
Hier bietet sich ein **wissensbasiertes System** wie z.B.: ein Expertensystem an, bei dem eine Trennung von Wissen und Programmsteuerung stattfindet. Das ermöglicht die zentrale und modulare Darstellung des Wissens, woraus sich eine Problemlösungskompetenz des Programmsystems und neue Problemlösungsmöglichkeiten ergeben.
Die komplette wissensbasierte Umsetzung ist aber nicht optimal, da sich ein wissensbasiertes System oft nicht für konventionelle Programmaufgaben eignet.
Deshalb ist eine **Kombination** aus wissensbasierten und objektorientierten Programmsystemen sinnvoll, um die Vorteile von Beiden nutzen zu können. Durch diese Kombination entsteht ein mehrteiliges Programmsystem, dessen erhöhte Komplexität wiederum mit den Entwurfsprinzipien

in Einklang gebracht werden muß.
Dies kann dadurch erreicht werden, indem das **Schichten-Entwurfsprinzip** bei der Bildung der Systemarchitektur angewandt wird[1]. Die Systemarchitektur basiert dabei auf Schichten mit verschiedenen Abstraktions-Niveaus.

Als **Ergebnis** entsteht ein komplexes Programmsystem, das sowohl ein objektorientiertes als auch ein wissensbasiertes Programmsystem kombiniert. Dessen Modularität und Transparenz wird dabei durch die Schichtenarchitektur gewährleistet.

1.3 Gliederung der Arbeit

In Kapitel 1 sind die Aufgabenstellung und die Anforderungen vorgestellt und definiert worden. Aus diesen Angaben und aus gewonnenen Erkenntnissen während der Diplomarbeit wurde der Lösungsansatz abgeleitet und hier vorgestellt, um die weitere Vorgehensweise in der Diplomarbeit offenzulegen. Entsprechend diesem Lösungsansatz, der Kombination eines objektorientierten und eines wissensbasierten Systems mittels Schichten-Architektur, ergeben sich die folgenden Theoriekapitel.

In Kapitel 2 erfolgt eine Einführung in die künstliche Intelligenz und Expertensysteme, um den Überblick und das nötige Wissen für die folgenden detailierten Theoriekapiteln zu gewährleisten. Dabei werden grundlegende Begriffe und Konzepte der künstlichen Intelligenz und der Expertensysteme erläutert. Weiters werden die Gebiete der künstlichen Intelligenz aufgeführt und die Verbindung mit der Informatik umrissen.

Kapitel 3 behandelt mit Repräsentation und Suche die beiden Schwerpunkte der künstlichen Intelligenz. Bei der Repräsentation werden zuerst die grundlegendenen Symbolsysteme Aussagenlogik und Prädikatenlogik und weiters die Wissensrepräsentationen Produktionsregel, semantisches Netz, Frame und hybride Repräsentation erläutert. Mit der Zustandsraumrepräsentation erfolgt die Überleitung zum Schwerpunkt Suche, wo zuerst die erschöpfenden, uninformierten Suchalgorithmen vorgestellt werden. Dann wird auf die intelligente heuristische Suche und deren Bedeutung für Expertensysteme eingegangen.

Kapitel 4 beschäftigt sich detailiert mit dem Expertensystem, dem für diese Diplomarbeit relevantem Teilgebiet der künstlichen Intelligenz. Zu Beginn werden allgemeine Aspekte des Expertensystems wie Einsatzgebiete, Entwicklung und Akteure behandelt. Dann wird das Produktionssystem, die Grundlage des Expertensystems, und folgend das regelbasierte Expertensystem erklärt.

[1] Während des Architekturentwurfs wird die Systemstruktur erstellt, d.h. es werden die grundlegenden Komponenten des Programmsystems und die Abhängigkeiten zwischen ihnen dargestellt.

Kapitel 5 behandelt mit Objektorientierung und Schichtenarchitektur die restlichen theoretischen Grundlagen, die entsprechend dem Lösungsansatz benötigt werden.

In Kapitel 6 erfolgt die Systemanalyse, in der zu Beginn die Szenarios bei der Programmentwicklung und die Rollen der Akteure behandelt werden. Weiters wird Aufgabenstellung und Lösungsansatz aus Kapitel 1 in den Abschnitten System-Stufen, Repräsentation und Programmsystem analysiert und präzisiert. Mit dem Abschnitt System-Schichten beginnt die eigentliche Systementwicklung, anschließend wird mit dem statischen Modell die fachliche Lösung und mit dem dynamischen Modell das Verhalten des Systems spezifiziert.

Kapitel 7 erörtert zuerst relevante Aspekte des gewählten Expertensystems. Dann erfolgt im Systementwurf die Umsetzung der in der Systemanalyse spezifizierten Anwendung. Im Zuge der Realisierung und Optimierung werden die Modelle der Systemanalyse überarbeitet und erweitert.

In Kapitel 8 wird das in den vorigen Kapiteln entwickelte Programmsystem mittels Sensordaten einer realen Erdölbohrung implementiert und getestet. Zuerst werden die Input-Daten und die Regeln erklärt, dann wird der Programmablauf anhand eines Programmprotokolls erörtert und schlussendlich erfolgt die Darstellung sämtlicher Daten einschließlich des Auswertungsergebnisses.

2 Einführung in künstliche Intelligenz

Zu Beginn dieses Kapitels werden die verschiedenen Ansätze der künstliche Intelligenz vorgestellt. Dann werden die Begriffe Intelligenz, Wissen, Problem und Symbolsystem definiert. Weiters wird im Vergleich zum konventionellen Programmsystem das Konzept des wissensbasierten Programmsystems aufgezeigt, und im Zuge dessen das Expertensystem eingeführt. Anschließend folgt ein Überblick über die Hauptschwerpunkte Repräsentation und Suchen, die KI-Gebiete und deren Gemeinsamkeiten. Abschließend wird die Beziehung zwischen KI und Informatik abgehandelt.

Ziel der künstlichen Intelligenz (KI) ist die Erforschung und Nachbildung intelligenten Verhaltens. Dazu repräsentieren die meisten KI-Programme Wissen in irgendeiner formalen Sprache, die dann durch Algorithmen manipuliert wird.

2.1 KI-Ansätze

Intelligenz ist zu komplex, um durch eine Theorie beschrieben zu werden. Nach [lug01] existieren Hierarchien von Theorien, die Intelligenz auf **verschiedenen Abstraktionsebenen** charakterisieren. Auf der untersten Ebene dieser Hierarchie sind neuronale Netze, genetische Algorithmen und andere Formen des emergenten Rechens.
Bei neuronalen Netzen werden mit Hilfe von Modellen, die der Struktur von Neuronen des menschlichen Gehirns nachempfunden sind, intelligente Programme erstellt. Das Wissen des Systems wird nicht mit Hilfe expliziter Symbole und Operationen dargestellt, sondern geht implizit aus dem gesamten Netzwerk von neuronalen Verbindungen und Schwellenwerten hervor.
Die **traditionellen rationalistischen Techniken** verwenden explizit repräsentiertes Wissen und sorgfältig entworfene Suchalgorithmen zur Implementierung von Intelligenz. Dabei wird von der Mathematik ausgegangen, wobei der Glaube an das logische Schließen als Muster der Intelligenz und an eine objektive Grundlage des logischen Schließens unterstellt wird. Logiker arbeiten auf der höheren Ebene der Deduktion, Abduktion, Induktion, Wahrheitskonservierung und zahlloser anderer Modi und Arten des Schließens.
Noch weiter oben in der Hierarchie befinden sich die Expertensysteme, intelligente Agenten und

natürlichsprachliche Programme.

Traditionelle KI-Programme reagieren auf Störungen für gewöhnlich äußerst überempfindlich; statt sich elegant zu verabschieden, neigen solche Programme eher dazu, entweder erfolgreich zu sein oder komplett fehlzuschlagen. Menschliche Intelligenz ist hier viel flexibler und kann unklare Eingaben sehr gut interpretieren. Neuronale Architekturen stellen weiters ein natürliches Modell für die Parallelverarbeitung dar, da jedes Neuron eine unabhängige Einheit ist. Menschen können in der Regel eine Aufgabe schneller ausführen, wenn sie mehr darüber in Erfahrung bringen, während Computer eher langsamer werden. Diese Verlangsamung ist durch den Aufwand bedingt, den sequentielles Durchsuchen einer Wissensbasis mit sich bringt. [lug01]

Hier wird weiters auf den traditionellen logischen Ansatz eingegangen.

2.2 Begriffe

Intelligenz

> *Intelligenz verknüpft zwei Dinge: Besitz von Wissen in Form von Daten und die Kombination dieser Daten zur Gewinnung neuer Erkenntnisse durch „Überlegung“ und „Nachdenken“. [hale90]*

Der Besitz von Wissen, die Datenfülle, bedeutet demnach allein noch keine Intelligenz, andererseits ist Intelligenz ohne Wissen undenkbar.

Wissen

Nach [hale90] und [gfh90] lassen sich folgende Wissensarten unterscheiden:

Implizites Wissen: Dieses Wissen steckt z.B.: in Form eines Algorithmus im Code eines Programmes. Der Algorithmus wird nicht aufgerufen, weil das Programm nach diesem Algorithmus gesucht hat, sondern weil der Programmierer wußte, daß er ihn im Programmablauf an dieser Stelle benötigt. Somit ist das Wissen, daß dieser Algorithmus an einer bestimmten Stelle ausgeführt werden muß, im Programmcode versteckt.

Explizites Wissen: Hier beinhaltet das Programm Wissen über die Problemlösungsmöglichkeiten und wendet diese entsprechend den Bedürfnissen der Aufgabenstellung an.
Systeme, die Wissen in expliziter Form enthalten, werden als **wissensbasierte Systeme** (z.B. Expertensysteme) bezeichnet.

Deklaratives Wissen: ist eine rein statische Ansammlung von beschreibenden Wissenselementen wie z.B.: Fakten, Objekten und Konzepten. Es wird beschrieben, was getan werden muß, d.h. in welcher Relation Entitäten der betrachteten Welt zueinander stehen. Bei deklarativen Wissen sind Informationen über die Zusammenhänge zwischen einzelnen Wissenselementen, vor allem darüber, wann und wo bestimmte Informationen eingesetzt werden, nicht Bestandteil des Wissens selbst. Eine getrennte, aktive Komponente verwendet dann dieses Wissen, um eine Lösung zu finden.

Prozedurales Wissen: ist nicht statisch im oben genannten Sinn, da eine entsprechende Aktion ausgeführt wird, z.B.: Regeln und Prozeduren. Es wird angegeben, wie eine Aufgabe gelöst werden muß, d.h. welche Aktionen auszuführen sind. Details der Aktion sind in Verarbeitungsanweisungen abgelegt.

Bei der Zuordnung von explizitem/implizitem Wissen versus deklarativem/prozeduralem Wissen wird deklarativ meist mit explizit assoziiert. Schwierig ist die Unterscheidung zwischen explizitem und implizitem prozeduralen Wissen. Bei explizitem prozeduralen Wissen erfolgt die Beschreibung der Auswahl eines bestimmten Algorithmus nicht prozedural, sondern deklarativ, und der Aufruf des Algorithmus unterliegt der expliziten Kontrolle des Programms.

Problem

> *Ein Problem kann durch Anfangszustand, gewünschten Zielzustand (Endzustand) und zustandstransformierende Operatoren (z.B.: Regeln) beschrieben werden. Eine Lösung ist eine Operatorfolge, die Anfangszustand mit Zielzustand verbindet. [gfh90]*

Diese einfache Sicht des Problemlösens kann auf verschiedenste Wissensrepräsentationen, egal ob prozedural oder deklarativ, angewandt werden. Operatoren können komplexe Problemlösungsverfahren wie das Lösen von Gleichungssystemen oder einfache Regeln bei Produktionssystemen sein.

Symbolsystem

In der Logik und der Informatik befindet sich in physischen Symbolsystemen Intelligenz[1].
Symbolsysteme sind Sammlungen von Mustern und Prozessen. Muster können Datenstrukturen,

[1] Die physische Symbolsystemhypothese wird von Kritikern angegriffen, die argumentieren, Intelligenz sei inhärent biologisch und existenziell und könne daher nicht symbolisch reprodruziert werden

Objekte, Prozesse und andere Muster beschreiben und interpretiert werden, wobei die Interpretation die Selektion und Anwendung von Operatoren erfordert. Prozesse sind durch Operationen in der Lage, Muster zu erzeugen, zu zerstören und zu ändern. Symbolsysteme können Probleme lösen, indem sie potenzielle Lösungen generieren und diese testen, d.h. indem sie suchen. In der Regel werden Lösungen gesucht, indem Symbolausdrücke erzeugt und dann so lange modifiziert werden, bis sie die für eine Lösung erforderlichen Bedingungen erfüllen.

Nach [lug01] wird intelligentes Verhalten von Menschen und Maschinen erreicht durch den Einsatz von:

1. **Symbolmustern** zur Repräsentation wichtiger Aspekte der Problemdomäne;
2. **Operationen** auf diesen Mustern, mit denen potenzielle Problemlösungen generiert werden;
3. **Suchverfahren** zur Auswahl einer Lösung aus diesen potenziellen Lösungsmöglichkeiten.

Daraus ergeben sich folgende Schwerpunkte der KI-Forschung und -Anwendungsentwicklung: Die Definition der Symbolstrukturen und Operationen, die für intelligente Problemlösungsverfahren erforderlich sind, sowie die Entwicklung von Strategien für das effiziente und korrekte Durchsuchen der von diesen Strukturen und Operationen generierten potenziellen Lösungen. [lug01]

2.3 Wissensbasiertes vs. konventionelles Programmsystem

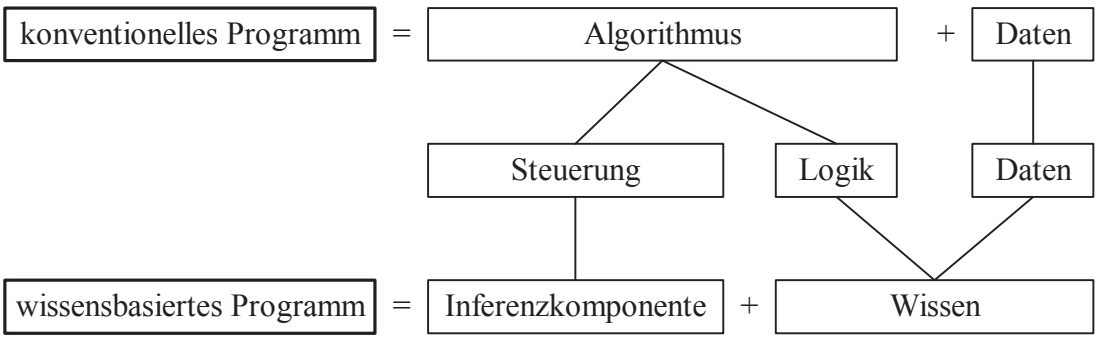


Abbildung 2.1: konventionelles und wissensbasiertes Programmkonzept.

Bei einem **konventionellen Programmsystem**[2] sind Logik (Fachwissen) und Steuerung (Datenverarbeitungswissen) direkt miteinander im Algorithmus[3] (Ablauflogik) verzahnt (siehe Abb.

[2] Ein konventionelles Programm ist ein streng formalisierter, eindeutiger und detaillierter Algorithmus.

[3] Ein Algorithmus ist eine Programmlösungsbeschreibung in Form von Arbeitsanweisungen, die festlegt, wie ein Problem gelöst werden soll.

2.1 nach [hale90]) und beeinflussen sich schon bei der kleinsten Änderung gegenseitig. Das Fachwissen ist hardcoded eingebettet, wie Atome in eine kristalline Struktur. Z.B.: ist die Abfrage 2 in Abb. 2.2 (siehe Seite 13) für sich allein praktisch bedeutungslos und nur in Verbindung mit Abfrage 1 korrekt anwendbar. Die Folge ist, daß bei Veränderungen der Struktur schwer zu überblickende Versetzungen eintreten können. Bei der Programmierung steht der Algorithmus und seine Optimierung im Mittelpunkt.
Konventionelle Programme eignen sich für Berechnungen, die auf konsistenten, exakt abgesicherten und wohl definierten Theorien beruhen. Dabei sind Datenmodelle und -strukturen nur elementar realisierbar. [hale90]

Ein **Expertensystem** ist ein wissensbasiertes Programmsystem, mit dem die Fachkompetenz von Experten (Expertenwissen) in einer Wissensbasis gebündelt und EDV-gerecht zur Lösung von Problemen bereitgestellt wird.

Expertensysteme enthalten die zwei **grundlegenden Komponenten**:

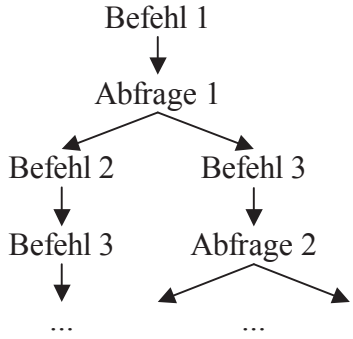
1. Wissensbasis oder Wissensbank (knowledge base),
2. Schlussfolgerungs- bzw. Inferenzmechanismus (inference engine).

Mit Hilfe der **Inferenzkomponente** werden für ein vorgegebenes Problem auf Grundlage des Wissens in der Wissensbasis entsprechende Schlussfolgerungen gezogen und eine Problemlösung vorgeschlagen. Sie ist eine Verarbeitungsstrategie, die Fähigkeiten elementaren Denkens besitzt, aber selbst über kein Wissen verfügt. [hale90]
Aufgrund dieser Architektur ergeben sich für das Expertensystem folgende **Vorteile**:
Die Zusammenfassung von Logik und Daten repräsentiert das Wissen in einer expliziten und tieferen Darstellungsform als beim konventionellen Programm. Die Steuerung erfolgt durch eine separate Inferenzkomponente. Aus der Trennung von Wissen und Steuerung und der expliziten Wissensdarstellung resultieren Problemlösungskompetenz des Programmsystems und neue Problemlösungsmöglichkeiten.
Dadurch können komplexe Probleme in Angriff genommen werden, für die keine gesicherten Theoriegebäude existieren, die schlecht strukturiert sind, konfliktiv sind, eine sehr große Zahl von Lösungsalternativen zulassen, oder nur durch vages bzw. unvollständiges Wissen charakterisiert sind. Die Trennung hat auch Vorteile für die Programmierung selbst.

Eine Wissensrepräsentation, die diese Trennung begünstigt, sind Regeln der Form „wenn X dann Y“. Abb. 2.2 veranschaulicht den Unterschied zwischen konventionellen anweisungsbasierten und dem in Expertensystemen häufig verwendeten regelbasierten Programmierstil. Regeln sind eine Darstellungsform für explizites prozedurales Wissen und ermöglichen die automatische

Schlussfolgerung von bekannten Informationen auf neue Informationen (siehe 3.3.1 Produktionsregel).

1. anweisungsbasierter Programmierstil:
   Programm = Sequenz von Befehlen und Abfragen



   Der Programmierer legt fest, was getan und in welcher Reihenfolge es getan wird.

2. regelbasierter Programmierstil:
   Programm = Menge von Regeln und Inferenzkomponente

   Regel 1: Wenn Situation X1, dann Aktion Y1.
   Regel 2: Wenn Situation X2, dann Aktion Y2.
   Regel 3: Wenn Situation X3, dann Aktion Y3.

   Der Experte legt fest, was getan wird, die Reihenfolge bestimmt die Inferenzkomponente.

Abbildung 2.2: Unterschied zwischen anweisungs- und regelbasiertem Programmkonzept.

Während beim anweisungsbasierten Programmierstil primitive Operationen in einer vom Programmierer fest vorgegebenen Reihenfolge sequentiell abgearbeitet werden, legt der Experte beim regelbasierten Programmierstil nur fest, was in einer bestimmten Situation getan werden soll. In welcher Reihenfolge die Regeln zur Problemlösung verwendet werden, entscheidet die Inferenzkomponente (Regelinterpreter). Während bei anweisungsbasierten Programmen der Kontrollfluß übersichtlich aber starr ist, weisen regelbasierte Programme gerade die umgekehrten Eigenschaften auf.
Der Nachteil der regelbasierten Programmierung ist der unübersichtliche Kontrollfluß, der große Regelsysteme schwer handhabbar macht. Daher werden ersetzend oder ergänzend andere Wissensrepräsentationen benutzt, die objektorientierte Strukturierung ermöglichen. [pup91]

2.4 Überblick

2.4.1 Repräsentation

Wissen über die Umgebung ist eine notwendige Voraussetzung für intelligentes Verhalten.

> *Wissensrepräsentation kann als Abbildung von Ausschnitten einer Welt (Anwendungsbereich, Problemdomäne) in eine für den Computer verarbeitbare Form aufgefaßt werden. Sie beschäftigt sich mit dem Problem, die gesamte Menge an Wissen, die für intelligentes Verhalten erforderlich ist, in einer verarbeitbaren formalen Sprache zu beschreiben. [gfh90]*

Mittels Konzeptualisierung d.h.: Begriffsbildung oder Versinnbildlichung erfolgt diese Formalisierung eines Ausschnitts der realen[4] Welt.

Bei der **Wahl der Repräsentation** ist darauf zu achten, sowohl die für den Problemlösungsprozeß wesentlichsten Zusammenhänge auch am klarsten und einfachsten darzustellen, und die effiziente Durchführung der Aufgaben zu ermöglichen. Je nach Aufgabe und Abstraktionsniveau wird ein mehr oder weniger großer und detaillierter Ausschnitt aus der realen Welt dargestellt. [gfh90]

Aus dem Anspruch, durch automatische Verarbeitung von Wissen intelligentes Verhalten zu erzielen, und aus der Verwandtschaft zur Informatik, ergeben sich **Anforderungen** an die Wissensrepräsentation und somit an die interne Struktur der Wissensbasis:

Verarbeitbarkeit: Es muß die Möglichkeit gegeben sein, aus bestehenden Wissen auf neues Wissen zu schließen z.B.: durch Regeln und Schlussfolgerungsmechanismen.

Flexibilität: Ein und derselbe Repräsentationsansatz soll geeignet sein, Wissen aus möglichst verschiedenen Anwendungsbereichen (Domänen) darzustellen, und unabhängig vom Anwendungsgebiet die unterschiedlichen Informationen effizient darzustellen.

Modularität: Die Wissensbasis soll leicht veränderbar und modular aufgebaut sein. Weiters soll die Wissensbasis von dem Inferenzmechanismus getrennt sein.

Verständlichkeit: Der Inhalt der Wissensbasis muß leicht verständlich darstellbar sein. [gfh90]

[4] Der Ausdruck „real“ bezieht sich hier nicht nur auf physisch vorhandene Dinge, sondern auch auf alle Ideen, Vorstellungen und Abstraktionen, die für die Lösung des Problems sinnvoll oder notwendig sind.

Wissensverarbeitung benötigt außer Schlussfolgerungsmechanismen auch eine sinnvolle Reihenfolge bei der Abarbeitung. Aus der Fülle der Wissensbasis muß dabei nach gerade relevanten Aussagen (bei einer Menge von Regeln z.B. die best anwendbare Regel) gesucht werden. [hale90]

2.4.2 Suchen

> *Suchen ist eine Problemlösungstechnik, die systematisch den Raum von Problemzuständen erforscht, d.h. aufeinander folgende und alternative Zustände des Problemlösungsprozesses.*
> *Je nach Vorgabe kann das Problem darin bestehen, eine beliebige einzelne Lösung oder alle Lösungen zu einem Problem, oder von möglichen Lösungen die Beste zu finden.*

Oft kann eine gestellte Aufgabe als Suchproblem formuliert werden. In manchen KI-Bereichen stellen reine Suchalgorithmen den Lösungsansatz dar, diese sind aber sehr aufwendig und können bei realen großen Problemen praktisch undurchführbar sein und somit versagen. Dieser Ansatz wird auch als **erschöpfende oder blinde Suche** bezeichnet und beinhaltet keine Intelligenz.
Menschen verwenden beim Problemlösen **intelligente Suchverfahren** und führen keine erschöpfende Suche durch. Die Suche kann durch eine auf Alltagswissen beruhende Sicht der menschlichen Problemlösungsverfahren wie z.B. Faustregeln und Tricks unterstützt werden. Diese Faustregeln und Tricks, sogenannte **Heuristiken**, statten die Suche mit Intelligenz aus und ermöglichen so eine effizientere Problemlösung.

2.4.3 KI-Gebiete

Nach [lug01] gliedert sich KI in eine Reihe von Teildisziplinen, die zwar grundsätzlich den gleichen Problemlösungsansatz verfolgen, sich aber mit verschiedenen Anwendungen beschäftigen.

Spiele spielen Brettspiele gehören zu den ersten Anwendungsgebieten der KI und lieferten entscheidende Anstöße im Bereich Zustandsraumsuche. Die Erzeugung von Suchräumen wird dadurch erleichtert, daß die meisten Spiele nach einer wohldefinierten Menge von Regeln gespielt werden und sich mühelos mit einfachen Formalismen auf einem Computer darstellen lassen. Da Spiele extrem umfangreiche Suchräume generieren können, erfordern sie leistungsfähige Techniken (Heuristiken) zur Ermittlung der im Problemraum zu untersuchenden Alternativen.
Aus diesen Gründen stellen Spiele eine fruchtbare Domäne für das Studium heuristischer Suchverfahren dar. [lug01]

Maschinelles Schließen und Theorembeweise Die Aufgabe von Theorembeweisern liegt in der automatischen Herleitung und Verifikation logischer Formeln. Da die Logik ein formales System ist, bietet sich ihre Automatisierung an. Eine Vielzahl von Problemen läßt sich in Angriff nehmen, indem die Problembeschreibung und relevante Hintergrundinformationen durch logische Axiome beschrieben und die Probleminstanzen als zu beweisende Theoreme behandelt werden. Da jedes einigermaßen komplexe logische System eine unendliche Zahl beweisbarer Theoreme generieren kann, ist eine Suchsteuerung durch effiziente Techniken (Heuristiken) nötig. [lug01]

Expertensysteme Expertensysteme (ES) sind Programme, mit denen das Spezialwissen und die Schlussfolgerungsfähigkeit qualifizierter Fachleute auf eng begrenzten Aufgabengebieten nachgebildet werden soll. Expertensysteme benötigen detaillierte Einzelkenntnisse über das Aufgabengebiet und Strategien, wie dieses Wissen zur Problemlösung benutzt werden soll.
Um ein Expertensystem zu bauen, muß das Wissen formalisiert, im Computer repräsentiert und gemäß einer Problemlösungsstrategie manipuliert werden.
Expertensysteme umfassen ein breites Feld sehr unterschiedlicher Anwendungsbereiche. Dabei wird angenommen, daß die Unterschiede zwischen den einzelnen Bereichen hauptsächlich in den Wissensinhalten liegen, während die jeweiligen Wissensrepräsentationen und Problemlösungsstrategien viele Ähnlichkeiten aufweisen. [pup91]

Während konventionelle Programme aus der Zusammenarbeit von Entwickler und Anwender hervorgeht, sind an der Entwicklung eines Expertensystems drei **Akteure** beteiligt:

1. **Anwender**: Sie sind üblicherweise Laien oder zumindest keine Spezialisten im vorgesehenen Problemgebiet.
2. **Experten**: Ihr Wissen wird in das System eingebracht. Sie besitzen im allgemeinen kein EDV- oder ES-spezifisches Wissen.
3. **Wissensingenieure**: Sie erfassen das Wissen der Experten, stellen es im Rechner dar und entwickeln das Expertensystem. [gfh90]

Weitere Gebiete:

Verstehen natürlicher Sprache und semantische Modellierung;
Modellierung menschlichen Verhaltens;
Planung und Design;
Maschinelles Lernen;
Parallele Verarbeitung und emergentes Rechnen.

Nach [lug01] gibt es dabei wichtige Merkmale, die allen KI-Gebieten gemeinsam zu sein scheinen:

- die Verwendung von Computern für das Schließen, die Mustererkennung, das Lernen oder andere Formen der Inferenz;
- die Konzentration auf Probleme, die sich nicht für algorithmische Lösungen eignen. Darin liegt der Einsatz der heuristischen Suche als KI-Problemlösungstechnik begründet;
- die Beschäftigung mit Problemlösungsverfahren, die mit ungenauen, fehlenden oder schlecht definierten Informationen arbeiten, und der Einsatz von Repräsentationsformalismen, die den Programmierer in die Lage versetzen, diese Unzulänglichkeiten auszugleichen;
- Nachdenken über die wichtigen qualitativen Merkmale einer Situation;
- der Versuch, sowohl auf Fragen der semantischen Bedeutung als auch der syntaktischen Form einzugehen;
- Antworten, die weder exakt noch optimal, aber in gewisser Weise ausreichend sind. Dies ergibt sich aus dem Einsatz heuristischer Problemlösungsmethoden in Situationen, in denen optimal oder exakte Ergebnisse entweder zu kostspielig oder nicht möglich sind;
- der Einsatz großer Mengen bereichsspezifischen Wissens beim Problemlösen. Dies ist die Grundlage von Expertensystemen;
- die Verwendung von übergeordneten Wissen (auf einer Metaebene angesiedeltes Wissen) zur effizienteren Steuerung von Problemlösungsstrategien.

2.5 Künstliche Intelligenz und Informatik

KI lässt sich als Zweig der Informatik definieren, der mit der Automatisierung intelligenten Verhaltens befasst ist. Somit können die theoretischen und angewandten Prinzipien der Informatik zugrunde gelegt werden. Zu diesen Prinzipien gehören die zur Wissensrepräsentation verwendeten Datenstrukturen, die zur Anwendung dieses Wissens erforderlichen Algorithmen sowie die Sprachen und Programmiertechniken, die zur Implementierung eingesetzt werden.
KI war stets damit befasst, die Fähigkeiten der Informatik zu erweitern. Aus einer Reihe von Gründen, wie z.B.: die schiere Größe und Komplexität der KI-Probleme, wurden von KI-Programmierern leistungsfähigere Programmiermethoden entwickelt. Dazu gehören neue Modelle und Techniken zur Wissensstrukturierung (z.B.: objektorientierte Systeme, Produktionssysteme), neue Programmiersprachen (z.B. objektorientiert), auf Expertensystemen basierende Programmgerüste und höhere Programmiersprachen (z.B.: Lisp, Prolog).

Viele dieser Techniken stellen Fortschritte im Bereich der Programmiersprachen und der Softwareentwicklungsumgebungen dar und gehören nun zu den Standardtools der Softwareentwicklung. [lug01]

3 Repräsentation und Suche

Zu Beginn werden die Begriffe Repräsentation und Abstraktion und die Anforderungen an eine Repräsentation und an eine Repräsentationssprache definiert. Anschließend werden die grundlegendenen Symbolsysteme Aussagenlogik und Prädikatenlogik und in diesem Kontext Interpretation, Repräsentation, Implikation, Modus Ponens, Variable und die Eignung als Grundlage für Expertensysteme erörtert. Weiters werden die Wissensrepräsentationen Produktionsregel, semantisches Netz, Frame und in diesem Kontext Implikation, Graph und Objektorientierung bzw. Objektbasierung erläutert. Darauf aufbauend wird die hybride Repräsentation erörtert. Die Zustandsraumrepräsentation stellt die Verbindung zur blinden bzw. erschöpfenden Suche her, welche die daten- und zielorientierte Suche, und die Tiefen- und Breitensuche umfaßt. Dann wird die intelligente heuristische Suche mit der Bestensuche als Beispiel aufgeführt und abschließend auf den Zusammenhang zwischen Heuristiken und Expertensystemen eingegangen.

3.1 Repräsentation

> *Die Repräsentation soll dazu dienen, die wichtigsten Merkmale einer Problemdomäne zu beschreiben und diese Daten einem Problemlösungsverfahren zugänglich zu machen.*
> *Die Abstraktion, also die Repräsentation lediglich derjenigen Informationen, die für einen gegebenen Zweck erforderlich sind, stellt ein unabdingbares Werkzeug für den Umgang mit Komplexität dar. Eine Repräsentation ist nur eine Abstraktion, also ein Symbolmuster, das eine gewünschte Entität bezeichnet und nicht die Entität selbst. Würde die falsche Repräsentation verwendet, wäre die Implementierung sehr mühsam, da diese Repräsentation von ihrer Struktur her für das Problem zu wenig geeignet wäre. [lug01]*

Ein gegebenes Problem kann sich für mehrere alternative Repräsentationen eignen. Eine der wichtigsten Aufgaben der Entwickler von KI-Programmen besteht in der Auswahl der Repräsentation.

Die Repräsentation sollte nach [lug01]:

1. sich dazu eignen, alle erforderlichen Informationen zu beschreiben;
2. ein natürliches Schema für die Darstellung des erforderlichen Wissens bieten;
3. die effiziente Ausführung des resultierenden Codes unterstützen.

Das Anliegen der KI ist eher das qualitative und nicht das quantitative Problemlösen, was eher durch Schließen als durch Berechnen und durch die Strukturierung umfangreichen und vielfältigen Wissens statt der Implementierung eines wohldefinierten Algorithmus geschieht.

Daraus ergeben sich nach [lug01] folgende **Anforderungen an eine KI-Repräsentationssprache:**

Qualitatives Wissen beschreiben In der KI-Programmierung ist es notwendig, die qualitativen Aspekte eines Problems zu beschreiben, **abstrakte Klassen** von Objekten und Situationen zu formulieren, in diesen Begriffen zu denken und anschließend darüber Schlussfolgerungen zu ziehen. Dabei ist die Auswahl der richtigen Repräsentationsform entscheidend, welche die zu beschreibenden Daten und deren Eigenschaften und Beziehungen direkt wiedergeben kann.

Fähigkeit, aus der Beschreibung einer Welt zusätzliches Wissen abzuleiten Es ist nicht notwendig, zu dieser Beschreibung manches Wissen explizit neu hinzuzufügen. Wir definieren stattdessen eine allgemeine **Regel**, die es dem System ermöglicht, aus den gegebenen Fakten auf diese Informationen zu schließen. Die Formulierung allgemeiner Schlussregeln ermöglicht sehr ökonomische Repräsentationen sowie den Entwurf von Systemen, die flexibel und allgemein genug sind, um mit einer Reihe unterschiedlicher Situationen umgehen zu können.

Allgemeine Prinzipien ebenso wie spezielle Situationen repräsentieren Das wird durch Verwendung von **Variablen** ermöglicht. Mit Variablen lassen sich allgemeine Klassen von Objekten oder Eigenschaften der realen Welt angeben und somit allgemeine Aussagen über Klassen bilden.

Auf Grund der Anforderung, qualitatives Wissen zu formalisieren, werden Variablen dynamischer verwendet und implementiert als in traditionellen Programmiersprachen. Die Regeln für Wertzuweisung, Typ und Wertebereich, die wir in konventionellen Sprachen finden, sind für Schlussfolgerungssysteme zu restriktiv. Gute Sprachen zur Wissensrepräsentation handhaben die Bindung von Variablennamen, Objekten und Werten dynamisch.

Komplexe semantische Bedeutungen wiedergeben Viele KI-Problemdomänen erfordern große Mengen strukturiertem zusammenhängendem Wissen. Eine gültige und gute Beschreibung

muß neben einer Auflistung der Bestandteile auch beinhalten, in welcher Weise diese Bestandteile zusammengefügt sind und wie sie zusammenarbeiten.
Das Wissen kann so strukturiert werden, daß es die natürliche Klassen-Instanz-Struktur der Domäne wiederspiegelt. Bestimmte Beziehungen bezeichnen die verschiedenen Abhängigkeiten von Klassen und ermöglichen einen Vererbungsmechanismus, mit dem das Wissen auf der obersten Abstraktionsstufe gespeichert werden kann. Eine solche **strukturierte Darstellung** ist für eine Reihe von Situationen wichtig, zu denen taxonomische Daten, z.B. die Klassifizierung von Pflanzen und die Beschreibung komplexer Objekte wie ein KFZ-Motor gehören.
Man kann semantische Beziehungen auch grafisch darstellen. Diese als **semantisches Netz** bezeichnete Art der Beschreibung, ist eine Grundtechnik zur Repräsentation semantischer Bedeutung. Da die Beziehungen durch Verknüpfungen des Graphen explizit angegeben werden, könnte ein Algorithmus, der Schlüsse über die Domäne zieht, einfach diesen Verknüpfungen folgen, um die relevanten Verbindungen herzustellen. Das ist viel effizienter als das Durchsuchen einer Datenbank, die prädikatenlogische Beschreibungen enthält.

Metawissen darstellen Ein intelligentes System sollte nicht nur bestimmte Dinge wissen, sondern auch wissen, was es weiß. Dieses übergeordnete Wissen, auch Metawissen genannt, stellt Wissen über Struktur, Aufbau und Verwendbarkeit des Wissens dar.

3.2 Symbolsysteme

3.2.1 Die Aussagenlogik

Ein gewisses Quantum an Wissen (Wissenseinheit) kann in Form einer Behauptung bzw. **Aussage** wie z.B. „es regnet am Dienstag“ formuliert werden. Ein Aussagensymbol repräsentiert eine Aussage über die Welt, die in Bezug auf den beschriebenen Zustand der Realität entweder wahr oder falsch ist. Aussagen können miteinander über logische Elementaroperationen (nicht, und, oder) zu Sätzen verknüpft werden.
Die Zuweisung von Wahrheitswerten zu den Sätzen wird als **Interpretation** bezeichnet, d.h. eine Annahme über deren Wahrheit in Bezug auf eine mögliche Welt. Jede mögliche Abbildung eines Wahrheitswertes auf eine Aussage entspricht einem möglichen Interpretationsrahmen. [lug01]

Da in der Aussagenlogik ein einzelnes Symbol für eine komplexe Aussage steht, ist es nicht möglich, auf die Komponenten einer einzelnen Aussage zuzugreifen.

3.2.2 Die Prädikatenlogik

Die Prädikatenlogik bietet diese Möglichkeit durch **Prädikate**, die Eigenschaften oder Beziehungen seiner Argumente bezeichnen.
Argumente sind Symbole, die Objekte der Domäne repräsentieren. Die Repräsentation logischer Sätze beschreibt die wichtigen Eigenschaften und Beziehungen der Problemdomäne. Statt den gesamten Satz „es regnet am Dienstag“ durch ein einzelnes Aussagensymbol zu repräsentieren, kann ein Prädikat namens Wetter eingeführt werden, das eine Beziehung zwischen einem bestimmten Tag und dem Wetter beschreibt: Wetter (Dienstag, Regen).

> *Beim Einsatz der Prädikatenlogik als **Repräsentation** werden Objekte und Beziehungen der Problemdomäne durch eine Menge prädikatenlogischer Ausdrücke beschrieben. Die Datenbank bzw. Wissensbasis mit diesen prädikatenlogischen Ausdrücken, die z.B. jeweils den Wahrheitswert W haben, beschreibt den „Zustand der Welt“. [lug01]*

Prädikatenlogische Ausdrücke erlauben somit die Repräsentation von Wissen. Um jedoch eine Wissensverarbeitung zu erreichen, bedarf es entsprechender Verarbeitungsmechanismen, die aus vorhandenem Wissen neues Wissen schließen.

Schlussregeln bzw. Implikationen (eine höherwertige logische Operation) stellen eine maschinell ausführbare Möglichkeit dar, prädikatenlogische Ausdrücke zu bearbeiten, auf deren einzelne Komponenten zuzugreifen und neue Sätze abzuleiten.
Mittels einer Schlussregel können aus prädikatenlogischen Sätzen neue prädikatenlogische Sätze erzeugt werden, z.B.: P => Q (P impliziert Q): „Wenn es regnet, dann wird der Boden nass“.
Die Prädikatenlogik bietet mit dem Konzept der **logischen Folgerung** eine Grundlage für eine formale Theorie des logischen Schließens. Die genaue Bedeutung von „widerspruchsfrei und vollständig logisch folgen aus“ ist überaus wichtig: Ein prädikatenlogischer Ausdruck X folgt logisch aus einer Menge prädikatenlogischer Ausdrücke S, wenn jede Interpretation und Variablenzuweisung, die S erfüllt, auch X erfüllt , oder einfacher gesagt: Der Ausdruck X ist wahr für jede Interpretation, die S erfüllt.
Für den **Modus Ponens**, die widerspruchsfreie Grundschlussregel, gilt somit: In allen Interpretationen, in denen die Menge prädikatenlogischer Sätze S: P und P => Q wahr sind, ist auch der Ausdruck X: Q wahr. Z.B.: P: „es regnet“ und P => Q: „Wenn es regnet, dann wird der Boden nass“. Wenn in der Interpretation die Aussage P wahr ist, es also tatsächlich regnet und auch die Implikation P => Q wahr ist, dann kann durch Anwendung des Modus Ponens die Aussage Q: „der Boden wird nass“ hinzugefügt werden.

Läßt sich mit einer Schlussregel jeder Ausdruck erzeugen, der logisch aus S folgt, dann gilt sie als vollständig. Die Regel Modus Ponens ist ein Beispiel für eine Schlussregel, die widerspruchsfrei und bei Einsatz geeigneter Strategien vollständig ist.

> *Der Hauptvorteil des Einsatzes allgemeiner Methoden wie des Modus Ponens zur Erzeugung von Zuständen besteht darin, daß mit dem resultierenden Algorithmus jeder beliebige Raum logischer Schlussfolgerungen durchsucht werden kann. Die Besonderheiten eines Problems werden mit Hilfe von prädikatenlogischen Annahmen beschrieben. Folglich ist ein Mittel gegeben, das Problemlösungswissen von seiner Steuerung und Implementation auf dem Computer zu trennen. [lug01]*

Zudem sind Ausdrücke mit **Variablen** zulässig. Bevor der Wahrheitsgehalt eines Ausdrucks mit Variablen ermittelt werden kann, müssen die Variablen selbst belegt werden. Solange die Variablen ungebunden sind, ist der Ausdruck weder wahr noch falsch. Jeder in der Interpretation zulässige Wert kann an Stelle der Variablen in den Ausdruck eingesetzt werden.
Es gibt zwei Möglichkeiten, Variable zu binden, d.h. Bedingungen an die Verwendung der Variablen zu stellen: Allquantor und Existenzquantor. Der **Allquantor** stellt eine Behauptung über alle zulässigen Werte der Variablen in dem Ausdruck dar. Gibt es auch nur einen zulässigen Wert, für den der Ausdruck nicht gilt, so gilt der gesamte Ausdruck nicht. Allquantifizierung wirft Probleme bei der Berechnung des Wahrheitswertes eines Ausdrucks auf, da zu dessen Bestimmung alle möglichen Werte der Variable getestet werden müssen. Wenn die Interpretationsdomäne unendlich groß ist, bezeichnet man die Prädikatenlogik als unentscheidbar. Der **Existenzquantor** stellt eine Behauptung über einen zulässigen Wert der Variable dar. Es genügt also, nur einen Wert für die Variable zu finden, für den der Ausdruck wahr ist, um die Richtigkeit des gesamten Ausdrucks zu zeigen. Trotzdem kann die Ermittlung des Wahrheitswertes eines Ausdrucks, der existenzquantifizierende Variable enthält, ebenso schwierig sein wie mit Allvariablen.

Beispiel zu Modus Ponens, Allvariable und Interpretation (siehe Abb.3.1 nach [lug01]):
S ist eine Menge prädikatenlogischer Sätze, I ist eine Interpretation und X ist ein Ausdruck, der logisch aus S folgt und von I gleichermaßen erfüllt wird.

Da die Aussagenlogik nicht mit Variablen arbeitet, haben Sätze lediglich eine endliche Anzahl von Wahrheitswerten und somit können alle möglichen Zuweisungen erschöpfend getestet werden.
Mithilfe von Wahrheitstafeln lässt sich die Gültigkeit von Ausdrücken überprüfen, die keine Variablen enthalten. Bei Ausdrücken mit Variablen ist die Entscheidung der Gültigkeit nicht immer möglich. [lug01]

In der Prädikatenlogik erster Ordnung können All- und Existenzaussagen nur über Objekte getroffen werden,während in der Prädikatenlogik zweiter Ordnung solche Aussagen auch über Mengen

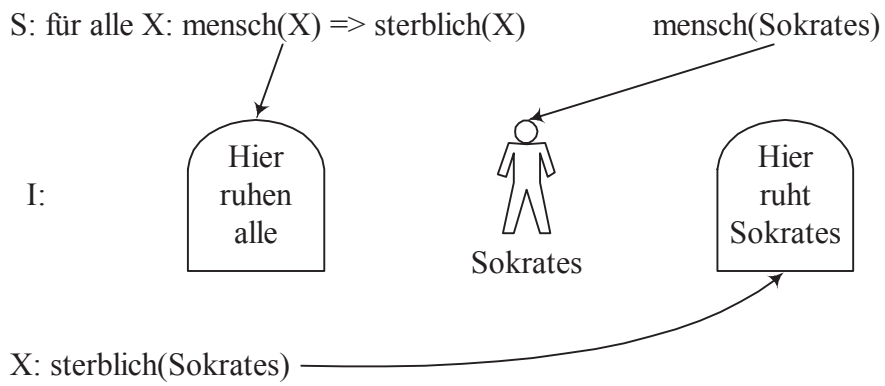


Abbildung 3.1: Beispiel zur Prädikatenlogik.

bzw. Eigenschaften von Objekten zulässig sind. Die relativ ausdrucksstarke Prädikatenlogik zweiter Ordnung ist weder vollständig noch entscheidbar, die weniger ausdrucksstarke Prädikatenlogik erster Ordnung ist zwar auch nicht entscheidbar, aber immerhin vollständig, und die einfache Aussagenlogik ist sowohl vollständig als auch entscheidbar.
Für Wissensrepräsentationen in Expertensystemen hat die Prädikatenlogik erster Ordnung gewichtige Nachteile hinsichtlich Mächtigkeit, Adäquatheit und Effizienz. In vielen Anwendungsbereichen sind das Wissen bzw. die Daten unsicher, unvollständig oder zeitabhängig, was in der Prädikatenlogik erster Ordnung schlecht oder gar nicht dargestellt werden kann. Für große Wissensbasen benötigt man adäquate, im Prädikatenkalkül erster Ordnung nicht vorhandene Strukturierungsmittel wie Hierarchien und Kontexte. [pup91]
Da die Prädikatenlogik Schlussregeln und die logische Folgerung direkt abbildet, eignet sie sich als Grundlage für Expertensysteme. Jedoch ist ihre praktische Bedeutung für Expertensysteme aufgrund der Nachteile hinsichtlich ihrer Darstellungsmöglichkeiten gering.
Deshalb werden meist stark reduzierte Teilsysteme verwendet, die allerdings oft um Ausdrucksmittel erweitert sind, wie z.B. um die Möglichkeit, eine Implikation mit einem Wahrscheinlichkeitswert zu versehen. [gfh90]

3.3 Wissensrepräsentationen

Formalismen, mit deren Hilfe Expertenwissen abgebildet werden können.

Ausgangspunkt bildet die historische Perspektive, der zufolge eine Wissensbasis als eine Abbildung zwischen Objekten und Relationen in einem Problembereich und den mathematischen Objekten und Relationen in einem Programm beschrieben werden kann. Das Ergebnis von Schlussfolgerungen in einer Wissensbasis sollte dem entsprechenden Ergebnis der Aktionen bzw. Beobach-

tungen in der Welt entsprechen.
Die Wissensrepräsentationssprache vermittelt die mathematischen Objekte, Relationen und Inferenzen, die den Programmierern zur Verfügung stehen. [lug01]

3.3.1 Produktionsregel

> *Die Produktionsregel (Implikation) ist eine Untermenge des Prädikatenkalküls zur einfachen Darstellung prozeduralen Wissens. Sie stellt eine maschinell ausführbare Möglichkeit dar, aus Voraussetzungen oder Behauptungen neue Fakten zu produzieren.*

Sie hat das Format „Bedingung => Aktion“ und wird in einem Expertensystem als Wenn-Dann-Regel repräsentiert, wobei die Prämisse der Regel, also der Wenn-Abschnitt, der Bedingung und die Konklusion, also der Dann-Abschnitt, der Aktion entspricht.

Die Regel verbindet eine oder mehrere Bedingungen mit ein oder mehreren Aktionen, und wenn die Bedingungen erfüllt sind, dann werden die verknüpften Aktionen ausgeführt. Eine Aktion kann die Behauptung eines neuen Faktes oder eine Prozedur zum Ausführen sein.

Implikation: Mit ihr können Kausalbeziehungen in Wenn-Dann-Form dargestellt werden. Die Gesamtaussage: „Wenn A, dann B“ bedeutet folgendes. Falls die Aussage A wahr ist, folgt hieraus, daß auch die Aussage B wahr ist. Aus einer gegebenen Information wird neue Information gefolgert. Weiters wurde in der Logik eindeutigkeitshalber festgelegt, die Gesamtaussage der Wenn-Dann-Verknüpfung als wahr zu betrachten, wenn die Eingangsaussage A falsch ist, damit keine fehlerhaften Schlüsse gezogen werden. Die Gesamtaussage: „Wenn A, dann B“ ist genau dann falsch, falls die Aussage A wahr und die Aussage B falsch ist, sonst ist die Gesamtaussage stets wahr. [hale90]

Wenn Wissen als Beziehung von Objekten zueinander oder komplizierte Objektklassifizierungen mit Abhängigkeiten zwischen einzelnen Klassen und Objektattributen vorliegt, dann ist die Repräsentationsform Produktionsregel zu flach.
Komplexe Objekte, die zudem in einem komplizierten Zusammenhang stehen, benötigen andere Repräsentationsmechanismen, die sowohl den inneren Aufbau eines Objektes als auch die Abhängigkeiten mehrerer Objekte untereinander beschreiben können. [hale90]

3.3.2 Semantisches Netz

Assoziationistische Theorien legen die Bedeutung von Objekten über ein Netzwerk von Assoziationen mit anderen Objekten in einer Wissensbasis fest um z.B.: komplexe, zusammengefaßte

Gegenstände objektorientiert zu beschreiben. Diese Theorie eignet sich hervorragend, Beziehungen zwischen einzelnen Objekten zu beschreiben.
Das Wissen kann über einfache Assoziationen hinaus hierarchisch strukturiert werden, wobei Informationen jeweils auf der höchstmöglichen taxonomischen Ebene gespeichert werden. Diese Strukturierung von Wissen wird in **Vererbungsstrukturen** formalisiert. Vererbungssysteme ermöglichen das Speichern von Informationen auf der Ebene der größtmöglichen Abstraktion. Auf diese Weise wird die Größe der Wissensbasis verringert und das Risiko von Inkonsistenzen bei der Aktualisierung von Einträgen minimiert. [lug01]
Die Strukturierung wird durch verschiedene Beziehungen bzw. Beziehungstypen erreicht, die wegen ihrer unterschiedlichen Bedeutung benannt werden müssen.
Es erfolgt eine Zusammenfassung von Objekten zu **Klassen**, die weiters hierarchisch geordnet sind. Diese Abhängigkeiten der Klassen zueinander werden durch Beziehungsoperatoren (Prädikat ist-eine-Art-von) und andere Zusammenhänge zwischen Objekten durch Beziehungsoperatoren (hat, benötigt) dargestellt. Die konkrete Ausprägung eines Objekts, die Instanz, kann durch die Relation ist-ein miteinbezogen werden. [hale90]

Graphen haben sich als ideales Werkzeug zur Formalisierung assoziationistischer Wissenstheorien herausgestellt, da mit ihnen Assoziationen und Relationen explizit dargestellt werden können. Diese grafische Darstellung wird als semantisches Netz bezeichnet.
Ein Graph besteht aus einer Menge von Knoten und einer Menge von Kanten. Der Graph ist gerichtet, wenn den Kanten eine Richtung zugeordnet ist z.B. durch einen Pfeil. Ein Weg durch einen Graphen verbindet eine Folge von Knoten über aufeinander folgende Kanten. [lug01]
Die **Knoten** eines solchen Graphen sind Konzepte bzw. Fakten und repräsentieren Objekte der realen Welt, Abstraktionen, Eigenschaften, Ereignisse und Zustände. Die **Kanten** des Graphen stellen konzeptuelle Relationen bzw. Assoziationen zwischen den Knoten dar. Sowohl Knoten als auch Kanten sind im Allgemeinen mit Beschriftungen versehen.
Da semantische Netze die Eigenschaften eines Konzepts im direkten Zusammenhang mit diesem Konzept darstellen, kann man sie als Variante der objektorientierten Darstellung ansehen. Historisch gesehen waren die Semantischen Netze der ältere Formalismus, aus dem sich im Lauf der Zeit die als „objektorientiert“ bezeichneten Charakteristika (z.B. Vererbungsmechanismen) herauskristallisierten. Der Unterschied zwischen Semantischen Netzen und den „eigentlichen“ objektorientierten Methoden liegt darin, daß letztere die Unterordnung von Eigenschaften unter das Objekt betonen (bei Frames z.B. stellt ein Attribut kein eigenständiges Objekt dar), während bei semantischen Netzen das Bestreben überwiegt, die Eigenschaften eines Konzepts unmittelbar durch andere, gleichwertige Konzepte darzustellen. [gfh90]

Semantische Netze können als Sammlung von Prädikaten interpretiert werden, somit kann die Prädikatenlogik zur rechnerischen Darstellung von semantischen Netzen verwendet werden. [hale90]

> *Die **Graphennotation** hat per se nur geringe Vorteile gegenüber der Notation in der Prädikatenlogik. Es ist nur eine andere Notationsform für Relationen zwischen Objekten. Die Mächtigkeit der Netzwerkrepräsentationen liegt in der Definition der Verbindungen und der durch diese Strukturen ermöglichten Schlussfolgerungsregeln, beispielsweise der Vererbung. [lug01]*

Die **Vorteile** der Graphennotation gegenüber einer rein logischen liegen in der Transparenz und Natürlichkeit der Repräsentation. Weiters wird eine effiziente und mächtige Verarbeitung ermöglicht, da die Stärken der Graphentheorie für Schlüsse über z.B. die strukturelle Organisation einer Wissensbasis genutzt werden können.

Darüberhinaus wäre eine übergeordnete Struktur wünschenswert, die es sowohl dem Programm als auch dem Programmierer erleichtern, mit komplexen Konzepten auf zusammenhängende Weise umzugehen.

3.3.3 Frames

Ein **Frame** (Rahmen) kann als statische Datenstruktur angesehen werden, mit deren Hilfe wohlverstandene stereotype Situationen dargestellt werden können. Es verfügt über hervorragende strukturelle Beschreibungsmöglichkeiten zur Darstellung des gesamten Wissens der Objekte.
Objekte beinhalten deklaratives (beschreibt Objekt) und prozedurales (beschreibt Verhalten) Wissen. Dieses Wissen umfaßt Aussagen über Struktur, Eigenschaften, Beziehungen, Zustände und Funktionen. Frames sind somit eine Mischung aus deklarativer und prozeduraler Wissensrepräsentation.

Eine Menge gleichartiger Objekte wird durch ihren Typ beschrieben. Dieser Typ beschreibt ein Konzept der realen Welt. Solch ein Frame beschreibt eine stereotype Situation mit ihren möglichen Eigenschaften, und gibt damit einen Rahmen für die genaue Festlegung dieser Eigenschaften vor. Ein solcher **generischer Frame** stellt also einen Prototyp für alle möglichen realen Einzelsituationen dieser Art dar. Tritt eine solche spezifische Situation nun tatsächlich auf, wird sie durch einen sogenannten **individuellen Frame** beschrieben. Dieser Frame stellt eine Instanz des Prototyps dar, d.h. seine Struktur ist der des generischen Frames gleich, er enthält aber die dem jeweils aktuell darzustellenden Sachverhalt entsprechenden individuellen Werte der im Prototyp festgelegten Eigenschaften. Wie bereits erwähnt, entspricht ein solcher Prototyp also einem Konzept der realen Welt, seine Instanzen entsprechen Objekten oder Sachverhalten der realen Welt. [gfh90]
Die generischen Frames der höheren Spezifikationsebene sind sogenannte **Klassen** und Unterklassen, die individuellen Frames der untersten Ebene sind **Instanzen** der Klassen. Instanzen werden dadurch definiert, daß jede Eigenschaft konkret mit Werten belegt wird.

Ein Frame besteht im Wesentlichen aus einem Namen und den Attributen, die bestimmte Facetten haben. Die Attribute, welche die Eigenschaften eines Objekts angeben, können unterschiedliche Facetten annehmen: Datentypdefinitionen, Wertebereiche, Standardwerte, Texte, Prozeduren, Methoden, Relationen usw..

Frames erweitern auf verschiedene Weise semantische Netzwerke und können ebenfalls durch Prädikate dargestellt werden. In einem semantischen Netzwerk werden alle Konzepte durch Knoten und Verbindungen auf derselben Spezifikationsebene repräsentiert.

> *Frames wurden zu dem **Zweck** entworfen, die impliziten Verknüpfungen zwischen Informationen in einer Domäne explizit durch Datenstrukturen darzustellen. Sie stellen für die Organisation ein geeignetes Mittel bereit, indem Dinge als strukturierte Objekte mit benannten Merkmalen und zugehörigen Werten repräsentiert werden. Ein Frame läßt sich dabei als **einzelne, komplexe Einheit** ansehen. [lug01]*

Vorteile dieser objektbasierten bzw. objektorientierten Repräsentation der Frames:

Frames erweitern die Mächtigkeit von semantischen Netzen, indem sie es ermöglichen, komplexe Objekte als ein einziges Frame zu repräsentieren und nicht als große Netzwerkstruktur. Auf diese Weise wird ein sehr natürliches Verfahren zur Repräsentation von stereotypischen Entitäten, Klassen, Vererbung und Standardwerten bereitgestellt. [lug01]
Weiters erleichtern sie die hierarchische Strukturierung von Wissen durch Klassen- und Vererbungsmechanismen wesentlich. Hier können neben einzelnen Relationen auch beliebige Attribute vererbt werden. Dadurch entsteht ein mächtiges Vererbungskonzept. Auch Instanzen können Attribute von ihren Klassen erben. Klassenbildung und daraus abgeleitete Aggregationen, Instanzierung, Definition von Abhängigkeiten und Vererbungsmechanismen stellen eine hohe Funktionalität dar. [hale90]
Eine zusätzliche Erweiterung ist die Objektkommunikation. Um Anwendungen zu realisieren, bei denen der Zustand eines Objekts den Zustand eines anderen Objekts beeinflußt, müssen die Objekte zur gegenseitigen Kommunikation befähigt werden. Objektbasierte Systeme übernehmen dafür Techniken wie Methoden und message passing der objektorientierten Programmierung zum dynamischen Austausch von Informationen. Bei rein objektorientierten Expertensystemen werden nacheinander die Methoden der Objekte ausgeführt. Diese können ihrerseits entweder Methoden anderer Objekte aufrufen oder Nachrichten an sie schicken.

> *Von einem höheren Standpunkt kann man die **regel- und die objektbasierte Wissensrepräsentation** als **zwei Sichtweisen desselben** zugrundeliegenden Netzwerks von Objekten und Beziehungen auffassen, bei dem einmal die Kanten und einmal die Knoten als das „Wichtigere“ angesehen werden. [pup91]*

3.3.4 Hybride Repräsentation

Oft sind bei einem Problem mindestens zwei Arten von Wissen zu repräsentieren:

- Daten bzw. Objekte, aus denen z.B. eine Interpretation abgeleitet wird, und
- Regeln, die diese Daten miteinander verknüpfen.

Meist ist es sinnvoll, einen Teil des Wissens objektorientiert, einen anderen regelbasiert darzustellen. Dabei werden mehrere verschiedene Repräsentationen nebeneinander genutzt, z.B. Regeln neben Objekten. Solche wissensbasierte Systeme werden als **hybrid** bezeichnet. Bei regelbasierten Systemen liegt das Wissen in Form von Regeln, bei Objektbasierten dagegen in Form strukturierter Objekte vor.
Bei einem solchen hybriden System interagieren die Regeln mit den in den Frames enthaltenen Informationen. Dies ermöglicht, Schlüsse über den kompletten Frame-Bereich durch Durchsuchen und Bearbeiten aller Frames zu ziehen. Diese **musterorientierten Regeln** können Variable enthalten, welche zum Vergleichen spezieller Eigenschaften mehrerer Instanzen einer Klasse genutzt werden können. Das ermöglicht mächtige und flexible Mustervergleiche und somit das Schreiben von sehr allgemeinen Regeln als Problemlösungsschritte. Das zu lösende Problem ist somit die Auswahl von Objekten durch Muster- und Größenvergleich ihrer Attribute.

Bei **regelbasierten Expertensystemen** geht es beim Entwurf darum, wie die Regeln zu organisieren sind und wie der Problemlösungsansatz zu strukturieren ist. Da hier mit beziehungslosen Fakten gearbeitet wird, muß bei komplexeren Sachverhalten die ganze Wissensbasis durchsucht werden.
Bei **objektbasierten Expertensystemen** ist die grundlegende Entwurfsüberlegung, welche Objekte benötigt werden und wie diese in Beziehung stehen um das Problem zu lösen. Hier wird das gesamte Problem als aus Objekten bestehend betrachtet und bietet im Vergleich zu regelbasierten Systemen weitere Möglichkeiten wie Klassenbildung, Vererbung, Methoden und musterorientierte Regeln. Da die in Beziehung stehenden Fakten gruppiert in einer einzelnen Frame-Struktur repräsentiert werden, kann so der gesamte komplexe Sachverhalt durch Inspektion dieses einen Frames erhalten werden.
Indem ein objektbasiertes System Regeln einsetzt, können die Vorteile beider Repräsentationen genutzt werden. Durch den Einsatz von musterorientierten Regeln und Variablen kann eine einzige Regel zum Bearbeiten aller Instanzen einer Klasse geschrieben werden. Ohne Variablen müßte für jede Instanz eine eigene Regel geschrieben werden. [dur94]

Gegenüberstellung von Regel und Frame:

Vorteile von Regeln: Eine Regel ist ein unabhängiges Stück Wissen und kann neue Informationen ableiten. Wenn Regeln Variable enthalten, stellen sie einen mächtigen Ansatz für generelles Problemlösen dar.

Nachteile von Regeln: Sie sind ineffizient für die Darstellung großteils prozeduralen Wissens. Manche Anwendungen benötigen kleine Prozeduren. Beim regelbasierten Ansatz müssen für jede Prozedur kleine externe Programme geschrieben werden, die dann von der Regel aufgerufen werden, was zur Systemkomplexitätserhöhung führt.
Sie sind ungeeignet für die Darstellung von Objekten und Objektinteraktionen z.B.: Anwendungen wie Simulationen. Die Änderung eines Objekts startet eine Welle von Änderungen durch andere in Beziehung stehende Objekte. Das Lösen solcher Aufgaben durch Regeln ist schwer, und erhöht die Komplexität des Systems. [dur94]

Vorteile von Frames: Frames mit Methoden und message passing sind eine natürliche Repräsentation von Objekten, deren Verhaltensweisen und Interaktionen mit anderen Objekten.
Durch die Verkapselung der Methoden innerhalb des Frames werden auch die dynamischen Details verborgen, anstatt sich als Serie von Regeln auszubreiten.
Weiters bieten Methoden eine mächtige und einfache Art zur Repräsentation von Prozeduren.

Nachteile von Frames: Das System kann sehr unübersichtlich werden.
Sie sind nicht für die Repräsentation von Heuristiken geeignet. Diese Heuristiken müßten dann angepaßt werden, wodurch die Wahrheit leiden könnte aber die Komplexität auf jeden Fall ansteigt. [dur94]

3.4 Zustandsraum

> *In der Zustandsraumrepräsentation eines Problems entsprechen die Knoten eines Graphen den partiellen Problemlösungszuständen, die Kanten dagegen den Schritten eines Problemlösungsprozesses. Die Knoten sind diskrete Zustände eines Problemlösungsprozesses, wie z.B.: die Ergebnisse logischer Schlussfolgerungen. Die Kanten sind Zustandsübergänge, die logischen Schlussregeln entsprechen. [lug01]*

Durch Darstellung eines Problems als Zustandsraumgraph mittels Graphentheorie läßt sich die tiefere Struktur eines Problems modellieren. Weiters ist die Graphentheorie das geeignete Werkzeug, um Schlüsse aus der Struktur von Objekten, Eigenschaften und Beziehungen zu ziehen. Die

Formalisierung der Graphentheorie ermöglicht zudem das Konzept der Zustandsraumsuche. Ein Problem wird gelöst, indem der Teilgraph des Zustandsraumgraphen für einen Lösungsweg gesucht wird. [lug01]

3.4.1 Repräsentation logischer Ausdrücke

Logische Ausdrücke bieten die Möglichkeit, Objekte und Beziehungen in einer Problemdomäne zu beschreiben, und Schlüsse wie der Modus Ponens erlauben es, logisch von diesen Beschreibungen auf neues Wissen zu schließen. Schlussregeln dienen dazu, die Kanten zwischen den Zuständen zu erstellen und zu beschreiben und definieren einen Raum, der nach einer Problemlösung durchsucht wird.

Es folgt hier ein Beispiel aus der Aussagenlogik, wie eine Menge logischer Beziehungen als Definition eines Graphen angesehen werden kann. Seien q, p, r usw.. Aussagen: q => p; r => p; v => q; s => r; t => r; s => u; s; t (siehe Abb.3.2 links). Aus dieser Menge von Aussagen und der Schlussregel Modus Ponens kann auf bestimmte Aussagen (p,r und u) geschlossen werden. Die Beziehungen zwischen den ursprünglichen Aussagen und diesen Folgerungen drückt der gerichtete Graph aus. Die Kanten entsprechen den logischen Implikationen (=>). Aussagen, die wahr sind (s und t), entsprechen den gegebenen Daten des Problems. Aussagen, die logisch aus dieser Aussagenmenge folgen, entsprechen den Knoten, die entlang eines direkten Wegs von einem Knoten folgen, der eine wahre Aussage repräsentiert. Ein solcher Weg entspricht einer Reihe von Anwendungen des Modus Ponens. Beispielsweise entspricht der Weg [s, r, p] der nachstehenden Folge von Schlüssen: s und s => r ergibt r, r und r => p ergibt p. [lug01]

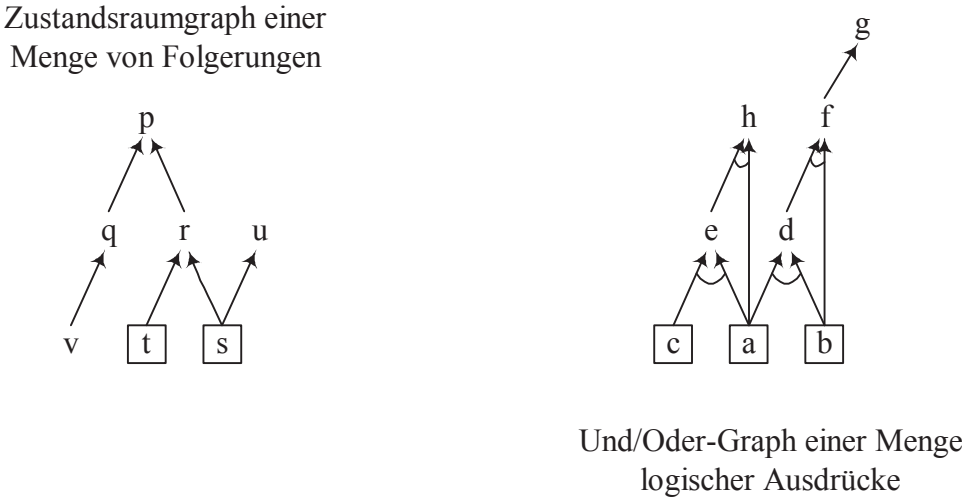


Abbildung 3.2: Beispiel zu Zustandsgraph und Und/Oder-Graph.

Um die logischen Beziehungen auszudrücken, die durch die Operatoren Und und Oder definiert werden, ist eine Erweiterung des grundlegenden Graphenmodells notwendig, der so genannte **Und/Oder-Graph**.
Sind die Prämissen einer Implikation durch den Operator Und miteinander verknüpft, werden die von diesem Knoten ausgehenden Kanten mit einem Bogen miteinander verbunden. Der Bogen drückt aus, das alle Prämissen der Implikation wahr sein müssen. Implikationen, deren Prämissen durch den Operator Oder miteinander verknüpft sind, können als getrennte Implikationen geschrieben werden.

Das nächste Beispiel aus der Aussagenlogik generiert einen Und/Oder-Graph. Ausgangssituation sind folgende wahre Aussagen: a; b; c; a and b => d; a and c => e; b and d => f; f => g; a and e => h (siehe Abb.3.2).

Die Suche im Und/Oder-Graphen erfordert etwas umfangreichere Aufzeichnungen als bei gewöhnlichen Graphen. Ist ein Oder-Zweig nicht brauchbar, wird der nächste Oder-Zweig getestet. Beim Durchsuchen der Und-Knoten müssen alle Und-Nachfolger eines Knotens als wahr bewiesen werden, damit sich auch der übergeordnete Knoten als wahr erweist.
Der Und-Knoten entspricht einer Problemzerlegung, wobei das Problem in Teilprobleme aufgeteilt ist und alle Teilprobleme gelöst werden müssen, um das ursprüngliche Problem zu lösen. Der Oder-Knoten entspricht einer Auswahl, einen Punkt in der Problemlösung, an dem zwischen alternativen Problemlösungswegen oder -strategien gewählt werden muß, wobei jede Alternative, sofern sie erfolgreich ist, ausreicht, das Problem zu lösen. [lug01]

3.5 Zustandsraumsuche

> *Die Zustandsraumsuche charakterisiert das Problemlösungsverfahren als den Prozeß, einen Lösungsweg vom Startzustand bis zu einem Ziel zu suchen. Ziel kann ein Zustand aber auch eine Eigenschaft des Lösungswegs selbst sein. Es ist die Aufgabe eines* Suchalgorithmus*, einen Lösungsweg durch einen solchen Problemraum zu finden. [lug01]*

Die Zustandsraumsuche ist die Suche nach Lösungen für eine konkrete Aufgabe, unter Zuhilfenahme des zur Verfügung stehenden Wissens. Ausgehend von einer Problemstellung wird Wissen kombiniert und angewandt, um eine Lösung zu erreichen. Beispiele sind: Ziehen von Schlussfolgerungen aus Fakten, Diagnose von Ursachen ausgehend von Beobachtungen und Analyse von Daten und Problemlösungsstrategien für eine bestimmte Aufgabe.
Beispiel: Das Problem sei, einen mechanischen Fehler an einem Kraftfahrzeug (KFZ) zu diagnostizieren. Die einzelnen Knoten repräsentieren einen temporären Wissenszustand über die mecha-

nischen Probleme des KFZ. Der Prozeß des Überprüfens der Fehlersymthome und des Schließens auf die Fehlerursache kann als Wegsuche durch Zustände zunehmenden Wissens betrachtet werden. Kanten, die den grundlegenden Diagnosetests entsprechen, führen zu Zuständen, die im Diagnosetest weiter angesammeltes Wissen repräsentieren. Der Weg legt fest, welche Diagnosetests durchgeführt werden, wobei mit jeder Entscheidung bestimmte Tests aus der weiteren Betrachtung ausgeschlossen werden. [lug01]

Der verwandte Begriff **Inferenz** wird entweder gleichbedeutend verwendet oder wird benutzt, um die spezielle Tätigkeit des logischen Schlussfolgerns zu bezeichnen. Die Grundlage der Zustandsraumsuche bildet die formale Logik, auch wenn das Wissen selbst nicht notwendigerweise in der Sprache der formalen Logik dargestellt wird. [gfh90]
Die Prädikatenlogik kann als formale Spezifikationssprache, bei der die Problemstruktur aufgrund der impliziten Darstellung eventuell unübersichtlicher wäre, aber auch zur Abbildung der Knoten eines Graphen auf den Zustandsraum eingesetzt werden. Mit dieser Repräsentation wird die Bestimmung, ob ein bestimmter Ausdruck aus einer gegebenen Menge von Annahmen logisch folgt, zu dem Problem, einen Weg zu finden. Die Aufgabe kann daher als Graphensuchproblem formuliert werden und mit Hilfe einer Suche gelöst werden. [lug01]

Eine Zustandsraumsuche,welche den gesamten Zustandsraum durchsucht, wird als **erschöpfende oder blinde Suche** bezeichnet. Obwohl diese erschöpfende Suche als Lösungsansatz für jedes Problem eingesetzt werden kann, ist sie aufgrund der schieren Größe realer Probleme zu ineffektiv und aufwendig und somit praktisch unmöglich. Diese Art der Problemlösung ohne jegliche Intelligenz spiegelt an sich nicht das Wesen intelligenten Verhaltens wider.

3.5.1 Strategien der Zustandsraumsuche

Strategien geben an, wie das Wissen zur Lösung von konkreten Problemen kombiniert und angewandt wird. Ein Zustandsraum kann in zwei Richtungen durchsucht werden: von den gegebenen Daten einer Probleminstanz zu einem Ziel hin oder vom Ziel zurück zu den Daten.

3.5.1.1 Daten- und zielorientierte Suche

Die **datenorientierte Suche**, auch **Vorwärtsverkettung** (forward chaining) genannt, geht von einem Anfangszustand aus und arbeitet sich zu einem Endzustand vor. Hierbei beginnt das Problemlösungsverfahren mit den gegebenen Fakten des Problems sowie einer Menge gültiger Regeln für Zustandsänderungen. Die Suche wird fortgesetzt, indem Regeln auf Fakten angewandt werden, um neue Fakten zu erhalten, die wiederum von den Regeln verwendet werden, um weitere Fakten zu generieren. Dieser Prozeß wird solange fortgesetzt, bis (hoffentlich!) ein Weg gefunden worden

ist, welcher der Zielbedingung genügt.
Die datenorientierte Suche verwendet das Wissen und die Beschränkungen aus den gegebenen Daten eines Problems, um die Suche auf Wege zu führen, die als Wahr bekannt sind.

Bei der **zielorientierten Suche**, auch **Rückwärtsverkettung** (backward chaining) genannt, wird von einem Endzustand (Ziel, Hypothese) ausgegangen und ein dazu passender Anfangszustand gesucht, bis ein Anfangszustand erreicht worden ist. Hierbei wird von einem bekannten oder vermuteten Ziel ausgegangen. Man sucht nach Regeln, die dieses Ziel generieren, und bestimmt die Bedingungen, die vorliegen müssen, damit die Regeln eingesetzt werden können. Diese Bedingungen werden zu neuen Zielen oder Teilzielen der Suche. Die Suche wird rückwärtsgerichtet anhand aufeinander folgernder Teilziele fortgesetzt, bis (hoffentlich!) die gegebenen oder vom Benutzer erfragten Fakten des Problems erreicht worden sind.
Die zielorientierte Suche verwendet somit Wissen über das gewünschte Ziel, um die Suche durch relevante Regeln zu leiten und Zweige des Raums auszuschließen. [lug01]

In Abb.3.2 (siehe Seite 31) beim Und/Oder-Graph versucht eine zielorientierte Strategie zur Bestimmung, ob h wahr ist, zuerst zu beweisen, daß sowohl a als auch e wahr sind. Daß a wahr ist, folgt sofort, doch für e müssen sowohl c als auch a wahr sein. Hat das Problemlösungsverfahren alle diese Kanten nach unten zu wahren Aussagen hin verfolgt, werden die Wahr-Werte in den Und-Knoten kombiniert, um zu verifizieren, daß h wahr ist.
Eine datenorientierte Strategie zur Bestimmung der Wahrheit von h beginnt dagegen mit den bekannten Tatsachen (c, a und b) und fügt gemäß der Beschränkungen der Und/Oder-Graphen neue Aussagen zu dieser Menge hinzu; e oder d könnten die ersten Aussagen sein, die in diese Faktenmenge aufgenommen werden. Diese zusätzlichen Aussagen ermöglichen es, auf neue Fakten zu schließen. Der Prozeß wird fortgesetzt, bis das gewünschte Ziel h bewiesen ist. [lug01]

Sowohl daten- als auch zielorientierte Problemlösungsverfahren durchsuchen den gleichen Zustandsraumgraphen; allerdings können sich die Reihenfolge und die Anzahl der durchsuchten Zustände unterscheiden.
Die bevorzugte Strategie wird von den Eigenschaften des Problems selbst bestimmt. Dazu gehören die Komplexität der Regeln, die Form des Zustandsraums sowie die Art und die Verfügbarkeit der Problemdaten. [lug01]

Die **datenorientierte Suche eignet sich** für Probleme mit folgenden Bedingungen:

- Alle oder zumindest die meisten Daten liegen in der ersten Problembeschreibung vor. Aus den vorhandenen Daten sollen neue Zustände bzw. Lösungsvorschläge generiert werden. Interpretationsprobleme passen häufig zu diesem Ansatz, da sie eine Datensammlung bereitstellen und vom System eine Interpretation auf höherer Ebene fordern;

- Es liegt eine hohe Anzahl potenzieller Ziele vor, doch es gibt nur wenige Möglichkeiten, die Fakten und gegebenen Informationen einer bestimmten Probleminstanz zu verwenden;
- Die Formulierung eines Ziels oder einer Hypothese ist schwierig.

Die **zielorientierte Suche eignet sich** unter folgenden Bedingungen:

- Ein Ziel oder eine Hypothese ist in der Problembeschreibung definiert oder kann auf einfache Weise formuliert werden. Viele Diagnosesysteme betrachten potenzielle Diagnosen in einer systematischen Weise und unter Verwendung von zielorientierten Verfahren bestätigen sie oder schließen sie aus;
- Es gibt sehr viele Regeln, die auf die Fakten des Problems anwendbar sind und daher eine steigende Anzahl von Folgerungen oder Zielen produzieren. Die frühzeitige Wahl eines Ziels kann die meisten dieser Zweige ausschließen und die zielorientierte Suche effektiver gestalten, da der Raum von Überflüssigem befreit wird;
- Die Daten des Problems stehen nicht zur Verfügung, sondern müssen vom Problemlösungsverfahren akquiriert werden. In diesem Fall kann die zielorientierte Suche die Datenbeschaffung erleichtern. [lug01]

Die sorgfältige Analyse eines bestimmten Problems ist durch nichts zu ersetzen. Zu berücksichtigen sind Aspekte wie Verzweigungsfaktor und Zustandsgenerierung beim Anwenden der Regeln in beiden Richtungen, die Verfügbarkeit der Daten und die Einfachheit der Festlegung potenzieller Ziele. [lug01]

Details zur Implementierung

Bei der Lösung eines Problems mit der ziel- oder datenorientierten Suche, muß ein Weg zwischen einem Startzustand und einem Ziel durch den Zustandsraumgraphen gefunden werden. Da es kein Orakel gibt, müssen Verfahren verschiedene Wege durch den Raum verfolgen, bis ein Ziel gefunden ist. **Backtracking** ist eine Technik zum systematischen Test aller Wege in einem Zustandsraum. Die Backtracking-Suche beginnt beim Startzustand und folgt einem Weg, bis entweder ein Ziel oder eine Sackgasse erreicht ist. Handelt es sich um ein Ziel, ist die Suche beendet und der Lösungsweg wird zurückgegeben. Handelt es sich dagegen um eine Sackgasse, wird ein Backtracking zum letzten Knoten auf dem Weg durchführt, der noch nicht geprüfte Geschwister enthält. Die Suche wird dann in diesen Zweigen fortgesetzt. [lug01]

3.5.1.2 Tiefen- und Breitensuche

Neben der Suchrichtung (daten- oder zielorientiert) ist auch die Reihenfolge, in der die Zustände im Graphen untersucht werden, entscheidend.

Bei der **Tiefensuche** wird ein Pfad soweit wie möglich nach unten verfolgt. Hier werden bei der Untersuchung eines Zustands zunächst alle untergeordneten Knoten und deren Nachfolger geprüft und dann erst dessen Geschwister berücksichtigt (siehe Abb.3.3 nach [hale90]). Die Tiefensuche dringt schnell tiefer in den Suchraum ein, wann immer es möglich ist. Geschwister werden nur dann berücksichtigt, wenn ein Zustand keine weiteren Nachfolger besitzt.
Die Tiefensuche läßt sich effizient implementieren, geht aber nachteilig schnell in die Tiefe und verstrickt sich so in Details.

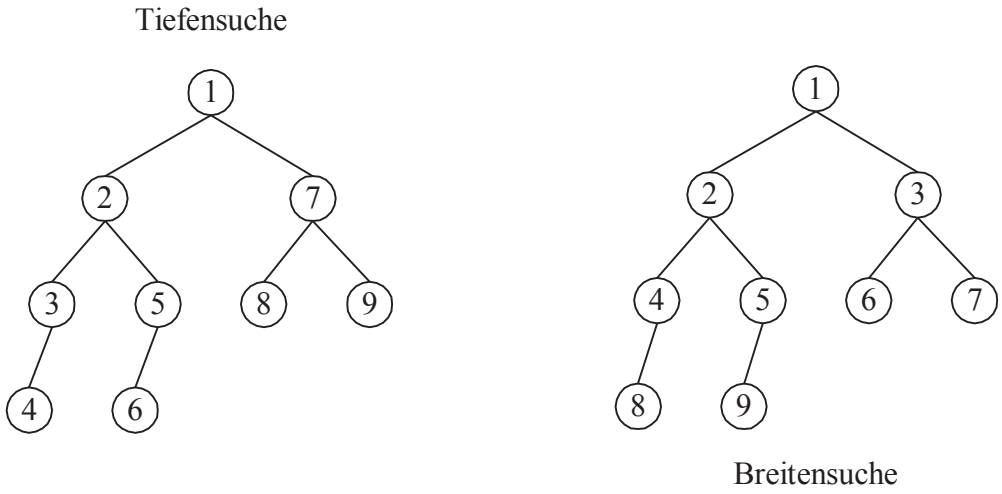


Abbildung 3.3: Beispiel zu Tiefen- und Breitensuche.

Die **Breitensuche** dagegen erforscht den Raum Ebene für Ebene. Nur dann, wenn auf einer gegebenen Ebene keine weiteren Zustände zu untersuchen sind, wechselt der Algorithmus zur nächsten Ebene (siehe Abb.3.3). Da bei der Breitensuche jeder Knoten einer Ebene berücksichtigt wird, bevor tiefer in den Raum eingedrungen wird, findet die Breitensuche daher garantiert den kürzesten Weg vom Start zum Ziel, benötigt dabei aber mehr Speicherplatz. [lug01]

Im Gegensatz zur Breitensuche ist bei einer Tiefensuche nicht garantiert, daß der kürzeste Weg zu einem Zustand gefunden wird, wenn der Algorithmus zum ersten Mal auf diesen Zustand trifft.
Wie bei der Wahl zwischen daten- und zielorientierter Suche zur Auswertung eines Graphen hängt auch die Wahl zwischen Tiefen- und Breitensuche vom zu lösendem Problem ab.
Zu den Auswahlkriterien gehören, wie wichtig es ist, den kürzesten Weg zu einem Ziel zu finden, der Verzweigungsfaktor des Raums, verfügbare Rechnerzeit und Speicherressourcen, und ob alle

Lösungen oder nur die erste Lösung erwünscht ist. [lug01]

Details zur Implementierung

Die beiden Suchalgorithmen verwenden zwei Listen, um den Fortschritt beim Durchsuchen des Zustandsraums zu verfolgen. Open listet die Zustände auf, die zwar generiert, deren Nachkommen aber noch nicht untersucht wurden. Die Reihenfolge, in der Zustände aus Open entfernt werden, bestimmt die Reihenfolge der Suche. In Closed werden die Zustände verzeichnet, die schon untersucht wurden. [lug01]

Alle nicht informierten Suchalgorithmen wie Tiefensuche und Breitensuche weisen beim Zeitverhalten im schlechtesten Fall eine exponenzielle Komplexität auf. Suchansätze, die diese Komplexität reduzieren, verwenden Heuristiken, um die Suche zu leiten. [lug01]

3.5.2 Heuristische Suche

> *Wenn die Zustandsraumsuche die Mittel zur Formalisierung des Problemlösungsprozesses zur Verfügung stellt, dann ermöglichen es Heuristiken, diesen Formalismus mit Intelligenz auszustatten. [lug01]*

Eine **Heuristik** ist nach [hale90] eine Strategie, durch die man in gezielter und effizienter Weise wenigstens eine brauchbare Lösung erhält. Dabei wird nicht garantiert, in allen Fällen eine optimale Lösung zu liefern, vielmehr soll in vernünftiger Zeit eine annehmbare Lösung gefunden werden.

Das Lösen eines umfangreichen Problems erfolgt durch Zerlegen des Problemlösungsraums in Teilräume mittels sinnvoller Suchregeln, den sogenannten Heuristiken. Wichtig bei Heuristiken ist, daß sie geeignete und problemspezifische Einschränkungen bzw. Abkürzungen nutzen, um ein intelligentes Lösungsverhalten zu erzielen. Heuristische Suchalgorithmen leiten die Suche auf die Pfade, die mit hoher Wahrscheinlichkeit zum Erfolg führen, und schließen wenig versprechende Zustände und deren Nachfolger aus. Dadurch werden unnütze oder offensichtlich dumme Anstrengungen vermieden, was eine Reduktion der Komplexität ergibt.
Heuristiken sind nicht absolut sicher und können somit versagen. Eine Heuristik ist nur eine informierte Vermutung, welche häufig auf Erfahrung oder Intuition beruht. Da nur beschränkte Informationen verwendet werden, kann eine Heuristik zu einer nur teilweise optimalen Lösung führen oder gar keine Lösung finden. Heuristiken besitzen im allgemeinen ausreichende Lösungsmöglichkeiten mit eben der Möglichkeit des Versagens bei Grenzfällen.

Heuristiken werden nach [lug01] für **zwei grundlegende Situationen** eingesetzt:

1. Ein Problem besitzt zwar eine exakte Lösung, doch die Rechenkosten für die Suche nach dieser Lösung sind unerschwinglich. Bei vielen Problemen (wie Schach) wächst der Zustandsraum explosionsartig. In solchen Fällen können erschöpfende Suchtechniken (wie Tiefen- oder Breitensuche) unter Umständen keine Lösung innerhalb eines praktikablen Zeitraums finden.
2. Ein Problem hat keine exakte Lösung, da die Problembeschreibung oder die verfügbaren Daten zweideutig sind, z.B. eine medizinische Diagnose: Eine gegebene Menge von Symptomen kann verschieden Ursachen haben. Heuristiken werden hier eingesetzt, um die wahrscheinlichste Diagnose zu wählen.

Es ist hilfreich, heuristische Algorithmen als zweigeteilt zu betrachten: das heuristische Maß und ein Algorithmus, der dieses Maß verwendet, um den Zustandsraum zu durchsuchen. Der Algorithmus wählt dann den Zustand mit dem höchsten heuristischen Wert und fährt dann fort.
Die einfachste Möglichkeit der Implementierung einer heuristischen Suche bietet die **Hill-Climbing-Strategie**. Sie expandiert den aktuellen Zustand während der Suche und bewertet dessen Nachkommen. Der beste Nachkomme wird für die weitere Expansion ausgewählt und weder dessen Geschwister noch der Vater werden weiterhin berücksichtigt. Da dieser Algorithmus den bisherigen Weg nicht aufzeichnet, ist keine Wiederherstellung (Backtracking) möglich, wenn die Strategie versagt. Ein Hauptproblem der Hill-Climbing-Strategien besteht in deren Tendenz, bei lokalen Maxima stecken zu bleiben. Trotz der Einschränkungen lässt sich Hill-Climbing effizient einsetzen, wenn dessen Bewertungsfunktion ausreichend informiert ist, um lokale Maxima und unendlich lange Wege zu vermeiden. [lug01]

Im Allgemeinen erfordert die heuristische Suche allerdings einen besser informierten Algorithmus, wie z.B. die Bestensuche.

3.5.2.1 Bestensuche

Die Bestensuche ist eine Variante der Breitensuche, wobei die beste Alternative je Ebene mittels Qualitätsfunktion weiterverfolgt wird. Sie sortiert die Zustände nach einer Bewertung des heuristischen Nutzens und berücksichtigt dann weder den tiefsten noch den flachsten sondern den besten Zustand. Wie die Tiefen- und Breitensuche verwendet auch die Bestensuche Listen, um Zustände aufzuzeichnen: In der Liste Open wird die aktuelle Grenze der Suche und in der Liste Closed werden die bereits besuchten Zustände verzeichnet. Zusätzlich ordnet der Algorithmus die Zustände

in Open nach einer heuristischen Bewertung, wie nahe sie sich am Ziel befinden. Damit befinden sich die besten Zustände am Anfang von Open. Führt die Heuristik die Bestensuche auf einen Weg, der sich als falsch herausstellt, wird der Algorithmus einige schon generierte nächstbeste Zustände aus Open abfragen und sich auf einen anderen Bereich des Raums konzentrieren. Somit erlaubt die Liste Open ein Backtracking aus Wegen, die kein Ziel produzieren.
Die Bestensuche ist ein allgemeiner Algorithmus, um jeden Zustandsraumgraphen heuristisch zu durchsuchen. Durch die Allgemeingültigkeit kann die Bestensuche in vielen Heuristiken eingesetzt werden, von subjektiven Schätzungen der Eignung eines Zustands bis hin zu hoch entwickelten Bewertungen, die auf Wahrscheinlichkeit beruhen. [lug01]

Ein weiterer Ansatz zur Implementierung von Heuristiken ist die Verwendung von Konfidenzintervallen bei Expertensystemen. Damit werden die Ergebnisse der Regeln gewichtet, um dann die Schlussfolgerungen auszuwählen, die am Erfolg versprechensten sind. Zustände mit extrem geringen Konfidenzwerten können frühzeitig ausgeschlossen werden. [lug01]

Andere KI-Repräsentationen wie Regeln, semantische Netze oder Frames verwenden Suchstrategien, die den bereits Vorgestellten gleichen.

3.5.3 Heuristiken und Expertensysteme

Mit dem Programm GPS (General Problem Solver) wurde versucht, allgemeine Prinzipien intelligenten Problemlösens zu entdecken. Der GPS lenkt die Suche mit einer allgemeinen Heuristik, die lediglich die syntaktische Form von Zuständen überprüft. Damit hoffte man zu erreichen, das sich der GPS als allgemeine, von der Wissensdomäne unabhängige Architektur zum intelligenten Problemlösen erweisen würde. Programme wie der GPS, die auf die Syntax bezogene Strategien beschränkt und für eine Vielzahl unterschiedlicher Anwendungen vorgesehen sind, werden auch als **Problemlöser mit schwachen Lösungsmethoden** bezeichnet. [lug01]

Komplexere Probleme benötigen bereits mehrere Heuristiken für die unterschiedlichen Situationen im Problemraum. Beispielsweise besitzen bei Spielen alle Knoten des Zustandsraums eine gemeinsame Repräsentation (z.B. eine Spielbrettbeschreibung), und so kann im gesamten Suchraum eine einzige Heuristik angewandt werden. Das steht im Gegensatz zu Systemen wie die Analyse von KFZ-Teilsystem-Problemen, bei denen jeder Knoten ein anderes Teilziel und jede Kante eine eigene Heuristik darstellt. Diese spezielle Heuristik ist lediglich für dieses KFZ-Teilsystem sinnvoll, bei der Diagnose von anderen KFZ-Teilsystemen ist sie nur wenig hilfreich und in Domänen, die nichts mit KFZ zu tun haben, ist sie völlig unbrauchbar.

> *Bei komplizierten realistischen Problemen werden situationsspezifische Problemlösungsheuristiken in die Syntax und in den Inhalt individueller Problemlösungstechniken integriert. Jeder Lösungsschritt enthält seine eigene Heuristik, die bestimmt, wann auf die relevanten Zustände im Raum bei einem passenden Muster die zugehörige Operation bzw. Heuristik angewandt wird. [lug01]*

Dieser Ansatz wird als **starke Problemlösungsmethode bzw. wissensbasierter Problemlöser** bezeichnet und setzt eine Menge Wissen über die jeweilige Situation voraus. Die starken Problemlösungsmethoden wie z.B. Expertensysteme unterscheiden sich durch ihren Einsatz von expliziten Wissen über eine bestimmte Problemdomäne deutlich von den schwachen Methoden.
Eine der Herausforderungen beim Entwurf eines wissensbasierten Programms besteht darin, große Mengen von domänenspezifischem Wissen zu akquirieren und zu strukturieren. [lug01]

4 Expertensystem

Zu Beginn dieses Kapitels wird der Begriff Expertensystem umfassend definiert. Dann werden die Einsatzgebiete, Stärken, Schwächen des Expertensystems aufgeführt. In weiterer Folge wird dessen Entstehungsprozeß und die daran beteiligten Akteure sowie deren Aufgaben angegeben. Anschließend wird das Produktionssystem, die Grundlage des Expertensystems, und seine Architektur definiert und erklärt. Dabei wird detailiert auf dessen Steuerungsmöglichkeiten und Vorzüge eingegangen. Darauf aufbauend wird das regelbasierte Expertensystem behandelt.

4.1 Definition

> *Starke Problemlösungsmethoden bzw. wissensbasierte Problemlöser wie z.B. Expertensysteme (ES) machen Gebrauch von für einen bestimmten Problembereich spezifischen Wissen, um für den jeweiligen Anwendungsbereich Leistung auf Expertenniveau zu bieten. Im Allgemeinen wird dieses Wissen beim Entwurf von Expertensystemen durch Unterstützung von Experten für einen bestimmten Bereich erworben, wobei das System die Methodologie und das Verhalten des menschlichen Experten nachbildet. Expertensysteme sind, wie auch menschliche Experten, insofern spezialisiert, als sie sich auf einen stark eingegrenzten Problembereich konzentrieren und über klar definierte Problemlösungsstrategien verfügen. [lug01]*

Die menschlichen Experten, die das Wissen des Systems bereitstellen, haben ihr tiefes theoretisches Verständnis des Problembereichs durch den Einsatz von Oberflächenwissen erweitert, um das durch Erfahrung im Problemlösen gewonnene Wissen verwenden zu können.

Tiefes Wissen: aus grundlegenden, wissenschaftlich anerkannten Prinzipien gewonnen z.B. Gesetze und Modelle aus Mechanik und Physik.

Oberflächenwissen: umfaßt informelle Tricks, Kunstgriffe und heuristische Verfahren und gründet sich auf Erfahrungswissen, das beim Lösen von konkreten Problemen erworben wurde.

Dieses heuristische Wissen stellt eine wichtige Ergänzung zu den Standardtheorien dar. Manchmal untermauern diese Regeln theoretisches Wissen in nachvollziehbarer Weise. Oft handelt es sich auch um Abkürzungen, deren Richtigkeit sich empirisch gezeigt hat. Expertenwissen setzt sich somit aus tiefem Wissen und einer Sammlung heuristischer Problemlösungsregeln zusammen, die sich erfahrungsgemäß in dem Bereich als effizient erwiesen haben. Ein Großteil dessen, was man langläufig als Intelligenz bezeichnet, scheint dabei den Heuristiken innezuwohnen, welche eine wesentliche Grundlage für intelligente Expertensysteme bilden. [lug01] und [hale90].

Nach [lug01] lassen sich **Expertensysteme** allgemein folgendermaßen **charakterisieren**:

1. Sie setzen heuristische Schlussfolgerungsverfahren ein, indem sie Wissen nutzen, um zu brauchbaren Lösungen zu gelangen.
2. Sie erlauben eine problemlose Modifizierung durch Hinzufügen und Löschen von Wissen aus der Wissensbasis.
3. Sie unterstützen die Beurteilung ihres eigenen Schlussfolgerungsverfahrens, indem sie sowohl Zwischenschritte anzeigen als auch Fragen zum Lösungsprozeß hinlänglich beantworten können.

Das Schließen innerhalb eines Expertensystems sollte Informationen über den Status seiner Problemlösungsvorgänge und Erklärungen zu den Entscheidungen bereitstellen, die das Programm trifft. Erklärungsroutinen behalten Informationen zum ausgewählten Weg durch den Suchgraphen und protokollieren Zwischenergebnisse. Erklärungen sind für einen menschlichen Experten, von großer Bedeutung, wenn es darum geht, Vorschläge von einem Computer anzunehmen. [lug01]

4.2 Allgemein

4.2.1 Einsatzgebiete

Die **Stärke** der Expertensysteme liegt dort, wo Problemlösungen auf unvollständigen Theorien basieren, das Wissen in Form von Erfahrungswissen vorliegt, oder nur vages bzw. diffuses Wissen existiert, das nur fragmentarisch über Regeln und Heuristiken formuliert werden kann. In diesen Fällen ist das vorhandene Wissen nicht so aufbereitet, daß man generell ausgelegte Algorithmen in konventioneller Form aufbauen könnte, es fehlt eine algorithmusierbare Ordnung des Wissens. [hale90]

Allgemeine **Problemkategorien** nach [pup91]:

- Interpretation: Ableitung von Situationsbeschreibungen aus Sensordaten.
- Diagnostik: Ableitung von Systemfehlern aus Beobachtungen.
- Überwachung: Vergleich von Beobachtungen mit Sollwerten.
- Design: Konfigurierung von Objekten unter besonderen Anforderungen.
- Planung: Entwurf einer Abfolge von Aktionen zum Erreichen eines Zieles.
- Simulation: Ableitung von möglichen Konsequenzen gegebener Situationen.

Verschiedene Problemtypen lassen sich mit denselben Strategien lösen. So besteht das Ziel bei der Interpretation, Diagnose und Überwachung meist darin, ein bekanntes Muster wiederzuerkennen. Dabei wird die Lösung aus einer Menge von vorgegebenen Alternativen ausgewählt. Im Gegensatz dazu wird beim Design und bei der Planung die Lösung aus kleinen Bausteinen zusammengesetzt, da es zu viele Kombinationen gibt, als daß eine Auswahl an Alternativen möglich wäre. Diese Bausteine sind bei der Planung Handlungen und beim Design Komponenten des zu entwerfenden Objektes. Die Simulation unterscheidet sich dadurch, daß kein vorgegebenes Ziel erreicht werden soll, sondern nur die Auswirkungen von Handlungen oder Ereignissen simuliert werden soll.

Expertensysteme bergen auch viele **Unzulänglichkeiten und Gefahren** in sich: die Richtigkeit des eingebrachten Wissens und der Lösungsvorschläge; Grenzfallverhalten; die Schwierigkeit, tiefere Kenntnisse des Problembereichs darzustellen; der Mangel an Robustheit und Flexibilität; die Unfähigkeit, tiefergehende Erklärungen zu bieten; Schwierigkeiten bei der Verifizierung; geringes Lernen aus Erfahrung. [lug01] und [hale90]

Die Vernachlässigung der **Einbindung und** somit der **Vorteile anderer Bereiche der Computerwissenschaften** (konventionelle Programmsysteme) bei der Entwicklung von Expertensystemen ist nachteilig. Die Grundidee der heuristischen Problemlösung sollte nicht dazu benutzt werden, alle Problembereiche mit dieser informellen Methode zu lösen. Heuristische Methoden sind nur dann zweckmäßig, wenn formalere Methoden nicht verfügbar oder nicht anwendbar sind. Deshalb sollten generische Teile der Problemlösung mit konventionellen Programmsystemen gelöst werden. [pri89]

4.2.2 Akteure

Hauptdarsteller beim Entwurf eines Expertensystems sind der Wissensingenieur, der Experte für den Problembereich und der Anwender.

Der **Anwender** legt die wichtigsten Einschränkungen für den Entwurf fest.
Der **Experte** für den Problembereich stellt das Wissen zur Verfügung. Im Allgemeinen handelt es sich dabei um jemanden, der über Erfahrungen im Problembereich und über Verfahren zur Problemlösung verfügt. Die Hauptaufgabe des Experten für den Problembereich besteht darin, diese Kenntnisse dem Wissensingenieur zugänglich zu machen.
Der **Wissensingenieur** ist der KI-Experte für Wissensrepräsentation und KI-Sprachen. Seine Aufgaben sind vergleichbar mit denen des Entwicklers eines konventionellen Softwaresystems. Ihm obliegt es, die Aufgabe des zu entwickelnden Systems genau zu definieren und Software und Hardware auszuwählen. Oft hat der Wissensingenieur zunächst keinerlei Kenntnisse über den Anwendungsbereich. Nachdem er sich einen Überblick verschafft hat, beginnt der Entwurf des Systems. Zuerst muß ein geeigneter Repräsentationsformalismus ausgewählt werden, weiters ist eine Suchstrategie festzulegen. Dann hilft er dem Experten bei der Formulierung des erforderlichen Wissens und implementiert dieses Wissen in einer passenden Wissensbasis. [lug01] und [gfh90]

4.2.3 Entwicklungs-Zyklus

Entwicklungs-Zyklus nach [gfh90]:

- Problemidentifikation:
  - Untersuchung des Problemgebietes;
  - Selektion abgrenzbarer Teilaufgaben und Wissensbereiche;
  - Festlegung der in diesem Entwicklungszyklus zu lösenden Aufgabenstellung. Die Lösbarkeit des Gesamtproblems soll dabei nicht aus den Augen verloren werden;
  - Entwurf eines (vorläufigen) Lösungsmodells.
- Wahl der geeigneten Repräsentationsform:
  - Identifikation geeigneter Konzepte zur Wissensdarstellung;
  - Erforschung der Problemlösungstechniken und –strategien des Experten.
- Formalisierung des Wissens:
  - Identifikation von charakteristischen Eigenschaften des Wissens;
  - Darstellung des Expertenwissens im Repräsentationsformalismus;
  - Modellierung der zugrundeliegenden Prozesse.
- Implementierung:
  - Auswahl geeigneter Softwarewerkzeuge;

  - Anpassung des formalisierten Wissens an deren Darstellung;
  - Aufbau der Wissensbasis.

- Validierung des Expertensystemverhaltens unter Mithilfe des Experten an Hand von Falldaten.

Dieser Zyklus muß zumindest partiell meist öfter durchlaufen werden.

Expertensysteme werden durch fortschreitende Annäherungen an das fertige System erstellt, wobei Fehler des Systems zu Korrekturen oder Erweiterungen der Wissensbasis führen. Das steht im Gegensatz zu konventionellen Entwicklungsprozessen wie dem Top-Down-Entwurf. [lug01]

4.3 Produktionssystem

> *Das Produktionssystem ist eine allgemeine Architektur zum musterorientierten Problemlösen. Es besteht aus einer Menge von Produktionsregeln, einem Arbeitsspeicher und einem Erkennen-Handeln-Steuerungszyklus. Dabei ist das Produktionssystem neben einem Rechenmodell zur Implementierung der Graphensuche auch ein echtes Modell des menschlichen Problemlösungsverhaltens.*

Die **Produktionsregeln** entsprechen dem Problemlösungsfertigkeiten des menschlichen Langzeitgedächtnisses und werden durch die Ausführung des Systems nicht geändert. Sie werden durch das Muster einer bestimmten Probleminstanz aktiviert. Weiters können neue Fertigkeiten hinzugefügt werden, ohne daß das zuvor vorhandene Wissen umkodiert werden muß.
Der **Arbeitsspeicher** des Produktionssystems entspricht dem Kurzzeitgedächtnis oder dem aktuellen Fokus der menschlichen Aufmerksamkeit und beschreibt den aktuellen Zustand bei der Lösung der Probleminstanz.
Produktionssysteme stellen damit ein Modell zur Kodierung menschlicher Expertise in Form von Regeln und zum Entwurf von musterorientierten Suchalgorithmen zur Verfügung, d.h. Aufgaben, die zentraler Bestandteil des Entwurfs von regelbasierten Expertensystemen sind. [lug01]

Um die Prädikatenlogik als Schlussfolgerungsmechanismus einsetzen zu können, wird eine Problemlösungskomponente, ein **Steuerungsmechanismus**, benötigt, der den Raum systematisch durchsucht und gleichzeitig sinnlose Wege und Schleifen vermeidet. Ein Steuerungsalgorithmus muß dabei alternative Möglichkeiten in irgendeiner sequenziellen Reihenfolge überprüfen. [lug01]

4.3.1 Definition

Ein Produktionssystem wird nach [lug01] definiert durch:

1. Die Menge an **Produktionsregeln**. Eine Produktionsregel setzt sich aus einem Bedingungs- und einem Aktionsteil zusammen (Bedingung => Aktion) und definiert ein Element des Problemlösungswissens. Bei dem Bedingungsteil handelt es sich um ein Muster, das festlegt, wann diese Regel auf eine Probleminstanz angewandt werden kann. Der Aktionsteil definiert den zugehörigen Problemlösungsschritt.
2. Der **Arbeitsspeicher** enthält eine Beschreibung des aktuellen Zustands der Welt in einem Schlussfolgerungsprozeß. Diese Beschreibung ist ein Muster, das zur Auswahl der geeigneten Problemlösungsaktionen mit dem Bedingungsteil der Produktionsregeln verglichen wird. Wenn der Bedingungsteil einer Regel mit dem Inhalt des Arbeitsspeichers übereinstimmt, dann kann die mit dieser Bedingung verknüpfte Aktion ausgeführt werden. Die Aktionen von Produktionsregeln sind speziell darauf ausgelegt, den Inhalt des Arbeitsspeichers zu ändern.
3. Der **Erkennen-Handeln-Zyklus** (siehe Abb.4.1). Der Steuerungsmechanismus von Produktionssystemen ist einfach: Der Arbeitsspeicher wird mit dem Beginn der Problembeschreibung initialisiert. Der aktuelle Status des Problemlösungsprozesses wird in Form von Mustern im Arbeitsspeicher dargestellt. Diese Muster werden mit den Bedingungen der Produktionsregeln verglichen. Daraus ergibt sich eine Teilmenge der Produktionsregeln, die so genannte Konfliktmenge, deren Bedingungen den Mustern entsprechen. Anschließend wird eine in der Konfliktmenge enthaltene Produktionsregel mittels Konfliktlösung ausgewählt und angewandt, indem ihr Aktionsteil ausgeführt wird. Somit wird der Inhalt des Arbeitsspeichers geändert, und andere Regeln werden anwendbar. Erneut wird der Steuerungszyklus mit dem nun veränderten Arbeitsspeicher wiederholt. Diese Produktionssystemschleife endet, wenn der Inhalt des Arbeitsspeichers mit keiner Regelbedingung mehr übereinstimmt.

Mit Hilfe eines Produktionssystems läßt sich der Suchraum eines Problems durchsuchen. Die Produktionsregeln sind die kodierte Menge von Schlussfolgerungen zur Änderung des Zustands innerhalb des Graphen. Wenn die Produktionsregeln als logische Implikationen formuliert sind, dann entspricht die Aktion der Regelanwendung der Anwendung des Modus Ponens. Durch die Regelanwendung wird ein neuer Zustand im Graphen erzeugt.
Das reine Produktionssystemmodell beinhaltet keine Mechanismen wie Backtracking, die eine Wiederaufnahme der Suche ermöglicht, falls die Suche in eine Sackgasse geführt hat. [lug01]

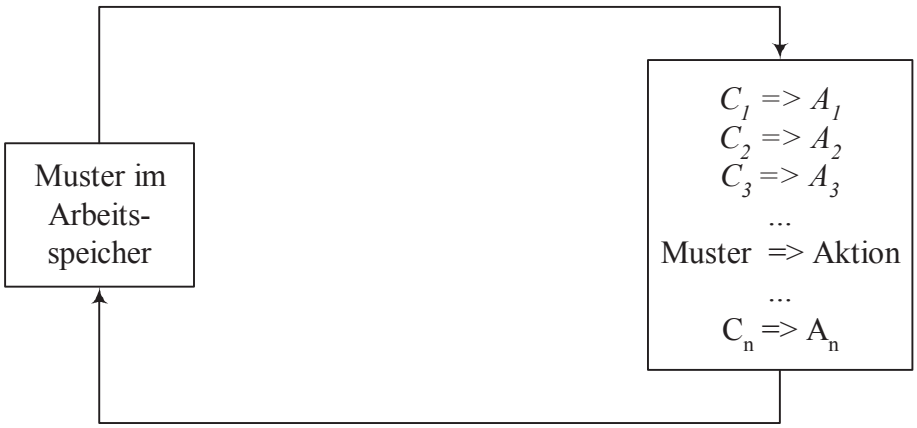


Abbildung 4.1: Produktionssystemschleife

4.3.2 Steuerung

Daten- oder zielorientierte Steuerung der Suche

Eine Möglichkeit der Steuerung resultiert aus der Wahl zwischen daten- und zielorientierter Suchstrategie (siehe Kapitel 3.5.11):

Die datenorientierte Suche beginnt mit einer Problembeschreibung, z.B. eine zu interpretierende Datenmenge, und gewinnt neues Wissen aus den Daten. Dies geschieht durch die Anwendung von Schlussfolgerungsregeln oder anderen zustandsgenerierenden Operationen auf die aktuelle Beschreibung der Welt und durch das Hinzufügen der Ergebnisse zur Problembeschreibung im Arbeitsspeicher.
Der aktuelle Zustand der Welt umfaßt damit Daten, deren Wahrheit vorausgesetzt bzw. aus der Anwendung von Regeln abgeleitet wurde. Im Erkennen-Handeln-Zyklus wird der aktuelle Zustand dann mit der Menge von Produktionsregeln verglichen. Wenn diese Daten mit den Bedingungen einer Produktionsregel übereinstimmen, dann ergänzt der Aktionsteil mittels Ändern des Arbeitsspeichers den aktuellen Zustand der Welt.

Um eine **zielorientierte Suche** in einem Produktionssystem zu implementieren, wird das Ziel in den Arbeitsspeicher eingefügt und mit dem Aktionsteilen verglichen. All diejenigen Produktionsregeln, deren Aktionen mit dem Ziel übereinstimmen, bilden die Konfliktmenge. Wenn die Aktion einer Regel mit dem Ziel übereinstimmt, wird die zugehörige Bedingung in den Arbeitsspeicher eingefügt und zum neuen Teilziel der Suche. Dieser Vorgang entspricht einer Zerlegung des Problemziels in einfachere Teilziele. Diese neuen Zustände werden daraufhin mit den Aktionsteilen anderer Produktionsregeln verglichen. Dieser Vorgang wird so lange fortgesetzt, bis ein Fakt gefunden oder vom Benutzer angegeben wurde. [lug01]

Es lassen sich beide Suchvarianten mühelos durch Produktionssysteme charakterisieren (siehe Abb.4.2), wobei eine Affinität zwischen datenorientierter Suche und Produktionsmodell existiert. Bei der zielorientierten Suche werden andere Produktionsregeln angewandt und damit ein anderer

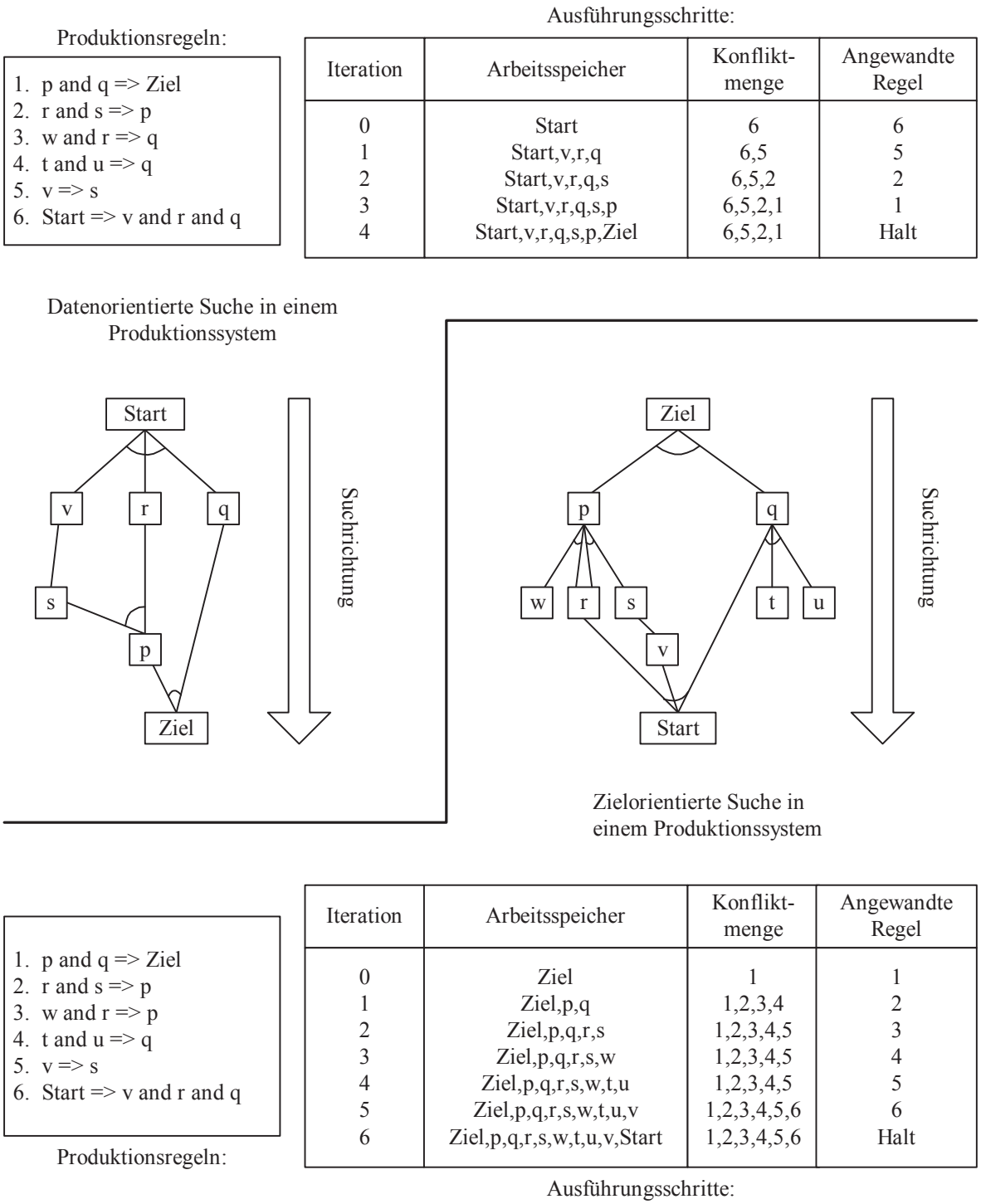


Abbildung 4.2: Beispiel zu daten- und zielorientiertem Produktionssystem.

Teilraum durchsucht als bei der datenorientierten Suche (siehe Abb.4.2).
Die Konfliktlösungsstrategie im Abb.4.2 ist einfach die, daß diejenige anwendbare Regel ausgewählt wird, die am längsten oder überhaupt noch nicht angewandt worden ist. Stehen unter diesen Bedingungen mehrere Regeln zur Auswahl, wird die Erste gewählt. [lug01]

Steuerung der Suche durch die Regelstruktur

Die Struktur der Regeln, einschließlich der Unterscheidung zwischen Bedingung und Aktion und der Reihenfolge, in der Bedingungen geprüft werden, legt fest, in welcher Art und Weise der Raum durchsucht wird.
Die Prädikatenlogik hat eine deklarative Semantik. D.h., prädikatenlogische Ausdrücke definieren einfach wahre Beziehungen in einer Problemdomäne und machen keinerlei Annahmen über die Reihenfolge, in der deren Komponenten interpretiert werden. Die Implementierung in einem Produktionssystem erzwingt, daß die Regeln in einer bestimmten Reihenfolge verglichen und angewandt werden. Aus diesem Grund bestimmt die jeweilige Form der Regelformulierung, ob und wie einfach eine entsprechende Regel für einen Problemzustand gefunden werden kann. [lug01]

> *Das Produktionssystem zwingt der deklarativen Sprache, die zur Formulierung der Regeln eingesetzt wird, eine prozedurale Semantik auf. [lug01]*

Der Programmierer kann die Suche durch die Struktur und die Reihenfolge der Produktionsregeln steuern.

Steuerung der Suche durch die Konfliktlösungsstrategie

Die wichtigsten Konfliktlösungsstrategien, die auch kombiniert werden können, sind nach [pup91]:

- Auswahl nach der Reihenfolge, z.B.:
  - Die erste anwendbare Regel feuert (einfachste und triviale Strategie).
  - Die aktuellste Regel feuert, d.h. die Regel, deren Vorbedingung sich auf möglichst viele neue Einträge in der Datenbasis bezieht.
- Auswahl nach syntaktischer Struktur der Regel, z.B.:
  - Spezifischere Regeln werden bevorzugt. Die syntaktisch größte Regel feuert, z.B.: die Regel mit den meisten Aussagen.
  - Auswahl mittels Zusatzwissen, z.B.:

- Die Regel mit der höchsten Priorität feuert. Dazu muß jeder Regel eine Priorität zugeordnet sein.
- Zusätzliche Regeln, sogenannte Meta-Regeln, steuern den Auswahlprozeß.

4.3.3 Vorzüge

Aufgrund der Architektur des Produktionssystems und der daraus resultierenden Eigenschaften ergeben sich nach [lug01] folgende grundlegende Vorteile:

Trennung von Wissen und Steuerung: Die Steuerung erfolgt über den Erkennen-Handeln-Zyklus der Produktionssystemschleife und das Problemlösungswissen ist in den Regeln selbst kodiert. Vorteilhaft an dieser Trennung ist u.a., daß die Wissensbasis abgeändert werden kann, ohne daß die Programmsteuerung hierzu geändert werden muß, und daß umgekehrt der Code für die Programmsteuerung geändert werden kann, ohne daß die Menge der Produktionsregeln geändert werden muß.

Eine natürliche Affinität zur Zustandsraumsuche: Die Komponenten eines Produktonssystems lassen sich mühelos auf die Konstrukte der Zustandsraumsuche abbilden. Die aufeinander folgenden Zustände des Arbeitsspeichers bilden die Knoten des Zustandsraumgraphen. Die Produktionsregeln stellen die Menge möglicher Übergänge zwischen Zuständen dar, wobei die Konfliktlösung die Auswahl eines Zweiges des Zustandsraums implementiert. Dies vereinfacht Implementierung, Testen und Dokumentation von Suchalgorithmen.

Modularität der Produktionsregeln: Durch das Fehlen jeglicher syntaktischer Interaktionen zwischen den Produktionsregeln können sie sich nur insofern beeinflussen, als daß sie die Muster im Arbeitsspeicher ändern. Sie können andere Regeln weder direkt aufrufen, noch können sie den Wert von Variablen in anderen Produktionsregeln festlegen. Der Gültigkeitsbereich der Variablen dieser Regeln ist auf die jeweilige Regel beschränkt. Diese syntaktische Unabhängigkeit kommt der inkrementellen Entwicklung entgegen, bei der Wissen nach und nach ergänzt, entfernt und geändert wird.
In einem System, das ausschließlich aus Produktionsregeln besteht, führt eine Änderung einer einzelnen Regel zu keinerlei globalen syntaktischen Nebeneffekten. Regeln können deshalb hinzugefügt und entfernt werden, ohne daß weitere Änderungen an dem umgebenden Programm erforderlich wären.

Musterorientierte Steuerung: Der Erfordernis nach besonderer Flexibilität in der Programmausführung für KI-Anwendungen kommt es entgegen, das die Regeln eines Produktionssystems in jeder beliebigen Reihenfolge angewandt werden können. Die Problembeschreibun-

gen, die den aktuellen Zustand der Welt bilden, legen die Konfliktmenge fest und folglich den jeweiligen Suchweg und die Lösung.

Möglichkeiten zur heuristischen Steuerung der Suche: Neben der Kodierung von Heuristiken im Wissensgehalt der Regeln selbst, stellen Produktionssysteme über die Konfliktlösungsstrategie eine weitere Möglichkeit zur heuristischen Steuerung zur Verfügung.

Verfolgung und Erklärung: Die Modularität der Regeln und die Art ihrer Anwendung erleichtern es, die Arbeitsweise eines Produktionssystems zu verfolgen.
Jede Regel entspricht einem einzelnen Element des Problemlösungswissens und die Kette der Regeln, die in einem Lösungsprozeß angewandt werden, spiegelt die Schlussfolgerungssequenz eines menschlichen Experten wider. Dagegen ist eine einzelne Codezeile oder eine Prozedur einer konventionellen Programmiersprache praktisch bedeutungslos.

Sprachunabhängigkeit: Das Steuerungsmodell von Produktionssystemen ist unabhängig von der für die Regeln und den Arbeitsspeicher gewählten Repräsentation, solange diese Repräsentation die Mustererkennung unterstützt.

Ein plausibles Modell menschlicher Problemlösungsverfahren.

4.4 Regelbasierte Expertensysteme

Das Produktionssystem ist der geistige Vorgänger der Architektur der heutigen Expertensysteme. Dabei wird Expertensystemen aber nicht notwendigerweise unterstellt, daß es menschliches Problemlösungsverhalten modelliert. [lug01]

4.4.1 Architektur

Um mit einem Expertensystem praktisch und effizient arbeiten zu können, gibt es neben den beiden Grundkomponenten Wissensbasis und Steuerungsmechanismus noch weitere **Komponenten**:

Interviewerkomponente: führt den Dialog mit dem Benutzer und/oder liest automatisch erhobene Meßdaten ein. Falls kein Benutzerdialog stattfinden, nennt man das Expertensystem auch eingebettetes System, ansonsten interaktives System.

Erklärungskomponente: macht die Vorgehensweise des Expertensystems transparent.

Wissenserwerbskomponente: ermöglicht es dem Experten, sein Wissen in das Expertensystem einzugeben und später zu ändern. [pup91]

Die **Wissensbasis** besteht aus der allgemeinen Wissensbasis und dem Arbeitsspeicher. Die **allgemeine Wissensbasis** bildet den Kern eines Expertensystems und besteht aus einer Menge von Produktionsregeln. Diese Bedingungs-Aktions-Regeln werden in einem Expertensystem als Wenn-Dann-Regeln repräsentiert. Fallspezifische Daten werden im **Arbeitsspeicher** (Datenbasis) gehalten.
Dazu gehören Fakten (Tatsachen), Schlussfolgerungen und für den Problembereich relevante Informationen. Zu diesen wiederum gehören z.B.: Daten für eine gegebene Instanz eines Problems, Teillösungen, Konfidenzmaße für Schlussfolgerungen und Informationen zu Sackgassen bei der Problemlösung. [lug01]
Nach Herkunft unterscheidet man das Wissen in Expertenwissen, fallspezifisches Wissen von Benutzern und Zwischen- und Endergebnisse, die von der Steuerungskomponente hergeleitet worden sind.
Fallspezifisches Wissen und Zwischen- und Endergebnisse sind typischerweise Faktenwissen, während das Expertenwissen sowohl aus Faktenwissen (z.B. Katalogwissen) als auch aus Ableitungs- und Kontrollwissen besteht. Während Ableitungswissen (z.B. Regeln) den Gebrauch des Faktenwissens steuert, steuert das Kontrollwissen (z.B. sog. Meta-Regeln) den Gebrauch von Ableitungswissen. [pup91]

Die **Steuerungskomponente** bzw. der Inferenzmechanismus wendet das Wissen an. Letztendlich ist der Inferenzmechanismus ein Interpreter für die Wissensbasis und führt den Erkennen-Handeln-Zyklus durch.
Die Anweisungen zur Durchführung des Steuerung sind dabei von den Produktionsregeln getrennt.

Bedeutung der Trennung zwischen Wissensbasis und Steuerungsmechanismus nach [lug01]:

1. Die Trennung ermöglicht eine natürlichere Repräsentation des Wissens. Wenn-Dann-Regeln z.B. modellieren sehr viel genauer die Verfahren zur Problemlösung, wie Menschen sie selbst beschreiben, als maschinennahe Anweisungen dies können.
2. Da die Wissensbasis von den Kontrollstrukturen des Programms auf unterer Ebene getrennt ist, können sich die Entwickler beim Entwurf des Expertensystems auf die Akquisition und die Strukturierung des zur Problemlösung benötigten Wissens konzentrieren und die Details der Implementierung vernachlässigen.
3. Die Trennung ermöglicht das Durchführen von Änderungen an der Komponente der Wissensbasis, ohne daß dadurch andere Programmkomponenten beeinflusst werden.
4. Die Trennung ermöglicht die Wiederverwendung der Steuerung und der Benutzerschnittstelle (z.B. Expertensystem-Shell).

4.4.2 Steuerung

Die Steuerung erfolgt entweder daten- oder zielorientiert.
Für viele Problembereiche scheint sich vor allem die Suche durch Vorwärtsverkettung zu eignen. Bei einem Interpretationsproblem sind z.B. die Daten größtenteils vorgegeben und es ist schwierig, Hypothesen oder Ziele zu formulieren. Daher liegt es auf der Hand, sich für ein vorwärtsverkettetes Schlussfolgerungssystem zu entscheiden, bei dem die Fakten im Arbeitsspeicher abgelegt werden und das System nach einer Interpretation sucht. Bei der **datenorientierten** Verarbeitung wird die Breitensuche häufiger verwendet. [lug01]

Details zur Implementierung

Ein einfacher Algorithmus besteht darin, daß man der Reihe nach die Vorbedingungen aller Regeln überprüft, die anwendbaren Regeln entsprechend der Konfliktlösungsstrategie sortiert und den Aktionsteil des Spitzensreiters der Konfliktmenge ausführt. Danach löscht man die Konfliktmenge und beginnt von neuem.
Eine erste Verbesserung dieses einfachen Algorithmus ist die Ausnutzung von Ähnlichkeiten zwischen den Vorbedingungen verschiedener Regeln. So können Regeln mit gemeinsamen Aussagen auch gemeinsam ausgeschlossen werden, wenn eine solche Aussage nicht zutrifft. Dazu werden die Regeln in Pfade eines Netzwerkes abgebildet. Da eine Aussage nur einmal im Netzwerk dargestellt wird, kreuzen sich die verschiedenen Pfade vielfältig, was den Effizienzgewinn bewirkt. Eine solche Transformation nennt man Regelkompilierung, welche es dem System ermöglicht, Regeln und Daten miteinander abzugleichen, indem es einem Zeiger auf diese Regel folgt.
Eine zweite Verbesserung basiert auf der Überlegung, daß eine Neuberechnung der Konfliktmenge weit aufwendiger ist als ihre Modifikation aufgrund von Änderungen. Von einem Zyklus zum nächsten ändert sich die Datenbasis nur durch die Ausführung der Aktion der ausgewählten Regel, die einzelne Daten hinzufügen oder löschen kann. Deswegen braucht man nur zu überprüfen, ob die gelöschten Daten die Anwendbarkeit einer Regel der alten Agenda zerstören und ob die hinzugefügten Daten neue Regeln anwendbar machen. Letzteres wird überprüft, indem man mit den neuen Daten als Ausgangspunkt das Regelnetzwerk durchläuft.
Diese beiden Verbesserungsideen sind im RETE-Algorithmus realisiert, wodurch die Ausführung des Prozesses unter Beibehaltung der Semantik wesentlich beschleunigt wird. [pup91]

Bei **zielorientierten** Systemen wird in Hinblick auf ein bestimmtes Ziel geschlossen. Dieses Ziel wird wiederum in Teilziele zerlegt, deren Erreichung den Vorgang dem Hauptziel näher bringen und die wiederum weiter zerlegt werden können. Daher wird diese Suchstrategie immer durch die

Hierarchie von übergeordnetem Ziel und Teilzielen geleitet.
Datenorientierte Strategien suchen dagegen wesentlich weniger zielgerichtet. Stattdessen wird die Traversierung des Suchgraphen ausschließlich durch die Reihenfolge der Regeln und durch das Erschließen neuer Informationen gesteuert. [lug01]

Der Entwickler verfügt über eine weitere Möglichkeit zur Steuerung der Suche, nämlich durch die **Anordnung der Regeln in der Wissensbasis**. Die prozedurale Interpretation der Regeln versperrt zwar einige der Vorteile der rein deklarativen Repräsentationen, sie spiegelt jedoch die Problemlösungsstrategie wider, die auch der menschliche Experte verwendet. Wir können beispielsweise die Prämissen einer Regel so anordnen, daß die Prämisse mit der geringsten oder besten Aussicht, erfüllt zu werden, zuerst weiterverfolgt wird.
Neben der Reihenfolge der Prämissen innerhalb einer Regel kann auch die Bedeutung einer Regel selbst heuristisch sein. In dem KFZ-Teilsystem-Beispiel (siehe Seite 32 und 39) sind alle Regeln heuristisch, und daher ist es möglich, daß das System zu falschen Schlüssen kommt. [lug01]

4.5 Expertensystem-Shell

Die Erstellung eines kompletten Expertensystems ist mit einem erheblichen Aufwand verbunden. Daher gibt es Systeme, die bis auf die problemabhängige Wissensbasis bereits alle Komponenten gebrauchsfähig enthalten. Jedoch muß die innere Logik der Schale (Shell) mit der inneren Logik des konkreten Problems übereinstimmen.
Beurteilungskriterien für den Einsatz einer Expertensystem-Shell sind: konkrete Problemstellung (Problemklasse); Hardware-Konfiguration; Problemumfang; Antwortzeiten; Qualität und Benutzerfreundlichkeit der Dialogkomponente; eingesetzte Inferenzstrategien (Vorwärts/Rückwärts); Möglichkeiten zum Behandeln mathematischer Formeln; Schnittstellen zu traditonellen Programmen; Hilfe im Bereich der Wissenserwerbskomponente; Portabilität; Support; Kosten. [hale90]

5 Objektorientierung und Schichtenarchitektur

Da bei dem zu erstellenden Programm-Muster neben wissensbasiert auch objektbasiert bzw. objektorientiert entwickelt werden soll, behandelt dieses Kapitel einige grundlegende Aspekte der Objektorientierung. Der zweite Teil dieses Kapitels beinhaltet die Grundlagen der Schichten-Architektur. Weiters wird deren Umsetzung sowohl bei konventionellen als auch bei integrierten wissensbasierten Systemen erörtert.

5.1 Klassisch prozedurale Programmierung

Die meisten **klassischen Programme** erzeugen spezifische Informationen durch Abarbeiten von Anfangsinformationen mittels einer Sequenz von Operationen. Systeme mit diesem ausschließlich funktionalen Ansatz lassen den natürlichen, ursprünglichen Informationskontext vermissen. Sowohl die Anfangsinformation als auch das Ergebnis der Abarbeitung ist meist losgelöst vom ursprünglichen Informations-Dasein im Anwendungsbereich und angesammelt als unnatürliche Kollektion von Input und Output-Daten. Der Anwender muß die Kollektion an Input-Daten bereitstellen und umarbeiten und die Ergebnisse dann wiederum in ihren logischen Kontext bringen. Die Objekte und Beziehungen des Anwendungsbereichs müssen also in eine dem Programm entsprechende Form umgewandelt werden.

Bei der klassischen Entwicklung ergeben sich folgende **Nachteile**:

Die damit erstellten Produkte (Dokumentation, Programme) sind wenig flexibel bezüglich zukünftiger Änderungen und Erweiterungen. Weiters sind umfangreiche Modelle der strukturierten Analyse schwierig zu lesen und zu ändern. Beim Übergang von der strukturierten Analyse zur Datenabstraktion entsteht ein eklatanter Strukturbruch. Die Durchgängigkeit zwischen den Phasen wird dadurch erheblich erschwert.

5.2 Objektorientierte Programmierung

Bei der **objektorientierten Entwicklung** hingegen werden in allen Phasen dieselben objektorientierten Konzepte verwendet. Dadurch wird eine bessere Durchgängigkeit über die Phasen Analyse, Entwurf und Implementierung hinweg erreicht, da kein Strukturbruch auftritt.

Daraus ergeben sich folgende **Vorteile**:

Die Transparenz und die Nachvollziehbarkeit werden verbessert. Weiters kann eine Änderung im Entwurf leicht in der Analyse nachgetragen werden und umgekehrt. [bahei99]

Die objektorientierte Programmierung (OO-Programmierung) stellt zum Teil neue **Konzepte** für die Entwicklung von Software-Systemen zur Verfügung. Zur Anwendung kommen die Konzepte Klasse, Objekt (siehe Kaptilel 3.3.3 Frames), Attribut, Operation, Botschaft, Vererbung, Polymorphismus und dynamisches Binden. [bahel99]
Als Beispiel sei hier lediglich die Klasse erwähnt. Die **Klasse** ist eine Abstraktion, die Gemeinsamkeiten von Objekten und Regeln zu ihrer Erzeugung beschreibt. Allein das Klassenkonzept bringt eine ganze Reihe von Vorteilen mit sich. Durch die Bildung von Kapseln kann eine Klasse leichter verstanden und geändert bzw. erweitert werden, ohne daß die anderen Klassen stark davon betroffen sind. Damit unterstützt Verkapselung die einfache Wiederverwendbarkeit und Erweiterbarkeit der entstandenen Systeme. [bahei99]
Für weitere Details sei auf [bahei99] und [bahel99] verwiesen.

Gegenüberstellung des Objekt-Begriffs in der KI- und der OO-Programmierung

Der Objektbegriff in der Wissensrepräsentation der KI ist grundlegend verschieden vom Objektbegriff der OO-Programmierung.
In der KI geht es um die Beschreibung der inneren Struktur von Objekten mit dem Ziel der Wissensdarstellung. In der OO-Programmierung hingegen geht es um die Modularisierung von Software mit dem Ziel der Wiederverwendung. Von der Beschreibung der inneren Struktur von Objekten wird dort gerade abstrahiert. Ein Objekt repräsentiert kein Wissen, sondern Funktionalität. Das Wissen, das nötig ist, um diese Funktionalität zu erbringen, wird gekapselt und verborgen. Ein Objekt wird nur über seine Funktion, also sein Verhalten definiert. In der Wissensrepräsentation wird ein Objekt über seine innere Struktur definiert. Funktionalität gibt es nicht, ausgenommen Get- und Set-Methoden. Grob gesagt ist es in der Wissensrepräsentation dann objektorientiert, wenn es nur Get- und Set-Methoden gibt und keine anderen Funktionen. In der OO-Programmierung hingegen ist es nur objektorientiert, wenn es keine Get- und Set-Methoden gibt, da diese die Kapselung durchbrechen.

5.3 Schichten-Architektur

Bei der Schichten-Architektur handelt es sich um ein Architektur-Muster. Dieses Muster unterteilt ein Programm in Schichten. Die Schichtenstruktur klärt die Systementwicklung durch Separierung von Teilen des Systems entsprechend gleicher Abstraktionslevel. Dadurch wird das System modular und transparent, was Verständnis und Weiterverwendung fördert.

5.3.1 Grundlagen

Schichten werden entsprechend ihrem Abstraktionsniveau erstellt und angeordnet. Die Komponenten innerhalb einer Schicht können beliebig aufeinander zugreifen. Zwischen den Schichten selbst gelten strengere Zugriffsregeln und die Schnittstellen sollen möglichst schmal sein. Weiters soll jede Schicht bzw. jede Komponente ihre eigenen Aufgaben allein durchführen und nicht Teile davon an Andere delegieren.

Nach [bahel99] ist eine **Schichten-Architektur sinnvoll, wenn:**

- die Dienstleistungen einer Schicht sich auf demselben Abstraktionsniveau befinden und
- die Schichten entsprechend ihrem Abstraktionsniveau geordnet sind, so daß eine Schicht nur die Dienstleistung der tieferen Schichten benötigt.

Wobei es folgende **Möglichkeiten** gibt:

- Schichten mit linearer Ordnung: Von Schichten mit höherem Abstraktionsniveau kann auf alle Schichten mit niedrigerem Abstraktionsniveau zugegriffen werden, aber nicht umgekehrt. Den Vorteilen der größeren Flexibilität und besserer Performance stehen die Nachteile von geringerer Wartbarkeit und Änderbarkeit gegenüber.
- Schichten mit strikter Ordnung: restriktiveres Modell, bei dem immer nur auf das nächstniedrigere Niveau zugegriffen werden darf.

Eine optimale Schichten-Architektur realisiert folgende **Entwurfsziele**:

- Wiederverwendbarkeit einzelner Schichten.
- Änderbarkeit und Wartbarkeit, solange die Schichtschnittstellen unverändert bleiben.
- Portabilität, da Hardwareabhängigkeiten in einer Schicht isoliert und modifiziert werden können. [bahei99]

5.3.2 Konventionelles Programm

Die klassischen Informationssysteme werden meist in einer Zwei-Schichten-Architektur entworfen. Sie besteht aus einer Anwendungsschicht, in der die Benutzeroberfläche und das Fachkonzept in einer einzigen Schicht fest verzahnt sind, und einer Datenhaltungsschicht.

Ein grundlegendes Prinzip im Entwurf von Software-Systemen besteht heute in der klaren Trennung zwischen Benutzeroberfläche und Fachkonzept. Die Fachkonzept-Klassen dürfen dabei nicht direkt mit dem System der grafischen Benutzeroberfläche (GUI-System) kommunizieren, sondern zwischen dem GUI-System und den Fachkonzept-Klassen befindet sich eine GUI-Schicht. Die **Drei-Schichten-Architektur** besteht dann aus: der GUI-Schicht, der Fachkonzeptschicht und der Datenhaltungsschicht. Die Fachkonzeptschicht entspricht dem funktionalen Kern der Anwendung und realisiert die fachlichen Anforderungen. Die GUI-Schicht realisiert die Benutzeroberfläche und die Dialogführung einer Anwendung. Sie baut die Benutzeroberfläche einer Anwendung auf, liest die Eingaben von den Benutzern und gibt die Ergebnisse aus. Dazu werden Objekte der Fachkonzept-Klassen erzeugt und manipuliert. Die Datenhaltungsschicht realisiert die jeweilige Form der Datenspeicherung.
Diese Trennung ermöglicht das Austauschen der Benutzeroberfläche, ohne daß das Fachkonzept geändert werden muß. [bahel99]
Wie oben erwähnt, erfüllt die GUI-Schicht zwei unterschiedliche Aufgaben. Das ist einerseits die Präsentation der Informationen und anderseits die Kommunikation mit der Fachkonzeptschicht. Entsprechend dieser Aufgaben kann eine separate Zugriffsschicht zur Fachkonzeptschicht gebildet werden. Die GUI-Schicht befaßt sich dann nur noch mit der Präsentation. Die Fachkonzept-Zugriffsschicht ist verantwortlich für alle Zugriffe auf die Fachkonzeptschicht. Die GUI-Schicht arbeitet normalerweise mit einer kleinen Menge von relativ einfachen Typen, während die Fachkonzeptschicht Typen beliebiger Komplexität besitzen kann. Die Zugriffsschicht paßt die Daten der Fachkonzeptschicht für die Präsentation durch die GUI-Schicht an. So verbirgt sie die komplexen Beziehungen der Fachkonzeptschicht vor der GUI-Schicht. Weiters kann eine separate Datenhaltungs-Zugriffsschicht implementiert werden, welche den Datenaustausch zwischen den Fachkonzept-Objekten und der Datenhaltungsschicht übernimmt. Man spricht in diesen Fällen von einer **Mehr-Schichten-Architektur**. [bahei99]

5.3.3 Programm mit integriertem wissensbasiertem System

Da sich ein konventionelles Programmsystem für die Umsetzung prozeduraler und generischer Programmaufgaben und sich ein wissensbasiertes Programmsystem für Heuristiken eignet, ist eine Kombination dieser beiden Programmsysteme sinnvoll. Dadurch können mehrere verschiedene Wissensformen des Anwendungsbereichs wie z.B.: bekannte Algorithmen, Frames und Heu-

ristiken zusammengeführt werden, um das beste Ergebnis zu erreichen. Das Resultat der Zusammenführung integriert verschiedene Arten von Programmsystemen zu einem kompletten System.

In der Struktur eines integrierten wissensbasierten Systems kann eine Schichtenorganisation basierend auf den Abstraktionslevel der zu bearbeiteten Information benutzt werden. Bei konventionellen Programmen sind die Schichten und Schnittstellen funktionalitätsbezogen, bei wissensbasierten Systemen hingegen eher informationsbezogen.
Die benachbarten Schichten kommunizieren über gut definierte Schnittstellen wie z.B.: Funktionen, Klassen- und Objektmethoden, Wissensbasen usw. [pri89]

Die Struktur eines integrierten wissensbasierten Systems nach [pri89]

Die Kernschicht des Systems beinhaltet das Problemlösungswissen für den Anwendungsbereich. Die äußeren Schichten verbinden den Kern stufenweise mit einer bestimmten Systemumgebung (siehe Abb.5.1). Der Kern manipuliert die Informationen, welche bedeutend für das Problemlösen sind, während die äußeren Schichten Daten und Informationen sammeln und für den Kern entwickeln.

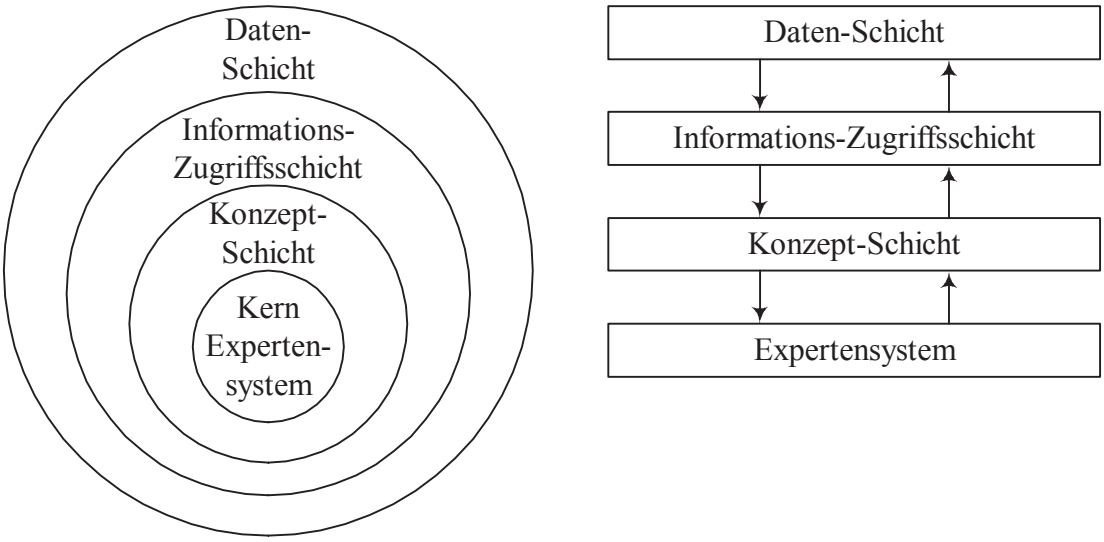


Abbildung 5.1: Schichtenarchitektur für integriertes wissensbasiertes System.

Daten-Schicht Das sind die physischen Datenspeicher, welche z.B. die auszuwertenden Daten enthalten.

Informations-Zugriffsschicht Die Informations-Zugriffsschicht umfaßt die funktionalen Programme, die als Informationsquellen für den zentralen Problemlösungsprozeß dienen.

Diese Schicht versorgt die Konzept-Schicht mit Informationen und steuert eventuell den Problemlösungsprozeß durch Botschaften.

Konzept-Schicht Sie stellt eine konzeptionelle Abstraktion der realen Welt zur Verfügung, welches für die Problemspezifikation und für den Problemlösungsprozeß benötigt wird. Sie beinhaltet Abstraktionen von konkreten Objekten, abstrakte Ideen, Ereignisse und Beziehungen der realen Welt. Die Abstraktionen basieren auf formalen und gut verstandenen Wissen über den Anwendungsbereich. Form und Umfang richten sich nach dem zu lösenden Problem und nach dem Bedarf der Expertensystem-Schicht, da deren Operationen auf die von der Konzept-Schicht bereitgestellten Informationen angewiesen ist.
Die Konzept-Schicht dient der Expertensystem-Schicht als Lieferant für die benötigten Informationen des Problems. Weiters werden die von der Expertensystem-Schicht produzierten Lösungen mit Hilfe der von der Konzept-Schicht definierten Konzepte repräsentiert.

Die **Vorteile** der expliziten Konzept-Schicht im Vergleich zu Lösungen, wo dieses Wissen in anderen Teilen des Systems verteilt und versteckt ist, sind:

- Flexibilität: Das explizite Modell kann auf verschiedene Arten, für verschiedene Probleme, mit verschiedenen Methoden verwendet werden.
- Erweiterbarkeit: stufenweise Ausbaubarkeit des Systems (des Modells und des Programms) ohne unerwartete Nebeneffekte.
- Integration: Das explizite Modell bietet eine konsistente Repräsentation der über die Anwendungen (Schichten) hinaus benutzten Informationen. [pri89]

Die Expertensystem-Schicht (Kern) Diese Schicht beinhaltet Wissensbasis und Inferenzmechanismus und damit den Kern des Problemlösens. Der Inferenzmechanismus führt den eigentlichen Problemlösungsprozeß durch Anwenden des Wissens der Wissensbasis aus.

Diese Systemstruktur unterstützt die Integration eines Expertensystems und damit eines wissensbasierten Problemlösens in andere konventionelle Programmsysteme. Es entsteht ein komplexes wissensbasiertes System, das andere Repräsentationen, Programme und Informationsspeicher benutzen kann, und somit die Vorteile aus mehreren verschiedenen Programmsystemen. [pri89]

6 Systemanalyse

In der Systemanalyse wird spezifiziert, was das System leisten soll. Ausgehend von Objekten der realen Welt wird durch Modellbildung und geeignete Abstraktion ein objektorientiertes Modell erstellt. Weiters wird die Architektur des Programms mit integriertem wissensbasiertem System festgelegt. Das daraus resultierende Modell bildet die fachliche Lösung des zu realisierenden Systems und besteht aus einem statischen und einem dynamischen Modell. Für Details bezüglich Systemanalyse sei auf [balhei99] verwiesen.

6.1 Akteure

Wie bereits bei der Aufgabenstellung erwähnt, werden am Institut für Automation die Programme von den Technik-Ingenieuren bzw. Technik-Studenten erstellt. Den Akteurrollen bei Expertensystemen (siehe Kapitel 4.2.2) entsprechend sind diese Techniker eine Mischung aus Fachexperte und Wissensingenieur mit mehr oder weniger umfangreichen Programmierfähigkeiten. (Nachfolgend werden solchermaßen befähigte Techniker als Fachexperte-Informatiker benannt).

6.1.1 Szenarios bei konventioneller Programmentwicklung

Der *Fachexperte-Informatiker* entwickelt das Programm, wobei oft zu Beginn das Konzept unklar ist bzw. fehlt oder im Laufe der Programmierung abhanden kommt. Eine weitere Schwierigkeit ist die Strukturierung des Programms, und die Aufrechterhaltung dieser Struktur während der Programmierung. Für diesen Fachexperten-Informatiker ist die Entwicklung eines wissensbasierten Programmsystems unzumutbar, da dieses Wissensgebiet selbst für Informatiker ein Randgebiet darstellt.
Aufgrund dieser Gegebenheiten ist es bereits für den Programmautor selbst oft schwer, das mit der Zeit gewachsene Programm komplett zu durchblicken und dann weiterzuverwenden.

Noch schwieriger wird es für *andere Fachexperten-Informatiker*, da bereits das Anwenden des

Programms problematisch und das entstandene Programmergebnis aufgrund mangelnder Transparenz nur bedingt vertrauenswürdig ist. Die Weiterverwendung des Programms und einzelner Programmteile ist nur mit sehr viel Aufwand möglich, weshalb häufig eine neue Programmentwicklung vorgezogen wird. Auch die Wissensweiterverwendung gestaltet sich schwierig, da das Wissen verzerrt und über das Programm verteilt ist.

Für einen einfachen *Anwender* ist die Benutzung eines solchen Programms und dessen Ergebnisse oft überhaupt nicht nachvollziehbar, und eine Weiterverwendung unmöglich.

6.1.2 Szenarios bei wissensbasierter Programmentwicklung

Für einen **neuen Anwendungsbereich** entwickeln *ein Wissensingenieur und ein Fachexperte* gemeinsam das Programm. Durch die Einbindung des Wissensingenieurs entsteht ein gut konzeptioniertes und strukturiertes Programmsystem. Dieses ist zwar in der Gesamtheit komplexer, aber die einzelnen Programmteile und Komponenten sind übersichtlicher und verständlicher, und somit nachvollziehbar und weiterverwendbar. Weiters ist aufgrund der Wissensbasierung ein Wissensmanagement möglich.

Für andere **Problemstellungen aus dem gleichen Anwendungsbereich bzw. bei Problemen mit ähnlichen Konzepten** kann *ein Fachexperte-Informatiker* das bereits entwickelte Programmsystem weiterverwenden, ohne einen Wissensingenieur einzuschalten.

Aufgrund der Modularisierung und Strukturierung ist dieses Programmsystem auch für **Andere** anwendbar und nachvollziehbar, und einzelne Programmteile können in anderen Projekten weiterverwendet werden. Durch die explizite Wissensdarstellung kann nun auch das Expertenwissen weiterverwendet oder geändert werden.

Auch ein einfacher *Anwender* kann solch ein Programm anwenden und eventuell durch kleine Änderungen z.B. in der Wissensbasis an seine Bedürfnisse anpassen.

Fachexperten-Informatiker und im kleineren Rahmen auch Anwender können weiters die Programmsteuerung ändern und somit den Programmablauf variieren.

6.2 System-Stufen

Aus der Beschreibung der Bohrauswertung in Kapitel 1 ergeben sich folgende System-Stufen:

Sensoren: Nur die auswertungsrelevanten Sensoren werden ausgewählt und später in die Sensor-Kanäle eingelesen, die restlichen Sensoren werden verworfen.
Hier findet die erste Datenreduzierung durch Selektieren der relevanten Sensoren statt.

Sensor-Kanäle: Neben dem Sensorwert (val) an sich sind auch Werte hinsichtlich des Zeitverhaltens (min, max, avg, slp) interessant. Diese Werte sind die sogenannten Basis- bzw. Sensor-Fakten. Für die einzelnen Sensoren wird jeweils eine unterschiedliche Auswahl an Fakten benötigt.
Dadurch wird die Datenmenge temporär erhöht, um die Möglichkeiten bei der nachfolgenden Auswertung zu erhöhen.

Ein-Sensor-Auswertungen: aus einzelnen Sensor-Fakten abgeleitete Aussagen.

Multi-Sensor-Auswertungen: komplexe Aussagen durch Verknüpfung von mehreren Ein--Sensor-Aussagen. Dies können verschiedene Ein-Sensor-Aussagen eines Sensors für qualifiziertere Aussagen, aber auch mehrere Sensoren für komplexere Aussagen sein.
Hier findet die letztendliche Datenverdichtung zu aussagekräftigen komplexen Fakten statt.

Report: beinhaltet die aus den komplexen Aussagen resultierenden komplexen Fakten.

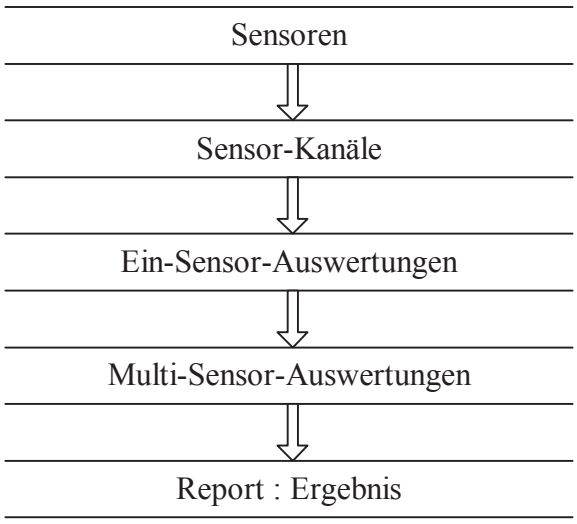


Abbildung 6.1: System-Stufen.

6.3 Repräsentation und Programmsystem

Die Wahl der optimalen Repräsentation bzw. des optimalen Programmsystems, welches das Problem am klarsten und einfachsten darstellt und die effiziente Ausführung ermöglicht, ist ein entscheidender Schritt im Lösungsprozeß.

Die Analyse der Bohrauswertung zeigt, das es viele Objekte bestimmter Typen wie Sensor-Kanal, Ein-Sensor-Auswertung und Multi-Sensor-Auswertung gibt. Weiters werden auch generische Berechnungen für das Zeitverhalten benötigt. Dieses **Wissen über generische Problemsituationen** ist von permanenter Natur. Veränderbare Informationen über den aktuellen Problemfall (individuelle Probleme) können dabei durch Instanzen dargestellt werden.
Diese generischen Teile der Anwendung werden am besten mit einem **konventionellen objektorientierten Programmsystem** gelöst, da z.B. Regeln für die Darstellung von strukturierten Objekten zu flach sind und die direkte Einbindung von prozeduralen Code nicht unterstützen.

Bei den *Versuch, wissensbasierte Systeme rein objektorientiert zu implementieren*, treten aufgrund der widersprüchlichen Konzepte Schwierigkeiten auf.
Bei einem wissensbasiertem System sind die Basisdaten wie z.B. die Objekte Multi-Sensor-Auswertungen und die Problemlösung wie z.B. das Interpretationswissen sauber voneinander getrennt. Dieses Interpretationswissen beschreibt komplizierte Abhängigkeiten innerhalb des Anwendungsbereichs. Es beinhaltet Größenvergleiche mit Basisfakten, um Ein-Sensor-Auswertungen zu erhalten und komplexe Größen- und Mustervergleiche von diesen Ein-Sensor-Auswertungen, um Multi-Sensor-Auswertungen zu erhalten.
Bei einem objektorientierten System ist aufgrund dessen Konzept der Methoden *die Problemlösung eng an die behandelten Daten gekoppelt und auf diese dezentral verteilt*. Weiters sind die Möglichkeiten zur Problemlösung durch die Gegebenheiten des objektorientierten Systems beschränkt oder nur sehr schwer umsetzbar. Z.B. sind die Größenvergleiche nur mittels Parameterwerte möglich, wobei deren Art und Anzahl vom Objekt und dessen Parameterliste abhängt.
Ein Multi-Sensor-Auswertungsobjekt hängt aber von einer variablen Anzahl von Größenvergleichen von Ein-Sensor-Auswertungsobjekten bzw. von Mustervergleichen verschiedener Ein-Sensor-Auswertungsobjekten untereinander ab. Diese variable Abhängigkeit kann nur mit hohem Programmieraufwand realisiert werden, oder es ist nur eine bestimmte maximale Anzahl von Größenvergleichen erlaubt. Eine flexiblere Lösung könnte mit dem Event-Listener-Konzept erreicht werden, jedoch würde dabei das Programm zu komplex, dezentral und langsam werden.

Nach [pri89] können solche Abhängigkeiten nicht mit Beziehungen oder Algorithmen beschrieben werden, sondern nur mit anderen Repräsentationstechniken wie Regeln.

Diese Schwierigkeiten können somit durch den Einsatz eines **Expertensystems** gelöst werden. Wie im Theorieteil beschrieben, ermöglicht solch ein Expertensystem aufgrund seiner Architektur die *Trennung von Steuerung und Wissen*. Dadurch kann das *Interpretationswissen zentral und dennoch modular* in der Wissensbasis gespeichert werden und die Inferenzkomponente erledigt dessen Anwendung.
Die generischen Objekte aus dem objektorientierten Teil, die in der Wissensbasis instantiiert werden, enthalten nur Attribute mit Basisdaten und deren Zugriffsroutinen. Das Interpretationswissen zur Problemlösung wird in Regeln realisiert, die keine Basisdaten, sondern nur Konstanten, Variablen, Vergleiche und Methodenaufrufe enthalten.

Durch die Kombination eines wissensbasierten und eines konventionellen objektorientierten Programmsystems entsteht ein **mehrteiliges hybrides Programmsystem**, welches die Vorteile von Beiden in sich vereint.

Vorteile der Objektorientierung

Sie stellt eine natürliche Repräsentation von stereotypischen Objekten, deren Verhaltensweisen und Interaktionen mit anderen Objekten dar. Möglichkeiten wie Klassenbildung, Vererbung, Prozeduren bzw. Methoden usw. ergeben die Mächtigkeit und Verständlichkeit dieses Programmsystems für generische Situationen.
Es unterstützt die Organisation großer Mengen von strukturiertem zusammenhängendem Wissen in komplexe Einheiten, und ist somit in der Lage, die impliziten Verknüpfungen zwischen Informationen eines Anwendungsbereichs explizit durch Datenstrukturen darzustellen. Dies bildet die Grundlage für die Anwendung von musterorientierten Regeln.

Vorteile der Wissensbasierung

Durch die modulare Darstellung von Heuristiken im Wissensgehalt der Regeln wird eine natürliche Repräsentation dieses Wissens erzielt. Sie ermöglichen eine variable und mächtige musterorientierte Steuerung. Darüberhinaus verfügt sie mit der Suchstrategie und der Konfliktlösungsstrategie neben den Regeln über andere Möglichkeiten zur heuristischen Steuerung.
Die Trennung von Wissen und Steuerung auf unterer Ebene ermöglicht das Durchführen von Änderungen an der Wissensbasis oder an der Programmsteuerung, ohne daß dadurch jeweils andere Programmkomponenten beeinflusst werden. Diese Trennung ermöglicht weiters die Wiederverwendung der einzelnen Komponenten.
Aufgrund der Modularität der Produktionsregeln ist die inkrementelle Entwicklung von Expertensystemen möglich, bei der das Wissen nach und nach ergänzt, entfernt und geändert wird. Die Modularität der Regeln und die Art ihrer Anwendung erleichtern es zudem, die Arbeitsweise des Expertensystems zu verfolgen und nachzuvollziehen.

In *hybriden Systemen* wird oft der regelbasierte Teil der Problemlösung in einer anderen Program-

miersprache implementiert als der objektorientierte und der logische Teil.

Entsprechend des Anforderungskatalogs wird das objektorientierte Programmsystem in **Java** realisiert.
Die Entscheidung bezüglich des Expertensystems fiel auf **Jess** (siehe Kap. 7.1), da es in Java programmiert ist, und sich somit leichter in ein Java-Programm einbetten läßt. Weiters basiert Jess auf dem bekannten Clips, das sich laut [met91] für Aufgabenstellungen wie diese gut eignet. Außerdem ist Jess für Studien- und Forschungszwecke kostenlos verfügbar.

Um *Transparenz und Modularität des komplexen hybriden Programmsystems* zu gewährleisten, wird das **Schichten-Entwurfsprinzip** bei der Bildung der Systemarchitektur angewandt.

6.4 System-Schichten

Die System-Stufen werden nun entsprechend dem gewählten Lösungsansatz und der damit verbundenen Repräsentationsform und Programmsystemwahl weiterentwickelt.
Dies geschieht nach dem Schichten-Entwurfsprinzip, wobei sich entsprechend dem Abstraktionsniveau der Programmsystemkomponenten bzw. deren Informationen dann folgende System-Schichten ergeben:

- **Sensoren:** entsprechen einer Daten-Schicht, welche die physischen Meßwerte enthält.
- **Sensoren-Kanäle:** entsprechen einer Informations-Zugriffsschicht, welche aus den Daten Basis-Fakten erstellt.
- **Sensoren-Fakten:** stellen eine Zugriffsschicht dar, welche die Darstellung der Basis-Fakten übernimmt.
- **Einfache Aussagen:** werden durch Tests der Sensor-Fakten in den Regeln realisiert.
- **Komplexe Aussagen:** ergeben sich durch Verknüpfung mehrerer einfacher Aussagen.
- **Komplexe Fakten:** sind eine weitere Zugriffsschicht, welche die komplexen Aussagen in Objekte aufnimmt.
- **Report:** ist das Auswertungsergebnis in Form einer physischen Datei.

Beschreibung der einzelnen Schichten

Sensoren Das sind die physischen Daten der Bohrung. Diese bestehen aus einer Vielzahl von Meßwertreihen, wobei jede Reihe eine Zeitangabe und einen Meßwert pro Sensor besitzt.
Die relevanten Sensoren werden ausgewählt und nachfolgend in Sensor-Kanal-Objekte eingelesen.

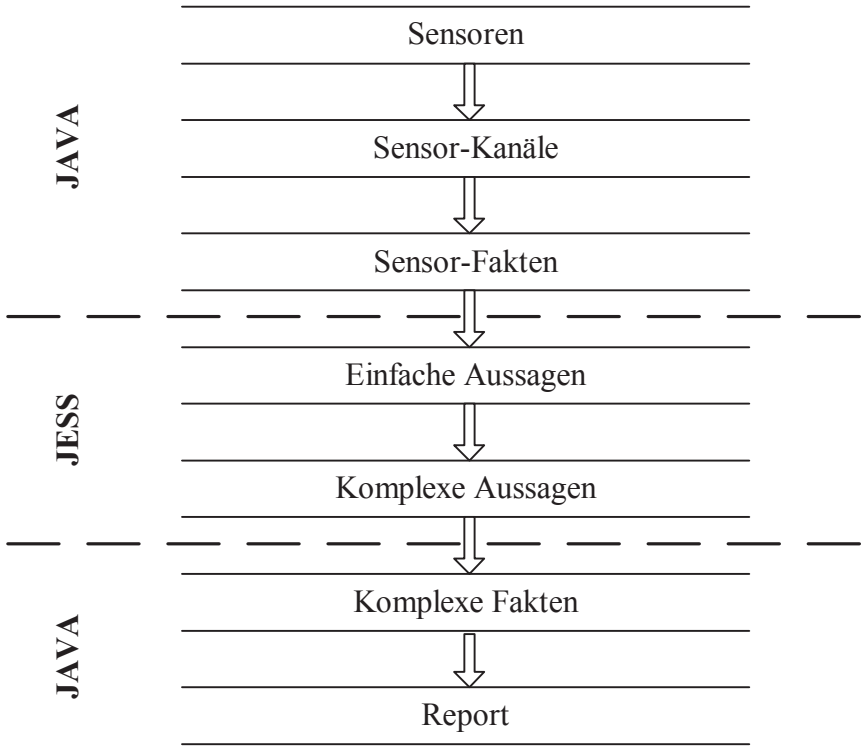


Abbildung 6.2: System-Schichten, Analyse.

Sensor-Kanal Sie nehmen die physischen Daten zur Bearbeitung und Berechnung für die nachfolgende Auswertung auf.

Jeder Sensor-Kanal besitzt einen Puffer, in dem die Werte einer bestimmten Anzahl vorangegangener Meßpunkte zwischengespeichert werden, um neben dem aktuellen Wert auch die zeitliche Entwicklung berechnen zu können. Diese Basis-Fakten werden vom Sensor-Kanal berechnet und dann dem entsprechenden Sensor-Fakt zugewiesen. Das Programm soll so gestaltet sein, daß nur die pro Sensor jeweils benötigen Sensor-Fakten angelegt werden.

Diese Objekte eignen sich nicht für die Darstellung der variablen Auswahl an Fakten.

Sensor-Fakt Diese separate Zugriffsschicht übernimmt die Repräsentation der Information und die Kommunikation mit Jess anstatt diese Aufgaben ebenfalls in der Sensor-Kanal-Schicht mit umzusetzen [1].

Diese Vermittlungsschicht kann mit einer kleinen Menge von relativ einfachen Typen arbeiten, während die darüberliegende Sensor-Kanal-Schicht eine höhere Komplexität aufweist. Die separate Zugriffsschicht ergibt Vorteile bezüglich Transparenz, Flexibilität und der Kommunikation zwischen Java und Jess.

[1] entsprechend Kapitel 5.3.2 Schichtenarchitektur bei konventionellen Programmen

Somit werden die generischen und prozeduralen Aufgaben: Importierung der Daten, deren Pufferung und Berechnung im konventionellen Java mittels Objekten und Methoden realisiert.

Einfache Aussagen Durch Testen der Sensor-Fakten im Bedingungsteil der Regeln werden einfache Aussagen über einzelne Sensoren getroffen. Neben einfacher Vergleiche der Fakten mit Werten und Fakten miteinander sind auch andere boolesche Funktionen möglich. Dafür sind zum Teil spezielle Informationen wie z.B. Grenzwerte nötig, welche der Wissensbasis hinzugefügt werden.

Komplexe Aussagen Durch Verknüpfung (and/or) mehrerer einfacher Aussagen im Bedingungsteil der Regeln werden komplexe Aussagen bezüglich mehrerer Sensor-Fakten getroffen.

Die eigentliche Interpretation erfolgt in Jess, wodurch eine zentrale Wissensdarstellung in Form von modularen Regeln und eine variable und mächtige Auswertung ermöglicht wird.

Komplexe Fakten Diese Schicht ist wiederum eine Zugriffsschicht ähnlich der Sensor-Fakten-Schicht und stellt die komplexen Aussagen dar.
Weiters übernimmt sie die Ausgabe in den Report. Das ist bei dieser Aufgabenstellung möglich, da die Ausgabe entsprechend einfacher und klarer Regeln erfolgen kann. Die Ausgabe mittels Methoden wird durch Messages aus dem Aktionsteil der Regeln nach deren Aktivierung und Feuerung aufgerufen.
Indem Objekte gesetzt werden, anstatt die Ausgabe gleich aus den Regeln heraus zu erledigen, ergeben sich weitere Möglichkeiten der Bearbeitung und auch der Auswertung (z.B. mehrstufig).

So ist auch diese generische Aufgabe in Java gelöst und hält den Jess-Teil frei von unnötigen und komplizierenden Ballast.

Report Das Ergebnis des Bohrdatenauswertung wird in Form einer physischen XML-Datei abgespeichert. Für jede in Jess erfolgte Interpretation wird die Zeitangabe und die Interpretation an sich ausgegeben.

6.5 Statisches Modell

Das statische Modell beschreibt die Klassen des Systems, die Assoziationen zwischen den Klassen und die Attribute des Systems.

Für die Verständlichkeit und Wiederverwendbarkeit des Programmsystems ist die Trennung von generischen Wissen und problemspezifischen Wissen (hier Interpretationswissen) sehr bedeutend.

Rules: Das Auswertungswissen wird getrennt in Jess entsprechend wissensbasiertem Konzepten in Form von Regeln im Programm Rules realisiert.

Gleiches trifft für das Wissen bezüglich der Sensorenselektion und der jeweiligen Auswahl an benötigten Sensor-Fakten zu. Deswegen werden auch diese Aufgaben aus dem Java-Teil herausgezogen und in einem eigenen Jess-Programm, Konfig, zusammengefaßt.

Konfig: Dieses Konfig enthält die Deklarationen der Sensor-Kanäle und damit indirekt die Deklarationen der abgeleiteten Sensor-Fakten, und weiters die Deklarationen der komplexen-Fakten.

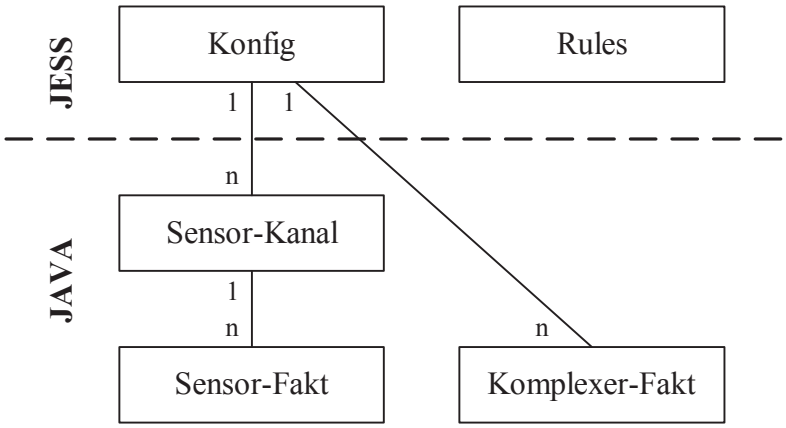


Abbildung 6.3: Klassendiagramm, Analyse.

Dadurch enthält der Java-Teil kein rein problemspezifisches Wissen und kann ohne Änderung für ähnliche Auswertungsaufgaben übernommen werden.

6.6 Dynamisches Modell

Das dynamische Modell beschreibt das Verhalten des zu entwickelnden Systems.

Eine Besonderheit dieses Programmsystems ist, daß hier Interaktionen zwischen den Programmsystemen Java und Jess im Vordergrund stehen, statt Interaktionen zwischen Benutzer und Programm.

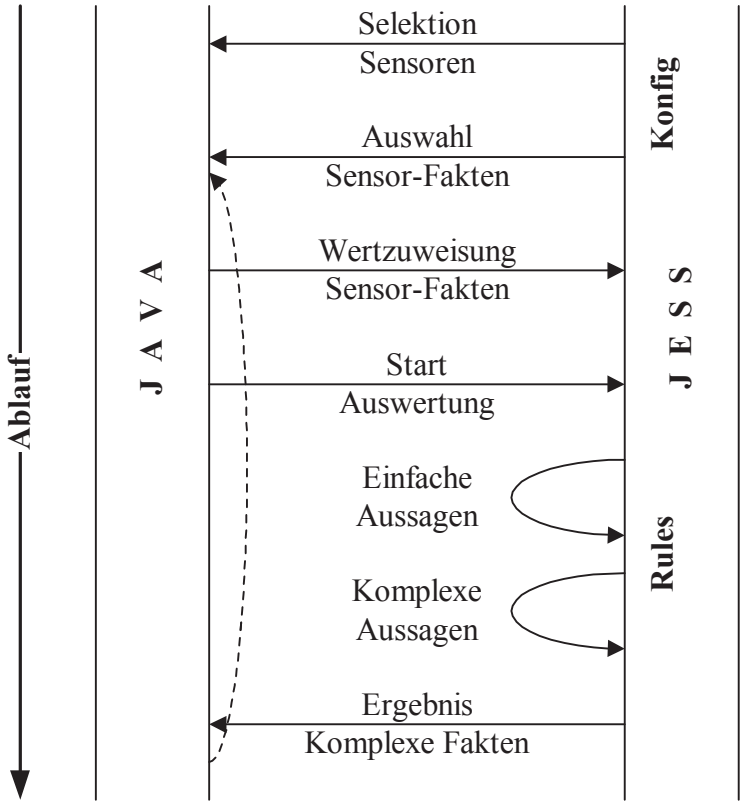


Abbildung 6.4: Sequenzdiagramm, Analyse.

In Abb. 6.4 ist vereinfacht das Zusammenspiel von Jess und Java dargestellt. Sie zeigt den ersten Durchlauf mit der Initialisierung und einer Auswertung. Weitere Auswertungsdurchläufe folgen entsprechend dem gestrichelten Pfeil.

Dieser Entwurf wurde auch vom Jess-Entwicklungsstand bestimmt. Da noch kein vollständiges Implementieren von Jess innerhalb eines Java-Programms möglich ist, wurde eine derart doppelt hybride Lösung angestrebt; hybrid in der Darstellung (Objekt und Regel) und hybrid bezüglich unterschiedlicher Programmsysteme (Java und Jess).

7 Systementwurf

Beim Entwurf geht es darum, die in der Systemanalyse spezifizierte Anwendung auf einer Plattform unter den geforderten technischen Randbedingungen zu realisieren. Dabei wird das Analyse-Modell unter den Gesichtspunkten Effizienz und Wiederverwendung überarbeitet. Es entsteht ein objektorientiertes Entwurfsmodell. Für Details bezüglich Systementwurf sei auf [balhei99]verwiesen.

7.1 Regelorientierter Jess-Teil

Hier wird nur auf relevante Punkte für den Entwurf und für die Implementierung eingegangen. Ergänzende Aspekte und andere Lösungsmöglichkeiten werden in nachfolgenden Kapiteln behandelt, für Details sei auf [fri01] verwiesen.

Grundlagen

Jess ist eine in Java geschriebene Programmbibliothek, welche als Interpreter für die Jess Sprache dient. Diese Sprache ist der Clips-Expertensystem-Sprache sehr ähnlich, welche ihrerseits eine hoch spezialisierte Form von Lisp ist.

Das regelbasierte System Jess führt eine Sammlung von Wissen in Form von Fakten, die **Wissensbasis**. Dabei müssen die Fakten eine gewisse spezifische Struktur aufweisen.
Jess verfügt über nur zwei **Konfliktlösungsstrategien**: depth (default) and breath. Bei der depth-Strategie feuert die kürzlichst aktivierte Regel. Bei der breadth-Strategie feuern die Regeln entsprechend der Reihenfolge ihrer Aktivierung.
Jess's zentrales Konzept ist die Vorwärtsverkettung. Darüberhinaus wird eine Art der Rückwärtsverkettung ermöglicht. [fri01]

Regeln

Eine Regel setzt sich aus einem Bedingungs- und einem Aktionsteil zusammen, welche durch das Symbol „=>“ getrennt sind.
Wird ein konventionelles Wenn-Dann einer prozeduralen Sprache zu einer bestimmten Zeit und in einer bestimmten Reihenfolge ausgeführt, so wird eine Regel hingegen immer dann exekutiert, wenn ihre Bedingungen neu zutreffen und der Inferenzmechanismus läuft (run).
Der Bedingungsteil besteht aus Mustern, welche zum Abgleich mit den Fakten der Wissensbasis benutzt werden. Eine Regel wird aktiviert, wenn die Muster in der Wissensbasis auftreten. Aktivierung bedeutet die Registrierung einer Regel, deren Bedingungen allesamt erfüllt sind. Jess-Regeln feuern bzw. exekutieren nur, während der Inferenzmechanismus läuft (jedoch können sie auch aktiviert werden, wenn er nicht läuft). Wenn die Regel feuert, werden die Aktionen im Aktionsteil ausgeführt.

Funktion **run** [integer]: startet den Inferenzmechanismus, wodurch die aktivierten Regeln feuern. Ist kein Argument bei der run-Funktion angegeben, dann stoppt der Inferenzmechanismus erst, wenn keine weiteren aktivierten Regeln zum Feuern vorhanden sind, oder wenn der Befehl halt aufgerufen wurde.
Ist hingegen ein Argument, welches die maximale Anzahl an Feuerungen bestimmt, angegeben, dann stoppt der Inferenzmechanismus spätestens nach dieser Anzahl an Feuerungen.

Eine Regel wird nur einmal von einem Satz Fakten aktiviert, wenn sie dann gefeuert hat, wird sie für diese Fakten nicht mehr feuern. Erst durch das Zurücksetzen der Fakten mittels des Befehls reset und das Neusetzen kann die Regel erneut feuern. [fri01]

Muster

Muster können Wildcards, verschiedenste Prädikate und Variable (beginnen mit ?: ?variable) enthalten. Eine Variable vertritt jeden möglichen Wert dieser Position in der Regel. Die im Bedingungsteil passenden Variablen sind dann im Aktionsteil derselben Regel verfügbar.
Weiters können diese Variablen getestet werden, um ihre Selektivität zu spezifizieren. Mittels dem Befehl „test“ können andere Mustervergleiche mit booleschen Funktionen und nicht nur mit der Wissensbasis realisiert werden. Dadurch sind z.B. Größenvergleiche von Variablen und Konstanten (defglobal: ?*defglobal*) möglich.

Zusammenspiel von Java und Jess

Jess kann direkt auf alle **Java**-Klassen und Bibliotheken zugreifen. So kann fast alles, was in Java-Code möglich ist, mit Jess realisiert werden. Aber z.B. ist das Definieren neuer Java-Klassen in Jess nicht möglich.

Durch das unterschiedliche Programmsystemkonzept von Jess mit Wissensbasis und Inferenzmechanismus ist die Realisierung grundlegend verschieden. Generische Aufgaben, die in Java einfach zu erledigen sind, verursachen in Jess sehr komplexe Konstrukte.

Das Erstellen und Manipulieren von Java-Objekten direkt von Jess aus ist mittels Java Reflection möglich. Jess besitzt einen Mechanismus zum automatischen Generieren eines Templates (Klassen in Jess) für die Repräsentation einer Java-Klasse. Dieses Template wird dann dazu benutzt, eine Repräsentation eines Objekts der Java-Klasse in Form eines Fakts in der Wissensbasis anzulegen. Mittels dem Befehl **defclass** wird ein solches Template generiert. Der Befehl **definstance** erzeugt dann eine Repräsentation eines spezifischen Java-Objekts in der Wissensbasis.

Die Repräsentation in der Wissensbasis kann **statisch** (ändert sich unregelmäßig, ähnlich eines Schnappschusses der Eigenschaften zu einem Zeitpunkt) oder **dynamisch** (ändert sich automatisch wenn immer sich die Objekteigenschaften ändern) sein, und wird beim Befehl definstance angegeben.
Bei der statischen Repräsentation können die definstance-Fakten in der Wissensbasis nur durch den Befehl reset entsprechend der Java-Objekte aktualisiert werden. [fri01]

Befehl **reset**: entfernt zuerst alle Fakten und Aktivierungen aus der Wissensbasis, setzt dann die initialen und definierten Fakten und setzt weiters die einzelnen registrierten definstance-Fakten.

Das Expertensystem stellt eine separate Schicht dar, welche über seine Wissensbasis mit der Konzept-Schicht kommuniziert. Dabei werden mittels des Mechanismus defclass und definstance Abbilder der Konzept-Schicht-Klassen und Objekte in der Wissensbasis erzeugt. Das Expertensystem arbeitet dann mit diesen Abbildern, wobei Änderungen eines Abbildes abhängig von dessen Art (statisch oder dynamisch) unterschiedliche Folgen ergibt.

Bei dem Entwurf eines Programmsystems, das Java und Jess verwendet, gibt es nach [fri01] verschiedene Architekturmöglichkeiten:

1. reines Jess-Programm, das auch das main() liefert, und kein Java benutzt;
2. Jess-Programm, welches Java-Bibliotheken und in Java geschriebene Jess-Befehle verwendet;
3. halb Jess- und Halb Java-Programm, wobei Jess noch das main() bereitstellt;
4. halb Jess- und Halb Java-Programm, wobei nun Java das main() bereitstellt;
5. Java-Programm, welches die Jess-Programme zur Laufzeit lädt;
6. reines Java-Programm, welches Jess komplett mittels dessen Java-Bibliotheken manipuliert. Diese Option wird jedoch noch nicht von Jess unterstützt.

Hier wird die 5. Möglichkeit gewählt, da hierbei der Großteil des Programms in Java realisiert werden kann. Nur die Spezialaufgaben werden in Jess umgesetzt, wodurch der Jess-Teil klein und lesbar bleibt.

Aufgrund dieser Realisierungsvariante wird ein Java main() benötigt, das Erzeugung, Steuerung und Laden der Jess-Komponenten übernimmt.

7.2 Objektorientierter Java-Teil

Bezüglich Informatik und Java sei auf [bahel99], [hoco01], [hoco02] und [ste02] verwiesen.

7.2.1 Statisches Konzept

FileReader

Der FileReader enthält das Java main(), in dem das Jess-Expertensystem erzeugt, gesteuert und die Jess-Komponenten eingelesen werden.
Weiters werden hier mittels eines XML-Parsers die Sensor-Daten aus der XML-Datei eingelesen und dann an die entsprechenden Sensor-Kanäle weiterverteilt.

Somit entspricht der FileReader der Informations-Zugriffsschicht und enthält mit dem Java main() und dem XML-Parser zentral die steuernden Mechanismen des Programmsystems.

ContextObject

Entsprechend der Gegebenheiten von Jess und unter Berücksichtigung der Schichtenarchitektur werden Basis-Fakten und komplexe Fakten mittels der gleichen Klasse ContextObject dargestellt. Das Attribut level dieser Klasse spezifiziert das ContextObject, ob es sich um ein Sensor-Fakt (level=0) oder einen komplexen Fakt (level>0) handelt. Damit können einzelne Levels selektiv angesprochen werden, was das gezielte Zurücksetzen einzelner Levels und komplexe Steuerungen ermöglicht.

Vorteile:

- Optimierung der Schichtenarchitektur (Schichtenstruktur zwiebelförmig mit gleicher Input- und Output-Schicht für das Expertensystem => mehrstufige Auswertung möglich);
- Vereinfachung der generischen Handhabung (Aktualisieren und Zurücksetzen).

Für jeden benötigten Fakt wird dann ein definstance-Objekt in der Wissensbasis angelegt und aktualisiert. Hier wurde die statische Repräsentation gewählt.

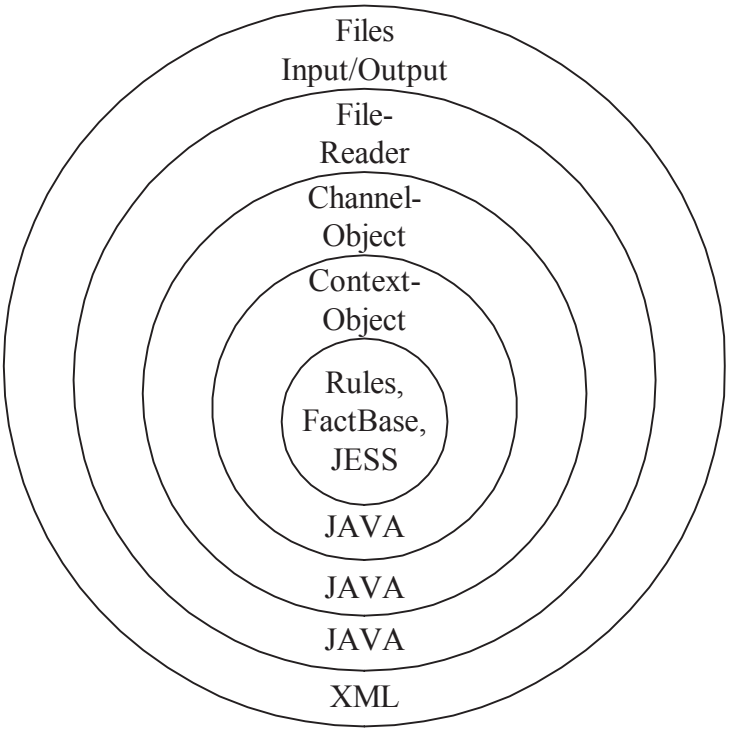


Abbildung 7.1: System-Schichten, Entwurf.

Container

Da der generische Teil variabel bezüglich der Problemstellungen sein soll (variable ChannelObjects und ContextObjects) und Jess das Zurücksetzen und Aktualisieren der Fakten zur erneuten Regelauslösung verlangt, benötigen die Sensor-Kanäle (ChannelObjects) und die Fakten (ContextObjects) eine Objektverwaltung.

Diese Objektverwaltung wird entsprechend dem Container-Entwurfsmuster realisiert. Die jeweils verwalteten Objekte werden dabei unter ihrem Namensattribut als Suchschlüssel in einer HashMap abgespeichert.

Beschreibung der einzelnen Klassen

FileReader:

Beinhaltet mit dem Java main() und dem XML-Parser die zentrale Ablaufsteuerung des Programmsystems.
Dies ergab sich auch aus der Wahl des SAX-Parsers zum Einlesen der XML-Input-Datei. Der SAX-Parser ist für diese Anwendung, die nur das sequentielle Einlesen ohne Traversierungen und ohne

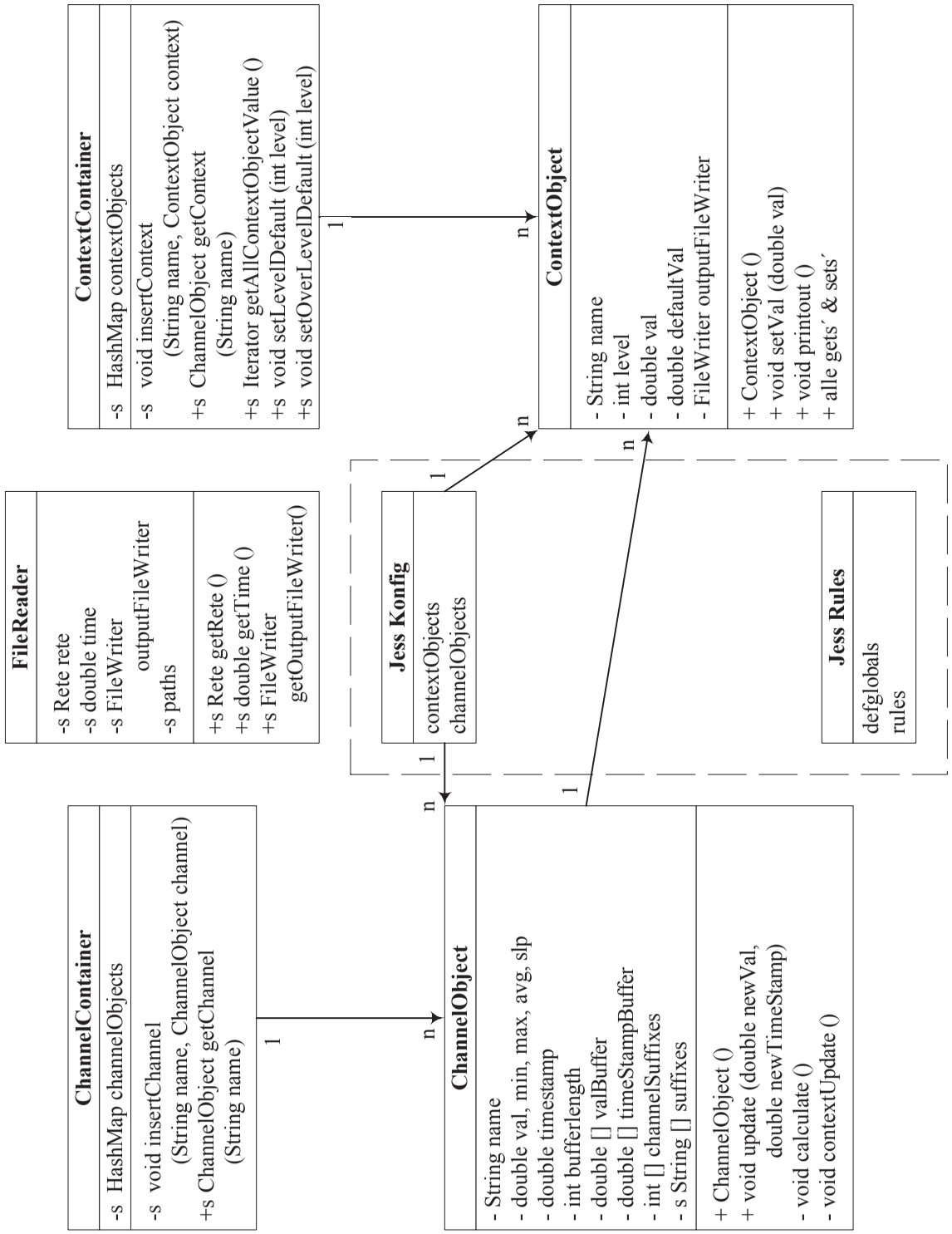


Abbildung 7.2: Klassendiagramm, Entwurf.

Änderungen benötigt, die treffende Entscheidung. Weiters ist er schnell und weist einen geringen Speicherbedarf auf.

FileReader
- s Rete rete : Dieses Objekt ist die Jess Rule-Engine - s double time : Zeitangabe der aktuellen Meßwertreihe - s FileWriter outputFileWriter : Handle des XML-OutputFile - s paths´ : Angabe der XML-File- und Jess-File-Pfade
+ s Rete getRete () + s double getTime () + s FileWriter getOutputFileWriter ()

Im Java main() wird das Jess-Expertensystem rete, die eigentliche Rule-Engine, mittels dessen Java-Bibliotheken erstellt und dann die Jess-Komponenten Konfig und Rules eingelesen. Dann wird der SAX-Parser erzeugt, initialisiert und gestartet, wodurch die Sensor-Daten eingelesen und auf die ChannelObjects weiterverteilt werden. Das Verteilen geschieht dynamisch über die Namen. Aus diesem Grund muß folgende Namenskonvention eingehalten werden. Das Namensattribut des ChannelObjects in Konfig muß ident sein mit XML-Tag des Meßwerts. Nach Einlesen des kompletten Datensatzes einer Zeiteinheit wird der Inferenzmechanismus der Rule-Engine gestartet. Nach Beendigung des Inferenzmechanismus werden die ContextObjects in den Defaultzustand zurückgesetzt (für den nächsten Durchlauf). Danach wird der nächste Datensatz geparst und wieder mittels der gleichen Rule-Engine abgearbeitet, bis das XML-Input-Fileende erreicht wird. Weiters wird auch das XML-Output-File erzeugt und initialisiert.

ChannelObject: Realisierung der Sensor-Kanäle.

Sie verarbeiten die eingelesenen Daten zu Sensor-Fakten und aktualisieren dann die entsprechenden ContextObjects mittels Aufruf der ContextObject-Methode setVal().
Bei der Erzeugung eines ContextObjects mittels Aufruf des Konstruktors erfolgt der Eintrag im ContextContainer.

Bei Deklaration der benötigten ChannelObjects in Konfig können die jeweils benötigten Sensor-Fakten ausgewählt werden, wodurch nur die auswertungsrelevanten Sensor-Fakten bzw. Context-Objects angelegt werden.
Entsprechend der Auswahl werden dann die vom ChannelObject jeweils abhängigen ContextObjects dynamisch mittels Java Reflection aktualisiert.

Sensor-Fakten bzw. ContextObjects:

val: aktuell eingelesener Meßwert dieses ChannelObjects.
min: kleinster Meßwert im Puffer valBuffer.

ChannelObject
- String name : Name des ChannelObjects und des Meßwerts im XML-InputFile - double val : Von XML-InputFile über FileReader eingelesene aktuelle Meßwert - double min, max, avg, slp : abgeleitete Werte des Zeitverhaltens des ChannelObjects - double timestamp : Zeitangabe der aktuellen Meßreihe - int bufferlength : Länge der Puffer - double [] valBuffer : Puffer der letzten Meßwerte zur Berechnung des Zeitverhaltens - double [] timeStampBuffer : Puffer der Zeitangaben - int [] channelSuffixes : Auswahl an zugehörigen ContextObjects des ChannelObjects - s String [] suffixes : Umsetzungstabelle für die ContextObject-Auswahl
+ ChannelObject () : Konstruktor, der ChannelObject erzeugt und in ChannelContainer einträgt und weiters die zugehörige ContextObject-Auswahl erzeugt + void update (double newVal, double newTimeStamp) : aktualisiert das ChannelObject und die zugehörigen ContextObjects - void calculate () - void contextUpdate ()

max: größter Meßwert im Puffer valBuffer.
avg: Mittelwert über alle Meßwerte im Puffer valBuffer.
slp: Differentialquotient zwischen ersten und letzten Meßwert und Zeitwert der beiden Puffer valBuffer und timeStampBuffer.

ContextObject: stellen Sensor-Fakten und komplexe Fakten dar.

ContextObject
- String name : Name des ContextObjects - int level : spezifiziert den Level eines ContextObjects in der Auswertung - double val : aktueller Wert - double defaultVal : Defaultwert, um Aktivierung zu erkennen und zum Zurücksetzen - FileWriter outputFileWriter : Handle des XML-OutputFile
+ ContextObject () : Konstruktor, der ContextObject erzeugt und in ContextContainer einträgt und weiters die Erstellung seiner Abbildung in der Jess Knowledge-Base veranlaßt + void setVal (double val) : setzt den Wert und veranlaßt die generische Ausgabe + void printout () : Ausgabe der ContextObject-Daten ins XML-OutputFile + alle gets´ & sets´

Bei der Erzeugung eines ContextObjects mittels Aufruf des Konstruktors erfolgt der Eintrag im ContextContainer. Weiters veranlaßt das ContextObject mittels definstance die Erstellung einer Abbildung seiner selbst in der Wissensbasis des Jess-Expertensystems rete.
Im ContextObject wird die Ausgabe generisch realisiert, da bei dieser Bohrdatenauswertung eine allgemeine Ausgaberegel möglich ist. Wenn immer ein ContextObject des höchsten Auswertungslevels aktiv gesetzt wird, also sein Wert ungleich dem Standardwert ist, dann wird mit dem Aufruf der Methode printout() die Ausgabe veranlaßt. Die printout() schreibt dann die ContextObject-Daten time und name in das XML-OutputFile.

Die Methode setVal() wird auch von Aktionsteil der Jess-Regeln aufgerufen, um komplexe Fakten zu setzen. Dabei unterscheidet sich die Jess-Syntax des Methodenaufrufs von der Java-Syntax.
Details: siehe Rules Seite 80.

ChannelContainer: Erledigt die Verwaltung der ChannelObjects, und ist damit Java-Teil intern.

ChannelContainer
- s HashMap channelObjects : Tabelle zur Verwaltung der ChannelObjects
- s void insertChannel (String name, ChannelObject channel) + s ChannelObject getChannel (String name)

ContextContainer: Erledigt neben der Verwaltung der ContextObjects mit dem Zurücksetzen auch komplexere Aufgaben im Zusammenhang mit dem Jess-Teil.

ContextContainer
- s HashMap contextObjects : Tabelle zur Verwaltung der ContextObjects
- s void insertContext (String name, ContextObject context) + s ChannelObject getContext (String name) + s Iterator getAllContextObjectValue () + s void setLevelDefault (int level) : setzt alle ContextObjects eines bestimmten Levels auf deren Defaultwerte zurück + s void setOverLevelDefault (int level) : setzt alle ContextObjects ab einem bestimmten Level auf deren Defaultwerte zurück

Konfig: Enthält die gesammelten aus FileReader und ChannelObject herausgezogenen Deklarationen.

Da die Deklarationen der Java-Objekte aus Jess-Konfig heraus geschehen, ist die Syntax der Konstruktoraufrufe verschieden.

Jess Konfig	
contextObjects :	Deklarationen
channelObjects :	Deklarationen

```
Java:    new ChannelObject("dmea", 20, {4});
Jess:   (new ChannelObject "dmea" 20 (create$ 4))
```

Die Angabe des Arrays int[] suffixIndexes in ChannelObject (z.B. {0,3,4}) geschieht in Jess mittels eines MultiFields (entsprechend (create$ 0 3 4)).

Rules: Enthält die Deklarationen der Grenzwerte für die Regeln und die Regeln selbst (Beispiel einer Regel: siehe Kapitel 8.1 Rules).

Jess Rules	
defglobals :	Deklarationen
rules :	Deklarationen

In Rules wird die Methode setVal() der ContextObjects benötigt, um diese gegebenenfalls zu setzen. Die Syntax des Aufrufs aus Jess ist ebenfalls verschieden.

```
Java:    contextObject.setVal(6.0);
Jess:   (set ?context val 6.0)
```

7.3 Dynamisches Konzept

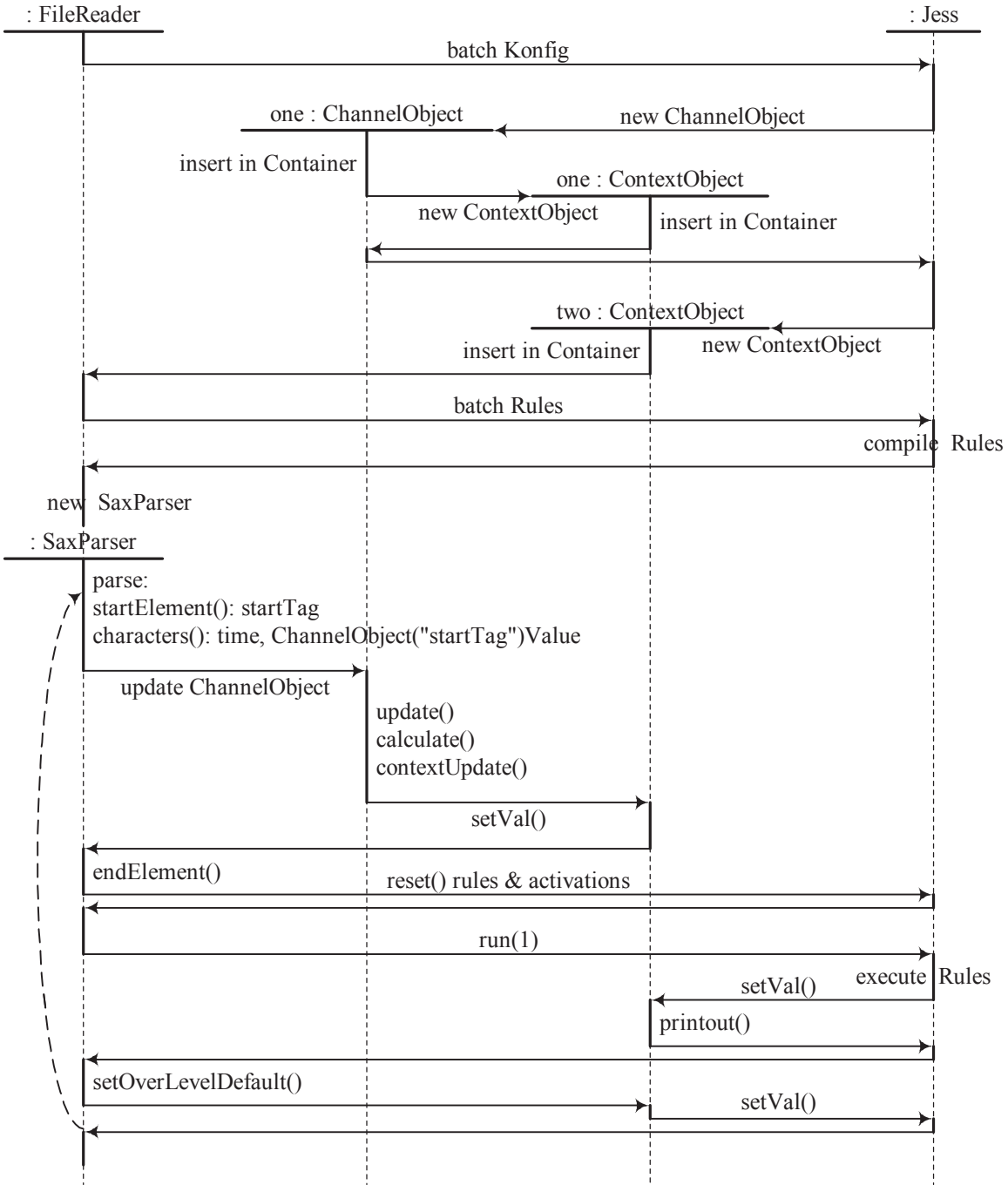


Abbildung 7.3: Sequenzdiagramm, Entwurf.

8 Prototypische Implementierung und Test

In der Implementierungsphase wird aus dem OOD-Modell das fertige Programm erstellt. Durch die prototypische Implementierung wird die Funktionsfähigkeit des entwickelten Modells nachgewiesen.

8.1 Rules

Das grundlegende Wissen über die Auswertungsregeln der Erdölbohrdaten stammt von Sonja Ernst und Dieter Wimberger. Dieter Wimberger stellte theoretische Regeln auf, die von Sonja Ernst entsprechend der vorhandenen Meßdaten angepaßt wurden. Weitere Änderungen erfolgten meinerseits aufgrund des neu hinzugekommenen wissensbasierten Konzepts der Regeln.

Sämtliche Regeln sind in Tabelle Seite 83 und im Anhang A.Programm Rules zu finden.

Input-Daten bzw. ChannelObjects:

```
bpos            block position
dmea            depth hole measured
mfia            mud flow in average
hkla            hook load average
woba            weight on bit average
```

Komplexe ContextObjects bzw. Auswertungsergebnisse:

```
wbeMoveDown     wellborne equipment run in hole
wbeMoveUp       wellborne equipment run out of hole
wrDown          wash in hole or ream in hole
wrUp            wash out of hole or ream out of hole
circ            circulation
genHole         generating hole
```

Exemplarische Erklärung der Regel zu genHole:
Dmea gibt die Bohrlochtiefe an. Das Ansteigen dieses Wertes ist signifikant für den Bohrfortschritt. Dies kann mittels des aus dmea abgeleiteten slp-Faktes realisiert werden. Weiters ist während der Lochbohrung woba und mfia größer null.

Detail zur Regelgestaltung:
WrDown & Up, wbeMoveDown & Up: Bei den entsprechenden Regeln wurde die Bedingung woba = 0, also das Nichteintreten, hinzugefügt. Damit sind wrDown & Up eindeutig gegenüber genHole verschieden. WbeMoveDown & Up wurden ebenfalls um diese Bedingung ergänzt, um die Regeln konsistent zu halten.

Aussagen / Fakten	bpos	dmea	mfia	hkla	woba
wbeMoveDown	slp > negbgn & slp < negend		= 0	> val	= 0
wbeMoveUp	slp > posbgn & slp < posend		= 0	> val	= 0
wrDown	slp > negbgn & slp < negend		> 0	> val	= 0
wrup	slp > posbgn & slp < posend		> 0	> val	= 0
circ			> 0		
genHole		slp > 0	> 0		> 0

Negbgn, negend, posbgn, posend und val sind fakten- und bohrspezifische Steigungs- und Grenzwerte, die eruiert und dann Jess zur Verwendung bereitgestellt werden müssen.

Beispiel einer Jess-Regel:

```
1    (defrule state-wbeMoveDown
2     (context (name "mfiaval") (val 0.0))
3     (context (name "wobaval") (val 0.0))
4     (context (name "hklaval") (val ?hklaval))
5     (context (name "bposslp") (val ?bposslp))
6     (test (> ?hklaval ?*hklaval*))
7     (test (and (> ?bposslp ?*bposslpnegbgn*)
                 (< ?bposslp ?*bposslpnegend*)))
8     (context (name "wbeMoveDown") (OBJECT ?context))
9     =>
10    (set ?context val 6.0))
```

Erklärung:

Zu 1: Regelkopf mit Angabe des Regelnamens state-wbeMoveDown.

Zu 2: Context ist die Abbildung von ContextObject. Hier wird überprüft, ob mfia-Meßwert gleich 0.0 ist, wobei das Objekt über seinen Namen angesprochen wird.

Zu 4: Erzeugen und Belegen der Variablen ?hklaval, die den Wert hklaval des ContextObjects mit dem Namen „hklaval“ beinhaltet. Dieses umständliche Verfahren zum Ansprechen ist erforderlich, da keine Objektreferenz zum direkten Ansprechen verfügbar ist.

Zu 6: Testen, ob der Wert hklaval des ContextObjects hklaval größer als die globale Variable ?*hklaval*, der hkla spezifische val-Grenzwert, ist.

Zu 8: Erzeugen der Objektreferenz ?context für das ContextObject mit dem Namen “wbeMoveDown”, um es dann im Aktionsteil setzen zu können.

Zu 10: Aktionsteil, in dem das komplexe ContextObject aktiv gesetzt wird.

8.2 Konfig

```
(new ChannelObject "dmea" 20 (create$ 4))
(new ChannelObject "bpos" 5 (create$ 4))
(new ChannelObject "hkla" 2 (create$ 0))
(new ChannelObject "mfia" 2 (create$ 0))
(new ChannelObject "woba" 2 (create$ 0))
```

Bei dmea und bpos wurde die Pufferlänge für dmeaslp und bposslp auf 20 bzw. auf 5 gesetzt, um die gewünschte Auswertungsempfindlichkeit zu erhalten. Bei den restlichen ChannelObjects wurde die Pufferlänge auf das Minimum 2 gesetzt, da hier nur die val-Werte benötigt werden.

```
(new ContextObject "wbeMoveUp" 1 0.0)
(new ContextObject "wbeMoveDown" 1 0.0)
(new ContextObject "wrDown" 1 0.0)
(new ContextObject "wrUp" 1 0.0)
(new ContextObject "genHole" 1 0.0)
(new ContextObject "circ" 1 0.0)
```

8.3 Ermittlung geeigneter Steigungs- und Grenzwerte

Die Grenzwerte wurden über grafische Darstellung der Input-Daten abgeleitet. Für die Steigungswerte und Pufferlängen wurden die abgeleiteten slp-Daten mittels mehrerer Programmdurchläufe

berechnet und durch Veränderung der Ausgabe ausgegeben. Durch deren grafischen Darstellung wurden die Grenzwerte und die Pufferlängen festgelegt, und dann das Auswertungsergebnis mittels eines Programmdurchlaufs überprüft. Dieser Zyklus wurde solange wiederholt, bis das Auswertungsergebnis den Vorstellungen entsprach.

```
(defglobal ?*hklaval* = 150)
(defglobal ?*bposslpposbgn* = 0.15)
(defglobal ?*bposslpposend* = 0.5)
(defglobal ?*bposslpnegbgn* = -0.5)
(defglobal ?*bposslpnegend* = -0.15)
```

8.4 Input-Daten

Die verwendeten Sensorendaten sind reale Meßwerte einer Erdölbohrung, die aber selektiert und leicht editiert wurden, um alle Zustände innerhalb eines vernünftigem Zeitrahmens zu beinhalten. Mittels eines eigenen VBA-Hilfsprogramms wurden die Input-Daten entsprechend dem Anforderungskatalog in XML-Format transformiert.

```
<channellist>
    <channels>
        <time>1</time>
        <bpos>21.038639</bpos>
        <dmea>663.85358</dmea>
        <hkla>370.5762</hkla>
        <mfia>1343.9821</mfia>
        <woba>21.00651</woba>
    </channels>
    <channels>
        <time>2</time>
        <bpos>21.038639</bpos>
        <dmea>663.85358</dmea>
        <hkla>370.78619</hkla>
        <mfia>1343.9821</mfia>
        <woba>20.796471</woba>
    </channels>
    ...
```

Der Tag „channels“ als Begrenzung einer Meßwertreihe ist zwingend, da FileReader diesen Tag zum Parsen benötigt.

8.5 Programmablauf

```
FileReader: outputFile
FileReader: jessKonfig einlesen

  Rules: ContextObjects

  +ContextObject: new Object: wbeMoveUp
  ==> f-0 (MAIN::context (class <External-Address:java.lang.Class>)
            (defaultVal 0.0) (level 1) (name "wbeMoveUp") (val 0.0)
            (OBJECT <External-Address:ContextObject>))
      Meldung aufgrund watch durch definstance-Befehl im Konstruktor
            von ContextObject.
      Ähnliche Meldungen tretten bei jedem ContextObject-Konstruktor-
            aufruf auf und werden nachfolgend nicht mehr aufgeführt.
    +ContextContainer: insertContext: wbeMoveUp
  +ContextObject: new Object: wbeMoveDown
    +ContextContainer: insertContext: wbeMoveDown
  +ContextObject: new Object: wrDown
    +ContextContainer: insertContext: wrDown
  +ContextObject: new Object: wrUp
    +ContextContainer: insertContext: wrUp
  +ContextObject: new Object: genHole
    +ContextContainer: insertContext: genHole
  +ContextObject: new Object: circ
    +ContextContainer: insertContext: circ

  Rules: ChannelObjects

  *ChannelObject: new Object: dmea
    *ChannelContainer: insertChannel: dmea
  +ContextObject: new Object: dmeaslp
  ==> f-6 (MAIN::context (class <External-Address:java.lang.Class>)
            (defaultVal 0.0) (level 0) (name "dmeaslp") (val 0.0)
            (OBJECT <External-Address:ContextObject>))
      Meldung aufgrund watch durch definstance-Befehl im Konstruktor
            von ContextObject.
      Ähnliche Meldungen tretten bei jedem ContextObject-Konstruktor-
            aufruf auf und werden nachfolgend nicht mehr aufgeführt.
    +ContextContainer: insertContext: dmeaslp
  *ChannelObject: new Object: bpos
    *ChannelContainer: insertChannel: bpos
  +ContextObject: new Object: bposslp
    +ContextContainer: insertContext: bposslp
  *ChannelObject: new Object: hkla
    *ChannelContainer: insertChannel: hkla
  +ContextObject: new Object: hklaval
    +ContextContainer: insertContext: hklaval
  *ChannelObject: new Object: mfia
```

```
    *ChannelContainer: insertChannel: mfia
  +ContextObject: new Object: mfiaval
    +ContextContainer: insertContext: mfiaval
  *ChannelObject: new Object: woba
    *ChannelContainer: insertChannel: woba
  +ContextObject: new Object: wobaval
    +ContextContainer: insertContext: wobaval

FileReader: jessRules einlesen

  Rules: DefGlobals
  Rules: Rules

  MAIN::state-wbeMoveDown: +1+1+1+1+1+1+1+1+1+1+2+2+2+2+2+2+t
  MAIN::state-wbeMoveUp: =1=1=1=1=1=1=1=1+1=1=2=2=2=2+2+2+t
  MAIN::state-wrDown: =1=1=1=1=1=1=1+1+1+2+2+2+2+2+2+2+t
  MAIN::state-wrUp: =1=1=1=1=1=1=1+1=1=2+2+2+2+2+2+2+t
  MAIN::state-genHole: =1=1=1+1=1+1+1+2+2+2+2+2+2+t
  MAIN::state-circ: =1=1=1+1=1+2+2+t

FileReader: inputFile
FileReader: parsen
-----------------
*FileReader: time: 1.0
+FileReader: Update: bpos=21.038639
  -ChannelObject: update, calculate, contextUpdate: bpos
    -ContextObject: setVal: bposslp=21.038639
+FileReader: Update: dmea=663.85358
  -ChannelObject: update, calculate, contextUpdate: dmea
    -ContextObject: setVal: dmeaslp=663.85358
+FileReader: Update: hkla=370.5762
  -ChannelObject: update, calculate, contextUpdate: hkla
    -ContextObject: setVal: hklaval=370.5762
+FileReader: Update: mfia=1343.9821
  -ChannelObject: update, calculate, contextUpdate: mfia
    -ContextObject: setVal: mfiaval=1343.9821
+FileReader: Update: woba=21.00651
  -ChannelObject: update, calculate, contextUpdate: woba
    -ContextObject: setVal: wobaval=21.00651

*FileReader: reset
  ==> Focus MAIN
  ==> f-0 (MAIN::initial-fact)
  ==> f-1 (MAIN::context (class <External-Address:java.lang.Class>)
            (defaultVal 0.0) (level 1) (name "circ") (val 0.0)
            (OBJECT <External-Address:ContextObject>))
      Ab hier abgekürzte Schreibweise: Weglassen der gleichen ersten
           und letzten Teile mit <External-Address:java.lang.Class>
  ==> f-2 (MAIN::context (defaultVal 0.0) (level 0)
```

```
            (name "hklaval") (val 370.5762)
  ==> f-3 (MAIN::context (defaultVal 0.0) (level 1)
            (name "wbeMoveDown") (val 0.0)
  ==> f-4 (MAIN::context (defaultVal 0.0) (level 0)
            (name "mfiaval") (val 1343.9821)
==> Activation: MAIN::state-circ :  f-4,, f-1
  ==> f-5 (MAIN::context (defaultVal 0.0) (level 1)
            (name "wrUp") (val 0.0)
  ==> f-6 (MAIN::context (defaultVal 0.0) (level 1)
            (name "wrDown") (val 0.0)
  ==> f-7 (MAIN::context (defaultVal 0.0) (level 0)
            (name "wobaval") (val 21.00651)
  ==> f-8 (MAIN::context (defaultVal 0.0) (level 1)
            (name "genHole") (val 0.0)
  ==> f-9 (MAIN::context (defaultVal 0.0) (level 0)
            (name "bposslp") (val 21.038639)
  ==> f-10 (MAIN::context (defaultVal 0.0) (level 0)
            (name "dmeaslp") (val 663.85358)

==> Activation: MAIN::state-genHole :  f-4, f-7, f-10,,,, f-8

  ==> f-11 (MAIN::context (defaultVal 0.0) (level 1)
            (name "wbeMoveUp") (val 0.0)

*FileReader: run
FIRE 1 MAIN::state-genHole f-4, f-7, f-10,,,, f-8
    -ContextObject: printout: genHole

*FileReader: setOverLevelDefault level:1
    ContextContainer: setOverLevelDefault 1
-----------------
*FileReader: time: 2.0
...
```

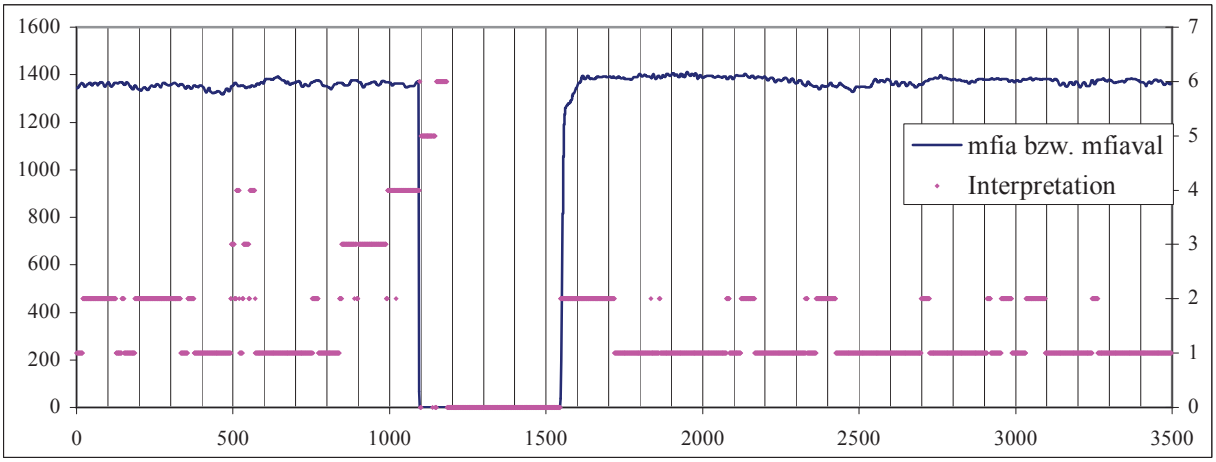
8.6 Output-Daten

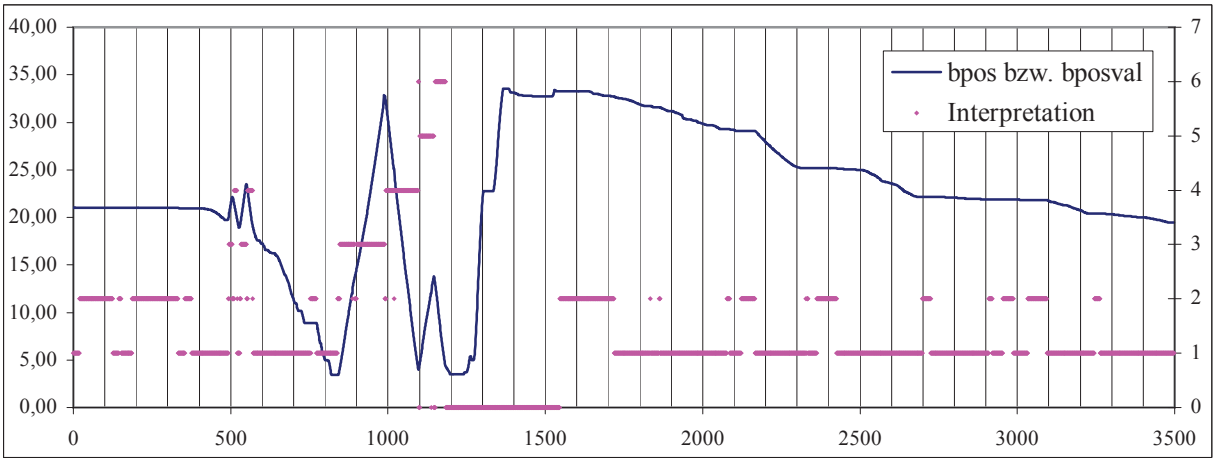
```
    <outputlist>
        <output>
            <time>1.0</time>
            <context>genHole</context>
        </output>
        <output>
            <time>2.0</time>
            <context>genHole</context>
        </output>
        ...
```

8.7 Diagramm-Auswertung

Zur Veranschaulichung werden hier die Input-Daten bzw. die abgeleiteten einfachen ContextObjects und die Auswertungsergebnisse bzw. die komplexen ContextObjects als Diagramme dargestellt. Zur Verdeutlichung der Zusammenhänge wurden die Diagramme der Input-Daten durch das Auswertungsergebnis (Interpretation) als Sekundärachse ergänzt.
Bei der Interpretation entsprechen die Werte folgendem Ergebnis:

```
6               wbeMoveDown
5               wbeMoveUp
4               wrDown
3               wrUp
2               circ
1               genHole
0               nothing
```





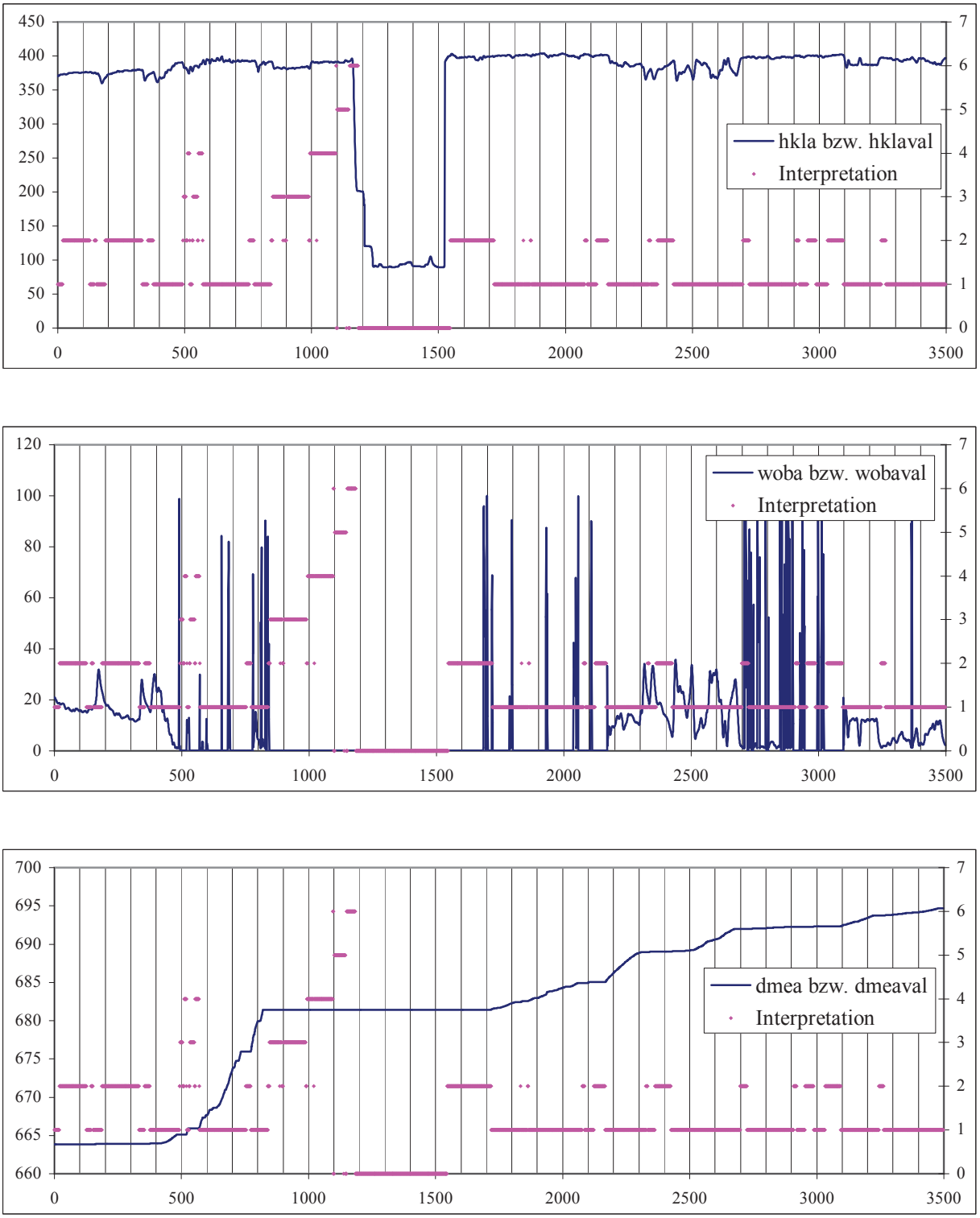


Abbildung 8.1: Diagramme der Input-Daten bzw. der einfachen ContextObjects.

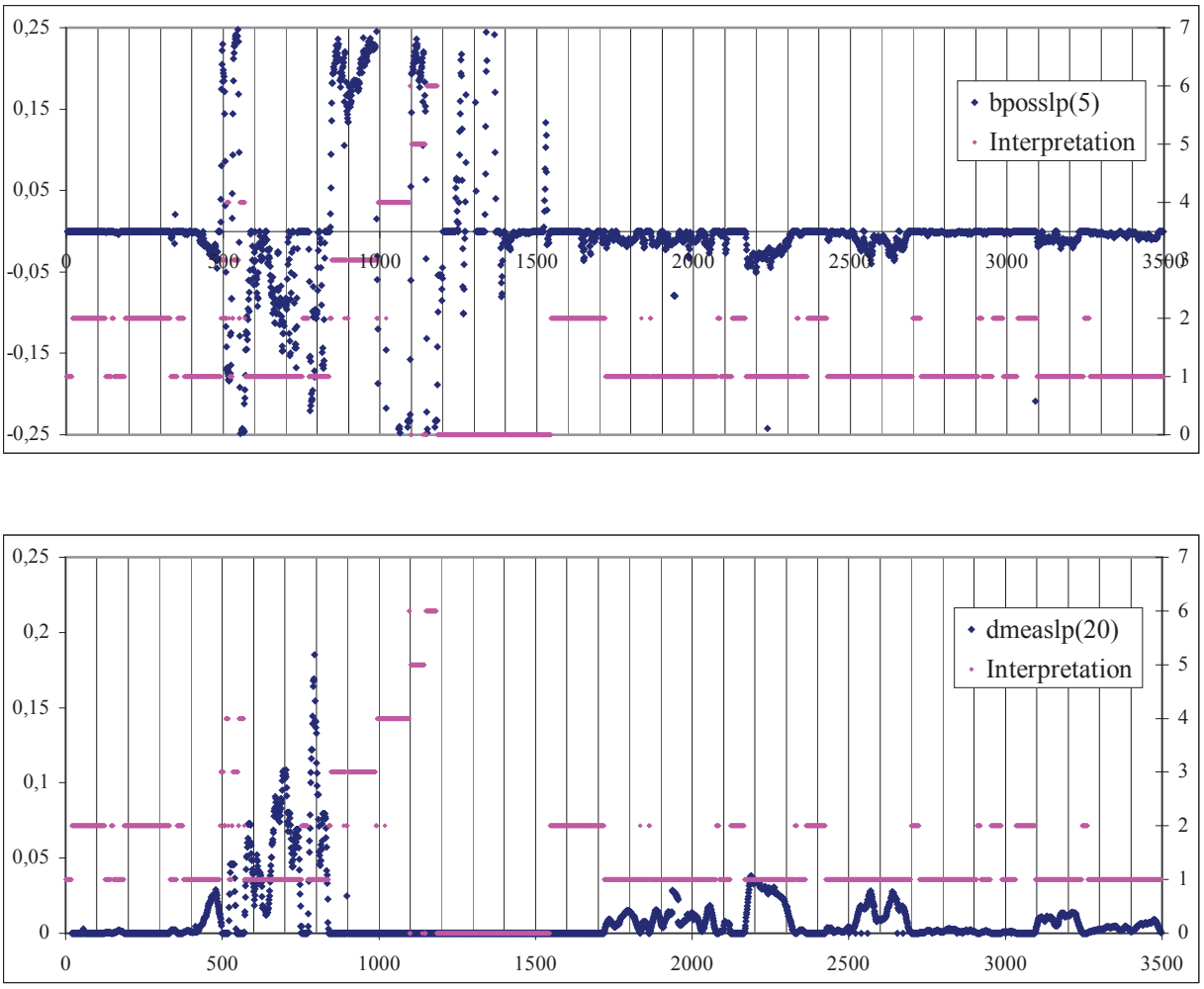


Abbildung 8.2: Diagramme der abgeleiteten einfachen ContextObjects.

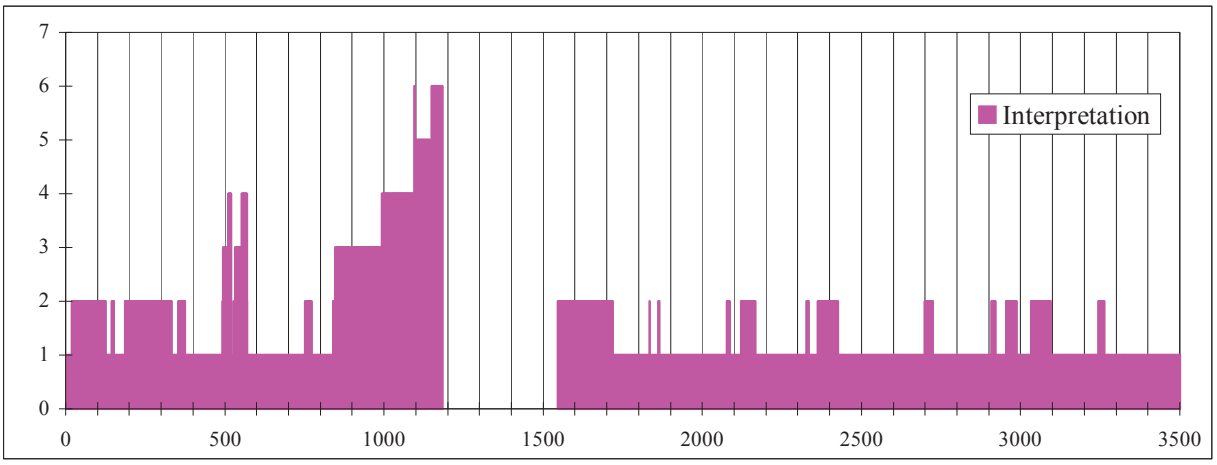


Abbildung 8.3: Diagramm des Auswertungsergebnisses der Bohrinterpretation.

8.8 Details zur Implementation und andere Möglichkeiten

Rete

Jess verwendet eine verbesserte Form des Rete-Algorithmus, um die Regeln mit der Wissensbasis abzugleichen. Dabei werden die Regeln als azyklischer gerichteter Graph implementiert, wobei folgende Optimierungen stattfinden (siehe 4.4.2). Strukturelle Ähnlichkeiten von Bedingungsmustern werden durch Regelkompilierung zusammengefaßt. Die zeitliche Redundanz wird vermieden, indem die Konfliktmenge aufgrund von Änderungen nur modifiziert und nicht komplett neuberechnet wird. Dadurch wird explizit ein Geschwindigkeitsgewinn auf Kosten von Speicherplatz erzielt, wobei somit der Speicherplatzbedarf nicht unbedeutend ist. (Details: siehe [fri01] und [win92].

Für die Regelstruktur ergibt sich folgende Empfehlung: Die spezifischsten Muster sollten weiter am Beginn, die transienten Muster hingegen weiter am Ende des Bedingungsteils einer Regel stehen.

Zusammenspiel von Java und Jess

Damit der externe Zugriff von Jess auf die Java-Objekteigenschaften name, level, val usw. möglich ist, muß die Java-Klasse, wenn ihre Objekteigenschaften private sind, die entsprechenden set- und get-Methoden als public bereitstellen.

Bei meinem Programm erfolgt die Repräsentation in der Wissensbasis statisch, da das für diese Bohrauswertung genügt und die effizientere Lösung darstellt. Dafür muß die Java-Klasse die Eigenschaften einer Java-Bean aufweisen, d.h. bestimmten Namens- und Methodenkonventionen entsprechen.
Um eine dynamische Repräsentation zu realisieren, muß die Java-Klasse darüberhinaus neben java.io.Serializable auch java.beans.PropertyChangeListener unterstützen. Dadurch kann Jess über eine Änderung des Java-Objekts automatisch benachrichtigt werden, und bleibt so kontinuierlich auf dem aktuellen Stand. Ich habe auch die dynamische Repräsentation probehalber realisiert und sie als Kommentar im Programm belassen [siehe Anhang Programm: Klasse ContextObject: Kommentar 1,2 und 3].

Java Reflection

Die jeweils abhängigen ContextObjects werden von ChannelObject dynamisch mittels Java Reflection aktualisiert. Diese Lösungsmöglichkeit ist ausreichend flexibel und benötigt wenig Zeit und Speicheraufwand.
Eine andere Lösungsmöglichkeit wären Java-Beans und java.beans.PropertyChangeListener.

[siehe Anhang Programm: Klasse ChannelObject: Kommentar 1,2,3 und 4].

Steuerung von Jess mittels Konfliktlösungsstrategie

Bei meinem Programm genügte die Angabe des Arguments 1 im Run-Befehl run[1] zur Steuerung von Jess.
Für die Realisierung komplexer Steuerungen bietet sich jess.JessEvent und jess.JessListener an [siehe fri01]. Damit können z.B. die gefeuerten Regeln erkannt und dann jeweils selektiv darauf reagiert werden. Dadurch ergeben sich viele Möglichkeiten für mehrstufige Auswertungen. Ich habe auch diese Möglichkeit getestet und sie als Kommentar im Programm belassen [siehe Anhang Programm: Klasse FileReader: Kommentar 1 und 3].

9 Resümee

Die Implementation des Prototypen für die Dateninterpretation einer Erdölbohrung und damit die Erstellung des Programm-Musters waren erfolgreich und die Ziele und Anforderungen wurden erfüllt. Die geforderten Entwurfsprinzipien Modularität und Transparenz wurden besonders durch folgende Punkte begünstigt:

- die durch das wissensbasierte Konzept des Expertensystems ermöglichte Trennung von Wissen und Steuerung;
- die durch Verwendung einer kompletten Expertensystemshell von der übernommenen Steuerung des Wissens;
- die durch Anwendung des Schichten-Entwurfsprinzips resultierende Schichtenarchitektur.

Es entstand ein Programmsystem bzw. ein Programm-Muster, welches die Verständlichkeit, Wiederverwendbarkeit und Erweiterbarkeit gewährleistet, und so Softwareengineering und Wissensmanagement ermöglicht.

Durch das wissensbasierte Konzept der Expertensystemshell ist eine flexible Änderung und Erweiterung der Regeln mittels der von Jess bereitgestellten Programmiersprache möglich.
Diese ungewohnte Programmiersprache von Jess stellt eine mögliche Schwierigkeit für zukünftige Anwendungen dar. Im Programmsystem wurde dies durch Auslagerung aller generischer Programmaufgaben in Java und dem sorgfältigen Erklären der restlichen in Jess verbliebenen Objekt-, Werte- und Regeldeklarationen soweit als möglich reduziert.
Eine komplette Behebung dieser Schwierigkeit wäre zukünftig durch Erstellung einer grafischen Oberfläche zur Eingabe und Änderung dieser Deklarationen möglich.

Zukünftige Erweiterungsmöglichkeiten sind die mehrstufige Auswertung, wobei die Steuerung entsprechend gestaltet werden muß, und die Online-Auswertung statt des Auslesens aus einer XML-Datei.

Anhang A

Programm

FileReader

```
import java.io.*;

import jess.*;
import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;

import javax.xml.parsers.*;

/** Die Klasse <code>FileReader</code> enthält main() und damit die zentrale
 * Ablaufsteuerung. Die Meßdaten werden mittels XML-Parser aus dem XML-InputFile
 * in die ChannelObjects eingelesen. Weiters Wird das Jess-Expertensystem mittels
 * dessen Java-Bibliotheken erstellt und dann die Jess-Files Konfig und Rules
 * eingelesen. Auch wird das XML-OutputFile intialisiert.
 * @author Steiner Engelbert
 * @version 1.10
 */
public class FileReader {

  /* da main Klassenmethode->kann direkt nur auf Klasseneigenschaften (static)
    zugreifen*/

  /** das Feld <code>rete</code> ist die Jess Rule-Engine. Um Jess in ein
   * Java-Programm einzubetten, wird diese Rule-Engine erzeugt und entsprechend
   * manipuliert.
   */
  private static Rete rete;
  /** das Feld <code>time</code> beinhaltet die Zeitangabe der aktuellen
   * Messwertreihe.
   */
  private static double time;
  private static FileWriter outputFileWriter;
  //Pfadangaben
  private static String jessKonfig = "C:/EST/Jess60/programs/drill10/Konfig.clp";
  private static String jessRules = "C:/EST/Jess60/programs/drill10/Rules.clp";
  private static String inputFile = "C:/EST/Jess60/programs/input_long.xml";
  private static String outputFile = "C:/EST/Jess60/programs/drill10/output.xml";

  /** Die Methode <code>main</code> erzeugt die Rule-Engine, liest die
   * Jess-ScriptFiles Konfig und Rules ein und öffnet sowohl XML-InputFile als
   * auch XML-OutputFile. Die Methode <code>main</code> parst das XML-InputFile,
   * aktualisiert währenddessen die ChannelObjects und über diese die ContextObjects
   * und startet und manipuliert die Rule-Engine nach jeder eingelesenen Meßwertreihe.
```

```
 */
public static void main(String[] args) throws Exception {
  //rete erzeugen
  rete = new Rete();
  /*1*/
  /*File-Handler outputFileWriter erzeugen für Ausgabe der Ergebnis-ContextObjects
    in XML-File muß vor Einlesen der Jess-Rules stattfinden, da outputFileWriter
    für ContextObjects benötigt wird*/
  try {
    File outFile = new File(outputFile);
    if (outFile.exists()) {
      outFile.delete();
    }
    outFile.createNewFile();
    outputFileWriter = new FileWriter(outFile);
    outputFileWriter.write("<outputlist>\r\n");
  } catch (IOException ecnf) {
    System.out.println("  FileReader Error Outputfile");
    ecnf.printStackTrace();
  }
  //test//System.out.println("FileReader: outputFile");
  //Jess-Konfig einlesen
  //test//System.out.println("FileReader: jessKonfig einlesen\r\n");
  rete.executeCommand("(batch " + jessKonfig + ")");
  //Jess-Rules einlesen
  //test//System.out.println("\r\nFileReader: jessRules einlesen");
  rete.executeCommand("(batch " + jessRules + ")");
  //rete.executeCommand("(reset)");
  /*File-Handler inFile fürs Einlesen des XML-File in die ChannelObjects
    erzeugen*/
  File inFile = new File(inputFile);
  //test//System.out.println("\r\nFileReader: inputFile");
  //SAXParser parser erzeugen
  SAXParserFactory factory = SAXParserFactory.newInstance();
  SAXParser parser = factory.newSAXParser();
  //Event-Handler fürs Parsen erzeugen
  DefaultHandler myHandler = new DefaultHandler() {

    String name;

    //public void startDocument(...)
    //public void endDocument(...)
    public void startElement(String uri, String localName, String qName,
                             Attributes attributes) {
      //startTag merken
      if ("".equals(qName.trim()) != true) {
        name = qName;
        //test//System.out.println(":"+qName);
      }
    }

    public void endElement(String uri, String localName, String qName) {
      /*starten der Regel-Abarbeitung bei endTag "channels":*/
      if ("channels".equals(qName.trim())) {
        try {
          /*reset vor run um die Activations von  voriger Abarbeitung zu löschen,
            Variable sind nicht zu löschen, da nur Objecte (contextObject)
            existieren*/
          //test//System.out.println("\r\n*FileReader: reset");
          rete.executeCommand("(reset)");

          /*Die Zahl 1 hinter run gibt die Anzahl der Rules-Firings an, nach
            der die durch run gestartete Abarbeitung abgebrochen wird*/
          //test//System.out.println("\r\n*FileReader: run");
          rete.executeCommand("(run 1)"); /*2*/
```

```
          } catch (JessException ej) {
            System.out.println("FileReader: Fehler: " + ej.getMessage());
            ej.printStackTrace();
          }
          //Zurücksetzen der ContextObjects für die nächste Abarbeitung
          //test//System.out.println("\r\n*FileReader: setOverLevelDefault level:1");
          ContextContainer.setOverLevelDefault(1);
          //Aufruf des Garbage Collectors
          System.gc();
        }
      }

      public void characters(char[] data, int start, int lenght) {
        /*XML-Daten mittels gemerkter startTag entweder time oder ChannelObject
          zuweisen*/
        String text = (new String(data, start, lenght)).trim();

        if ("".equals(text) == false) {
          double val = Double.parseDouble(text);
          if (name.equals("time")) {
            time = val;
            //test//System.out.println("-----------------");
            //test//System.out.println("*FileReader: time: " + val);
          } else {
            /*System.out.println("+FileReader: ReadIn: " + name + "=" +
                                          val + " (time:"+time+")");*/
            ChannelObject channel = ChannelContainer.getChannel(name);
            if (channel != null) {
              //test//System.out.println("+FileReader: Update: " + name + "=" +
              //test// val /*+ " (time:"+time+")"*/);
              channel.update(val, time);
            }
          }
        }
      }
    };
    //Parsing starten
    //test//System.out.println("FileReader: parsen");
    parser.parse(inputFile, myHandler);

    outputFileWriter.write("</outputlist>\r\n");
    outputFileWriter.close();
  }

  /** Die Methode <code>getRete</code> gibt die Rule-Engine zurück.
   */
  public static Rete getRete() {
    return rete;
  }

  /** Die Methode <code>getTime</code> gibt die Zeitangabe der aktuellen
   * Meßwertreihe zurück.
   */
  public static double getTime() {
    return time;
  }

  /** Die Methode <code>getOutputFileWriter</code> gibt den
   * XML-OutputFile-Handler zurück
   */
  public static FileWriter getOutputFileWriter() {
    return outputFileWriter;
  }
}
```

```
/*3*/

/*1
    rete.addJessListener(new JessEventHandler());
    rete.setEventMask(JessEvent.DEFRULE_FIRED);*/
/*2
    System.out.println("Start: "+System.currentTimeMillis());
    System.out.println("Stop: "+System.currentTimeMillis());*/
/*3
    Nachfolgender Code durch 1 im JESS-run-Befehl (run 1) unnötig geworden

    class JessEventHandler implements JessListener {
      public void eventHappened(JessEvent je) {

        if ((je.getType()) == JessEvent.DEFRULE_FIRED) {
          try {
            FileReader.getRete().executeCommand("(reset)");
            ContextContainer.setOverLevelDefault(1);
            //System.out.println(": reset nach JessEvent.DEFRULE_FIRED");
          } catch (JessException e) {
            System.out.println(e.getMessage());
          }
        }
      }
    }*/
```

ChannelObject

```
//import jess.*;

import java.lang.Class;
import java.lang.reflect.Field;
import java.util.Arrays;

/*1*/

/** Die Klasse <code>ChannelObject</code> verarbeitet die von FileReader
 * eingelesenen Daten und verarbeitet diese zu ContextObjects. Ein ChannelObject
 * erzeugt die jeweiligs benötigte Auswahl an ContextObjects und aktualisiert
 * diese dann dynamisch.
 * @author Steiner Engelbert
 * @version 1.10
 */
public class ChannelObject /*2*/ {
  //Attribute
  private String name;
  private double val;
  private double min, max, avg, slp;
  private double timeStamp;

  private int bufferLength;

  private double[] valBuffer;
  private double[] timeStampBuffer;

  /** das Feld <code>suffixes</code> enthält die Umsetzungstabelle zur Angabe der
   * Auswahl an ContextObjects. Im ChannelObject-Konstruktoraufruf aus Jess heraus
   * wird die Auswahl mittels eines MuliFields angegeben. Wegen Jess und um
   * String-Fehler auszuschliessen erfolgt die Angabe der Auswahl mittels Zahlen.
   * Diese Zahlen werden mittels suffixes umgesetzt, um z.B. die
   * ContextObject-Benennung durchzuführen.
   * Index-Wert-Paare: [0 val], [1 min], [2 max], [3 avg], [4 slp]
   */
```

```
  private static String[] suffixes = {"val", "min", "max", "avg", "slp"};
  private int[] channelSuffixes;
  //aus ChannelObject zu erstellende ContextObjects

  /** Der Konstruktor <code>ChannelObject</code> erzeugt ein ChannelObject, fügt es
   * in den ChannelContainer ein und erzeugt die jeweilige Auswahl an ContextObjects.
   * Alle von ChannelObjects erzeugte ContextObjects haben Level 0.
   * @param <code>name</code> gibt den Namen des ChannelObjects an und muß mit
   * dem Tag-Name des entsprechenden Meßwertes im XML-InputFile übereinstimmen.
   */
  public ChannelObject(String name, int bufferLength, int[] suffixIndexes) {
    this.name = name;
    if (bufferLength >= 2) {
      this.bufferLength = bufferLength;
    } else {
      System.out.println("Die Buffergröße darf nicht kleiner als 2 sein und "
          + "wird somit auf 2 gesetzt");
      this.bufferLength = 2;
    }
    channelSuffixes = new int[suffixIndexes.length];
    System.arraycopy(suffixIndexes, 0, channelSuffixes, 0, suffixIndexes.length);
    valBuffer = new double[bufferLength];
    Arrays.fill(valBuffer, 0);
    timeStampBuffer = new double[bufferLength];
    //test//System.out.println(" *ChannelObject: new Object: " + this.name);
    ChannelContainer.insertChannel(name, this);
    //Erzeugen der in Jess definierten Channel-Contexte
    for (int i = 0; i < suffixIndexes.length; i++) {
      String newContextObjectName = name + suffixes[suffixIndexes[i]];
      ContextObject newContextObject = new ContextObject(newContextObjectName, 0, 0);
      //alle von ChannelObject abgeleitete ContextObjects erhalten Level 0)
/*3*/
    }
  }

  public void update(double newVal, double newTimeStamp) {
    /*test// System.out.println(" -ChannelObject: update, calculate, contextUpdate: "
                                                                + this.name);*/
    val = newVal;
    timeStamp = newTimeStamp;
    System.arraycopy(valBuffer, 1, valBuffer, 0, bufferLength - 1);
    System.arraycopy(timeStampBuffer, 1, timeStampBuffer, 0, bufferLength - 1);
    valBuffer[bufferLength - 1] = newVal;
    timeStampBuffer[bufferLength - 1] = newTimeStamp;
    calculate();
    contextUpdate();
  }

  private void calculate() {
    //test//System.out.println(" -ChannelObject: calculate " + this.name);
    double sumVal = 0;
    max = valBuffer[0];
    min = valBuffer[0];
    for (int i = 0; i < bufferLength; i++) {
      if (valBuffer[i] > max)
        max = valBuffer[i];
      if (valBuffer[i] < min)
        min = valBuffer[i];
      sumVal = sumVal + valBuffer[i];
    }
    avg = sumVal / bufferLength;
    slp = (double) ((valBuffer[bufferLength - 1] - valBuffer[0]) /
        (timeStampBuffer[bufferLength - 1] - timeStampBuffer[0]));
  }
```

```
  private void contextUpdate() {
    //test//System.out.println(" -ChannelObject: contextUpdate " + this.name);
    for (int i = 0; i < channelSuffixes.length; i++) {
      //Suffix des zugehörigen ContextObjects
      String suffix = suffixes[channelSuffixes[i]];
      double v = 0;
      try {
        //Wert v von ChannelObject für abgeleitetes ContextObject reflektieren
        //ClassObject, welches das ChannelObject this repräsentiert
        Class c = this.getClass();
        //FieldObject val,min,max or slp des ChannelObject this
        Field f = c.getDeclaredField(suffix);
        //Wert des FieldObjects des ChannelObject this
        f.setAccessible(true);
        v = f.getDouble(this);
        f.setAccessible(false);
      } catch (NoSuchFieldException ensf) {
        System.out.println("  ChannelObject: reflection: " + ensf.getMessage());
        ensf.printStackTrace();
      } catch (Exception es) {
        System.out.println("  ChannelObject: reflection: " + es.getMessage());
        es.printStackTrace();
      }
      //ändern des zugehörigen ContextObjects
      ContextContainer.getContext(this.name + suffix).setVal(v);
    }
  }
/*4*/
}

/*1
    import java.io.Serializable;
    import java.beans.PropertyChangeSupport;
    import java.beans.PropertyChangeEvent;
    import java.beans.PropertyChangeListener;*/
/*2
    implements Serializable*/
/*3
    Kommunikation ChannelObject->ContextObject mittels PropertyChangeListener
      statt Reflexion
      this.addPropertyChangeListener(new PropertyChangeListener () {
        public void propertyChange (PropertyChangeEvent event) {
          if (event.getPropertyName().compareTo(suffix)==0) {
          newContextObject.setValue();
          }
        }});*/
/*4
    private PropertyChangeSupport pcs = new PropertyChangeSupport(this);

      public void addPropertyChangeListener(PropertyChangeListener pcl) {
        pcs.addPropertyChangeListener(pcl);
      }

      public void removePropertyChangeListener(PropertyChangeListener pcl) {
        pcs.removePropertyChangeListener(pcl);
      }*/
```

ContextObject

```
import java.io.Serializable;
import java.io.FileWriter;
import java.io.IOException;
/*1*/
import java.lang.reflect.Array;

import jess.*;

/** Die Klasse <code>ContextObject</code> stellt die Fakten für Jess dar.
 * @author Steiner Engelbert
 * @version 1.10
 */
public class ContextObject /*implements Serializable*/ {

  private String name;
  private int level;
  //zum selektiven stufenweisen Ansprechen für mehrstufige Auswertungen
  private double val;
  private double defaultVal;
  private FileWriter outputFileWriter;

  /** Der Konstruktor <code>ContextObject</code> erzeugt ein ContextObject, fügt es
   * in den ContextContainer ein. Weiters veranlaßt <code>ContextObject</code> die
   * Erstellung einer Abbildung seiner selbst in der Knowledge-Base von Jess.
   * @param <code>level</code> spezifiziert den Level eines ContextObjects in der
   * Auswertung.
   */
  public ContextObject(String name, int level, double defaultVal) {
    this(name, level, defaultVal, defaultVal);
  }

  /** Der Konstruktor <code>ContextObject</code> erzeugt ein ContextObject, fügt es
   * in den ContextContainer ein. Weiters veranlaßt <code>ContextObject</code> die
   * Erstellung einer Abbildung seiner selbst in der Knowledge-Base von Jess.
   * @param <code>level</code> spezifiziert den Level eines  ContextObjects in der
   * Auswertung.
   */
  public ContextObject(String name, int level, double defaultVal, double val) {
    //test//System.out.println(" +ContextObject: new Object: " + name);
    this.name = name;
    this.level = level;
    this.defaultVal = defaultVal;
    this.val = val;
    this.outputFileWriter = FileReader.getOutputFileWriter();
    try {
      /*durch den jess-Befehl definstance werden facts in der knowledge-base erzeugt,
        welche die zugehörigen java-objects repräsentieren durch Angabe der
        static-Klausel werden die facts nur bei reset auf den Stand der
        java-objects aktualisiert, default ist dynamic-Klausel, wo die facts
        sofort automatisch mittels PropertyChangeListener aktualisiert werden*/
      Funcall f = new Funcall("definstance", FileReader.getRete());
      f.add(new Value("context", RU.ATOM));
      f.add(new Value(this));
      f.add(new Value("static"));
      f.execute(FileReader.getRete().getGlobalContext());
    } catch (JessException ej) {
      System.out.println("ContextObject: Fehler: " + ej.getMessage());
    }
    ContextContainer.insertContext(name, this);
  }

  /** Die Methode <code>setVal</code> setzt den Wert des ContextObjects.
```

```
   * Weiters wird hier die generische Ausgabe veranlaßt, da bei dieser
   * Bohrauswertung eine allgemeine Ausgaberegel möglich ist.
   */
  public void setVal(double val) {
    this.val = val;
    /*2*/

    /*test: Ausgabe der ContextObjects(level=0)
    if (this.level == 0) {
      System.out.println("  -ContextObject: setVal: " + this.name + "=" + val);
    }*/

    //generische Ausgabe über level in java, da allgemeine Ausgaberegel möglich
    //auch individuellere Ausgabe mittels jess oder java ist möglich
    if ((this.level >= 1) && (this.val != this.defaultVal)) {
      printout();
    }

    //slp-Ausgaben zur Bestimmung der Grenzwerte in den Regeln
    /* if ("dmeaslp".equals(this.name)){
       try{
        this.outputFileWriter.write(((int) FileReader.getTime()) + " " +
                                                              this.val + "\r\n");
        this.outputFileWriter.flush();
      }catch (Exception ex) {
      System.out.println("  ContextObject Error Outputfile: "+ex.getMessage());
      ex.printStackTrace();}
     }*/
  }

  /** Die Methode <code>printout</code> schreibt folgende ContextObject-Daten in
   * das XML-OutputFile: time, name, val.
   */
  public void printout() {
    try {
      this.outputFileWriter.write("<output>\r\n");
      this.outputFileWriter.write("  <time>" + FileReader.getTime() + "</time>\r\n");
      this.outputFileWriter.write("  <context>" + this.name + "</context>\r\n");
      //this.outputFileWriter.write("  <val>"+this.val+"</val>\r\n");
      this.outputFileWriter.write("</output>\r\n");
      //test:KurzAusgabe
      /*this.outputFileWriter.write(((int) FileReader.getTime()) + " " +
                                                    ((int) this.val) + "\r\n");*/
      this.outputFileWriter.flush();
      System.out.println("  -ContextObject: printout: " +
          FileReader.getTime() + " " + this.name);
    } catch (IOException eio) {
      System.out.println("  ContextObject Error Outputfile: " +
          eio.getMessage());
      eio.printStackTrace();
    }
  }

  public double getVal() {
    return this.val;
  }
/*3*/
  public String getName() {
    return this.name;
  }

  public void setName(String name) {
    this.name = name;
  }
```

```
  public int getLevel() {
    return this.level;
  }

  public void setLevel(int level) {
    this.level = level;
  }

  public double getDefaultVal() {
    return this.defaultVal;
  }

  public void setDefaultVal() {
    this.defaultVal = defaultVal;
  }
}

//Unnötige Programmteile durch statische definstance-facts

/*1
import java.beans.PropertyChangeSupport;
import java.beans.PropertyChangeListener;*/
/*2
      int tmp = this.val;
       pcs.firePropertyChange("val", new Integer(tmp),
        new Integer(val));*/
/*3
  private PropertyChangeSupport pcs = new PropertyChangeSupport(this);

  public void addPropertyChangeListener(PropertyChangeListener pcl) {
    pcs.addPropertyChangeListener(pcl);
  }

  public void removePropertyChangeListener(PropertyChangeListener pcl) {
    pcs.removePropertyChangeListener(pcl);
  }*/
```

ChannelContainer

```
import java.util.*;

/** Die Klasse <code>ChannelContainer</code> verwaltet die ChannelObjects
 * entsprechend dem Container-Muster. Dabei wird mit den Methoden insertChannel
 * und getChannel auf die HashMap zugegriffen.
 * @author Steiner Engelbert
 * @version 1.10
 */
public class ChannelContainer {

  private static HashMap channelObjects = new HashMap(10);

  public static void insertChannel(String name, ChannelObject channel) {
    channelObjects.put(name, channel);
    //test//System.out.println("  *ChannelContainer: insertChannel: " + name);
  }

  public static ChannelObject getChannel(String name) {
    ChannelObject channel = (ChannelObject) channelObjects.get(name);
    /*Ob Channel mit diesem Namen gefunden wurde, muß vom Aufrufer
      überprüft werden*/
    //test//System.out.println("  -ChannelContainer: getChannel: "+name);
    return channel;
  }
```

```
  public static Set getAllChannelObjectKey() {
    return channelObjects.keySet();
  }

  public static Collection getAllChannelObjectValue() {
    return channelObjects.values();
  }
}
```

ContextContainer

```
import java.util.*;

/** Die Klasse <code>ContextContainer</code> verwaltet die ContextObjects
 * entsprechend dem Container-Muster. Dabei wird mit den Methoden insertContext
 * und getContext auf die HashMap zugegriffen. Weiters stehen die
 * Methoden <code>setLevelDefault</code> und <code>setOverLevelDefault</code>
 * zur Verfügung, um generisches Ansprechen der ContextObjects zu ermöglichen.
 * @author Steiner Engelbert
 * @version 1.10
 */
public class ContextContainer {

  private static HashMap contextObjects = new HashMap(10);

  public static void insertContext(String name, ContextObject context) {
    //test//System.out.println("  +ContextContainer: insertContext: " + name);
    contextObjects.put(name, context);
  }

  public static ContextObject getContext(String name) {
    //test//System.out.println("  -ContextContainer: getContext: "+name);
    ContextObject context = (ContextObject) contextObjects.get(name);
    if (context == null) {
      System.out.println("  x ContextContainer: getContext: NoSuchElement " + name);
      return null;
    } else {
      return context;
    }
  }

  public static Iterator getAllContextObjectValue() {
    return contextObjects.values().iterator();
  }

  /** Die Methode <code>setLevelDefault</code> setzt alle ContextObjects eines
   * bestimmten Levels auf deren Defaultwerte zurück.
   * @param <code>level</code> gibt den Level der zurückzusetzenden ContextObjects an.
   */
  public static void setLevelDefault(int level) {
    //System.out.println("  ContextContainer: setLevelDefault " + level);
    for (Iterator allContexts = ContextContainer.getAllContextObjectValue();
         allContexts.hasNext();) {
      ContextObject conObj = (ContextObject) allContexts.next();
      if (conObj.getLevel() == level) {
        conObj.setVal(conObj.getDefaultVal());
      }
    }
  }

  /** Die Methode <code>setLevelDefault</code> setzt alle ContextObjects ab einem
   * bestimmten Level auf deren Defaultwerte zurück.
```

```
   * @param <code>level</code> gibt die Level der zurückzusetzenden
   * ContextObjects an.
   */
  public static void setOverLevelDefault(int level) {
    /*zuerst hier manuelles reset der ContextObjects und dann automatisches
      reset durch reset-Befehl im FileReader der KnowledgeBase-ContextObjects*/
    //test//System.out.println("  ContextContainer: setOverLevelDefault " + level);
    for (Iterator allContexts = ContextContainer.getAllContextObjectValue();
         allContexts.hasNext();) {
      ContextObject conObj = (ContextObject) allContexts.next();
      /*test//System.out.println(
                      "  -ContextContainer setOverLevelDefault: ConObj Name: "
                        + conObj.getName() + " Level: " + conObj.getLevel());*/
      if (conObj.getLevel() >= level) {
        conObj.setVal(conObj.getDefaultVal());
      }
    }
  }

  public static Set getAllContextObjectKey() {
    return contextObjects.keySet();
  }
}
```

Konfig

```
;;Kommentar
;;(watch all)
;;(watch compilations)
;;(watch activations)
;;(watch rules)
;;(watch facts)

(defclass context ContextObject)
;;wbeMoveUp wbeMoveDown wrDown wrUp genHole circ

;;(printout t " Rules: ContextObjects" crlf crlf)
(new ContextObject "wbeMoveUp" 1 0.0)
(new ContextObject "wbeMoveDown" 1 0.0)
(new ContextObject "wrDown" 1 0.0)
(new ContextObject "wrUp" 1 0.0)
(new ContextObject "genHole" 1 0.0)
(new ContextObject "circ" 1 0.0)

;;(printout t crlf " Rules: ChannelObjects" crlf crlf)
(defclass channel ChannelObject)
;;mfia woba hkla dmea bpos
(new ChannelObject "dmea" 20 (create$ 4))
(new ChannelObject "bpos" 5 (create$ 4))
(new ChannelObject "hkla" 2 (create$ 0))
(new ChannelObject "mfia" 2 (create$ 0))
(new ChannelObject "woba" 2 (create$ 0))
```

Rules

```
;;(printout t crlf " Rules: DefGlobals" crlf)
;;defglobal als Grenzwerte
(defglobal ?*hklaval* = 150)
(defglobal ?*bposslpposbgn* = 0.15)
(defglobal ?*bposslpposend* = 0.5)
(defglobal ?*bposslpnegbgn* = -0.5)
(defglobal ?*bposslpnegend* = -0.15)

;;(printout t " Rules: Rules" crlf crlf)
(defrule state-wbeMoveDown
  (context (name "mfiaval") (val 0.0))
  (context (name "wobaval") (val 0.0))
  (context (name "hklaval") (val ?hklaval))
  (context (name "bposslp") (val ?bposslp))
  (test (> ?hklaval ?*hklaval*))
  (test (and (> ?bposslp ?*bposslpnegbgn*) (< ?bposslp ?*bposslpnegend*)))
  (context (name "wbeMoveDown") (OBJECT ?context))
  =>
  (set ?context val 6.0)
;;  (printout t "  =>state wbeMoveDown" crlf)
)

(defrule state-wbeMoveUp
  (context (name "mfiaval") (val 0.0))
  (context (name "wobaval") (val 0.0))
  (context (name "hklaval") (val ?hklaval))
  (context (name "bposslp") (val ?bposslp))
  (test (> ?hklaval ?*hklaval*))
  (test (and (> ?bposslp ?*bposslpposbgn*) (< ?bposslp ?*bposslpposend*)))
  (context (name "wbeMoveUp") (OBJECT ?context))
  =>
  (set ?context val 5.0)
;;  (printout t "  =>state wbeMoveUp" crlf)
)

(defrule state-wrDown
  (context (name "wobaval") (val 0.0))
  (context (name "mfiaval") (val ?mfiaval))
  (context (name "hklaval") (val ?hklaval))
  (context (name "bposslp") (val ?bposslp))
  (test (> ?mfiaval 0))
  (test (> ?hklaval ?*hklaval*))
  (test (and (> ?bposslp ?*bposslpnegbgn*) (< ?bposslp ?*bposslpnegend*)))
  (context (name "wrDown") (OBJECT ?context))
  =>
  (set ?context val 4.0)
;;  (printout t "  =>state wrDown" crlf)
)

(defrule state-wrUp
  (context (name "wobaval") (val 0.0))
  (context (name "mfiaval") (val ?mfiaval))
  (context (name "hklaval") (val ?hklaval))
  (context (name "bposslp") (val ?bposslp))
  (test (> ?mfiaval 0))
  (test (> ?hklaval ?*hklaval*))
  (test (and (> ?bposslp ?*bposslpposbgn*) (< ?bposslp ?*bposslpposend*)))
  (context (name "wrUp") (OBJECT ?context))
  =>
  (set ?context val 3.0)
;;  (printout t "  =>state wrUp" crlf)
)
```

```
(defrule state-genHole
  (context (name "mfiaval") (val ?mfiaval))
  (context (name "wobaval") (val ?wobaval))
  (context (name "dmeaslp") (val ?dmeaslp))
  (test (> ?mfiaval 0))
  (test (> ?wobaval 0))
  (test (> ?dmeaslp 0))
  (context (name "genHole") (OBJECT ?context))
  =>
  (set ?context val 1.0)
;;  (printout t "  =>state genHole" crlf)
)

(defrule state-circ

 (context (name "mfiaval") (val ?mfiaval))
  (test (> ?mfiaval 0))
  (context (name "circ") (OBJECT ?context))
  =>
  (set ?context val 2.0)
;;  (printout t "  =>state circ" crlf)
)
```

Anhang B

Verzeichnisse

Abbildungsverzeichnis

Literaturverzeichnis

[bahei99] Balzert, Heide: Lehrbuch der Objektmodellierung: Analyse und Entwurf; Spektrum Akademischer Verlag Heidelberg Berlin, 1999.

[bahel99] Balzert, Helmut: Lehrbuch Grundlagen der Informatik: Konzepte und Notationen in UML, Java und C++, Algorithmik und Software-Technik, Anwendungen; Spektrum Akademischer Verlag Heidelberg Berlin, 1999.

[chsu94] Chen, Zhisong; Suen, Y. Ching: Measuring the Complexity of Rule-Based Expert Systems; Expert Systems With Applications, Vol.7, No.4, pp.467-481, 1994.

[daja91] Dawant, Benoit M.; Jansen, Ben H.: Coupling Numerical and Symbolic Methods for Signal Interpretation; IEEE Transactions on Systems, Man, Cybernetics, Vol.21, No.1, pp. 115-124, January/February 1991.

[dur94] Durkin, John; Expert systems: design and development; Prentice-Hall, Inc., 1994.

[fri01] Friedman-Hill, Ernest J.: Jess, The Java Expert System Shell; http://herzberg.ca.sandia.gov/jess/docs/index.html; (Stand 07.12.2001)

[gfh90] Gottlob, Georg; Frühwirt, Thomas; Horn, Werner (Herausgeber): Expertensysteme. Springer Verlag Wien; 1990.

[gup94] Gupta, Uma G.: The Academic Quality of AI Journals and the Role of AI in the MIS Curriculum: Perspectives of Business Faculty; Expert Systems with Applications, Vol.7, No.4, pp.581-588, 1994.

[hale90] Hartmann, Dietrich; Lehner, Karlheinz: Technische Expertensysteme; Springer Verlag Berlin Heidelberg, 1990.

[hay85] Hayes-Roth, Frederick: Rule-Based Systems; Communications of the ACM, Vol.28, No.9, pp.921-932, September 1985.

[hel96] Helbig, Hermann: Künstliche Intelligenz und automatische Wissensverarbeitung; Verlag Technik, 1996.

[hoco01] Horstmann, Cay S.; Cornell, Gary: Core Java: Band1-Grundlagen; Markt+Technik Verlag, 2001.

[hoco02] Horstmann, Cay S.; Cornell, Gary: Core Java: Band2-Expertenwissen; Markt+Technik Verlag, 2002.

[lug01] Luger, George F.: Künstliche Intelligenz: Strategien zur Lösung komplexer Probleme; Pearson Studium, 2001.

[met91] Mettrey, William: A Comparative Evaluation of Expert System Tools; IEEE Computer, pp.19-31, February 1991.

[paxi91] Pau, L. F.; Xiao X.:A Knowledge-Based Sensor Fusion Editor; IEEE Transactions on Systems, Man, and Cybernetics, Vol.21, No.5, pp. 1251-1259, September/October 1991.

[pepr00] Perraju, T. S.; Prasad, B. E.; Inference analysis in multiple rule firing systems; Knowledge-Based Systems 13, pp. 171-176, 2000.

[pri89] Priha, I.: Integrated knowledge systems; Artificial Intelligence in Engineering, Vol.4, No.2, pp. 70-78, 1989.

[pup91] Puppe, Frank: Einführung in Expertensysteme. Springer Verlag Berlin Heidelberg, 1991.

[scle86] Schnupp, Peter; Leibrandt, Ute: Expertensysteme. Nicht nur für Informatiker; Springer Verlag Berlin Heidelberg, 1996.

[ste02] Steyer, Ralph: Java2; Markt+Technik Verlag 2002.

[wiei01] Witten, Ian; Eibe, Frank: Data Mining; Hanser Verlag, 2001.

[win92] Winston, Patrick Henry: Artificial Intelligence; Addison-Wesley Verlag, 1992.