DISKI

Dissertationen zur Künstlichen Intelligenz

Mit Unterstützung des Fachbereichs 1 „Künstliche Intelligenz" der Gesellschaft für Informatik e.V. herausgegeben von

Eine Architektur zur Anwendung symbolischer Lernverfahren in der Robotik

von
Jürgen Kreuziger

infix

Dipl.-Inform. Jürgen Kreuziger
Universität Karlsruhe
Institut für Prozeßrechentechnik und Robotik
Kaiserstr. 12
76128 Karlsruhe

Zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften
von der Fakultät für Informatik
der Universität Karlsruhe (Technische Hochschule)
genehmigte Dissertation

Erster Gutachter: Prof. Dr.-Ing. R. Dillmann
Zweiter Gutachter: Prof. Dr. rer. nat. A Waibel

Tag der mündlichen Prüfung: 24. Mai 1994

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Kreuziger, Jürgen:
Eine Architektur zur Anwendung symbolischer Lernverfahren
in der Robotik / von Jürgen Kreuziger. – Sankt Augustin :
Infix, 1994
(Dissertationen zur künstlichen Intelligenz ; 68)
Zugl.: Karlsruhe, Univ. Diss., 1994
ISBN 3-929037-68-8
NE: GT

Reproduziert von einer Druckvorlage des Autors
Umschlaggestaltung: Art und Media, Bonn
Druck und Verarbeitung: Hundt Druck GmbH, Köln
Printed in Germany

ISSN 0941-5769
ISBN 3-929037-68-8

Geleitwort

Charakteristisch für heutige industrielle oder autonome Robotersysteme ist die Komplexität ihrer Programmierung beziehungsweise des initialen Wissensaufbaus und die fehlende automatische Verbesserung des Systemwissens während des Einsatzes des Robotersystems. Zukünftige Anwendungen von Manipulatoren und mobilen Systemen für Aufgaben in natürlichen Umgebungen werden nur dann möglich sein, wenn sowohl praktikable den Menschen einbeziehende Programmierparadigmen entwickelt werden als auch die Integration von Adaptions- und Lerntechniken in derartige Systeme erreicht wird. Im Forschungsgebiet des Maschinellen Lernens wird seit vielen Jahren an der Entwicklung von Adaptions- und Lernverfahren gearbeitet. Aufgrund der hohen Komplexität der Anwendungsdomäne Robotik wurden bisher jedoch nur wenige Versuche unternommen, Verfahren des Maschinellen Lernens auch in der Robotik einzusetzen.

Die vorliegende Arbeit hat die Definition einer Systemarchitektur für Robotersysteme zum Gegenstand, die Lernkomponenten für die unterschiedlichen Aspekte einer Robotersteuerung integriert. Die Architektur basiert auf einer Menge kooperierender, aber auch konkurrierender Teilsysteme, die jeweils spezifische Wissensrepräsentationen und zugehörige Lernverfahren einsetzen können. Im Gegensatz zu bisherigen Arbeiten auf diesem Gebiet findet keine isolierte Betrachtung des Lernvorgangs statt, sondern die Lernprozesse sind in den gesamten Systemablauf eingebettet. Damit ist eine Integration von Planung, Ausführung, Reaktion und Lernen in einem einheitlichen Formalismus möglich.

Innerhalb der Systemarchitektur werden als Schwerpunkte der Arbeit das Lernen von Regelwissen aus Beispielen, das Lernen von generischen Objektbeschreibungen aus 2D-Kantenbildern und das Lernen von für die Manipulation durch Roboter relevanten statischen und dynamischen Objekteigenschaften untersucht. Mit diesen Verfahren ist das System in der Lage, neue Manipulationsaufgaben innerhalb einer Manipulationsklasse zu lösen, neue Objekte mit Hilfe der gelernten Modellhierarchie zu klassifizieren und Eigenschaften dieser Objekte zu bestimmen. Für alle drei Lernziele wurden neue Methoden entwickelt bzw. existierende Verfahren erweitert, da derartige Fragestellungen in bisherigen Arbeiten ausgeklammert oder nur sehr vereinfacht untersucht wurden.

Die Akquisition von Regelwissen geschieht in dem entwickelten System entweder automatisch oder durch Interaktion mit dem Benutzer durch geeignete Anwendung von Generalisierungs- und Spezialisierungsoperatoren. Durch die Einbindung des Benutzers als Tutor wurde ein für die Robotik wichtiges neues Programmierparadigma des „Programmierens durch Vorführen" entworfen. Zusätzlich zu dem gelernten Handlungswissen kann das System eine Klassifikation der zu manipulierenden Objekte durchführen. Zu diesem Zweck wurde in der Arbeit ein unüberwachtes Lernverfahren weiterentwickelt, so daß

eine automatische Klassifikation von einfach strukturierten Objekten mit numerischen, nominalen und relationalen Attributen ermöglicht wird. Die Methode zur Durchführung von durch taktile Sensoren unterstützten Experimenten zur Bestimmung weiterer Objektinformationen oder des Verhaltens eines Objektes bei einer Aktion führt zu einer genaueren Beschreibung anfangs unbekannter Manipulationsobjekteigenschaften.

Die durchgeführte Realisierung der in der Arbeit entwickelten Systemarchitektur führte zu einem lernenden Steuerungssystem, das mit einem PUMA 260–Roboter experimentell evaluiert wurde. Die Ergebnisse der vorliegenden Arbeit sind nicht nur für die Robotik von großem Interesse, sondern liefern auch einen Fortschritt für die praktische Umsetzung der Grundlagen auf dem Gebiet des Maschinellen Lernens.

Karlsruhe, im Juli 1994 R. Dillmann

Vorwort

Die vorliegende Veröffentlichung ist eine überarbeitete Fassung meiner von der Fakultät für Informatik der Universität Karlsruhe genehmigten Dissertation. Sie entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Prozeßrechentechnik und Robotik der Universität Karlsruhe. Die Arbeit wurde im Rahmen der Forschungsarbeiten des Sonderforschungsbereichs 314, Projekt R2 „Lernverfahren für autonome, mobile Robotersysteme" der Deutschen Forschungsgemeinschaft erstellt.

Ich bedanke mich ganz herzlich bei meinem Doktorvater Prof. Dr.-Ing. R. Dillmann, der mich während der gesamten Arbeit sehr unterstützt und mich stets ermutigt hat, dieses „Neuland" der lernenden Robotersysteme immer weiter zu betreten.

Bei Herrn Prof. Dr.rer.nat. A. Waibel bedanke ich mich sehr für das Interesse an meiner Arbeit und die Übernahme des Korreferats.

Allen Mitarbeitern des Instituts für Prozeßrechentechnik und Robotik danke ich für die tolle Atmosphäre bei uns, in der es jeden Tag wieder Spaß gemacht hat zu arbeiten. Ganz besonders dankbar bin ich Michael Kaiser und Xiaoqing Cheng für die wertvollen Anregungen und für die viele Zeit, die Ihr Euch genommen habt, um meine Arbeit Korrektur zu lesen.

Ich bedanke mich auch insbesondere bei „meinen" Studentinnen und Studenten Antje Bredehöft, Stephan Cord, Thomas Diederich, Michael Fankanowsky, Markus Hauser, Andreas Lewark, Martin Stockwald, Steffen Tritschler, Ngoc Long Tu, Berthold Weismann und Walter Wenzel für die vielen Stunden der Diskussion und Euer Engagement bei den Implementierungsarbeiten und den Experimenten.

Zum Schluß ein großes Dankeschön an Ursula und meine Eltern – für alles.

Karlsruhe, im Juli 1994 Jürgen Kreuziger

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Unter dem Begriff *Roboter* werden oft nur Manipulatoren, d.h. Roboterarme, im industriellen Einsatz verstanden, z.B. für Schweiß-, Lackier-, Einlege- oder Montageaufgaben. Diese Maschinen sind programmierbar und können eine feste Folge von Einzeloperationen mit hoher Geschwindigkeit und Positioniergenauigkeit wiederholt durchführen. Ihre Programmierung erfolgt entweder durch exaktes Abfahren der gewünschten Bahn oder durch Verwendung einer Roboterprogrammiersprache, in der die nötigen Bewegungsbefehle angegeben werden. Diese Tätigkeit wird im allgemeinen von Experten durchgeführt. Ein Merkmal derartiger Roboter ist, daß das vorgegebene Programm im wesentlichen starr abläuft. Neben einem initialen Abgleich zwischen dem Modell, auf dem die Programmierung basierte, und der Realität in Form einer Anwesenheitsüberprüfung oder exakten Positionsbestimmung von zu manipulierenden Objekten kann häufig als Reaktion auf ein unerwartetes externes Ereignis nur ein „Nothalt" ausgeführt werden. Aus diesem Grund sind die Anforderungen an die Umgebung des Roboters im allgemeinen sehr restriktiv, und der Aufwand für deren Bereitstellung in einer Labor- oder Fabrikumgebung ist entsprechend hoch.

Bei der Anwendung von *Telemanipulatoren* liegen nahezu umgekehrte Verhältnisse vor. Der Manipulator führt keine sich wiederholende Tätigkeit aus und wird daher nicht programmiert, sondern durch einen menschlichen Bediener ferngesteuert. Die Anforderungen der Aufgaben bezüglich Geschwindigkeit und Positioniergenauigkeit müssen dementsprechend geringer sein. Der Bediener sollte jedoch auf die jeweils aktuelle Situation mit einer möglichst adäquaten Aktion reagieren können. Zu diesem Zweck werden ihm – eventuell aufbereitete – Informationen über die Manipulationsumgebung zur Verfügung gestellt, z.B. Videobilder, Kraftmeßwerte oder/und Simulationsergebnisse. Durch die Einbindung menschlicher Planungs- und Entscheidungsfähigkeiten in den Zyklus zwischen Sensormessung und Aktionsausführung sind auch Manipulationen in natürlichen, komplexeren Umgebungen möglich. Nachteil der Telemanipulation ist, daß jede Aufgabe wieder vollständig durch den menschlichen Bediener gelöst werden muß.

Zukünftige, wichtige Anwendungsgebiete für Robotersysteme sind der Einsatz in natürlichen Umgebungen für Arbeiten im Weltall oder unter Wasser, für Demontage- oder all-

gemein Recyclingaufgaben, als Haushalts- oder Büroroboter oder als Unterstützung für bedürftige Menschen. Für diese Aufgaben sind die Methoden der Industrierobotik und der Telemanipulation nur sehr bedingt geeignet. Hauptprobleme sind die Inflexibilität des einen und die ständige Benutzerbeanspruchung des anderen Ansatzes.

Alle Aufgaben zeichnen sich durch hohe Anforderungen an die sensorbasierte Informationsaufnahme, an Informationsverarbeitungsprozesse und an Aktoren zur aktiven Manipulation der Umwelt aus. Seit über 20 Jahren wird versucht, diesen ständigen Zyklus automatisch, d.h. *autonom*, durchzuführen. Die Idee ist, daß das autonome System selbständig eine gestellte Aufgabe in entsprechende Teilschritte zerlegt und Aktionen ausführt, die zu einer Lösung der Aufgabe führen. Dabei wird jedoch meist von Annahmen ausgegangen, die nur in einer abgegrenzten, künstlichen Welt erfüllt werden können, wie z.B. dem Vorliegen einer exakten Modellbildung, dem Verfolgen einer einzigen Aufgabe und der nur eingeschränkten Notwendigkeit eines reaktiven Verhaltens.

Zur Lösung der oben genannten Aufgabengebiete sind neben entsprechend geeigneten Sensoren und Aktoren vor allem perzeptive Komponenten zur Weiterverarbeitung und Interpretation der sensorischen Signale sowie planende Komponenten zur Ableitung aktorischer Signale notwendig. Wichtig ist, daß ein autonomer Roboter kooperierend arbeiten kann, d.h., bestimmte Aufgaben löst er selbständig, für andere kann er mit dem Bediener kommunizieren.

Sämtliche Komponenten des Systems benutzen bei ihren Entscheidungen *Wissen*, das entweder implizit oder explizit vorliegt. Implizites Wissen steckt beispielsweise in einem fest vorgegebenen Algorithmus, der aus Eingabewerten eine bestimmte Ausgabe erzeugt. Wissen wird als explizit bezeichnet, wenn es unabhängig von der Problemlösungskomponente vorliegt und damit leichter erweitert oder modifiziert werden kann.

Probleme sind in beiden Fällen die *Akquisition* und *Modifikation* des Wissens, d.h. wie Wissen in das System eingebracht und wie es verändert werden kann. Ziel ist ein korrektes und vollständiges Wissen, d.h. Wissen, das insgesamt „richtig" ist und alle möglichen Fälle behandelt. In einem System von der Komplexität eines autonomen Roboters muß geklärt werden, wie und wann eine Akquisition und Modifikation von Wissen stattfinden soll.

Die Arbeiten der letzten Jahre haben gezeigt, daß eine vollständige Akquisition von Wissen vor dem eigentlichen Einsatz des autonomen Systems nicht möglich ist. Entweder bleibt die Menge der bearbeitbaren Aufgaben stark eingeschränkt oder die Komplexität der einzelnen Aufgaben muß entsprechend niedrig sein. Diese Schwierigkeiten der Wissensakquisition treten auch in allen anderen Gebieten, in denen wissensbasierte Systeme eingesetzt werden, auf.

Eine Lösung dieses Problems ist der Einsatz von ausgefeilten Methoden zur *Benutzerinteraktion*. Diese können reichen von Frage-Antwort-Dialogen, um Entscheidungen zu fällen, die das System selbst nicht ableiten kann, bis zur Demonstration von Lösungsmöglichkeiten für eine Aufgabe und dem entsprechenden Aufbau von Wissen. Dieses *Programmieren durch Vorführen* stellt die natürlichste Art dar, insbesondere aktorisches Wissen zu vermitteln. Auf diese Art lernen auch Menschen das ganze Spektrum von elementaren Fertigkeiten bis hin zu Handlungsfolgen. Allerdings stellt dieses Programmierparadigma hohe Anforderungen an ein System, das zum einen die Benutzervorführungen interpretieren und zum anderen das vorhandene Wissen so modifizieren können muß, daß das System aus der Vorführung „profitiert".

Ein Kennzeichen der meisten bisherigen Ansätze zum Thema autonome Systeme ist, daß eine solche Modifikation, d.h. Verbesserung, von Wissen nicht stattfindet. Vielmehr wird für eine Aufgabe immer wieder neu eine Lösung berechnet, eventuell treten immer wieder die gleichen Fehler auf, unabhängig davon, ob die Aufgabe zum ersten oder bereits zum wiederholten Mal ausgeführt wird - eine *Adaptions-* und/oder *Lernfähigkeit* ist also nicht gegeben. Gerade Lernen, d.h. die Akquisition und Verbesserung von Wissen, ist aber Grundlage jeden „intelligenten" Verhaltens und ist notwendige Voraussetzung für die Erschließung neuartiger Anwendungen für Roboter allgemein. Ein Lernvorgang kann sowohl durch Beobachtung und Interaktion mit dem Benutzer als auch durch Analyse der gefundenen Lösung und der entstandenen Probleme stattfinden. In Robotersystemen kann sowohl aktorisches Wissen, z.B. zur Bestimmung der nächsten durchzuführenden Aktion, als auch sensorisches Wissen, z.B. zur Objektklassifikation, durch Lernvorgänge aufgebaut und sukzessive verbessert werden.

Die Forderung nach höherer Flexibilität von autonomen Systemen, nach neuen Programmiermöglichkeiten und vor allem nach dem Einsatz von Lernverfahren führt zu völlig neuen Anforderungen und Schwierigkeiten bei dem Entwurf einer geeigneten Architektur für ein solches Robotersystem. Die Formulierung dieser Anforderungen sowie die Entwicklung einer diese erfüllenden Systemarchitektur und geeigneter Lernmethoden sind Gegenstand dieser Arbeit.

Die Ergebnisse, die in dieser Arbeit mit Hinblick auf autonome Systeme erzielt wurden, lassen sich auch auf die anfangs erwähnten Gebiete der industriellen Robotik und der Telerobotik übertragen. Die Anforderungen sind dort im Grunde eine Teilmenge der hier untersuchten, so daß bestimmte Lösungen eventuell nur in vereinfachter Form benötigt werden.

1.2 Problemstellung

Heutige Architekturen für autonome Robotersysteme erfüllen die genannten Forderungen nach Flexibilität, Adaptions- und Lernfähigkeit für zukünftige autonome Systeme nur zum Teil. Aus diesem Grund müssen grundlegende Konzepte erarbeitet werden, mit denen die gewünschten Eigenschaften erreicht werden können. Insbesondere soll ein System sehr *flexibel* agieren bzw. reagieren können. Eine zeitlich getrennte Betrachtung von Planung, Ausführung und Umweltmodellierung ist daher nicht möglich, vielmehr ist ein integrierter Ansatz, der auch Lernvorgänge ermöglicht, notwendig. Dies gilt vor allem bei der Behandlung von Unsicherheiten, unerwarteten Ereignissen und bei der Manipulation in teilweise unbekannten Umgebungen.

In bisherigen Ansätzen wurde häufig gefordert, daß sich die Behandlung von unerwarteten Ereignissen im wesentlichen auf nicht geplante Zustände beschränkt, die durch Unsicherheiten in Sensorik, Aktorik und Modellen verursacht werden. Durch eine Analyse des „Fehlerzustandes" und entsprechende Fehlerbehebungsmaßnahmen soll dann eine Fortsetzung des Normplanes möglich werden. Tatsächlich ist in einer natürlichen Umwelt, in der auch andere aktive Komponenten vorkommen und häufig unerwartete Ereignisse auftreten, ein ständig reaktives Verhalten des Systems und eventuell sogar die gleichzeitige Betrachtung mehrerer konkurrierender Ziele notwendig. Parallel dazu muß das System strategisch die Lösung einer Aufgabe verfolgen.

Eine andere häufig gestellte Forderung ist, daß für jedes Objekt der Umgebung ein exaktes 3D-Modell und – eventuell nach einer initialen Positionsbestimmung der Objekte – ein Umweltmodell zur Verfügung stehen, wodurch dann viele Probleme auf rein geometrisches Schlußfolgern zurückgeführt werden können. Diese Annahme ist jedoch in natürlichen Umgebungen und bei Aufgaben, in denen teilweise unbekannte Objekte in der Manipulationsumgebung vorliegen können bzw. durch externe Einflüsse in sie gelangen, nicht zulässig. Daneben ist in den oben genannten zukünftigen Anwendungen eine vollständige Modellierung aller Objekte a priori auch weder möglich noch immer notwendig. Eine Lösungsmöglichkeit ist die Verwendung generischer, prototypischer Objektmodelle unterschiedlichen Abstraktionsgrades, die eine ganze Klasse von ähnlichen Objekten beschreiben und je nach Aufgabenbedarf unterschiedlich detailliert werden. Damit wird eine Behandlung unbekannter Objekte möglich, indem die ähnlichste bekannte Objektkategorie bestimmt und dementsprechend agiert wird.

Für alle Bereiche der Perzeption und Planung ist das im System abgebildete Wissen entscheidend für die Leistungsfähigkeit des autonomen Systems. Dabei ist die Definition einer geeigneten Wissensrepräsentation genauso wichtig wie die Bestimmung passender Verarbeitungsmechanismen. Das Wissen soll durch Lernen modifiziert und erweitert werden, es muß also in der jeweils gewählten Repräsentation auch lernbar sein. Eine möglichst homogene Darstellung der verschiedenen Wissensinhalte erleichtert diese Aufgabe und ermöglicht vor allem möglichst einheitliche Lernverfahren.

Lernfähigkeit ist aber nicht nur ein „Modul", das zu einem bestehenden System hinzugenommen werden kann - wenn es sich nicht nur um das Lernen eines Klassifikators, sondern z.B. um das Lernen von Regelwissen handelt. Vielmehr muß die gesamte Architektur und Repräsentation dahingehend entwickelt werden. Genau dies ist bisher jedoch noch nie für ein Manipulationssystem durchgeführt worden.

In einem derart komplexen System wie einem autonomen Roboter lassen sich vielfältigste Ansatzpunkte für Lernverfahren finden. Daher müssen die Lernaufgaben und Lernziele sehr genau analysiert und formuliert werden. Beispiele dafür sind der automatische Aufbau von Klassifikatoren zur Auswertung von Sensorsignalen, das Lernen von aktorischem Wissen, d.h. welche Aktion sollte in einer bestimmtem Situation als nächstes ausgeführt werden, oder das Lernen von elementaren Fertigkeiten (engl. skills), d.h. wie kann eine bestimmte Teilaufgabe, z.B. ein Fügevorgang, sensorbasiert gelöst werden.

Bisherige Arbeiten, die sich mit dem Einsatz von Techniken des Maschinellen Lernens (ML) für reale Roboter beschäftigen, betrachten zumeist nur einen sehr eng begrenzten Teilaspekt einer Steuerungsarchitektur. Wie das erlernte Wissen im gesamten Kontext angewendet und vor allem die benötigten Beispiele während eines Systemeinsatzes extrahiert werden können, wurde fast nie berücksichtigt. Eine vergleichende Übersicht und Klassifikation dieser Ansätze fehlt bisher völlig.

Im Bereich des Maschinellen Lernens wurden in den letzten Jahren eine Reihe von Methoden entwickelt, wie für bestimmte Repräsentationen einige Lernaufgaben - vor allem das Lernen von Klassifikatoren - gelöst werden können. Leider sind die gestellten Anforderungen an die Repräsentation derart restriktiv, daß diese Lernmethoden nicht ohne Modifikation auf die oben angesprochenen Probleme übertragen werden können. Dies ist insbesondere auch deshalb schwierig, weil im ML Annahmen über Art, Umfang und

Verfügbarkeit von Daten und Beispielen gemacht werden, die so in der Robotik nicht zutreffend sind. Ein weiteres wichtiges Problem für das Erlernen von aktorischem Wissen ist die Tatsache, daß in entsprechenden ML-Arbeiten fast ausschließlich die Planung betrachtet wurde, ohne die Ausführung oder eine Rückwirkung derselben auf den Lernvorgang zu berücksichtigen.

Außerdem sind für die notwendigen Weiterentwicklungen der ML-Methoden prinzipielle Probleme von Lernverfahren zu lösen. Dies ist erstens die Auswahl der günstigsten Modifikation des bestehenden Wissens, um die neue Information einarbeiten zu können (Vorzugskriterium, engl. bias) . Zweitens muß bestimmt werden, wie auf das wirklich relevante Wissen in der Menge der vorliegenden Informationen fokussiert werden kann. Dies wird in ML-Arbeiten fast nie untersucht, da ohnehin nur das relevante Wissen vorgegeben wird. Schließlich muß in irgendeiner Form der Lernerfolg bestimmt werden. Während dies beim überwachten Lernen eines Klassifikators durch die Messung der Klassifikationsgenauigkeit auf einer Testmenge möglich ist, ist eine Möglichkeit zur Messung des Lernerfolges beim Lernen von aktorischem Wissen nicht offensichtlich. Insbesondere läßt sich der Nutzen, der durch das Programmierparadigma des Programmierens durch Vorführen, d.h. durch das Einbringen von Benutzerlösungen für eine Aufgabe, erbracht wird, nicht einfach an Kennzahlen ablesen. Vielmehr ist dadurch eine neuartige, komfortablere und mächtigere Möglichkeit zur Wissensakquisition gegeben.

Für die Benutzerinteraktion stellt sich das Problem, wie die Art der Vorführung gewählt werden soll. Eine Variante ist, daß der Lehrer mit Hilfe von Eingabemedien oder durch direkte Bewegung des Roboterarms die Aufgabe unter Verwendung des realen Roboters durchführt. Die andere Variante besteht darin, daß der Lehrer unter Verwendung seiner eigenen Sensorik und Aktorik die Aufgabe löst. In beiden Fällen werden die Aktionen vom System aufgezeichnet. Die Interpretation dieser Aufzeichnungen und die Integration in das Systemwissen können dann teilweise automatisch, teilweise benutzergestützt ablaufen. Die Interaktionsmöglichkeiten reichen dabei von grafischen Hilfsmitteln zur Verarbeitung der Daten bis hin zu Frage-Antwort-Dialogen zwischen System und Benutzer.

Ein wichtiger Aspekt bei Benutzervorführungen ist der Grad der Wiederverwendbarkeit des aufgebauten Wissens. Während bei traditionellen Teach-In-Methoden in der Industrie das „Wissen" nur für die eine zu lösende Aufgabe verwendet werden kann, sollte das Wissen in einem fortgeschritteneren Ansatz natürlich auch für andere, ähnliche Aufgabenstellungen eingesetzt werden können.

Insgesamt stellen sich also für die Anwendung von Lernverfahren in Robotersystemen vor allem die Fragen nach einer geeigneten Architektur, adäquaten Wissensrepräsentationsmechanismen und geeigneten Lernverfahren. Für den letzten Punkt sind insbesondere die Erweiterung von ML-Techniken auf die komplexen Problemstellungen der Robotik wichtig und die Entwicklung einer prinzipiellen Methodik, wie ein Robotersystem aus eigenen Erfahrungen, durch Experimente oder durch Benutzerinteraktion Wissen aufbauen kann, das in zukünftigen Aufgaben angewendet werden und durch weitere Lernschritte sukzessive verbessert werden kann.

1.3 Zielsetzung

Ziel dieser Arbeit ist der erstmalige Entwurf und die Realisierung einer Systemarchitektur für ein lernendes Robotersteuerungssystem. Dazu sollen die Anforderungen an eine geeignete Systemarchitektur herausgearbeitet und bereits bekannte Konzepte aus der Literatur analysiert werden. Dies soll zu der Entwicklung einer integrierten Lernarchitektur für den Teilbereich der Manipulation führen, die als Basis für die anderen Schwerpunkte der Arbeit dienen soll.

Zu diesem Zweck ist neben geeigneten Wissensrepräsentationsformen ein möglicher Ablaufmechanismus zu formulieren, durch den eine entsprechende Aktivierung der jeweils wichtigsten Teilkomponente bzw. der wichtigsten Teile der Wissensbasen stattfindet. Außerdem sind Kommunikationsmechanismen zwischen den einzelnen Teilkomponenten zu definieren und die Interaktionsmöglichkeiten mit einem Benutzer, Lehrer und/oder Programmierer herauszuarbeiten. Dadurch soll erstmals eine Realisierung des neu definierten Programmierparadigmas des Programmierens von Robotern durch Vorführen erreicht werden.

Sämtliche Entwurfsentscheidungen müssen insbesondere unter Berücksichtigung der Integrationsfähigkeit von Lernverfahren getroffen werden. Dazu soll eine Analyse der bereits bekannten Techniken des Maschinellen Lernens durchgeführt und zum ersten Mal eine einheitliche Betrachtung der bisher veröffentlichten Arbeiten zum Thema Maschinelles Lernen in der Robotik vorgenommen werden.

Dies soll unter anderem auch zu einer Definition der möglichen Lernaufgaben in einem autonomen Robotersystem führen. Neben der angesprochenen prinzipiellen Probleme, wie der Definition geeigneter Vorzugskriterien und der notwendigen Fokussierung des vorhandenen Wissens, sollen dann drei Lernziele genauer betrachtet werden.

Das *Lernen von Regelwissen*, das im konkreten Fall im Rahmen der Planung und Ausführung von Manipulationsaufgaben eingesetzt wird, stellt einen weiteren Kern der Arbeit dar. Neben einer formalen Definition der Lernaufgaben sind geeignete Generalisierungs- und Spezialisierungsverfahren zu definieren, die eine sukzessive Verbesserung des Systemwissens durch systeminterne Lernvorgänge oder durch Integration von Benutzervorführungen ermöglichen. Dazu sind auch Methoden zu entwickeln, wie die sinnvollste Anwendung eines Lernoperators bestimmt werden kann. Bei der Definition der Lernverfahren ist insbesondere die nachfolgende Anwendung des erlernten Wissens zu berücksichtigen. Für die Interaktion mit dem Benutzer sind eine Beschreibungsmöglichkeit für die elementaren Ziele des Systems und die möglichen Eingriffe des Benutzers zu bestimmen. Mit diesen Entwicklungen soll damit das erste Mal eine Möglichkeit geschaffen werden, aktorisches Wissen für ein reales Robotersystem zu erlernen.

Ein weiterer Schwerpunkt der Arbeit ist das *Lernen* und die *Verwendung von generischen Objektbeschreibungen*. Neben der geeigneten Repräsentation für einzelne Objekte sind insbesondere neue Repräsentationen für daraus abgeleitete Objektklassen und deren hierarchische Anordnung zu bestimmen. Zum ersten Mal soll auf diesen Beschreibungen, die auch relationale und strukturelle Elemente beinhalten, ein unüberwachtes Lernverfahren definiert werden, das den inkrementellen Aufbau einer Objekthierarchie ermöglicht. Das Verfahren soll weiterhin das überwachte Einlernen von Objekten ermöglichen. Anschließend muß eine Klassifikation von Objekten stattfinden können. Für Lernen und Klassifikation

ist ein Zuordnungsproblem zwischen konkreter Objektansicht und Modellbeschreibung zu lösen.

Das dritte Lernziel stellt die *Bestimmung von statischen und dynamischen Objekteigenschaften* dar. Dazu soll ein Konzept zur Durchführung taktiler Experimente des Robotersystems zur Bestimmung von geometrischen Objekteigenschaften, sowie zur Durchführung von Aktionsexperimenten entwickelt werden.

Die entwickelte Architektur und die genannten Lernmethoden sollen dann anhand einfacher, aber realer, robotikspezifischer *Experimente* validiert werden.

Gegenstand dieser Arbeit ist nicht die bereits vollständige integrierte Realisierung aller Komponenten und ebenso nicht die Berücksichtigung aller möglichen sensorischen und aktorischen Komponenten, z.B. 3D-Vision oder sehr komplexer Greiferkinematiken. Dies muß Gegenstand zukünftiger Weiterentwicklungen bleiben. Vielmehr ist das Ziel, erstmals eine Architektur zu entwerfen, die diese Integration ermöglicht, und für einige Bereiche neuartige geeignete Lernmethoden zu entwickeln.

Im Rahmen dieser Arbeit wurde ausschließlich der Teilbereich der symbolischen Lernverfahren untersucht. Subsymbolische Lernverfahren, wie z.B. Neuronale Netze, sind bereits erfolgreich für die „unteren" Ebenen einer Robotersteuerungsarchitektur, z.B. für die Vorverarbeitung von Sensordaten oder für das Lernen von sensorgekoppelten Bewegungen, eingesetzt worden. Wegen der zugrundeliegenden Repräsentationen der subsymbolischen Ansätze, im wesentlichen Gewichtsmatrizen, ist eine Kommunikation mit einem menschlichen Lehrer über die Wissensinhalte nur schwer möglich. Die Architektur wurde so entwickelt, daß derartige Teilkomponenten eingebracht werden können, eine weitere Untersuchung fand aber nicht statt.

1.4 Übersicht

Die vorliegende Arbeit löst sowohl grundlegende Fragestellungen der Robotik als auch des Maschinellen Lernens. Bisher wurden nur sehr wenige Arbeiten auf diesem Gebiet der lernenden Robotersysteme durchgeführt. Aus diesem Grund führt Kapitel 2 sowohl in die Problematik autonomer Systeme als auch des Maschinellen Lernens ein. Dies soll zum einen die Fülle komplexer Probleme der Robotik verdeutlichen, die bei einem Systementwurf und beim Entwurf von geeigneten Lernverfahren für ein Robotersteuerungssystem zu berücksichtigen sind. Zum anderen soll der Stand der Technik beim Maschinellen Lernen aufgezeigt werden, wodurch die Schwierigkeiten bezüglich der Anwendung auf Robotikfragestellungen deutlich werden. Neben der Einführung der benötigten Konzepte findet eine kritische Bewertung der bereits vorhandenen Ansätze statt, die dann auch in die Anforderungsanalyse einfließt.

In Kapitel 3 wird erstmals eine detaillierte Analyse des Einsatzes von Lernverfahren in der Robotik durchgeführt. Zunächst werden mögliche Lernaufgaben herausgearbeitet und die Anwendbarkeit existierender Lernverfahren untersucht. Danach wird eine Klassifikationsmatrix für bestehende Arbeiten eingeführt und eine Übersicht über den Stand der Technik gegeben.

Die Untersuchungen von Kapitel 2 und 3 führen dann zur Anforderungsanalyse, die in Kapitel 4 beschrieben wird. Dort werden grundlegende Eigenschaften formuliert, die die zu entwickelnde Systemarchitektur aufweisen sollte.

Kapitel 5 definiert die einzelnen Komponenten der Systemarchitektur, die Kommunikation zwischen ihnen, die Aktivierung jeweils einer Komponente und die gewählte Wissensrepräsentation. Anschließend wird das planende Verhalten des Systems analysiert und das Vorgehen zur Erreichung eines Elementarzieles beschrieben.

Das Lernen von Regelwissen für die einzelnen Komponenten der entwickelten Architektur ist Gegenstand von Kapitel 6. Dort werden Generalisierungs- und Spezialisierungsmöglichkeiten betrachtet, ein Vorzugskriterium definiert und der genaue Ablauf beim automatischen und benutzergestützten Lernen formuliert.

In Kapitel 7 wird ausführlich das Lernen von generischen Objektbeschreibungen behandelt. Neben der speziellen Wissensrepräsentation werden Lernoperatoren definiert, eine Bewertungsmöglichkeit für die entstehenden Ballungen angegeben und die Lern- und Klassifikationsverfahren entwickelt.

Auf das Lernen von Objekteigenschaften wird in Kapitel 8 kurz eingegangen. Es wird das entwickelte Verfahren zur Durchführung von taktilen Explorationsexperimenten und zur Planung von Aktionsexperimenten angegeben und das Zusammenspiel mit den anderen Systemkomponenten erläutert.

Eine experimentelle Evaluierung der neu entwickelten Systemarchitektur und der Lernmethoden wird in Kapitel 9 durchgeführt. An Hand von einfachen realen Manipulationsaufgaben werden das Programmieren durch Vorführen, die dabei durchzuführenden Lernschritte und die Lernergebnisse beschrieben. Die Evaluierung des Ansatzes zum Lernen von generischen Objektbeschreibungen wird ebenfalls vorgenommen.

Die Arbeit endet mit einer Zusammenfassung der wichtigsten Ergebnisse und einem Ausblick auf weiterführende und ergänzende Arbeiten. Im Anhang werden einige kurze Details zur durchgeführten Realisierung angegeben.

In den einzelnen Kapiteln sind algorithmische Beschreibungen der entwickelten Verfahren angegeben, um das detaillierte Vorgehen nachvollziehen zu können. Es ist aber weder zum Verständnis der Arbeit als Ganzes noch zum Verstehen der prinzipiellen Vorgehensweise der einzelnen Verfahren notwendig, diese Beschreibungen genau zu verfolgen. Vielmehr wurde diese formale Beschreibungsweise gewählt, weil sich dadurch das exakte Vorgehen der einzelnen Methoden sehr viel besser angeben läßt als durch eine rein textuelle Aufzählung der einzelnen Schritte. Allerdings wurden aus Gründen der Lesbarkeit und der Länge der Verfahren einige Spezialfälle und Methoden zur Effizienzsteigerung nicht in die algorithmischen Beschreibungen aufgenommen.

Kapitel 2

Grundlagen: Autonome Robotersysteme und Maschinelles Lernen

In diesem Kapitel werden die Grundlagen für die nachfolgende Arbeit gelegt. Zunächst werden die Fähigkeiten, die ein autonomes System auszeichnen sollten, herausgestellt. Anschließend werden aus der Literatur bekannte Architekturkonzepte analysiert. Für die wichtigsten Teilaspekte autonomer Systeme werden dann grundlegende Begriffsbildungen erläutert, die vorhandenen Ansätze verglichen und in einigen Teilbereichen neue Sichtweisen begründet, die in den Entwurf der in dieser Arbeit vorgestellten Architektur eingeflossen sind (siehe auch Kapitel 4).

Der nachfolgende Abschnitt (Kap. 2.2) führt dann in das Gebiet des Maschinellen Lernens ein. Dazu werden die wichtigsten Grundbegriffe, die für die Arbeit relevant sind, definiert. Auf dieser Basis wird eine Analyse der wichtigsten bisher entwickelten Methoden durchgeführt und jeweils die Vor- und Nachteile diskutiert. Nach der getrennten Analyse von autonomen Robotersystemen einerseits und Lernverfahren andererseits kann dann im nachfolgenden Kapitel 3 eine genaue Untersuchung der Anwendbarkeit und der konkreten Anwendungen von Lernverfahren in der Robotik stattfinden.

2.1 Autonome Robotersysteme

In diesem Abschnitt werden die Ziele und Teilkomponenten autonomer Robotersysteme erläutert. Aufbauend auf einer Analyse der gewünschten Eigenschaften autonomer Systeme werden verschiedene prinzipielle Möglichkeiten untersucht, ein Gesamtsystem zu strukturieren (→ Architektur). Anschließend werden einzelne Teilkomponenten näher beschrieben, mit deutlichem Schwerpunkt auf den für diese Arbeit relevanten Aspekten. Obwohl die Ausführungen vor allem auf Forschungsarbeiten basieren, die ein vollständig autonomes System anstreben, sind sie ebenso relevant für Systeme, die daneben ein kooperatives Vorgehen zwischen Benutzer und Roboter ermöglichen wollen. Diese Systeme können auch teilautonom genannt werden, weil sie bestimmte Teilaufgaben selbständig lösen, bei anderen Teilaufgaben oder einigen neuen Aufgaben auf eine Unterstützung des Benutzers angewiesen sind. Auf diese erweiterte Sicht, die dieser Arbeit zugrunde liegt, wird im folgenden nicht mehr explizit hingewiesen.

2.1.1 Übersicht und Ziele

Seit Ende der 60er Jahre wird intensiv an Gesamtlösungen und Realisierungen von Teilkomponenten für autonome Systeme gearbeitet. Während der Industrieroboter, der heute für Schweiß-, Handhabungs- oder Montageaufgaben eingesetzt wird, in einer Umgebung und an Aufgaben arbeitet, die entweder selbst statisch und deterministisch sind oder die entsprechend entwickelt werden, soll ein autonomes System in unstrukturierter, teils unbekannter Umgebung arbeiten können. Dabei soll ein vorgegebenes Ziel durch planende Aktivitäten des Systems mittel- und langfristig erreicht werden. Kurzfristig ist auf unvorhergesehene Ereignisse in der Umwelt zu reagieren. Für die Beeinflussung der Umwelt und deren Erfassung stehen dem System aktorische Komponenten in Form eines Manipulators, eines Greifers oder eines Fahrwerks sowie sensorische Komponenten wie Kameras, Kraft-Momenten-Sensoren oder Ultraschallsensoren zur Verfügung.

Typische Aufgaben, die mit Hilfe eines Manipulators gelöst werden sollen, reichen von kollisionsfreien positions- und/oder kraftgeregelten Bewegungen des Endeffektors, Greifen, Transportieren und Ablegen eines Objektes, über Fügeaufgaben, Manipulationsaufgaben bis hin zu komplexen Montageaufgaben. Im Bereich der mobilen Roboter stellen sich Aufgaben wie das Durchführen einer kollisionsfreien Fahrt zu einem Zielpunkt und die Kartographierung einer teilweise unbekannten Umgebung. In den meisten Fällen müssen die Aktionen des Manipulators bzw. des mobilen Systems dabei durch Sensoren geführt und/oder überwacht werden.

Viele Untersuchungen waren zunächst auf die einfachere Domäne der Mobilität eingeschränkt, in der sich ein Fahrzeug in einer eventuell teilweise unbekannten Umgebung bewegt, dabei Hindernissen ausweicht, und mittelfristig einen vorgegebenen Zielpunkt erreicht. In diesem Bereich konnten in den letzten 5-10 Jahren deutliche Fortschritte erzielt werden, so daß einige der Teilprobleme von reaktiver Kollisionsvermeidung, lokaler Bahnplanung, Positionsbestimmung, globaler Routenplanung und Kartographierung inzwischen als weitgehend gelöst gelten. Für die Probleme des Zusammenspiels und des Informationsaustausches zwischen reflexiven Verhaltensweisen und kognitiven Ebenen wurden bisher allerdings erst ansatzweise Lösungen gefunden.

Im Bereich des Einsatzes von Manipulatoren ist die Idee der Autonomie noch bei weitem nicht so weit realisiert worden wie im Bereich der Mobilität. Dies liegt vermutlich zum einen an der starken Industrieorientierung vieler Forschungsarbeiten. In der Industrie ist zur Zeit der Einsatz autonomer Systeme nur bedingt notwendig, da die Umgebungsbedingungen mittels aufwendiger Techniken so gestaltet werden, daß eine genaue Kenntnis aller möglichen Situationen, die behandelt werden müssen, vorliegt. Zum anderen sind bei der Manipulation aber im Gegensatz zu mobilen Systemen auch komplexere Geometrien und Kinematiken zu berücksichtigen und die ausführbaren Aktionstypen diffiziler und vielfältiger.

Eine feste Definition eines autonomen Robotersystems gibt es nicht. In [Dil91] wird ein autonomes System als „ein informationsverarbeitendes System definiert, welches die Fähigkeit hat, eine ihm gestellte Aufgabe auch unter Einwirkung nicht explizit vorgesehener Ereignisse und Umweltzustände erfolgreich durchzuführen". Diese Definition spricht jedoch einige wichtige Charakteristika nur implizit an, so daß im folgenden ein autonomes System über die Menge der Eigenschaften und Forderungen definiert wird, die es haben sollte, bzw. die an es gestellt werden.

Definition 2.1 (autonomes Robotersystem) *Ein Robotersystem wird als* ***autonom*** *bezeichnet, wenn es die folgenden Eigenschaften aufweist*[1]*:*

1. *Das Robotersystem setzt Sensorik zur Wahrnehmung der Umwelt ein und besitzt die Fähigkeit zur Situationsanalyse.*
2. *Das System ist in der Lage unsichere und unvollständige Informationen zu behandeln. Dazu besitzt es insbesondere Vorverarbeitungs- und Fusionsmethoden für die aufgenommenen Sensorwerte.*
3. *Basierend auf der aktuellen Aufgabenstellung und den Informationen über den Zustand der Umwelt plant das System die beste als nächstes auszuführende Aktion.*
4. *Das System überwacht die eigenen Aktionen und reagiert auf wichtige neue Informationen oder unerwartete Ereignisse, die z.B. durch Aktionen anderer Systeme entstehen können.*
5. *Das Robotersystem ist robust, d.h. beim Ausfall von Komponenten oder starken unerwarteten Veränderungen in der Umwelt weist das System ein kalkulierbares Verhalten auf und reagiert möglichst schnell und sicher.*
6. *Die Entscheidungen des Systems können nicht rein reaktiv auf bestimmte Sensorwerte ablaufen. Das System nutzt daher allgemeine und domänenspezifische Wissensbasen.*
7. *Das System ist adaptiv und lernfähig, d.h. es paßt sich an bestimmte aufgabenspezifische Gegebenheiten an und ermöglicht die initiale Akquisition und die sukzessive Modifikation und Erweiterung von Wissen durch Analyse eigener Erfahrungen oder durch Unterstützung von außen.*

Die Anwendungsgebiete für autonome Systeme sind vielfältiger Natur, insbesondere wenn sowohl Manipulations- als auch Mobilitätskomponenten eingesetzt werden. Die wichtigsten derzeit angestrebten Einsatzgebiete sind:

- Reparaturarbeiten: Austausch von Komponenten, Manipulation eines Systems eventuell basierend auf einer eigenständigen Analyse des aufgetretenen Fehlers oder Problems
- Service- und Reinigungsarbeiten: Überwachung von Gebäuden oder Reinigungsarbeiten aller Art
- Arbeiten in Büroumgebungen oder im Haushalt: Unterstützung bei automatisierbaren Tätigkeiten, Übernahme von Transportaufgaben
- Unterstützung bedürftiger Menschen: Hilfe bei Tätigkeiten, die der Mensch nicht selbständig durchführen kann
- Recycling, Demontage: Zerlegung von Gegenständen aller Art zur getrennten Entsorgung bzw. Weiterverwendung von Teilkomponenten

[1] vgl. [Lee89]

- Arbeiten in für Menschen gefährlichen oder unzugänglichen Bereichen: Aufgaben im Weltall oder unter Wasser
- Industrieeinsatz: Wesentlich flexiblere Montagen für Automatisierungsaufgaben bei kleinen Stückzahlen, für die keine spezielle Umgebung aufgebaut werden kann.

Zur Zeit sind erst wenige prototypische Realisierungen autonomer Systeme bekannt, die tatsächlich für einige der vorgesehenen Gebiete eingesetzt werden könnten. Im wesentlichen handelt es sich dabei um mobile Systeme. Der Hauptgrund dafür, daß erst wenige Ansätze für autonome Manipulatoren realisiert wurden, liegt darin, daß erst wenige der genannten Eigenschaften für diese Domäne erreicht wurden. Ein Konsens über eine geeignete Architektur und wiederverwendbare Teilkomponenten ist noch nicht erreicht worden.

2.1.2 Architekturen für autonome Robotersysteme

Ein autonomes System besitzt als Komponenten Sensorik, Aktorik, Module zur Ableitung von Aktionen (strategische und reaktive Planung), zur Wissenshaltung (Wissensbasen), zur Wissensakquisition und zur Kommunikation mit einem Benutzer. Der prinzipielle Aufbau des Gesamtsystems sowie das Zusammenspiel der Teilkomponenten wird in einer Architektur festgelegt. Vereinfacht kann eine Architektur für ein autonomes System als Menge von sensorischen, modellierenden und aktorischen Komponenten verschiedenen Abstraktionsgrades aufgefaßt werden. Die Integration von lernenden Komponenten wurde bisher nur sehr spezialisiert für einige Teilkomponenten betrachtet (s. Kapitel 3). Je nachdem, ob die funktionalen Gemeinsamkeiten verschiedener Abstraktionsebenen, z.B. sämtliche Sensorverarbeitungskomponenten, in den Vordergrund gestellt werden oder die Komponenten einer Ebene, die zusammengenommen ein bestimmtes Verhalten des Systems bewirken können, z.B. „Bewege Dich ohne Kollision", spricht man von *funktionsorientiert* bzw. *verhaltensorientierten* Architekturen. Die konkrete Realisierung einer funktionsorientierten Architektur kann sich natürlich intern oder nach außen auch in verschiedenen „Verhalten" äußern. Andererseits besitzt die Realisierung eines Verhaltens oft intern eine funktionsorientierte Struktur, die aus der Aufnahme und Verarbeitung von Sensorwerten und dem Ansprechen der Aktorik besteht.

Eine Architektur kann zudem *hierarchisch* oder/und *verteilt* sein. Eine hierarchische Struktur liegt dann vor, wenn Perzeption oder Planung nicht einstufig ablaufen, d.h. wenn Sensorinformationen oder Aktionsvorgaben auf unterschiedlichen Abstraktionsniveaus vorliegen und verarbeitet werden. Von einer verteilten Architektur wird gesprochen, wenn die einzelnen Komponenten eigenständig ihre speziellen Aufgaben lösen und über ein entsprechendes Kommunikationsmedium mit den anderen Komponenten Informationen austauschen können, um so gemeinsam eine Lösung für die gestellte Aufgabe abzuleiten.

Funktionsorientierte Architekturen

Albus entwickelte bereits Anfang der 80er Jahre das hierarchische, funktionsorientierte Steuerungssystem NASREM (NASA Standard Reference Model), das aus den drei verschiedenen Komponenten Aktionsauswahl/Planung, Sensorverarbeitung und Modellierung

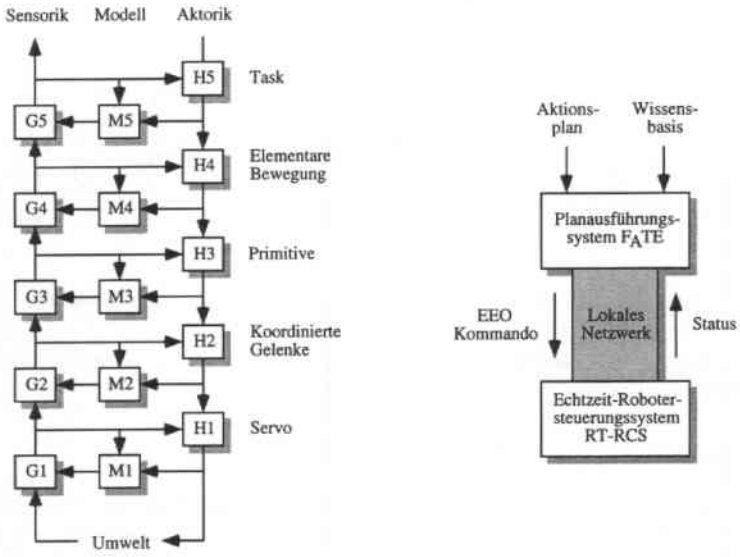

Abbildung 2.1: Funktionsorientierte Architekturen bei NASREM und KAMRO

besteht [Alb84, LGCS89] . Bei der Aktionsauswahl verarbeitet jede einzelne Ebene des Systems die einkommenden Befehle und erzeugt daraus eine Folge einfacherer Kommandos für die nächsttiefere Ebene. Umgekehrt wird auf der Sensorikseite von unten nach oben eine Abstraktion der aufgenommenen Sensorwerte vorgenommen. Sowohl die Aktionsauswahl als auch die Sensorverarbeitung können auf einem Modell basieren, das durch neue Sensorwerte entsprechend aktualisiert werden kann.

Ein weiteres Beispiel für eine hierarchische, funktionsorientierte Roboterarchitektur ist das Steuerungssystem des KAMRO (Karlsruher Autonomer Mobiler Roboter) [HHM88, RD89, HR90b, HR91]. Dieser mobile Zweiarmroboter besitzt im wesentlichen eine Architektur, die aus zwei Ebenen besteht: Einem Planausführungssystem [Hör92a, Hör92b] zur Auswahl und Parametrisierung einer Aktion während der Ausführung einer Aufgabe und einem Echtzeitsteuerungssystem, das die generierten Elementaroperationen in sensorgestütze Ansteuerungen der Manipulatoren umsetzt. Sensorik wird zum einen im Regelkreis zur Ausführung kraftgeregelter Elementaroperationen eingesetzt. Zum anderen wird über eine Overhead- bzw. zwei Handkameras die exakte Lage der modellierten Montageobjekte bestimmt. Die Montagefolge wird off-line bestimmt und als Montagevorranggraph an das Planausführungssystem weitergegeben.

Andere Beispiele für funktionsorientierte Systemarchitekturen sind WISMO (s. Kapitel 3.5.4) . In den bisherigen Arbeiten zu funktionsorientierten Architekturen spielt das Thema Reaktivität nur eine sehr untergeordnete Rolle. Eine echte Reaktion ist häufig nur innerhalb einer fest definierten elementaren Teiloperation möglich oder wenn das System

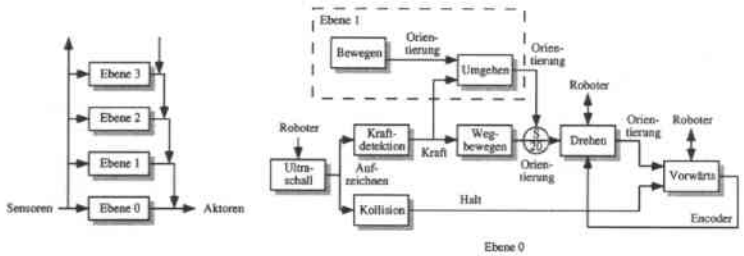

Abbildung 2.2: Subsumption-Architektur (links: abstrakt, rechts: konkret)

einen Fehlerzustand detektiert. Eine Untersuchung von Lernverfahren wurde lediglich bei WISMO durchgeführt.

Verhaltensorientierte Architekturen

Mit der Subsumption-Architektur [Bro85, Bro90] hat Brooks den Begriff der verhaltensorientierten Architektur geprägt. Im Vordergrund stehen hier jeweils eigenständige Komponenten für die unterschiedlichen Verhaltensweisen, die das System zeigen soll. Wichtig ist also nicht der innere Aufbau, d.h. die konkrete Realisierung, sondern das nach außen sichtbare Verhalten. Eine Klasse solcher Verhalten wird als Kompetenzebene bezeichnet, z.B. „Bewege Dich ohne anzustoßen", „Exploriere die Umwelt", „Baue eine Karte auf", „Nimm Änderungen der statischen Umwelt wahr". Jede Kompetenzebene enthält dabei alle niedrigeren Kompetenzebenen und hat selbst einen horizontalen Aufbau. Die Subsumption-Architektur wurde sowohl für mobile Systeme, für Gehmaschinen als auch im Manipulationsbereich [Con89a, Con89b, Con90] eingesetzt.

Ein Vorteil der Subsumption-Architektur, den Brooks nennt, ist, daß die Architektur im Gegensatz zur funktionsorientierten auch dann noch arbeiten kann, wenn einzelne Komponenten ausfallen. Falls sensorische oder aktorische Teilkomponenten versagen, muß dafür entsprechende Redundanz im System vorliegen. Die Frage ist natürlich, in wieweit höhere Kompetenzebenen noch sinnvolles Verhalten erzeugen können, wenn in den unteren ein Fehler auftritt; die tatsächliche Ansteuerung der Aktorik geschieht nur auf unterster Ebene.

Ebenso kann bisher schlecht beurteilt werden, wie umfangreich ein solches System werden kann und vor allem, wie es von außen vorgegebene Ziele erreichen kann. Bisher wurden relativ feste Strukturen realisiert, die zwar ein abgekoppeltes Verhalten erreichen können, deren Zusammenspiel mit einem System, das z.B. vom Benutzer vorgegebene Aufgaben löst, aber unklar ist.

Aufgrund der Schwierigkeit, die Einflüsse höherer Ebenen auf niedrige als Benutzer vorgeben zu müssen und exakt die Kombinationen der Sensorsignale für die Aktivierung eines Verhaltens auszuwählen, wurde von [MB90] der Einsatz von Lernverfahren in der Subsumption-Architektur untersucht. Dabei wurden binäre Sensoren, eine konjunktive

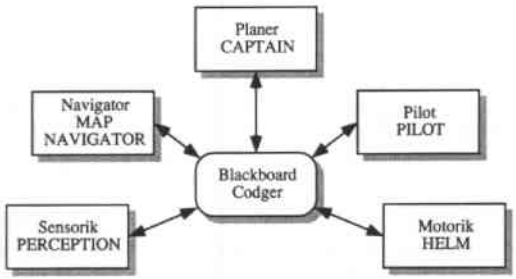

Abbildung 2.3: NavLab (Systemarchitektur)

Verknüpfung der Sensorsignale für die Aktivierung eines Verhaltens und eine direkte positive oder negative Rückmeldung über das Verhalten aus der Umgebung unterstellt. Durch diese Annahmen war es möglich, die Korrelation von Sensormessungen und Feedback für ein Verhalten als Ausgangspunkt für einen Lernvorgang zu bestimmen, und das Vorwärtsgehen einer sechsbeinigen Gehmaschine konnte beispielsweise erfolgreich trainiert werden. Ob das gewählte Vorgehen tatsächlich auch für komplexere Aufgaben eingesetzt werden kann, in denen insbesondere ein Feedback nur für eine Folge von Aktionen geliefert wird, ist unklar. Für die Anwendung in der Manipulation wurde Lernen nur als erstrebenswertes Ziel formuliert [Con89a], konkrete Untersuchungen dazu wurden bisher nicht veröffentlicht.

Hierarchisierung

Eine wichtige Eigenschaft einer Steuerungsarchitektur für ein autonomes System ist die hierarchische Verarbeitung und Ableitung von Sensor- bzw. Aktorinformationen. Durch die Einführung von Abstraktionsebenen kann dem unterschiedlichen Zeitbedarf und Zeithorizont der verschiedenen zu lösenden Teilaufgaben Rechnung getragen werden. Während höchste Planungsebenen im Minuten- oder nur im Stundentakt neue Vorgaben liefern müssen, liegt diese Zeitspanne bei der Regelung auf unterster Ebene im Millisekundenbereich. Analog werden auf höchster Ebene in der Sensorverarbeitung komplexe Verarbeitungen durchgeführt, während auf unterster Ebene reflexartig auf einen nahezu unverarbeiteten Sensorwert reagiert werden kann. Die meisten vorgeschlagenen Steuerungsarchitekturen für autonome Robotersysteme arbeiten hierarchisch.

Verteilung

Eine Systemarchitektur heißt *verteilt*, wenn sie aus einer Menge relativ eigenständiger Komponenten besteht, die über ein Kommunikationsmedium Informationen austauschen und dadurch gemeinsam eine Aufgabenlösung erreichen. Ein häufig verwendetes Kommunikationsmedium ist ein sogenanntes Blackboard [HR85], eine Datenstruktur, in die jede der Komponenten Informationen hineinschreiben oder herauslesen kann. Das bekannteste Beispiel für eine verteilte funktionsorientierte Architektur ist NavLab [GS87] , ein System

zur Navigation eines Kleintransporters durch teilweise unbekanntes Terrain (s. Abbildung 2.3).

Durch die Unabhängigkeit der einzelnen Komponenten voneinander erhält man die Möglichkeit zur parallelen Abarbeitung der einzelnen Teilprobleme. Eine Synchronisierung erfolgt über das Kommunikationsmedium. Zwar sind auch hierarchische Architekturen teilweise parallelisierbar; die Aufrufstruktur – ähnlich zu Unterprogrammaufrufen – ist aber fest vorgegeben.

Eine verteilte Architektur ermöglicht im allgemeinen die Realisierung der Aufgaben einzelner Komponenten auf unterschiedlichen Rechnern. Andererseits liegt bei einer Verteilung von Systemaufgaben auf mehrere Rechner nicht unbedingt eine echt verteilte Architektur vor. Häufig werden zeitkritische Teilprobleme auf eigene Rechnerkarten ausgelagert. Eine allgemeine Kommunikationsstruktur oder Probleme globaler Wissensbasen etc. werden dann aber nicht betrachtet.

Hierarchisierung und Verteilung sind keine Systemeigenschaften, die gegensätzliche Bedeutung haben oder sich ausschließen. Vielmehr basiert die in dieser Arbeit entwickelte verteilte Architektur auf einer Menge von funktionsorientierten Komponenten, die entsprechend kommunizieren und selbst wieder hierarchisch aufgebaut sind.

2.1.3 Wissensrepräsentation und Modellierung

In der Robotik sind je nach Anwendung eine Vielzahl von Modellen im Einsatz (siehe auch [Dil91]), auf denen Entscheidungen oder Berechnungen basieren können:

- Umweltmodell: Beschreibung der Roboterarbeitszelle, die aus dem Roboter, anderen aktiven oder passiven Komponenten und den zu manipulierenden Objekten besteht
- Geometriemodell: Geometrische Beschreibung eines Objektes oder einer Komponente mittels eines Kanten-, Flächen- oder Volumenmodells
- Kinematikmodell: Beschreibung der spezifischen Bewegungsfähigkeit eines mechanischen Systems durch Angabe der kinematischen Kette und der Bewegungsbereiche der einzelnen Armelemente
- Dynamikmodell: Beschreibung der auftretenden Kräfte und Momente bei einer Bewegung bzw. der notwendigen Kräfte/Momente zur Durchführung einer bestimmten Bewegung
- Technologisches Modell: Zusätzliche technologische Objekteigenschaften und Leistungsdaten der Betriebsmittel

Zur Repräsentation dieses Wissens werden im allgemeinen sehr spezielle Methoden verwendet. Bei geometrischen Modellen werden beispielsweise Begrenzungsflächenmodelle (engl. boundary representation) oder Körpermodelle (engl. constructive solid geometry, CSG) jeweils mit entsprechenden Instanziierungen von Basiselementen verwendet. Kinematik-

und Dynamikmodell werden im wesentlichen durch Parametersätze der jeweiligen Konventionen repräsentiert, d.h. des Denavit-Hartenberg- bzw. Lagrange- oder Newton-Euler-Ansatz.

Planungswissen wird in vielen Fällen implizit, d.h. in einem festen Algorithmus, repräsentiert, der starr abläuft und z.B. auf einem Montagevorranggraph basiert.

2.1.4 Aktorik und Sensorik

Aktorik und Sensorik sind die Bindeglieder zwischen Robotersystem und realer Umwelt. Bei der Aktorik werden bei der Manipulation verschiedene Armkinematiken eingesetzt, an die ein weites Spektrum von Endeffektoren von Zweifingergreifern über Spezialwerkzeugen bis hin zu Mehrfingergreifern angeflanscht werden kann. Aufgabe der Sensorik ist das Messen von inneren Zuständen des Roboters und von externen Zuständen der Umwelt. Verschiedene Typen von Sensoren sind laut [Dil91]: Interne Sensoren, Sicherheitssensoren, Kraftsensoren, taktile Sensoren, Abstands- und Näherungssensoren und optische Sensoren. In dieser Arbeit wird keine Forschung zur Entwicklung von Sensor- oder Aktorkomponenten durchgeführt, sondern entsprechende, vorhandene Hardware verwendet (s. Anhang A).

2.1.5 Planung

Die Planungskomponente eines autonomen Robotersystems ist dafür zuständig, basierend auf vorgegebenen Aufgaben oder Zielen, einem eventuell vorhandenen Weltmodell und aufgenommenen Sensorwerten, die nächste oder eine Folge von auszuführenden Aktionen zu bestimmen. Für eine solche Aufgabe können die typischen Programmiermethoden für Roboter (Teach-In und explizites Programmieren), wie sie heute für industrielle Umgebungen eingesetzt werden, nicht verwendet werden. Diese erlauben im allgemeinen nicht, ein Modell der Umwelt aufzubauen und kontinuierlich zu korrigieren. Entscheidungen, die auf Sensorwerten basieren, sind ebenfalls nur sehr eingeschränkt möglich – alle prinzipiellen Ablaufmöglichkeiten müssen explizit in das Programm codiert werden. Eine Aufgabenstellung muß genau einem Aufruf eines der vorliegenden Programme entsprechen.

Planungsverfahren der Künstlichen Intelligenz

Bereits seit Ende der 60er Jahre wurden Planungsmethoden entwickelt, die als langfristiges Ziel auch den Einsatz in Robotern vorsahen [FN71, Fah74]. Obwohl diese Methoden erste Schritte ermöglichten und auch heute noch als Basis in verschiedenen Systemen eingesetzt werden, machen sie zuviele Annahmen, die für die Anwendung bei einem realen Roboter unrealistisch sind.

Eine wichtige Annahme bei diesen traditionellen Planungsmethoden ist, daß sich die Welt während der Planung nicht ändert, d.h., daß eine Trennung in Planungs- und Ausführungsphase strikt möglich ist. Wilkins nennt ein System dagegen reaktiv, wenn es in einer akzeptablen Zeit auf jede Änderung in der Welt reagieren kann, die auch auftreten kann, während das System läuft [Wil88]. Die Eigenschaft der Reaktivität und insbesondere das Zusammenspiel mit einer strategischen Planung ist von grundlegender Bedeutung für die

Fähigkeiten und die möglichen Einsatzgebiete autonomer Systeme. Aus diesem Grund wird auf diese beiden Aspekte im nachfolgenden Abschnitt ausführlich eingegangen.

Im Bereich der klassischen KI-Planung gibt es bereits einige Ansätze zur Integration von Lernverfahren, vor allem der Einsatz von deduktiven Lernverfahren (s. Abschnitt 2.2.5) zur Ableitung von Makrooperatoren oder Operatorpräferenzen und der Einsatz des Lernens aus Analogien (s. Abschnitt 2.2.6) zur Lösung von Planungsproblemen durch Berücksichtigung bereits gelöster Planungsaufgaben. Dagegen gibt es nur wenige Arbeiten, die den Einsatz von Lernmethoden in einem reaktiven Planer untersuchen [Sch87, BM89, Ger90].

Planungssysteme für Roboter

Parallel zu den prinzipiell allgemein verwendbaren Planungsverfahren der KI wurden spezielle Planungsansätze für Robotersysteme entwickelt, die im Grunde aber rein geometrische Schlußfolgerungen und keine echte Planung mehr durchführen.

Dazu gehören die Off-line-Systeme zur Bestimmung einer möglichen Menge von Montagesequenzen [HdMS89, Wol89, FW90, HL90, HL91, LS90, HdMS91b, HdMS91a] ebenso wie Ansätze zur geometrischen Bewegungs- und Greifplanung [LPJM+87, JLP90] und zur impliziten Programmierung, die alle notwendigen Bewegungen aus vorliegenden Modellen ableiten: AUTOPASS [LW77], LAMA [LPW77], RAPT [PAB80], SHARP [LPT85], SPAR [HK90].

In existierenden Robotersystemen mit Planungskomponente besteht die Planung häufig nur in der Auswahl eines Skelettplanes und dessen Konkretisierung für die vorliegende Aufgabenstruktur. Als eigentliches Hauptproblem wird das On-line-Scheduling der verfügbaren Ressourcen betrachtet[2] [XB90, Hör92a]. Ein solches eher statisches Vorgehen ist zwar für abgegrenzte Aufgabenbereiche und bei bestimmten Annahmen über die Umgebung möglich, für ein autonomes System ist es aber nur bedingt geeignet.

Die erwähnten Ansätze betrachten nicht, wie on-line Entscheidungen basierend auf Sensorwerten berücksichtigt werden können, wie mit Unsicherheiten in der Modellierung umgegangen oder wie die exakte Modellierung überhaupt erreicht werden kann. Eine automatische Analyse von auftretenden Fehlern ist in derartige Systeme zwar integrierbar [Bro82], allerdings wird auch dazu wieder ein exaktes Modell vorausgesetzt. Die Übertragung der Lösungen der modellbasierten Robotik auf autonome Systeme wird häufig dadurch versucht, daß entweder eine vollständige Modellierung der Umwelt postuliert wird oder in einem a priori Schritt die Umwelt rekonstruiert wird, d.h. wieder ein exaktes Umweltmodell vorliegt, und darauf die bekannten Methoden angewandt werden. Dabei stellt sich die Frage, ob der Aufwand einer exakten Modellierung der realen Umwelt lohnt bzw. prinzipiell überhaupt möglich ist, oder ob nicht durch andere Vorgehensweisen das Problem besser gelöst werden kann. Eine alternative Vorgehensweise, die auch dem menschlichen Verhalten näher kommt, ist daher die bedarfsorientierte Modellierung der Umwelt – von T. Mitchell als „model as few as necessary" bezeichnet [Mit91].

[2] Die Betrachtung von Xia von Planung und Scheduling als zwei Dimensionen der gleichen prinzipiellen Aufgabe könnte mit auch im ML gebräuchlichen Termini übersetzt werden mit: Planung als der Übergang vom Abstrakten zum Konkreten und Scheduling als der Übergang vom Allgemeinen zum Speziellen, d.h. bei der Planung werden abstrakte Operatoren immer weiter verfeinert und während des Scheduling werden die Operanden immer genauer festgelegt.

2.1.6 Reaktivität und Strategie

> "For a system that deals with complex problems in a real world, as opposed to a simulated one, it is undesirable to solve an entire problem with an epistemologically adequate plan. There are too many reasonably likely outcomes for each real-world operation."[Sac74, S. 133]

Ein wichtiges Merkmal einer Interaktion mit der realen Welt ist, daß unsicher ist, in welchem Zustand sich die Welt vor und nach einer Aktion exakt befindet. Dies kann dadurch bedingt sein, daß sich entweder die Umgebung dynamisch ändert, z.B. durch andere unabhängige Agenten oder Prozesse, daß eine Aktion fehlerhaft durchgeführt wird, oder daß eine Vorhersage der Zukunft aufgrund der Vielzahl der zu berücksichtigenden Einflüsse einfach nicht möglich ist. Ein autonomes Robotersystem sollte adäquat mit dieser Eigenschaft umgehen können. Dies ist durch ein klassisches Off-line-Planungssystem nicht möglich. Vielmehr müssen Methoden entwickelt werden, wie on-line möglichst „gute" Entscheidungen gefällt werden können.

Ein planendes System, das seine Pläne als Antwort auf die sich ändernden Situationen während der Ausführung erzeugt oder ändert, heißt *reaktiver Planer* [Fir87]. Diese Reaktivität kann durch ein weites Spektrum von Möglichkeiten erreicht werden:

1. Einbau aller möglichen Varianten in den Aktionsplan, d.h. alle möglichen Zustände werden betrachtet und eine entsprechende Reaktion vorgesehen. Ein solcher Ansatz (z.B. universal plans [Sch87]) ist nur in einer sehr eng begrenzten Domäne mit wenigen möglichen Reaktionen einsetzbar. Andernfalls entstehen riesige Planstrukturen, weil alle Alternativen eingebaut werden müssen, von denen jeweils nur ein Bruchteil tatsächlich benötigt wird.

2. Definition von elementaren Reaktionsmodulen, d.h. eine Reaktion wirkt sich nur innerhalb einer festen Teilaufgabe aus. Das Spektrum der darin möglichen Alternativen und der Ablauf sind zumeist wie in Punkt 1 fest vorgegeben. Firby [Fir87] entwickelte ein Konzept für einen reaktiven Planer, der auf einer Menge von reaktiven Aktionspaketen (engl. reactive action packages) aufbaut, die als eigenständige Prozesse ein Planziel verfolgen, bis es erreicht ist.

3. Definition von reaktiven Verhaltensregeln, die bei jeder Systementscheidung auf Anwendbarkeit hin überprüft werden. Es wird also immer on-line entschieden, wie auf den aktuellen Weltzustand unter Berücksichtigung der mittel- oder langfristigen Zielvorgabe geeignet reagiert werden soll [AC87, GL87].

Die dritte Variante ist, von der Sichtweise der Reaktivität aus, die geeignetste und wurde in den meisten der wenigen bisher veröffentlichten Arbeiten zu reaktiven Planern verwendet. Sie soll hier auch als Grundlage der Repräsentation reaktiven Verhaltens angestrebt werden.

Einerseits sind reaktive Regeln zwar direkt anwendbar für eine direkte Kopplung zwischen Sensorik und Aktorik, andererseits ist eine Lösung für das Zusammenspiel mit einem eher strategisch orientierten Planer aber nicht offensichtlich. Aus diesem Grund gibt es eine

Reihe von Arbeiten [AC87, Mae90, Bro91], die eine rein reaktive Architektur fordern. Diese enthält keinerlei Planungskomponenten und baut im allgemeinen kein explizites Wissen über die Umwelt auf. Im Kontext derartiger Ansätze besteht dann ein Lernvorgang meist nur im Aufbau von einfachen Stimulus-Response-Regeln. Wissen, das zur strategischen Lösung von neuen ähnlichen Aufgaben eingesetzt werden kann, wird nicht erlernt.

Ein autonomes Robotersystem, das auch eine größere, sinnvolle Aufgabe lösen soll, kann nicht nur aus einer reaktiven Komponente bestehen. Vielmehr müssen gleichzeitig strategische und reaktive Aspekte bei der Planung berücksichtigt werden. In [LHM91] wird die strategische Planung im Vergleich zur klassischen KI-Planung neu definiert[3]. Sie besteht nicht mehr aus der Erzeugung einer Sequenz von Operatoren, die den Zielzustand ergeben, wenn sie auf den aktuellen Weltzustand angewendet wird, sondern entspricht einem System, das kontinuierlich ein reaktives System verändert, so daß dessen Verhalten zielgerichteter wird. Das reaktive System enthält eine Menge von Sensor-Aktor-Kopplungen, die Reaktionen genannt werden. Die strategische Planung interagiert mit der reaktiven Planung in einer ähnlichen Weise wie die reaktive Planung mit der Umwelt.

Das Zusammenspiel zwischen strategischem und reaktivem Planer sollte demnach nicht ablaufen wie im obigen Punkt 2, wo vom strategischen Planer für ein spezielles Teilproblem ein reaktives System aufgerufen wird, das dieses Problem selbständig löst und anschließend die Kontrolle an das strategische System zurückgibt. Vielmehr sollten beide Komponenten parallel ablaufen und auch der strategische Planer auf neue Sensorwerte oder daraus abgeleitete Informationen reagieren können und dadurch parallel, wenn nötig, sein langfristiges Vorgehen ändern. Eine externe oder eigene Aktion kann also zu einer Änderung auf reaktiver und/oder strategischer Ebene führen.

Aus diesem Grund ist eine explizite Trennung in ein strategisch planendes und ein reaktives Teilsystem relativ künstlich, da ein fließender Übergang zwischen Aktionen besteht, die noch als direkte Reaktion zu betrachten sind, und solchen Aktionen, die bereits eine Strategie verfolgen. Daher soll im folgenden nur noch von „reaktiv" und/oder „(strategisch) planend" als Eigenschaft des Gesamtsystems gesprochen werden.

Definition 2.2 (reaktives, planendes System) *Ein System heißt* ***reaktiv****, wenn es auf Ereignisse in der Umwelt mit entsprechenden Aktionen antworten kann. Ein System heißt (strategisch)* ***planend****, wenn es eine Aufgabe, d.h. ein Ziel, prinzipiell verfolgt und dafür notwendige Aktionen ableitet.*

Diese Sichtweise auf Reaktivität und Planung eines Systems läßt sich natürlich nicht mit den klassischen KI-Planungsverfahren verwirklichen. Geeignet sind inkrementelle, opportunistische Planungsansätze. *Opportunismus* bedeutet, daß zu jedem Zeitpunkt untersucht wird, welche Aktion möglich und sinnvoll ist, und nicht versucht wird, eine vorher festgelegte Aktionsfolge strikt durchzuführen. In [FK85] wurde ein solcher Ansatz erstmals für Schedulingaufgaben formuliert. Fox und Kempf definieren eine weitere wichtige Eigenschaft eines solchen Systems, die sie *least commitment scheduling* nennen. Heute wird auch von *deferred planning* gesprochen. Die Idee ist, daß die Festlegung auf eine Alternative,

[3]Dort wird die Komponente zur strategische Planung als Planer, die Komponente zur reaktiven Planung als Reaktor bezeichnet.

wenn ein Ziel auf mehrere verschiedene Arten erreicht werden kann, zum spätest möglichen Zeitpunkt geschieht.

Letztendlich führt jede reaktive oder planende Entscheidung zu einer Aktion des Systems, die interner Natur (Ableitung von Informationen) oder externer Natur (Ansteuerung der realen Aktorik) sein kann. Im Normalfall wird die Ansteuerung der Aktoren durch einen Satz von Elementaroperationen vorgenommen, von deren interner Abarbeitung bzw. Struktur dann abstrahiert wird. Eine solche Elementaroperation basiert im allgemeinen auf einer Regelungsfunktion, die im Grunde ebenfalls reaktives Wissen darstellt, z.B. für die Positions- und/oder Kraftregelung einer Achse bzw. Koordinate. Der Übergang zwischen traditionellen Regelungsaufgaben und reaktivem Verhalten höherer Ebene ist fließend.

Prinzipiell sollte das gesamte Aufgabenspektrum also mit einer homogenen Komponente zur Ableitung planenden und reaktiven Verhaltens auskommen können. Aus Effizienzgründen und weil häufig bestimmte Teilmodule bereits festliegen, z.B. Hardware mit fester Softwareschnittstelle, wird aber doch eine Trennung durchgeführt.

2.1.7 Fehlerbehandlung

In mehreren vorgeschlagenen Architekturen für autonome Robotersysteme gibt es eine explizite Komponente zur Fehlerzustandserkennung und -behebung (z.B. [MHM+89]). Ein Fehlerzustand kann dadurch auftreten, daß durch Unsicherheiten in Sensorik, Aktorik oder zugrundeliegendem Modell oder auch durch Einwirkung externer Ereignisse eine Aktion nicht so ausgeführt wird bzw. werden kann, wie gewünscht. Zur Lösung dieser Problematik wurden folgende Lösungsvarianten untersucht:

1. Einbau von expliziten Sensormessungen in den Aktionsplan zur Überprüfung von notwendigen Bedingungen a priori [Gin87, Gin90, HK90, Wer93]. Diese können dann unterschiedlich behandelt werden: Entweder für einen erkannten Fehlerzustand existiert eine Routine, die den ursprünglich gewünschten Zustand herstellt, oder in Abhängigkeit der Sensorwerte wird einer der vorher geplanten Pfade weiterverfolgt (z.B. [Sch87]).
2. Bestimmung der Notwendigkeit einer Sensormessung während der Laufzeit auf der Basis von Wahrscheinlichkeiten dafür, daß eine Abweichung vorliegt [Abr91]
3. Ständige Reaktion auf aktuelle Sensorwerte (→ reaktives Planen) [Fir87, GL87]

Die unter Punkt 1 und 2 genannten Methoden sind für relativ statische Umgebungen geeignet, in denen Fehlerzustände nur durch Abweichungen durch Modellierungs- oder Fertigungsungenauigkeiten auftreten können. Ein Einsatz dieser Methoden in einer dynamischen Umgebung, wie sie für ein autonomes System normalerweise vorliegt, ist nicht möglich. Durch eigene Aktionen oder durch Aktionen anderer Agenten können signifikante Unterschiede zwischen erwarteter und tatsächlicher Welt entstehen. Eine On-line-Entscheidung ist dann auf jeden Fall notwendig. Der erste Ansatz setzt außerdem voraus, daß a priori bekannt ist, welche Fehlerzustände überhaupt auftreten können. Ansonsten könnten keine passenden Sensormessungen off-line in den Plan eingebaut werden.

Die Behebung eines Fehlerzustandes kann dann entweder dadurch erfolgen, daß das System versucht, den eigentlich gewünschten Zustand zu erreichen, oder indem es versucht, an irgendeine Stelle des vorgesehenen Lösungspfades zurückzukehren. In den meisten Ansätzen wird davon ausgegangen, daß, wenn die Fehlerbehebung nicht in der Lage ist, die ursprünglichen Ziele zu erreichen, auf der Basis der neuen Umweltsituation eine Neuplanung erfolgen muß. Der Übergang zwischen Wissen für die Planung einer Fehlerzustandsbehebung und für die Planung der eigentlichen Aufgabenlösung muß also fließend sein. Es stellt sich also die Frage, ob eine explizite Trennung sinnvoll oder überhaupt möglich ist.

In dynamischen Umgebungen könnte es zu einer ständigen Fehlerzustandsbehebung kommen, da sogar während einer laufenden Behebung eine weitere nötig werden kann. Ein Fehlerzustand in diesem Sinne bedeutet, daß die Erwartung an einen zukünftigen Zustand nicht mit der Realität übereinstimmt. Diese Erwartung basiert auf einem Modell, das nicht vollständig ist und auch nicht sein kann. Das System muß daher prinzipiell in der Lage sein, auf aktuelle Umweltsituationen, d.h. neue sensorische Informationen, geeignet zu reagieren. Ob diese durch einen „Fehler" entstehen oder durch eine Einwirkung von außen, spielt für das System keine Rolle.

Sehr viel wichtiger ist die eventuelle Inkorrektheit des Systemwissens, die z.B. zu ungewünschten Aktionen führen kann, die dann tatsächlich auf einem Fehler des Systems beruhen. Dieser Fehler liegt aber im Wissen selbst und nicht in der falschen Ausführung. Ein derartiger Fehler ist im allgemeinen nur durch Kommunikation mit einem Experten behebbar.

2.1.8 Kritische Bewertung

Die Forschung auf dem Gebiet der autonomen Systeme steht für viele Teilbereiche noch am Anfang. Für mobile Systeme sind zwar eine große Anzahl von Arbeiten durchgeführt worden und mehrere Probleme bereits gelöst, für den Bereich der Manipulation sind bisher jedoch nur wenige Ansätze erarbeitet worden. Neue Trends führen weg von den stark modellbasierten Ansätzen der 70er und 80er Jahre, die versuchten, die Aufgaben durch geometrisches Schlußfolgern zu lösen. Wichtige Eigenschaften einer Architektur für ein autonomes Robotersystem sind die gleichzeitige Betrachtung von reaktiven und strategischen Aspekten. Eine wichtige offene Frage ist, wie ein autonomes Robotersystem sich tatsächlich über die Zeit verbessern kann. Eine mögliche Antwort können Ergebnisse der Forschungen liefern, die im Gebiet des Maschinellen Lernens durchgeführt werden.

2.2 Maschinelles Lernen

> „The ability to learn, to adapt, to modify behavior is an inalienable component of human intelligence. How can we build truly artificially intelligent machines that are not capable of self-improvement? Can an expert system be labeled 'intelligent', any more than the Encyclopedia Britanica be labeled intelligent, merely because it contains useful knowledge in quantity?" [Car89]

Dieser Abschnitt schafft die Grundlagen dieser Arbeit aus der Sicht des Maschinellen Lernens. Neben einer Übersicht über die Ziele des Maschinellen Lernens und die unterschiedlichen Klassen von Lernverfahren werden wichtige Grundbegriffe eingeführt, die in den nachfolgenden Kapiteln verwendet werden.

2.2.1 Übersicht und Ziele

Ein herausragendes Kennzeichen von Intelligenz ist die Fähigkeit zu lernen. Auf vielfältige Art und Weise nimmt beispielsweise der Mensch Informationen auf, kombiniert sie mit bereits bestehendem Wissen oder verbessert bereits bestehende Fähigkeiten. Es ist außerordentlich schwer, eine formale Definition dafür zu geben, worin Lernen besteht. Daher liegen die unterschiedlichsten Definitionsversuche vor, die zum Teil inhaltlich stark voneinander abweichen, je nach dem Fachgebiet, auf das sie sich beziehen. Stärker algorithmisch aus der Sicht eines informationsverarbeitenden Systems aus gesehen, wurde Lernen folgendermaßen definiert:

Definition 2.3 (Lernen) ***Lernen*** *ist jede Veränderung in einem System, die es ihm erlaubt, dieselbe Aufgabe oder eine Aufgabe gleicher Art beim nächsten Mal besser zu lösen [Sim83].*

Definition 2.4 (Lernen) ***Lernen*** *bedeutet Konstruktion oder Modifikation von Repräsentationen dessen, was erfahren wird [Mic86].*

Eine notwendige Ergänzung der Definition 2.3 ist, daß ein Lernvorgang auch dann vorliegt, wenn eine Aufgabe überhaupt danach ausgeführt werden kann. Diese initiale Wissensakquisition ist sicher ebenso wichtig wie die nachfolgende Verbesserung. Eine Definition mit ingenieursmäßiger Ausrichtung wird in [Dil88] zitiert:

Definition 2.5 (Lernendes System) *Ein System wird als* ***lernend*** *bezeichnet, wenn es in der Lage ist, unbekannte Eigenschaften eines Prozesses oder seiner Umgebung durch schrittweises Handeln und Beobachten zu erfassen. Die dadurch gewonnene Erfahrung wird benutzt, um Vorhersagen, Klassifikationen und Entscheidungen durchzuführen, damit ein vorgegebenes optimales Systemverhalten erreicht werden kann. (Fu, 1964, Saridis, 1977)*

Eine Definition von Psychologen lautet:

Definition 2.6 (Lernen) ***Lernen*** *ist der Prozeß, durch den eine Aktivität in Reaktion auf eine Umweltsituation entweder neu entsteht oder verändert wird - vorausgesetzt, daß die Besonderheiten der Aktivitätsänderung nicht als angeborene Reaktionstendenz, Reifungsvorgang oder Momentanzustand des Organismus (z.B. Erschöpfung, Drogenwirkung u.a.) erklärbar sind. [LE92]*

Zusammenfassend läßt sich Lernen dadurch charakterisieren, daß Information, die in einer bestimmten Art repräsentiert wird, entweder initial erworben oder sukzessive erweitert, korrigiert oder verbessert wird. Die Forschungsrichtung des Maschinellen Lernens

untersucht, wie derartige Lernvorgänge in informationsverarbeitenden Systemen realisiert werden können. Dabei lassen sich drei wesentliche Lernziele angeben, die auch die Schwierigkeit aufzeigen, alle Varianten von Lernvorgängen in einer einzigen Definition zu erfassen.

1. Der Erwerb von Fertigkeiten (engl. skill acquisition): Lernen/Entwicklung von motorischen und kognitiven Fähigkeiten durch Anweisung oder Training
2. Der Erwerb von Wissen (engl. knowledge acquisition): Lernen von Aktionsfolgen und ihrer Auswirkungen, Beschreibungen, Modellen, Regeln, deklarativem Wissen
3. Die Verbesserung vorhandenen Wissens: Neuorganisation, Umorganisation und Transformation von Wissen

Während bei dem Erwerb von Fertigkeiten nur wenig explizites Wissen, d.h. kommunizierbare Information, aufgebaut wird, geht es bei dem Erwerb von Wissen gerade darum. Der Erwerb von Fertigkeiten ist auch in der Robotik im wesentlichen der Aufbau von motorischen oder sensorischen Fähigkeiten durch wiederholte Anwendung [Mic86], um z.B. eine kraftgeregelte Bewegung möglichst fehlerfrei und effizient durchzuführen. Für derartige Anwendungen haben sich subsymbolische Lernverfahren als erfolgversprechend gezeigt. In dieser Arbeit steht der Erwerb von Wissen im Vordergrund. Daher wird im folgenden der Teil des Maschinellen Lernens, der sich damit beschäftigt – die sogenannten symbolischen Lernverfahren –, stärker untersucht.

Der Begriff der symbolischen Lernverfahren sagt aus, daß explizite Symbole manipuliert werden. Er ist nicht zu verwechseln mit der Unterscheidung von Newell in Symbol- und Wissensebene [New82]. Newell definiert speziell das Lernen auf der Symbolebene, bei dem sich die Gesamtmenge des Wissens nicht ändert, d.h. die deduktive Hülle des im System kodierten Wissens gleichbleibt, und das Lernen auf der Wissensebene, das eine Vergrößerung des Gesamtwissens, d.h. einen Wissenserwerb, darstellt.

In der Vergangenheit wurde bereits mehrfach versucht, die unterschiedlichen entwickelten Lernmethoden in verschiedene Klassen einzuteilen, beispielsweise nach der Art der Lernstrategie, nach dem verfolgten Lernziel oder nach der Art der Wissensrepräsentation [CMM83]. Michalski entwickelte in [MK90, Mic91] eine aktuellere, auch neueste Lernmethoden enthaltende Multikriterien-Klassifikation für Lernverfahren (s. Abbildung 2.4). Er unterscheidet nach dem Zweck des Lernens, dem Typ der Eingabeinformation, der verwendeten Inferenzmethode und der Rolle des vorhandenen Hintergrundwissens. Im folgenden soll als wichtigstes Unterscheidungskriterium die zugrundeliegende Inferenzmethode verwendet werden. Dementsprechend sind folgende drei Bereiche zu unterscheiden:

1. Induktive Lernverfahren (Synthetische Lernverfahren): Als Inferenzmethode kommt die Induktion zum Einsatz. Auf der Basis einer Menge von Beispielen wird versucht, eine möglichst plausible allgemeingültige Aussage abzuleiten, die als induktive Hypothese bezeichnet wird.
2. Deduktive Lernverfahren (Analytische Lernverfahren): Durch Analyse bereits vorhandenen Wissens wird versucht, dieses in eine Form zu bringen, die zu einer effizienteren Anwendung führt. Die deduktive Hülle des vorhandenen Wissens wird dabei nicht erweitert.

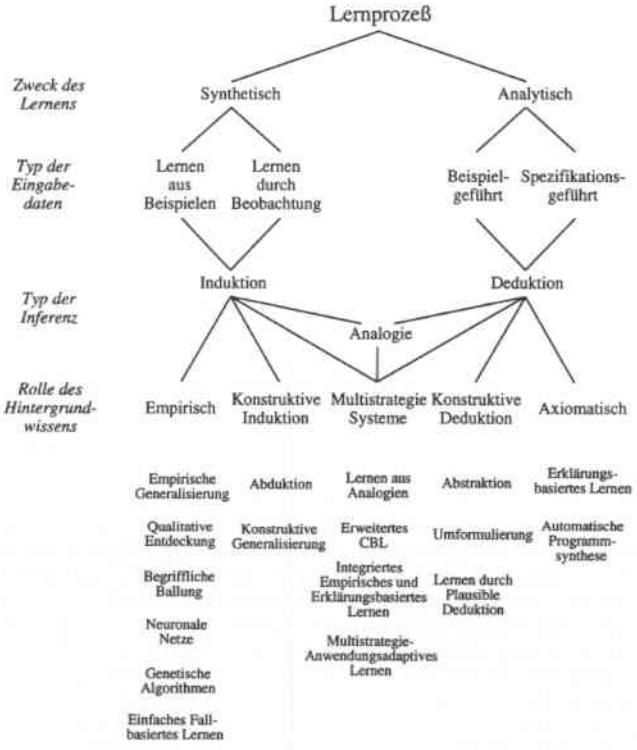

Abbildung 2.4: Klassifikation von ML-Techniken

3. Mischformen: Eingesetzt werden verschiedene Kombinationen von induktiven und/oder deduktiven Lernverfahren, z.B. Lernen aus Analogien.

Zusätzlich zu den Kriterien von Michalski wird häufig eine Unterscheidung in symbolische und subsymbolische Lernverfahren vorgenommen. Bezüglich der verwendeten Inferenzmethode sind die subsymbolischen Ansätze induktive Lernverfahren. Während die symbolischen Verfahren auf explizitem Wissen operieren, das auch zur Kommunikation mit dem Benutzer verwendet werden kann, manipulieren subsymbolische Verfahren im wesentlichen Informationen, die für den Benutzer keine explizite Bedeutung haben (Gewichte in einem Netz, Übergangswahrscheinlichkeiten in einem Automaten, etc.). Typische Repräsentationsformen, die mit symbolischen Methoden manipuliert bzw. erzeugt werden, sind aussagen- oder prädikatenlogische Ausdrücke, Regeln, Entscheidungsbäume, semantische Netze, Frames und Grammatiken.

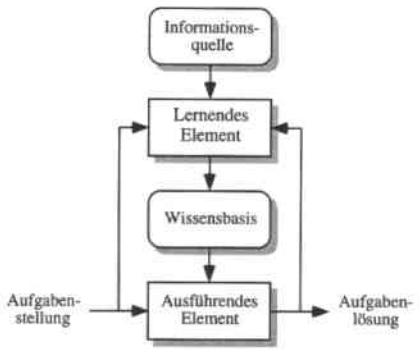

Abbildung 2.5: Einfaches Modell eines lernenden Systems

In den meisten ML-Arbeiten wurden die entwickelten Lernmethoden sehr isoliert betrachtet, d.h. im Vordergrund steht die Verarbeitung der Menge von Beispielen oder ähnlichem durch einen geeigneten Lernalgorithmus. Wie diese Komponente hingegen in einem Gesamtsystem die notwendigen, relevanten Daten herausfiltern kann und wie sich die eventuell noch fehlerhaften Lernergebnisse auf die Ausführung einer Aufgabe auswirken, wurde nur vereinzelt untersucht (siehe auch Kapitel 3.5). Dieses Integrationsproblem stellt sich nicht so deutlich, wenn die Lernaufgabe ist, einen Klassifikator abzuleiten, der eine Menge von Eingabewerten einer bestimmten Klasse zuordnet. Dieser Vorgang kann tatsächlich abgekoppelt oder a priori ablaufen. Schwieriger ist diese Betrachtung bei aktorischem Wissen, das zeitlich verkettet angewendet wird, und bei dem die Bewertung nicht exakt zum Zeitpunkt der Anwendung fehlerhaften Wissens geschieht. Ein einfaches Modell eines lernenden Systems ist in Abbildung 2.5 dargestellt. Dieses Modell zeigt zwar ein mögliches Zusammenspiel von lernendem und ausführendem Element über die Wissensbasis, schränkt die Sichtweise aber stark ein, indem nur Aufgabenstellung und -lösung vom ausführenden Element in das Lernelement eingehen.

In den folgenden Unterkapiteln werden zunächst einige Grundbegriffe definiert und dann die wichtigsten Lernmethoden sowie die Problematik der Integration mehrerer Lernstrategien erläutert. Auf dieser Grundlage wird im nachfolgenden Kapitel der Einsatz von Lernverfahren in der Robotik analysiert.

2.2.2 Grundbegriffe

In diesem Abschnitt werden einige Grundbegriffe des Maschinellen Lernens erläutert, die in den nachfolgenden Abschnitten und Kapiteln immer wieder verwendet werden. Ein allgemeiner Konsens über die Begriffsbildung im ML ist noch nicht erreicht worden, so daß es sich auch im folgenden um Definitionsversuche handelt.

- Induktion vs. Deduktion vs. Abduktion: Bereits in der Einleitung dieses Abschnitts wurde darauf eingegangen, daß die zugrundeliegende Inferenzmethode eines Lern-

$$\frac{\begin{array}{l}P(x_1) \rightarrow Q(x_1)\\ P(x_2) \rightarrow Q(x_2)\\ \cdots\\ P(x_n) \rightarrow Q(x_n)\end{array}}{\forall x : P(x) \rightarrow Q(x)}\ \text{induktiv} \qquad \frac{\begin{array}{l}P(x) \rightarrow Q(x)\\ P(x)\end{array}}{Q(x)}\ \begin{array}{l}\text{deduktiv}\\ \text{(Modus pon.)}\end{array} \qquad \frac{\begin{array}{l}P(x) \rightarrow Q(x)\\ Q(x)\end{array}}{P(x)}\ \text{abduktiv}$$

Abbildung 2.6: Inferenzmethoden: Induktion, Deduktion, Abduktion

verfahrens ein wichtiges Unterscheidungskriterium darstellt. Am Beispiel der Regel $P(x) \rightarrow Q(x)$ werden die unterschiedlichen Inferenzmethoden in Abbildung 2.6 erläutert. Der induktive Schluß ist die Ableitung einer allgemeinen Aussage aus speziellen Angaben, z.B. Beispielen. Beispiel: Rabe_1, Rabe_2 usw. sind schwarz, dann sind alle Raben schwarz. Ein deduktiver Schluß ist jeder klassische logische Schluß, wie z.B. der Modus ponens. Beispiel: Wenn x ein Rabe ist, dann ist x schwarz. Objekt_1 ist ein Rabe, also muß Objekt_1 schwarz sein. Durch die Anwendung deduktiver Schlüsse erhält man kein wirklich neues Wissen.

Eine Eigenschaft deduktiver Schlüsse ist, daß sie wahrheitserhaltend sind, d.h. wenn die Voraussetzungen wahr sind, dann ist auch das deduktiv abgeleitete Wissen wahr. Diese Eigenschaft gilt für einen induktiven Schluß nicht.[4] Dieser ist jedoch falschheitserhaltend, d.h. falls eine der Voraussetzungen falsch ist, dann ist sicher auch der induktive Schluß falsch. Die abduktive Inferenz ist der Vollständigkeit halber auch angegeben, spielt für maschinelle Lernverfahren bisher aber keine entscheidende Rolle. Entsprechend der Eigenschaften der Induktion und Deduktion werden auch *synthetische* bzw. *analytische* Lernverfahren unterschieden.

- überwacht vs. unüberwacht: In ML-Arbeiten werden häufig überwachte und unüberwachte Lernverfahren unterschieden. Dabei steht die Aufgabe des Lernens eines Klassifikators im Vordergrund, d.h. einer Zuordnung von Eingabedaten zu einer von mehreren Klassen. Von *überwachtem* Lernen wird gesprochen, wenn zum Training, also für den Lernvorgang, Beispiele, repräsentiert durch einen Eingabevektor, zur Verfügung stehen, für die auch eine Klassenzugehörigkeit oder allgemeiner der gewünschte Ausgabevektor angegeben ist. Diese Information stammt entweder vom Benutzer/Lehrer des Systems oder einem anderen System, das die gewünschte Ausgabe bereits erzeugen kann.

  Ein Lernvorgang heißt dementsprechend *unüberwacht*, wenn diese Klassenzuordnung fehlt und das System selbständig Gruppierungen bilden muß, die einer „Klasse" entsprechen könnten. Angewandt auf die Robotik ist diese Trennung in überwachte und unüberwachte Lernverfahren nur schwierig durchzuhalten, wenn es sich nicht nur um reine Klassifikatoraufgaben handelt. Eine mögliche Unterteilung wäre überwacht, d.h. der Lehrer gibt alles vor, teilüberwacht, d.h. der Lehrer gibt an bestimmten Punkten eine Hilfestellung, und unüberwacht, d.h. das System führt alles

[4] Aus diesem Grund hat es auch viele unterschiedliche Meinungen über den Sinn induktiver Schlüsse gegeben. Popper hält reine Induktion daher nur für sinnvoll in Zusammenhang mit einer deduktiven Überprüfung.

selbständig durch. Für das Lernen durch Experimentieren hieße unüberwacht also, daß das System die durchzuführenden Experimente bestimmt, die Experimente plant, das Ergebnis bewertet und einen entsprechenden Lernvorgang durchführt. Im folgenden werden die beiden Begriffe jedoch eingeschränkt auf die obige Sichtweise verwendet.

- inkrementell vs. nicht-inkrementell: *Inkrementelles* Lernen bedeutet, daß nicht alle zu lernenden Beispiele vor der Anwendung des Lernverfahrens bekannt sind, sondern, daß sukzessiv neue Beispiele in das Lernergebnis integriert werden können. Das bedeutet allerdings auch, daß nicht alle bereits eingelernten Beispiele wieder vollständig untersucht werden dürfen, sonst wäre jedes nicht-inkrementelle Verfahren trivialerweise auch inkrementell. Durch ein inkrementelles Vorgehen beim Lernen wird eine Verknüpfung von Lernen und Ausführung möglich [GLF89], d.h. das gelernte Wissen wird bei nachfolgenden Aufgaben eingesetzt, und das führt wiederum zu neuen Lernvorgängen, d.h. zu einer Erweiterung oder Korrektur des Wissens. Durch zusätzliche Beispiele oder Gegenbeispiele wird eine Hypothese entweder weiter generalisiert oder spezialisiert. Typische Beispiele inkrementeller Lernverfahren sind der Version Space Ansatz [Mit82], ID5R [Utg89] und die Begriffsbildungsverfahren UNIMEM [Leb87], COBWEB [Fis87] und CLASSIT [GLF89].

- Begriffs- vs. Regellernen (Klassifikator- vs. Planungswissen): Die meisten Arbeiten zum ML beschäftigen sich damit, wie für einen bestimmten Begriff eine geeignete hinreichende und notwendige Beschreibung gefunden bzw. wie aus einer Menge von Begriffen der entsprechende ausgewählt werden kann. Die Aufgabe dieses *Begriffslernens* läßt sich auch als Klassifikationsaufgabe auffassen. Gesucht ist eine Abbildung der Eingabewerte auf einen bestimmten Ausgabevektor, speziell eine Klassenzuordnung, d.h. $f(\vec{x}) = \vec{y}$, gesucht ist f. Dieses f soll beispielsweise aus einer Menge von vorgegebenen Beispielpaaren $((\vec{x_1}, \vec{y_1}), (\vec{x_2}, \vec{y_2}), \ldots)$ allgemein abgeleitet werden.

  Die Aufgabe des *Regellernens* soll hier definiert werden als die Formulierung von Regeln, die, nacheinander angewendet, einen Ausgangszustand in einen gewünschten Zielzustand überführen. Dazu gehört auch, daß die Auswahl der nächsten Regel vom jeweils gültigen Zustand erfolgen muß, und damit eine On-line-Entscheidung notwenig ist. Diese Aufgaben sind typische Probleme für Planungssysteme. Wichtig ist hier die verkettete Anwendung der Ergebnisregeln. Das Lernen von Regeln ist zu unterscheiden von der Repräsentation von Begriffen durch Regeln, z.B. „Wenn das Objekt bestimmte Eigenschaften hat, dann gehört es zur Klasse k_1".

- Abstraktion vs. Generalisierung: Zu unterscheiden sind die beiden häufig fälschlicherweise als äquivalent bezeichneten Begriffe der *Abstraktion* und der *Generalisierung*. Im Sprachgebrauch wird jedoch der Unterschied bereits deutlich: „Man abstrahiert von bestimmten Details." vs. „Diese Aussage kann man generalisieren/verallgemeinern." Abstraktion bedeutet eine Reduzierung der Informationen über einen Begriff, einen Plan, usw. *Konkretisierung* ist der inverse Vorgang, d.h. die Hinzunahme von weiteren Details. Bei der Abstraktion und Konkretisierung wird im allgemeinen die Beschreibungssprache verändert. Durch eine Generalisierung wird die Menge der Objekte, auf die eine bestimmte Eigenschaft zutrifft, vergrößert. Die inverse Operation ist die *Spezialisierung*. Michalski erläutert in [Mic91] den Unterschied folgendermaßen: Eine Aussage $d(S) = w$ bedeute, daß das Attribut d für die

Menge der Objekte in S den Wert w annimmt. Wird diese Aussage in $d(S) = w'$ geändert, wobei w' eine abstraktere Beschreibung darstellt, liegt eine Abstraktion vor. Wird die Aussage hingegen in $d(S') = w$ geändert, wobei S' eine Obermenge von S ist, liegt eine Generalisierung vor.

2.2.3 Wissensrepräsentation

In den Arbeiten zum ML werden verschiedenste Wissensrepräsentationsformen eingesetzt. Die am häufigsten verwendeten sind dabei:

- Einfache Attribut-Wert-Paare: Ein Beispiel wird durch eine feste Anzahl von Attributen und deren konkreter Wertbelegung repräsentiert. Die Werte können nominal oder numerisch sein. Beispiele: COBWEB [Fis87, Fis89], CLASSIT [GLF89].
- Aussagen- oder prädikatenlogische Ausdrücke (inklusive Hornformeln): Ein Beispiel ist durch einen entsprechenden Ausdruck beschrieben. Das Resultat des Lernvorgangs ist entsprechend dargestellt. Beispiele: ML-SMART [BGS88, BGGS89], Inductive Logic Programming [Mug93]
- Regeln: Ergebnisse eines Lernschritts können als Klassifikationsregel dargestellt werden. Außerdem werden Regeln zur Darstellung von Planungsoperatoren verwendet. Beispiele: AQ [MMHL86], Makrooperatoren
- Entscheidungsbäume: Das Resultat eines Lernvorgangs wird als Baum dargestellt mit inneren Entscheidungs- und äußeren Klassifikationsknoten. Beispiele: ID3 [Qui86], ID5R [Utg89]
- Semantische Netze: Beispiele für den Lernvorgang oder Resultate sind beschrieben durch eine relationale Struktur. Beispiel: ANALOGY [WBKL83, WBKL84]

2.2.4 Induktive Lernverfahren

Unter Induktion versteht man den Prozeß des plausiblen Schließens vom Speziellen zum Allgemeinen. Grundlage für die Plausibilität ist im allgemeinen eine große Zahl von zutreffenden Fällen. Haupteinsatzgebiete für induktive Verfahren sind das Begriffs- und das Regellernen. Das Ziel einer induktiven Begriffsbildung ist die Ableitung einer allgemeinen Begriffsbeschreibung aus einer Folge von Beispielen (und Gegenbeispielen) für einen bestimmten Begriff. Eine solche Begriffsbeschreibung kann nur eine induktive Hypothese darstellen, deren Güte z.B. durch Anwendung auf eine Testmenge bestimmt werden kann. Induktive Lernverfahren werden auch synthetisch genannt, weil sie - im Gegensatz zu deduktiven Verfahren - tatsächlich neues Wissen aufbauen, d.h. Wissen, das nicht durch streng logische Deduktion aus dem vorhandenen Wissen abgeleitet werden kann. Ein Problem induktiver Lernverfahren ist, daß meist eine große Menge an Beispielen benötigt wird, um eine „vernünftige" induktive Hypothese aufbauen zu können.

Ausgehend von einer Menge konkreter Fälle können in einem induktiven Vorgang im allgemeinen verschiedene Begriffsbeschreibungen abgeleitet werden, die eine mögliche Verallgemeinerung darstellen. Ein wichtiges Problem ist dann, in dieser Hypothesenmenge eine

„beste" Hypothese auszuwählen. Zu diesem Zweck muß ein sogenanntes Vorzugskriterium (engl. bias) definiert werden.

Definition 2.7 (Hypothesenvorzugskriterium) *Ein **Hypothesenvorzugskriterium** V definiert eine totale Ordnung auf der Menge aller möglichen Hypothesen. Eine Hypothese H wird gegenüber H' bevorzugt, wenn $H >_V H'$.*

Mögliche konkrete Beschreibungen für ein solches Vorzugskriterium sind:

- Wähle die einfachere Hypothese (Occam's Razor).
- Wähle die für den Benutzer verständlichere Hypothese.
- Wähle die Hypothese, deren Deskriptoren am einfachsten bestimmt werden können (Meßaufwand).
- Wähle die Hypothese, die die höchste Klassifikationsgenauigkeit besitzt.
- Wähle die Hypothese, die den geringsten Berechnungs- und Speicheraufwand benötigt.
- Wähle die Hypothese mit den wenigsten Konjunktionen.
- Wähle die Hypothese, die dem kleinsten Entscheidungsbaum entspricht.

In der Praxis können diese Vorzugskriterien aufgrund der notwendigen Formalisierbarkeit durch einen Hypothesenraumbias oder einen Präferenzbias ausgedrückt werden. Beim Hypothesenraumbias wird der Raum der möglichen Hypothesen eingeschränkt, indem beispielsweise nur logische Konjunktionen, lineare Schwellwertfunktionen etc. betrachtet werden. Für den Präferenzbias wird eine Funktion definiert, die einen Vergleich zwischen zwei Hypothesen zuläßt. Die Zeitkomplexität eines induktiven Verfahrens ist wesentlich von der Wahl des Vorzugskriteriums abhängig [SD90, S. 49].

Induktive Lernverfahren können weiter unterteilt werden in die Verfahren der empirischen Induktion und der konstruktiven Induktion. Während empirische Verfahren neben den konkreten Beispielen oder Beobachtungen nahezu kein weiteres Wissen einsetzen, verwenden konstruktive Verfahren a priori gegebenes Hintergrundwissen. Dieses Wissen kann in verschiedener Form gegeben sein: Regeln eines Experten für die Domäne, logische Implikationen und Äquivalenzen, heuristische Prozeduren zur Erstellung neuer Begriffe, etc. Eine Möglichkeit, dieses Wissen einzusetzen, sind abduktive Schlüsse, die weitere Aussagen über die vorliegenden Beispiele ableiten. Eine andere Möglichkeit ist die deduktive Ableitung der Belegung eines zusätzlichen Attributs aus der Attributbelegung eines Beispiels mit Hilfe des Hintergrundwissens. Dieses Attribut kann dann ebenfalls zur Klassifikation herangezogen werden (Konstruktive Generalisierung) .

Ergänzend zu der Klassifikation von Michalski wird in Abbildung 2.7 der Teilbereich der empirischen Induktion, der in dieser Arbeit die wichtigste Rolle spielt, noch weiter strukturiert. Im folgenden wird kurz auf die Zielsetzungen und Probleme der einzelnen Techniken des induktiven Lernens eingegangen und auf die wichtigsten Arbeiten verwiesen.

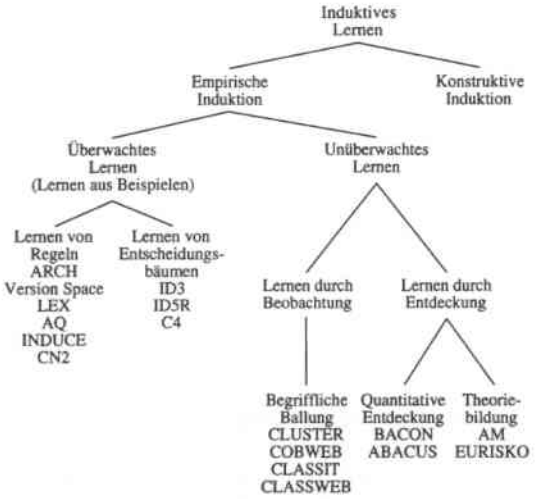

Abbildung 2.7: Induktive Lernverfahren

Empirische Induktion

Der Teilbereich der empirischen Induktion kann laut Abbildung 2.4 in die folgenden Gebiete weiter unterteilt werden: Empirische Verallgemeinerung oder überwachtes induktives Lernen, Begriffliche Ballung, Qualitative Entdeckung, Neuronale Netze, Genetische Algorithmen und einfaches fallbasiertes Lernen. Da als Wissen nur genau die Merkmale der vorliegenden Beispiele verwendet werden, wird diese Art der Induktion als *empirisch* bezeichnet. In einigen Arbeiten wurden verschiedene induktive Lernverfahren experimentell verglichen [WK89, FM89, MSTG89]. In [KHW91] werden fünf empirische, induktive Lernverfahren hinsichtlich ihrer Klassifikationsgenauigkeit, ihres Trainingsaufwandes und ihrer Verständlichkeit für den Benutzer im Rahmen eines weltweiten Vergleichs von 27 Lernverfahren [TBB+91] auf einer fest definierten Menge von Testbeispielen untersucht.

Empirische Verallgemeinerung Zum Bereich der empirischen Verallgemeinerung sind die mit Abstand meisten Arbeiten zu rechnen. Ziel ist die Verallgemeinerung von gegebenen Beispielbeschreibungen zu einer generellen Begriffs- oder Regelbeschreibung. Die in einem Schritt (nicht-inkrementell) oder sukzessive (inkrementell) aufgebaute Begriffs- oder Regelbeschreibung soll auf der Basis der vorliegenden Beispiele eine möglichst allgemeingültige Form annehmen.

Das Hauptproblem bei der empirischen Verallgemeinerung ist das Vorgehen, wie eine sinnvolle induktive Hypothese auf der Basis der a priori oder nacheinander vorliegenden Beispiele bestimmt werden kann. Dazu sind geeignete Generalisierungs- und Spezialisierungsoperatoren zu definieren, die die Veränderung der Hypothese auf der Basis eines oder

mehrerer Beispiele angeben. Michalski hat bereits in [Mic83] eine Großzahl der möglichen induktiven Generalisierungsoperatoren aufgeführt - die Spezialisierungsoperatoren sind dazu jeweils invers:

- Vergrößerung des Wertebereichs
- Schließen von Intervallen
- Aufsteigen in einer Taxonomie
- Erweiterung gegen Negativbeispiele
- Löschen von Bedingungsteilen von Regeln
- Hinzufügen von Alternativen
- Umwandlung von Konstanten in Variablen
- Umwandlung von konjunktiven Beschreibungen in disjunktive Beschreibungen

Für das Begriffslernen besteht die Aufgabe entweder darin, eine charakterisierende oder diskriminierende Beschreibung für jeden einzelnen Begriff (jede Klasse) zu bestimmen oder eine Struktur aufzubauen, die eine Einordnung eines neuen Beispiels zu einem Begriff zuläßt. Die resultierenden Begriffsbeschreibungen können dann z.B. durch aussagen- oder prädikatenlogische Formeln (Version Space [Mit82]), durch Wenn-Dann-Regeln (AQ [MMHL86], CN2 [CN89, CB91]) oder durch Entscheidungsbäume (ID3 [Qui86], ID5R [Utg89]) dargestellt werden. Sind für die vorliegenden Beispiele bereits der zuzuordnende Begriff, z.B. die Klasse, abgegeben, spricht man von überwachtem Lernen oder Lernen aus Beispielen (s. Abbildung 2.7).

Mitchell hat in seinem Version Space Ansatz [Mit82] erstmals das Problem, eine Begriffsbeschreibung aus Beispielen zu erlernen, vollständig als Suche im Raum möglicher induktiver Hypothesen untersucht. Ein Version Space ist die Menge aller Beschreibungen für einen Begriff, die zu allen Trainingsdaten (positiven und negativen) konsistent ist. Die möglichen Begriffe stehen untereinander durch „Spezieller-als"- bzw. „Allgemeiner-als"-Kanten in Verbindung. Damit läßt sich der aktuelle Version Space durch die Menge der möglichen allgemeinsten und speziellsten Begriffsbeschreibungen darstellen. Eine der stärksten Voraussetzungen des Version Space Ansatzes, nämlich die Konsistenz mit allen vorliegenden Daten, wurde von [Hir89] in seinem Ansatz zum „Incremental Version Space Merging" fallengelassen.

Auch die meisten nachfolgenden Arbeiten verwenden in irgendeiner Form ein Suchverfahren, das häufig durch einen Präferenzbias gesteuert wird. Als Suchverfahren können die bekannten Suchtechniken der KI eingesetzt werden: Vollständige Suche wie Tiefen- oder Breitensuche oder heuristische Suche wie Best-First-, Strahlsuche oder Hill Climbing. Daneben kann die Art der Suche im Raum der Hypothesen als Unterscheidungskriterium dienen. Die Hypothesen werden im allgemeinen bezüglich einer Ist-allgemeiner-Relation angeordnet. Dadurch ist eine Suche vom Speziellen zum Allgemeinen oder umgekehrt oder ein gemischter Ansatz [FW93] möglich.

Eine Anwendung der obigen induktiven Operatoren mit Schwerpunkt auf der Erweiterung gegen Negativbeispiele untersuchte Michalski in den verschiedensten AQ-Verfahren [MMHL86], die eine charakterisierende Beschreibung für jede Klasse liefern.

Eine grundlegend andere Methode verfolgen die Lernverfahren zum Lernen von Entscheidungsbäumen. Ziel ist eine Struktur, ein Entscheidungsbaum, die eine Einordnung eines Beispiels in eine der Klassen zuläßt. Ausgehend von der Wurzel wird in jedem inneren Knoten des Baumes ein Attribut getestet und je nach Wert einer der Nachfolgerbäume ausgewählt. Jedes Blatt des Baumes repräsentiert eine Klassenzuordnung. Zum Aufbau des Baumes wird über der Menge der jeweils betrachteten Beispiele das Attribut als nächstes Testattribut bestimmt, das den höchsten Informationsgewinn bringt.

Viele der grundlegenden Verfahren basieren zunächst auf einer Darstellung der Beispiele als Attribut-Wert-Paare mit diskreten Wertebereichen. Einige wurden jedoch später auf kontinuierliche Wertebereiche, auf die Behandlung unbekannter Attributwerte, auf den Umgang mit verrauschten Daten und auf die Berücksichtigung unterschiedlicher Kosten zur Bestimmung der Attributwerte erweitert.

Neben der Ableitung einer expliziten Beschreibung für jeden einzelnen Begriff/jede Klasse ist es möglich, einzelne Instanzen abzuspeichern (engl. instance based learning) und dann ähnlich zu einem Nächster-Nachbar-Klassifikator die tatsächliche Klassifikation durchzuführen [AK89, AKA91].

Die bisher beschriebenen Lernverfahren zur empirischen Induktion beschränkten sich ausschließlich auf das Lernen von Begriffen. Beim komplexeren Lernen von Regeln sollen „Bausteine" einer möglichen Lösungskette (z.B. für einen Planer) gelernt werden. Das Lernen von Begriffen kann dabei als Teilaufgabe auftreten, wenn die Bedingung einer Regel als Begriff aufgefaßt wird, der möglichst genau charakterisiert werden soll.

Die allgemeinste Definition der Aufgabe von Regellernverfahren ist es, eine eventuell leere Menge von Regeln der Form $B \rightarrow K$ (Bedingung impliziert Konklusion) so zu modifizieren, daß sie korrekt sind und in der Anwendung richtig verwendet werden. Nach [BSP85] können bei Benutzung der Regelmenge folgende Fehler auftreten: 1. Faktische Fehler: Eine Regel ist falsch, 2. Steuerungsfehler: Die Regeln sind richtig, aber der auf ihnen basierende Ablauf ist falsch. Dies gilt sowohl für Regeln, die zu allgemein sind, als auch für Regeln, die zu speziell sind. Der Ablauf eines Regellernprogramms ist dann: Bis die Regeln zufriedenstellend sind: 1. Finde einen Fehler in einer Regel durch einen Kritiker, 2. Modifiziere die Regel, um den Fehler zu beseitigen.

Aufgabe des Kritikers ist es, in der Folge der angewandten Regeln die fehlerhaften Stellen herauszufinden und dem Modifizierer eine Menge von Tripeln der Art (Bewertung der Instanz, d.h. positiv/negativ; Regel; Kontext, d.h. Variablenbindungen) zu übergeben. Eine typische Kritikertechnik ist die Verwendung von idealen Traces, d.h. die exakte Angabe, welche Regeln angewendet werden sollen und in welcher Reihenfolge. Damit können sowohl faktische als auch Steuerungsfehler erkannt werden. Die fehlerhafte Stelle ist bei dem Vergleich von idealem Trace und realer Anwendung die erste Stelle einer abweichenden Regelanwendung. Die Anwendungen der Regeln vorher waren korrekt, die jeweiligen Kontexte können daher als positive Trainingsinstanzen angesehen werden. Der Kontext am Unterscheidungspunkt gilt als negatives Beispiel für diese Regel. Das Ordnen der Regeln kann entweder durch explizite Ordnungsregeln oder durch die Vergabe von Prioritäten an die einzelnen Regeln geschehen.

Hauptproblem dieses Ansatzes ist natürlich die Forderung nach der Vorgabe eines idealen Trace. In den bisherigen Ansätzen wurde angenommen, daß dieser entweder von einem Benutzer vorgegeben oder von einem Programm erzeugt wird, der bzw. das das zu lernende Wissen schon besitzt. Dies wird beispielsweise dadurch realisiert, daß exakt die zu lernenden Regeln existieren oder daß ein vollständiger Ableitungsbaum aufgebaut wird und die kostengünstigste Regelfolge als ideal angesehen wird. Diese Forderung ist für komplexere Aufgabenstellungen nicht haltbar. Erstens will der Benutzer nicht damit belastet werden, jede einzelne Regelanwendung explizit vorzugeben und zweitens ist es ihm eventuell auch gar nicht möglich, bei einer großen Menge von Regeln zu beurteilen, welches die nächste sein soll. Hier müssen also prinzipiell neue Techniken entwickelt werden, wie ein Benutzer Beispiele vorführen kann, wie ein System diese analysieren und daraus lernen kann. Die in dieser Arbeit entwickelten Methoden werden detailliert in Kapitel 6 behandelt.

Begriffliche Ballung Ziel *begrifflicher Ballungsverfahren* (engl. conceptual clustering), die auch als *Lernen durch Beobachtung* bezeichnet werden, ist die Strukturierung einer Menge von Beispielen in einzelne Ballungen oder Klassen. Im Gegensatz zum Lernen aus Beispielen, d.h. dem überwachtem Lernen, sind die Beispiele hier keiner Klasse zugeordnet, d.h. die Verfahren sind unüberwacht. Die Beispiele einer Ballung sollten dann möglichst homogen sein, die Beispiele verschiedener Ballungen möglichst disjunkt. Um diese Eigenschaft bewerten zu können, muß ein Ballungsbewertungskriterium definiert werden.

Die bisher entwickelten Verfahren zur begrifflichen Ballung unterscheiden sich in folgenden Punkten: 1. Vorgabe der Anzahl der notwendigen Ballungen a priori oder automatische Bestimmung der besten Anzahl, 2. Einstufige Bestimmung der Ballungen oder eine ganze Hierarchie von Ballungsbeschreibungen als Resultat, 3. Vorgabe aller Beispiele oder inkrementeller Einbau neuer Beispiele möglich und 4. Ableitung einer expliziten Ballungsbeschreibung oder nicht. Diese Ableitung einer Beschreibung kann entweder nachträglich mit den oben beschriebenen Begriffslernverfahren durchgeführt oder während des Aufbaus vorgenommen werden. Als Beispielrepräsentation untersuchen nahezu alle bisherigen Arbeiten ausschließlich Attribut-Wert-Paare.

Beispiele für derartige Systeme sind CLUSTER [MS83], CLUSTER/S [SM86], die nichtinkrementell eine feste Anzahl von einstufigen Klassen bestimmen. Die Beschreibungen werden durch das gleiche Verfahren wie bei AQ abgeleitet. EPAM [FS84], UNIMEM [Leb86, Leb87], COBWEB [Fis87, Fis89] und CLASSIT [GLF89, Gen90] liefern hierarchische Strukturen und können inkrementell arbeiten. Die notwendige Anzahl von Ballungen ergibt sich automatisch. Eine Beschreibung der Klassen wird durch Wahrscheinlichkeitsverteilungen erreicht, durch die sogar eine Vorhersage fehlender Attributwerte für ein zugeordnetes neues Beispiel möglich ist. Während in COBWEB symbolische, diskrete Werte als Attributwerte zugelassen sind, behandelt CLASSIT bei einem ansonsten sehr ähnlichen Vorgehen numerische Attributwerte.

Eine Erweiterung von COBWEB zur Behandlung von Beispielen, die nicht durch Attribut-Wert-Paare sondern durch Relationsbeschreibungen ausgedrückt werden, wird kurz in [AT91] mit COBWEB_R vorgestellt.

Qualitative und quantitative Entdeckung Michalski führt in seiner Klassifikation lediglich die qualitative Entdeckung auf. Ebenso wichtig sind jedoch die Arbeiten zu

quantitativer Entdeckung. Beide Ansätze zusammengenommen, werden als *Lernen durch Entdeckung* bezeichnet (s. Abbildung 2.7). Ziel ist es, quantitative oder qualitative Zusammenhänge zwischen Werten abzuleiten. Dies sind z.B. aufgenommene Meßwerte, für die ein expliziter funktionaler Zusammenhang bestimmt werden soll. Aus diesem Grund wird dieser Teilbereich auch *funktionale Induktion* genannt. Die wenigen bisher entwickelten Verfahren (z.B. BACON [LZSB86], ABACUS [FM90]) unterscheiden sich darin, ob davon ausgegangen wird, daß für eine beliebige Kombination von Eingangswerten der Ausgangswert, z.B. durch ein allwissendes Orakel, bekannt ist, ob die funktionalen Strukturen vorgegeben sind und nur entsprechende Parameter bestimmt werden müssen, und ob die Funktionswerte durch eine einzige oder mehrere Teilfunktionen wiedergegeben werden. In ABACUS wurden mit AQ für diese Teilfunktionen entsprechende Bedingungen für den Zeitpunkt des Einsatzes generiert.

Subsymbolische Lernverfahren Die Trennung in symbolische und subsymbolische Lernverfahren ist ursprünglich dadurch begründet, daß symbolische Verfahren auf rein logik-orientierten expliziten Ausdrücken arbeiten, während subsymbolische Verfahren stärker numerisch orientiert Parameter verändern, die für einen menschlichen Benutzer nur noch schwer zu interpretieren sind. Die Ausnutzung von a priori vorhandenem Wissen ist nur sehr eingeschränkt möglich. Von den zugrundeliegenden Inferenzverfahren aus gesehen sind die Verfahren induktiv. Größte Bedeutung haben die Verfahren erlangt, die auf künstlichen neuronalen Netzen als Struktur arbeiten. Diese Methoden können auch wieder überwacht (z.B. Perzeptron, Backpropagation) oder unüberwacht (z.B. Kohonennetze) sein.

Andere Verfahren, die als subsymbolisch bezeichnet werden, sind die Genetischen Algorithmen [DJ90], die ihre Basisoperatoren an den genetischen Vorgängen der Evolution orientieren, wie Mutation, Crossover, Rekombination und Selektion.

Eine starke Ähnlichkeit zum Lernen von Regeln hat der Bereich des Reinforcement Learning, bei dem es im wesentlichen um das Lernen der richtigen Auswahl einer Aktion in einer Situation geht. Im Grunde werden die gleichen Probleme behandelt, nämlich, wie eine Bewertung der Kette aussehen kann, wo in der Ableitungskette Veränderungen vorgenommen werden sollen und wie die Auswirkung auf die einzelne Auswahl berechnet werden kann.

Konstruktive Induktion

Im Bereich der konstruktiven Induktion [Mic83, WE87, EW91] wird untersucht, wie Hintergrundwissen den induktiven Lernprozeß steuern oder verbessern kann. Als Hintergrundwissen gelten dabei sämtliche Informationen, die nicht explizit in den Beispielen vorliegen. Dies kann im einfachsten Fall eine Hierarchie sein, die den Zusammenhang zwischen verschiedenen Attributwerten herausstellt. Diese Hierarchie kann natürlich auch in Form von prädikatenlogischen Formeln dargestellt werden. In allgemeineren Ansätzen werden beliebige in PL1 oder Hornlogik ausdrückbare Formeln als Hintergrundwissen betrachtet. Die Ausnutzung von Hintergrundwissen wird im wesentlichen im Bereich des Lernens aus Beispielen, d.h. des überwachten Lernens untersucht (z.B. ML-SMART [BGS88, BGGS89]).

In [TLI91] wird analysiert, wie Hintergrundwissen im Bereich der begrifflichen Ballung eingesetzt werden kann.

2.2.5 Deduktive Lernverfahren

Bei den deduktiven oder analytischen Lernverfahren wird versucht, bereits vorhandenes Wissen zu analysieren und in eine effektivere oder „operationalere", d.h. direkt vom System verwendbare, Form zu bringen. Dies kann dadurch geschehen, daß Teile des Wissens umformuliert oder expliziter gemacht werden. Vorgegeben ist ein meist als konsistent und vollständig angenommenes Bereichswissen, wie z.B. Planungs- oder Klassifikationswissen, ein oder mehrere Beispiele, z.B. konkrete Planungsaufgaben und -lösungen oder ein Beispiel für einen schon bekannten Begriff [MKKC86, DM86, MCK+89]. Durch eine gerechtfertigte Generalisierung, die Reformulierung von Wissen, das Erzeugen von Makros oder das Erzeugen von Kontrollwissen soll die Leistungsfähigkeit des Systems verbessert werden. Vordergründig handelt es sich bei dieser Lernmethode vor allem um ein „Speedup Learning", tatsächlich kann aber auch eine Verbesserung der Problemlösungsfähigkeiten erreicht werden, weil durch diesen Speedup bestimmte Problemlösungen überhaupt erst untersucht werden können. Im Gegensatz zu induktiven Verfahren ist die Anwendung deduktiver Lernverfahren wahrheitserhaltend, d.h. aus bestehendem wahren Wissen wird durch deduktive Lernverfahren wieder wahres Wissen abgeleitet.

Das deduktive Lernen kann laut Abbildung 2.4 weiter untergliedert werden in die axiomatische Deduktion, die reines erklärungsbasiertes Lernen und die automatische Programmsynthese umfaßt, und die konstruktive Deduktion mit Abstraktion und deduktiver Generalisierung.

Die wichtigste Methode der deduktiven Lernverfahren ist das erklärungsbasierte Lernen (EBL). Dabei wird vom Grundansatz her von folgenden gegebenen Dingen ausgegangen:

1. Zielbegriff: Eine Beschreibung des zu lernenden Begriffs, die nicht das Operationalitätskriterium erfüllt.
2. Trainingsbeispiel: Ein Beispiel für den Zielbegriff
3. Bereichstheorie: Regeln und Fakten, die erklären, warum das Trainingsbeispiel ein Beispiel für den Zielbegriff ist.
4. Operationalitätskriterium: Ein Prädikat, das die Form spezifiziert, in der die erlernten Beschreibungen vorliegen sollten.

Gesucht ist eine Generalisierung des Trainingsbeispiels, die eine geeignete Beschreibung des Zielbegriffs darstellt und das Operationalitätskriterium erfüllt. Das erklärungsbasierte Lernen geschieht in zwei Schritten: 1. Erklärung des Beispiels und 2. Generalisierung der Erklärung. Im ersten Schritt wird das Beispiel unter Zuhilfenahme des Hintergrundwissens erklärt, d.h. es wird eine Begründung auf der Basis der Bereichstheorie dafür abgeleitet, daß das Trainingsbeispiel tatsächlich zum Zielbegriff gehört. Diese Erklärung wird soweit durchgeführt, bis alle Elemente der Erklärung dem Operationalitätskriterium genügen. Der zweite Schritt besteht in der Analyse der Erklärung selbst, um eine neue

allgemeingültige Beschreibung für den Begriff zu bilden. Bestimmte Merkmale und Randbedingungen des Beispiels werden soweit als möglich generalisiert, wobei die Erklärung erhalten bleiben muß. Der auf diese Weise entstandene allgemeine Begriff umfaßt nun neben dem ursprünglichen Beispiel auch alle weiteren, die mit der gleichen Erklärungsstruktur bearbeitet werden können.

Einen weiteren Schwerpunkt erklärungsbasierter Lernverfahren bildet die Klasse der Verfahren, die Makrooperatoren für Planungssysteme lernen . Basierend auf einer Liste oder einer strukturierten Darstellung von angewendeten Operatoren bei der Lösung einer Planungsaufgabe soll ein einziger Operator abgeleitet werden. Dieser Makrooperator oder „Chunk" soll bei einer zukünftigen Planungsaufgabe als zusätzlicher Operator eingesetzt werden. Dadurch wird die Tiefe des Suchraums zwar verringert, gleichzeitig erhöht sich jedoch die Breite des Suchraums, da die Menge anwendbarer Operatoren vergrößert wird (→ Utility problem). Der Zusammenhang mit dem obigen reinen EBL ist, daß eine Erklärung und Generalisierung dafür abgeleitet werden muß, wann der Makrooperator angewendet werden kann.

Schließlich können auch die Ansätze der *Wissensübersetzung* (engl. knowledge compilation) zu den deduktiven Lernverfahren gerechnet werden, obwohl, begründet durch deren Herkunft aus Diagnoseexpertensystemen, diese Eingruppierung selten durchgeführt wird. Nach [Byl91] ist Wissensübersetzung die Anwendung einer Funktion c auf Wissen K. Daraus entsteht $K' = c(K)$. Dieses Wissen sollte nun in irgendeiner Form nützlicher sein als K. Beispiel: Die Übertragung von Modellwissen in explizite Diagnoseregeln. Auch hier wird kein wirklich neues Wissen gelernt, sondern das Wissen wird in eine effizientere Darstellungsform transformiert.

Probleme, die in allen deduktiven Lernverfahren auftreten, sind zum einen die Forderung, daß das Hintergrundwissen korrekt, vollständig und anwendbar ist. Zum anderen ist die Nützlichkeit von zusätzlich abgeleitetem deduktivem Wissen geeignet zu bewerten, d.h. ob eine andere Form der Begriffsbeschreibung oder ein Makrooperator zum vorhandenen Wissen hinzugenommen werden soll, oder dadurch insgesamt längere Klassifikations- oder Planungszeiten entstehen als vorher. Für diese Probleme gibt es bereits erste Lösungsansätze [Min88], auf die hier allerdings nicht eingegangen werden soll.

Konstruktive Deduktion

Als konstruktive Deduktion wird ein Vorgang bezeichnet, bei dem basierend auf Hintergrundwissen Beschreibungen von einer Repräsentationsform in eine andere überführt werden, dabei aber die „wichtigen" Informationen erhalten bleiben. Ein Beispiel ist die Abstraktion, bei der eine sehr detaillierte Beschreibung in eine weniger detaillierte überführt wird. Bleibt die Repräsentationsform gleich, aber der Gültigkeitsbereich für eine Aussage wird auf Grund von Hintergrundwissen deduktiv vergrößert, liegt eine deduktive Generalisierung vor.

2.2.6 Integration verschiedener Lernstrategien

Obwohl sich die Mehrzahl aller Arbeiten des Maschinellen Lernens auf eine bestimmte Lernstrategie beschränkt, gibt es schon seit Mitte der 80er Jahre Bestrebungen, ver-

schiedene Methoden zu kombinieren. Diese führten zum einen zum Bereich des Lernens durch Analogien und dem sehr ähnlichen Gebiet des fallbasierten Lernens und Schließens. Zum anderen wurden verstärkt Untersuchungen angestellt, induktive und erklärungsbasierte oder symbolische und subsymbolische Verfahren gemeinsam einzusetzen. Dadurch sollen entweder starke Anforderungen an eines der Lernverfahren, z.B. der Korrektheit und Vollständigkeit des Bereichswissens, überwunden oder die Vorteile verschiedener Verfahren genutzt werden. In den 90er Jahren wurde das aufgabenadaptive Lernen mit mehreren Strategien (engl. multistrategy learning) eingeführt, bei dem mehrere Lernstrategien zur Verfügung stehen und für die aktuelle Aufgabe jeweils entschieden werden soll, welche jeweils am besten eingesetzt werden sollte.

Lernen aus Analogien

Eine Kombination von induktiven und deduktiven Komponenten bildet das Lernen aus Analogien. Dabei soll eine Lösung für ein neues Problem oder eine Beschreibung für einen neuen Begriff dadurch gefunden werden, daß in einer Wissensbasis bereits bekannter Lösungen bzw. Begriffe ein ähnlicher Fall gesucht wird und die Informationen darüber auf den neuen Fall übertragen werden. Dazu muß eine Ähnlichkeitsmetrik definiert werden, die die Ähnlichkeit zweier Probleme angibt. Für ein gefundenes ähnliches Problem muß eine Ähnlichkeitsabbildung definiert werden, mit der die Transformation des Wissens durchgeführt werden kann. In einem letzten Schritt muß das übertragene Wissen, das im wesentlichen induktiven Charakter hat, noch auf Konsistenz mit sonstigem Wissen geprüft und entsprechend in die Wissensbasis eingebaut werden. In diesem Schritt können selbst wieder induktive Lernverfahren angewendet werden, um eine Generalisierung oder Spezialisierung der Komponenten der Wissensbasis zu erreichen.

Die Arbeiten im Bereich des Lernens durch Analogien lassen sich den zwei Anwendungsgruppen Problemlösung oder Planen und Bilden oder Verfeinern von Begriffen zuordnen. Die Arbeiten zum Einsatz von Analogien bei der Problemlösung [Car83, Car86] untersuchen, wie für ein neues Problem eine passende Lösung aus der bekannten Lösung bzw. den Lösungsweg eines ähnlichen, bereits gelösten Problems abgeleitet werden kann. Ziel beim Bilden und Verfeinern von Begriffen [KC86, Hal89] ist die Bestimmung einer genaueren Beschreibung für einen neuen Begriff durch Übertragen von Informationen über einen bekannten analogen Begriff (Bsp.: Sonnensystem → Atommodell von Rutherford).

Integration von induktivem und erklärungsbasiertem Lernen

In [Leb86] wurde erstmals die Integration von induktivem und erklärungsbasiertem Lernen in UNIMEM untersucht. EBL wird dabei nicht auf ein einziges Beispiel angewandt, sondern auf eine Generalisierung, die aus einer größeren Menge von Beispielen entstanden ist. Eine Kombination empirischer Generalisierung und erklärungsbasierten Lernens wird in [Dan87, Cai90, PK90] durchgeführt, um unvollständiges oder inkorrektes Hintergrundwissen behandeln zu können. Eine Kombination von erklärungsbasiertem Lernen und dem Version space-Ansatz wird in [Hir89] untersucht. Hirsh wendet zunächst erklärungsbasiertes Lernen auf seine Trainingsdaten an und anschließend darauf seine abgewandelte Form des Version Space Ansatzes. Neuere Ansätze zur Integration von verschiedenen Lernkomponenten sind [BL90, KRTD90, EW91].

Eine Kombination von EBL, Lernen aus Analogien, empirischem Lernen und Lernen durch Befragen des Benutzers wird im System DISCIPLE [TK90, Tec91] eingesetzt. DISCIPLE ist als Learning Apprentice System in der Lage, aus Beispielproblemlösungen des Benutzers allgemeine Problemlöseregeln aufzubauen. Im Gegensatz zu Systemen wie LEAP [MMS85] oder GENESIS [DM86] geht dieser Ansatz jedoch nicht davon aus, daß ein vollständiges Domänenwissen zur Verfügung steht und lediglich mit Hilfe von EBL ein Beispiel bearbeitet werden muß, um die gewünschte allgemeine Problemlöseregel zu erhalten, sondern das Wissen wird durch Benutzerinteraktion sukzessive aufgebaut.

Einen umfassenderen Systementwurf zur Integration verschiedener Lernverfahren im Bereich Planen und Problemlösen stellt Prodigy [CG87, CG90, Etz91, KME91, MCK+89, VC90, VC91] dar. In dieser Architektur werden die Methoden Lernen durch Analogien, Lernen von Makrooperatoren, Lernen durch Experimentieren gemeinsam eingesetzt. Der zugrundeliegende Planer, um den alle Komponenten gruppiert sind, basiert auf einer Means-Ends-Analysis und verwendet eine Bereichs- und Operatorrepräsentation ähnlich zu STRIPS.

Integration von symbolischen und subsymbolischen Methoden

Im Hinblick auf fortgeschrittenere Systeme ist eine Kombination von symbolischen und subsymbolischen Lernverfahren sinnvoll, um die jeweiligen Stärken dieser Ansätze gemeinsam für eine Anwendung nutzen zu können. Während die getrennte Anwendung derartiger Verfahren, z.B. auf den unterschiedlichen Ebenen eines Robotersteuerungssystems, problemlos denkbar ist, steht die Forschung bezüglich einer echten Integration beider Techniken noch am Anfang [HR90a, MS91, ST89b, ST89a, TSN90, TS92]. Ziele, die langfristig erreicht werden sollen, sind unter anderen die Extraktion symbolischer Informationen aus subsymbolischen Teilkomponenten und eine Steuerung oder auch Vorbelegung, d.h. Initialisierung, der subsymbolischen Komponenten durch die symbolischen.

2.2.7 Kritische Bewertung

Die Forschung im Maschinellen Lernen hat in den letzten 10 Jahren eine Vielzahl von unterschiedlichen Methoden hervorgebracht, die viele verschiedene Probleme lösen - leider oft nur künstlich geschaffene, stark vereinfachte Probleme. Eine vereinheitlichende Untersuchung oder Beschreibung der verschiedensten Lernmethoden oder eine klar definierte Terminologie gibt es bisher nicht. Als Beispiele werden oft sehr konstruierte, einfache Fälle untersucht, so daß die Übertragung der Lernmethoden auf reale Probleme vielfach noch offen ist.

Trotzdem sind eine Reihe von Basisideen vorhanden, die zumindest prinzipiell auch für komplexere Anwendungen eingesetzt bzw. weiterentwickelt werden können. In einigen Arbeiten wurden z.B. für einige Teilprobleme von Robotersystemen Lernverfahren entwickelt bzw. angewendet. Das nächste Kapitel beschäftigt sich mit einer Analyse dieser Arbeiten und einer genauen Betrachtung der prinzipiellen Probleme und Einsatzmöglichkeiten von Maschinellem Lernen in autonomen Robotersystemen.

Kapitel 3

Einsatz von Lernkomponenten in Robotiksystemen

Nachdem im letzten Kapitel die Grundlagen von autonomen Robotersystemen und des Maschinellen Lernens getrennt erläutert wurden, untersucht dieses Kapitel ausführlich den Einsatz von Lernmethoden in Robotiksystemen (siehe auch [Kre92]). Neben der prinzipiellen Analyse der Einsatzmöglichkeiten (Kap. 3.2 und 3.3) wird eine Einordnung der verschiedenen bestehenden Ansätze in ein neu entwickeltes Klassifikationsschema vorgenommen (Kap. 3.4). Anschließend werden die bekannten Arbeiten kurz erläutert und kritisch bewertet (Kap. 3.4 und 3.5). Eine wichtige Informationsquelle für Lernvorgänge ist die Interaktion mit einem Lehrer. Aus diesem Grund wird im letzten Abschnitt dieses Kapitels (Kap. 3.6) auf Methoden eingegangen, wie ein Mensch Vorführungen für ein Robotersystem vornehmen und diesem so Beispiele für Problemlösungen liefern kann.

3.1 Übersicht

Ein wichtiger Aspekt bei der Entwicklung von autonomen Systemen ist das Problem der initialen Akquisition von Wissen und der sukzessiven Korrektur und Verbesserung dieses Wissens im Laufe des Einsatzes des Systems. Eine mögliche Lösung dieser Problematik ist der Einsatz von Methoden des Maschinellen Lernens.

Die Anwendung von Methoden des Maschinellen Lernens in der Robotik ist insbesondere dann von Bedeutung, wenn das zu lösende Aufgabenspektrum eine oder mehrere der folgenden Eigenschaften erfüllt:

- Aktionen in teilweise unbekannter Umgebung: Die Aufgabe kann a priori nicht vollständig geplant werden; das System muß zunächst oder besser kontinuierlich Wissen über seine Umwelt akquirieren.
- Auftreten unerwarteter Ereignisse: Während der Lösung einer Aufgabe treten Ereignisse auf, mit denen nicht gerechnet wurde. Das System muß lernen, wie darauf adäquat reagiert werden kann und wie der ursprüngliche Plan abgeändert werden kann, falls ein Auftreten dieser Ereignisse ausgeschlossen werden soll.

- Unsicherheiten in Aktorik und Sensorik: Um eine Aufgabe trotzdem sicher zu lösen, muß sich der Roboter sowohl aufgabenspezifisch als auch aufgabenunabhängig entsprechend adaptieren.
- Mittlere bis hohe Komplexität der Programmierung bzw. der Wissensbasen: Ist das zur Lösung von Aufgaben nötige Wissen derart umfangreich, daß es einem menschlichen Experten nicht mehr sinnvoll möglich ist, dasselbe zu formulieren, müssen neue Arten der Programmierung, z.B. das angesprochene Programmieren durch Vorführen, eingesetzt werden und das System daraus entsprechend lernen.
- Unvollständige oder inkorrekte Wissensbasen: Im allgemeinen ist es nicht möglich, für ein System von der Komplexität eines autonomen Roboters vollständige und korrekte Wissensbasen zu erzeugen. Daher ist es wichtig, daß das Systemwissen durch Lernmethoden erweitert und korrigiert werden kann.

Die Ziele des Einsatzes von ML-Techniken sind eine kurzfristige Adaption des Systems an seine aktuelle Umwelt und langfristig ein Lernvorgang, der zu geeigneteren Aktionen bei zukünftigen Aufgabenstellungen führt. Dieser Lernvorgang soll eine bessere Perzeption, eine schnellere und geeignetere Reaktion auf Informationen und eine bessere Antizipation der Auswirkungen eigener Aktionen bewirken. Neben diesen Kriterien, die eine Steigerung der Performanz, d.h. der Effizienz, fordern, geht es beim Einsatz von Lernverfahren auch um die prinzipielle (initiale) Wissensakquisition, d.h. den Aufbau von Wissensbasen für autonome Robotersysteme.

In den vergangenen Jahren ist das Interesse am Einsatz von Methoden des ML in der Robotik zunehmend gestiegen. Dabei werden sowohl Anwendungen für Manipulatoren als auch für mobile Systeme untersucht. Lerntechniken, die eingesetzt werden, reichen von Auswendiglernen und Lernen durch Instruktion über induktive Verfahren, Lernen aus Analogien bis hin zu deduktiven Ansätzen und dem Einsatz von fallbasierten Methoden. Viele dieser Systeme behandeln jedoch nur ein sehr spezielles Teilproblem der Robotik und nehmen oft Vereinfachungen an, die den Schritt von einer simulierten Anwendung hin zu einer realen sehr schwierig, wenn nicht sogar unmöglich machen. Grund dafür ist zu einem großen Teil die Tatsache, daß zur Zeit eine große Diskrepanz zwischen Anwendungen des Maschinellen Lernens und den Anforderungen der Robotik besteht. Auf diese Diskrepanz wird in Abschnitt 3.3 eingegangen.

3.2 Lernziele in der Robotik

In diesem Abschnitt werden die verschiedenen Einsatzmöglichkeiten von Lernverfahren in den Teilkomponenten eines autonomen Robotersystems untersucht. Der Schwerpunkt der Untersuchungen liegt auf der echten Wissensakquisition, d.h. der Lernvorgang wirkt sich auf das zugrundeliegende Systemwissen aus und nicht nur auf die Ausführung einer aktuellen Aufgabe. Letzteres ist eher als Adaption oder Optimierung zu bezeichnen.

Bei dieser (Nach-)Optimierung wird im allgemeinen eine vorgegebene oder über herkömmliche modellbasierte Ansätze erhaltene Aktionsfolge so adaptiert, daß die gestellte Aufgabe möglichst effizient und sicher gelöst wird. Gegenstand der Adaption sind im wesentlichen

die Parameter der einzelnen Positionierbefehle. Beispiel: Beim wiederholten automatischen Einstecken eines Bauteils in eine Grundplatte werden über eine Menge von gleichen Aufgaben ständig ungewollte Kräfte gemessen. Dies kann durch eine Ungenauigkeit der Zuführ-/Positioniereinrichtung oder eine Verstellung des Roboterarms bedingt sein. Ziel ist es, die Kraftmessungen so auszuwerten, daß eine Korrektur der angefahrenen Position wieder zu einer Bewegung mit geringeren auftretenden Kräften führt.

3.2.1 Perzeption

Aufgabe der perzeptiven Komponenten eines Systems ist die Verarbeitung von Sensorsignalen aller Art. Dazu gehören Vorverarbeitung, Merkmalsextraktion und Klassifikation. Insbesondere für den Bereich der Klassifikation können viele der in Kapitel 2.2 aufgeführten Lernverfahren eingesetzt werden, z.B. Lernen aus Beispielen, Lernen aus Beobachtung oder EBL.

Eine wichtige Form des Lernens von Klassifikatoren ist das *Lernen zur Klassifikation von Objekten* aus Sensorsignalen, z.B. aus 2D- oder 3D-Vision, taktilen Daten, Ultraschall- oder Infrarotdaten. Ziel ist in diesen Fällen immer, eine möglichst gute Abbildung von einer Menge von aus den aufgenommenen Sensorwerten extrahierten Merkmalen auf die möglichen Klassen zu bestimmen. Diese Klasse kann selbst wieder Merkmal für eine höhere Kategorisierung sein, so daß sich insgesamt eine Abstraktionshierarchie von Merkmalen ergibt. Beispiele auf den einzelnen Ebenen sind Kantentypen, Objekttypen oder Szenentyp. Dabei kann ein ganzes Spektrum von Klassifikationsaufgaben untersucht werden, je nach der a priori vorhandenen Modellierung der auftretenden Objekte und der Abstraktionsebene der extrahierten Merkmale. Sind beispielsweise alle möglichen Objekte vorher bekannt, so kann ein Klassifikator aufgebaut werden, der, wenn möglich, eine Aufteilung des Merkmalsraums bestimmt. Sind nicht alle Objekte bekannt, so muß ein inkrementelles Lernen während des Einsatzes möglich sein.

Neben der Zuordnung der vorliegenden Objekte aus einer punktuellen Menge von Messungen gehört die *Situationsanalyse* zu den wichtigsten Kandidaten für den Einsatz von Lernverfahren in der Perzeption. Aufgabe ist die Zuordnung - im allgemeinen einer Historie - von gemessenen Sensorwerten zu einem bestimmten erreichten Zustand. Aus der Folge gemessener Kraft-/Momentenwerte kann beispielsweise abgeleitet werden, daß der Zielzustand einer Fügeaufgabe erreicht ist.

Eine weitere Lernaufgabe, die Planungskomponenten im Bereich der Perzeption entspricht, ist das *Lernen von Sensoreinsatzplänen/-strategien*, d.h. wie und wann welche Sensoren eingesetzt werden sollten, um eine möglichst erfolgreiche oder möglichst effiziente Durchführung einer Aktion zu erreichen.

3.2.2 Modellierung

Der Begriff Modellierung zielt hier nicht auf die verschiedenen Modelle ab, die zur Repräsentation von Objekten, Roboter etc. verwendet werden, sondern auf die Frage, wie eine teilweise unbekannte Umgebung mit teilweise unbekannten Objekten intern modelliert werden kann. Eine erste Lernaufgabe ist die *Akquisition eines Weltmodells*, d.h. der Aufbau einer Repräsentation der aktuellen Umgebung des Systems unter Ausnutzung seiner

perzeptiven Fähigkeiten. Dazu gehört die Bestimmung von wahrnehmbaren Objekten und Hindernissen. Diese Aufgabe kann zunächst als reines Auswendiglernen angesehen werden, da entsprechende Wissensinhalte einfach abgespeichert werden. Es ist jedoch dabei zu berücksichtigen, daß die zugrundeliegenden Sensordaten im allgemeinen verrauscht und fehlerhaft sein können, so daß oft eine Fusion mehrerer Sensormessungen oder mehrerer verschiedener Sensoren durchgeführt werden muß. Im Bereich der mobilen Systeme wird diese Lernaufgabe als *Kartographierung* bezeichnet. Dieser Begriff wurde bei der Manipulation bisher nicht verwendet, drückt das Problem bei teilweise unbekannten Umgebungen aber sehr gut aus. Die Komplexität dieser Lernaufgabe ist wesentlich von der gewünschten Abstraktionsstufe der Modellierung abhängig. Dies hängt direkt mit den perzeptiven Fähigkeiten des Systems zusammen. So kann aus einer Menge von Ultraschallmessungen in einem Raster lediglich eingetragen werden, ob eine bestimmte Zelle frei oder besetzt ist, oder das Ziel einer kognitiven Karte angestrebt werden, in der letztendlich sogar eine Klassifikation von Objekten stattfindet.

Ein weiteres Lernziel ist die *Akquisition von Objekteigenschaften*, d.h. die Bestimmung, welche Eigenschaften ein (unbekanntes) Objekt besitzt, wie z.B. Massenverteilung oder stabile Lagen, oder wie es sich bei bestimmten Aktionen verhält, z.B. das Verhalten bei einer Schiebebewegung. Diese Eigenschaften können entweder durch eine reine Beobachtung des Systems oder durch eine Durchführung von aktiven Experimenten bestimmt werden. Dabei ist der Übergang fließend zwischen Eigenschaften, die dem Objekt bzw. der Objektklasse direkt als Information zugeordnet werden können, und solchen, die sich auf aktorisches, planendes Wissen auswirken. Beispielsweise kann aus derartigen Experimenten eine geeignete Strategie für eine bestimmte Aktion an einer Objektklasse abgeleitet werden.

3.2.3 Planung

Die planenden Komponenten des Robotersystems sind zuständig für die richtige Auswahl von Aktionen basierend auf dem Weltzustand und der zu lösenden Aufgabe. Zunächst kann der Spezialfall betrachtet werden, daß eigentlich keine echte Planung durchgeführt wird, sondern ein vorgegebenes Roboterprogramm abläuft. Durch das *Lernen von Roboterprogrammen* kann versucht werden, eine automatische Programmierung durch eine Menge von beispielhaften Lösungen, z.B. eines Benutzers, vorzunehmen. Ziel ist ein verallgemeinertes Programm, das die verschiedenen Varianten enthält. Dieses Programm wird dann bei einer erneuten Aufgabe aufgerufen und abgearbeitet.

Ein autonomes System kann seine Aufgaben nicht gemäß eines starren Algorithmus lösen. Vielmehr ist eine Planung durchzuführen, welche Operationen angewendet werden müssen. Wird diese Planung nicht direkt unter Berücksichtigung der elementaren Operatoren ausgeführt, sondern werden größere Strukturen berücksichtigt, so kann ein *Lernen von Aktions- oder Skelettplänen* vorgenommen werden. Beim Einsatz muß einer dieser Pläne ausgewählt und die konkreten Parameter je nach Aufgabenstellung richtig instanziiert werden. Diese Lernaufgabe ist sehr ähnlich zu dem in Abschnitt 3.2.1 erwähnten Lernen von Sensoreinsatzplänen.

Die geeignete Auswahl eines vorhandenen Aktionsplans kann zu einem *Lernen von Zusammenhängen zwischen Aufgabeneigenschaften und Aktionsplänen* führen, d.h. welcher

Aktionsplan für eine Aufgabe mit bestimmten Merkmalen am erfolgversprechendsten ist.

Werden die Pläne nicht aus größeren - eventuell sogar nur einer - Teilkomponente zusammengesetzt, werden im allgemeinen Planungsoperatoren oder -regeln definiert. Der sich daraus ergebende Bereich des *Lernens von Aktionsregeln* entspricht von der Grundidee her dem Regellernen aus Kapitel 2.2.4. Neben dem Aufbau einer korrekten und möglichst vollständigen Menge von Aktionsregeln ist auch das Lernen der richtigen Anwendungsreihenfolge wichtig. Deduktive Lernverfahren können z.B. eingesetzt werden, um Anwendungsprioritäten oder Makrooperatoren aufzubauen.

In einem homogen aufgebauten System ist der Übergang von planenden Aktionsregeln zu reaktiven Situations-/Aktionszusammenhängen fließend. Da mehrere der bisherigen Systeme aber ausschließlich derartige reaktive Kopplungen untersuchen, wird auch das *Lernen von reaktivem Verhalten* als eigenes Lernziel angesehen, mit stärkerer Betonung der direkten Reaktion.

In Kapitel 2.1.7 wurde die Aufgabe der häufig vorgesehenen Fehlererkennungs- und Fehlerbehebungskomponente erläutert. Analog zum Lernen von Klassifikatoren und Aktionsregeln können hier das *Lernen von Fehlerklassifikatoren* sowie das *Lernen von Fehlerbehebungsregeln* untersucht werden. In diesem Bereich sind bisher allerdings nur wenige Arbeiten bekannt. Wie schon oben erläutert, ist eine echte Trennung in Fehlerbehebung und normale Aktionsplanung aber auch nicht notwendig.

3.2.4 Steuerung und Regelung

Die Planung muß sich auf eine Menge von Elementaroperationen abstützen, die von der realen Aktorik ausgeführt werden können. Diese Elementaroperationen stellen ähnliche reaktive Strukturen dar wie die oben erwähnten. In Arbeiten zum *Lernen von Elementaroperationen* wird das Lernen von Fuzzyregeln oder neuronalen Netzen zu diesem Zweck untersucht.

Auf der Ebene der Steuerung kann an mehreren Stellen das *Lernen von elementaren Abbildungen* untersucht werden. Während das direkte oder inverse kinematische oder dynamische Problem oder verschiedenste Regelungsprobleme früher algorithmisch gelöst wurden, gibt es inzwischen sehr viele Arbeiten, die diese Abbildung z.B. durch neuronale Netze lernen lassen. Vorteil ist häufig die wesentlich höhere Effizienz bei der Ausführung, z.B. bei der Berechnung der inversen Kinematik, allerdings auf Kosten höheren Speicherbedarfs. Problematisch ist, daß in vielen Arbeiten speziell für das Lernen von Reglern gefordert wird, daß der zu modellierende Regler schon vorhanden ist und dadurch Beispiele in beliebiger Anzahl erzeugt werden können.

3.2.5 Benutzervokabular

Neben den jeweils spezifischen Lernzielen ist ein prinzipielles Ziel immer vorhanden, wenn eine Kommunikation zwischen System und Benutzer stattfindet, nämlich das *Lernen von Benutzervokabular und dessen Semantik*. Ziel ist es dann, für Begriffe, die der Benutzer beim Training oder bei der Formulierung von Aufgaben verwendet, eine adäquate interne Darstellung zu bestimmen. Dadurch sollen zum einen die Informationen des Benutzers

operational gemacht werden, und zum anderen sollen Systemzustände oder -vorgänge mit den Worten des Benutzers beschrieben werden können. Ein Beispiel für eine derartige Lernaufgabe ist die Umsetzung von Begriffen für zu erreichende Teilziele in bestimmte geometrische Bedingungen, die dazu erfüllt werden müssen.

3.3 Robotik vs. ML-Anwendungen

Obwohl im letzten Abschnitt eine große Zahl von Lernzielen in der Robotik aufgezeigt wurde, ist die Anzahl der auf diesem Gebiet durchgeführten Arbeiten relativ gering. Dieser Abschnitt soll anhand einiger wichtiger Merkmale aufzeigen, welche Gründe dazu bisher geführt haben. Eine erste kurze Übersicht über die speziellen Probleme der Anwendung von Maschinellem Lernen in Bereichen der technischen Automatisierung wurde in [CWD+91] gegeben. In diesem Abschnitt wird zwar schwerpunktmäßig der Bereich der Robotik betrachtet, viele Aussagen treffen aber auf die Unterschiede zwischen ML- und technischen Anwendungen allgemein zu.

Wissensrepräsentation

Im Bereich Maschinelles Lernen werden die vorhandenen Algorithmen im wesentlichen auf Beispielbeschreibungen angewandt, die sehr einfacher Natur sind (s. Kapitel 2.2.3). Die am häufigsten verwendete Wissensrepräsentation sind Attribut-Wert-Paare, d.h. die Beschreibung eines Beispiels (Objekt, Vorgang) durch eine feste Menge unterscheidbarer Merkmale und deren konkreten Wertausprägungen, oder einfach strukturierte Objekte. Weitere Wissensrepräsentationsformen, die insbesondere für die resultierenden Begriffsbeschreibungen verwendet werden, sind Regeln oder einfache aussagen- oder prädikatenlogische Darstellungen. Die Domänen der Attribute sind häufig diskret, d.h., es ist lediglich eine feste Menge symbolischer Werte zugelassen. In wenigen Arbeiten werden Pläne bzw. Planungsregeln behandelt, die dann aber der typischen STRIPS-Notation entsprechen (Operatoren mit einfachen Vor- und Nachbedingungen). In der Robotik sind komplexe Geometrie-, Kinematik- und Dynamikmodelle im Einsatz (s. Kapitel 2.1.3). Es werden strukturierte Repräsentationen verwendet, die aber wesentlich komplexer als die im ML verwendeten sind. Die Wertebereiche der Attribute sind häufig kontinuierlich numerisch. Eine allgemeingültige Diskretisierung von Wertebereichen ist nicht möglich. Das Ablaufwissen eines Robotersystems ist in den meisten Fällen prozedural codiert und steht damit einer Behandlung durch Verallgemeinerung von Regeln, Operatorbedingungen usw. zunächst nicht zur Verfügung. Die teilweise eher deterministischen Planungen führen dazu, daß oft einfache Planschablonen eingesetzt werden und keine Planung im eigentlichen Sinne durchgeführt wird. Aus diesem Grund können viele Probleme aber auch prinzipiell nicht gelöst werden. Echte autonome Systeme müssen wie in Kapitel 2.1.6 erläutert über ein mächtiges Planungssystem mit reaktiven und strategischen Eigenschaften verfügen.

Datenquelle

Im Maschinellen Lernen stammen die Trainings- und Testdaten in den meisten Fällen vom Benutzer, in der Robotik kann ebenfalls der Benutzer als Lehrer eingesetzt werden, daneben sind aber Daten zu verarbeiten, die aus Sensormessungen stammen. Eines der Hauptprobleme ist dann, die für den Lernalgorithmus relevanten Daten herauszufiltern. In ML-Anwendungen werden dem System nur die relevanten Daten vorgegeben, in realen

Anwendungen muß aber auch bestimmt werden, von welchen Merkmalen der Beispiele ein Parameter überhaupt abhängen kann. Weiterhin ist zu berücksichtigen, daß Sensormessungen zur Bestimmung von Attributwerten Kosten in Form von Verarbeitungszeit verursachen. Diese Problematik wurde im ML bisher selten untersucht [Tan90].

Korrektheit der Daten

Sensoren lieferen im allgemeinen keine sicheren und exakten Daten. Aus diesem Grund müssen die angewandten ML-Techniken mit gestörten und verrauschten Daten und damit inkonsistenten Beispielmengen umgehen können. Diese Problematik kann von einigen Weiterentwicklungen der Basisverfahren behandelt werden (z.B. χ^2-Test in ID3 [Qui86]), für viele Verfahren ist aber noch unklar, wie eine entsprechende Erweiterung aussehen kann. Eine wesentlich höhere Stabilität gegenüber verrauschten Daten weisen subsymbolische Verfahren auf, so daß untersucht werden muß, ob derartige Verfahren als vorverarbeitende Teileinheiten eingesetzt werden können.

Planen und Ausführen

Die enge Kopplung von Planen und Ausführen wurde im ML bisher fast nie berücksichtigt, sondern beide Komponenten werden als sequentiell nacheinander auszuführende Prozesse betrachtet, wobei das Augenmerk der ML-Arbeiten ausschließlich auf der Planung liegt. Soll das System aber nicht nur effizienter planen können (deduktive Ansätze), sondern tatsächlich auch das zugrundeliegende Planungswissen verbessern können, so muß die Auswirkung des eventuell fehlerhaften Planungswissens bei der Ausführung berücksichtigt werden. Dabei ist auch zu untersuchen, wie ein solches System überhaupt lernen kann, d.h., wie und wann durch Benutzerinteraktion oder durch eigene Analyse neues Planungswissen aufgebaut werden kann. Die Problematik der Rückwirkungen einer Aktion auf die Planung ist insbesondere bei reaktiven Verhaltensweisen wichtig. Auch hier sind bisher erst wenige Untersuchungen durchgeführt worden.

Determiniertheit von Aktionen

Aus den eben erläuterten Gründen wird im allgemeinen davon ausgegangen, daß die aus der Planung resultierenden Aktionen deterministisch sind, d.h., daß über die Nachbedingungen eines Operators exakt festgelegt werden kann, welche Auswirkungen die Aktion auf die Umwelt hat. Diese Annahme ist aufgrund von Unsicherheiten und unerwarteten Ereignissen oder Situationen in der Robotik nicht haltbar. Dies gilt insbesondere für autonome Systeme, in deren Arbeitsbereich sich auch andere aktive Agenten befinden können.

Zeitanforderungen

Im ML ist das Ziel im wesentlichen, den Lernerfolg überhaupt zu erhalten, während eine hohe Effizienz der Verfahren nur selten im Mittelpunkt steht. Ein hoher Aufwand für den eigentlichen Lernvorgang scheint dem Lernen inhärent zu sein. Allerdings müssen in der Robotik zumindest einige der Lernergebnisse in Echtzeit angewendet werden können.

Beispieldomänen

Ein Merkmal vieler ML-Arbeiten ist, daß eine sehr eng abgegrenzte Problemstellung untersucht wird, die meist leicht zu formalisieren ist, wie Spiele oder einfache Klassifikationsaufgaben. Häufig wird nur der eigentliche Lernvorgang analysiert, eine Anwendung und die daraus entstehende Notwendigkeit der sukzessiven Korrektur des Wissens wird nicht

berücksichtigt. In der Robotik ist die Domäne fest vorgegeben, und Lernmethoden müssen sich darauf ausrichten.

Die aufgezeigten Differenzen zwischen ML-Anwendungen und der Robotik, speziell autonomen Systemen, sollen in dieser Arbeit ein deutliches Stück beseitigt werden. Dazu müssen sowohl neue Lerntechniken entwickelt als auch geeignete Repräsentationsformen für autonome Systeme gefunden werden. Bevor im nächsten Kapitel die genauen Anforderungen zusammengefaßt werden, soll in den nächsten Abschnitten der Stand der Technik bei der Anwendung von Maschinellem Lernen in der Robotik aufgezeigt werden.

3.4 Klassifikation vorhandener Ansätze

In diesem Abschnitt wird erstmals der Versuch unternommen, bisherige Ansätze zum Einsatz von Lernverfahren in der Robotik vergleichend zu betrachten und in einem einheitlichen Klassifikationsschema anzuordnen. Eine ausführlichere Darstellung sowie eine detaillierte Beschreibung jedes Ansatzes und eine tabellarische Übersicht gibt [KC92, Cor92].

Grundlage für die Klassifikation (s. Abbildung 3.1) sind zum einen die in Kapitel 3.2 erarbeiteten Lernziele eines Robotersystems, zum zweiten die verwendete Lernmethode und zum dritten die Anwendungsdomäne. Das Lernziel stellt hier das Hauptunterscheidungskriterium dar. Eine Abgrenzung ist in vielen Fällen möglich, da die meisten Ansätze nur ein oder zwei Teilaspekte berücksichtigen, ohne das Zusammenspiel mit anderen Teilkomponenten zu untersuchen. Bisher gibt es nur vier Arbeiten, die auch eine Robotersystemarchitektur als Ganzes betrachten. Auf sie wird im nachfolgenden Abschnitt eingegangen. Zusammenfassend wurde die folgende Teilmenge von Lernzielen betrachtet:

- Lernen von Roboterprogrammen
- Lernen von Aktionsplänen (Aktorik, Sensoreinsatz)
- Lernen von Aktionsregeln
- Lernen von reaktivem Verhalten
- Lernen von elementaren Abbildungen
- Lernen von Fehlererkennungs-, Fehlerdiagnose- und Fehlerbehebungsregeln
- Lernen von Klassifikatoren zur Perzeption

Die übrigen Lernziele wurden in diesem Themenbereich noch nicht bearbeitet bzw. es gibt zwar Ansätze, z.B. Objekteigenschaften zu extrahieren (siehe auch Kapitel 8), aber dabei wird nichts gelernt.

Als eingesetzte Lerntechniken wurden die in Kapitel 2.2 eingeführten Methoden betrachtet. Dabei wurde im wesentlichen in induktive und deduktive Lernverfahren unterschieden. Eine weitere Untergliederung wurde vorgenommen in solche Lernverfahren, die direkt Eingaben eines Benutzers auswerten, und solche, die die notwendigen Beispiele automatisch ableiten. Allerdings war eine klare Zuordnung nicht immer möglich.

	Induktives Lernen		Deduktives Lernen	
	Benutzer	Automatisch	Benutzer	Automatisch
Roboter-programme	Andreae 85 Sato und Hirai 87 Heise 89 Münch 94	Dufay und Latombe 84	Selfridge und Levas 83 Levas 84	
Aktionspläne	Tan 90		Segre 88	
Aktionsregeln	Chen 86 Kadie 88 Simon 90 Kreuziger 92+93	Carbonell et al 87 Langley et al 89 Sobek und Laumond 89 Christiansen 90 Barbehenn und Hutchinson 91 Kreuziger 92+93	Laird et al 90	Bennett 89+90 Laird et al 90
Reaktives Verhalten	Bartenstein und Inoue 87	Vaaler et al 91 Moore 90 Lin 91		Mitchell 89+90
Perzeption		Winston 83+84 Pati88 Langley et al 89 Tan 90 Morik 93 Kreuziger 94		
Fehlerbehebung			Zheng und Daneshmend 91	

Abbildung 3.1: Anwendungen von ML in der Robotik

Die allerwenigsten Ansätze verwenden Standardlernverfahren aus dem Maschinellen Lernen, sondern es wurden zumeist eigene, passende Ansätze entwickelt. Da diese Verfahren in den meisten Fällen nur sehr grob beschrieben wurden, war eine Klassifizierung und Gruppierung der einzelnen Arbeiten äußerst schwierig. Als weiteres Problem erwies sich, daß nur in sehr wenigen Fällen eine genaue Beschreibung der Anwendung oder des konkreten Experiments vorlag, so daß eine fundierte Bewertung sehr erschwert wurde. Fast alle Arbeiten greifen ein sehr spezielles Teilproblem der Robotik heraus und versuchen, dafür eine Methode zu entwickeln. Es wurden nur solche Arbeiten in die Übersicht aufgenommen, die tatsächlich Fragestellungen der Robotik betrachten und nicht nur die prinzipielle Übertragbarkeit ihrer Arbeiten auf derartige Probleme postulieren.

Bezüglich der Anwendungsdomäne können die aufgeführten Arbeiten in folgende Kategorien eingeteilt werden:

- Manipulator: [SL83, DL84, Che86, BI87, Seg88b, Hei89, LTI+89, Moo90, CTM90, LHYT90, LR90, SM90, Sim90, Ben90, Tan90, TS90a, VS91, Kre92, KW94, MKKD94]
- Mobiles System: [BI87, LR90, Mit90, BH91, MR93]

- Blockweltaufgaben (Anspruch ist jedoch der Einsatz in autonomen Systemen): [SH86, RS87, FI87, CG87, Kad88, Tal88, SL89, ZD91]

Die aufgeführte Klassifikation ließe sich für das Lernen von elementaren Abbildungen auf unteren Steuerungsebenen noch weiter aufgliedern. Dort werden vornehmlich subsymbolische Lernverfahren [KH89, Ber91] eingesetzt und zwar für die folgenden Teilaufgaben:

- Inverse Kinematik: Lernen der Abbildung von Weltkoordinaten in Robotergelenkwinkelstellungen [AG90, AAMR87, JL90]
- Inverse Dynamik: Lernen der Abbildung von TCP-Geschwindigkeiten auf die Kräfte und Momente in den Robotergelenken [Sha90]
- Trajektorienverfolgung: Lernen des Nachfahrens einer bestimmten Trajektorie durch den Manipulator [AAMR87, AR89, AR90, Kur90, YY90]
- Sensor-Motor-Steuerung (engl. sensorimotor integration): Lernen einer Abbildung von bestimmten Sensormustern auf passende Aktorikkommandos [Mil87, Kup88, Mel88, PG88, SK91]

Im folgenden wird eine eine kurze Übersicht über die aufgeführten Arbeiten vor allem mit symbolischen Lernverfahren gegeben. Darin sollen die Lernziele und die entwickelten Methoden aufgezeigt und eine kurze Bewertung vorgenommen werden.

3.4.1 Lernen von Roboterprogrammen

Einen der ersten Ansätze zur Anwendung von Lernverfahren in der Roboterprogrammierung stellt die Arbeit von Selfridge und Levas [SL83, LS84] dar. In diesem rein deduktiven Ansatz wurde eine Vorführung durch den Benutzer, die textuell eingegeben wurde, durch vorhandene Regeln sukzessive erklärt. Dadurch ergab sich eine hierarchische Erklärungsstruktur für eine Aufgabe, die auf unterster Ebene aus einer Folge von eingenommenen Relationen bestand, die vom Benutzer eingegeben wurden. Diese Erklärungsstruktur wurde anschließend als Programm für eine erneute Aufgabenlösung eingesetzt. Schwierigkeiten dieses Ansatzes sind die bereits in Abschnitt 2.2.5 angesprochenen Probleme: Es ist relativ unwahrscheinlich, daß aus einem einzigen Beispiel bereits das allgemeingültige Vorgehen für eine bestimmte Aufgabe abgeleitet werden kann. Dies ist auch dadurch begründet, daß die Lösung einer Aufgabe nicht ausschließlich auf der Menge der „Normpfade" verläuft.

Von Dufay und Latombe [DL84] wurde das induktive Lernen von 2D-Fügevorgängen mit geometrischen Unsicherheiten untersucht. Gegeben war ein Planungssystem, das die Aufgabe prinzipiell lösen konnte. Außerdem wurde eine Ausführungsüberwachung eingesetzt, die alle aktuell erreichten Relationen bestimmen konnte. In einer Trainingsphase wurden mehrere Ausführungstraces für eine bestimmte Aufgabe – speziell das Fügen eines Stiftes – erstellt. Diese wurden in der nachfolgenden Induktionsphase zu einem einzigen Graphen zusammengefaßt. Aus diesem wurde dann automatisch ein Roboterprogramm erstellt. Das gewünschte Ergebnis ist ähnlich zu den „universal plans" von Schoppers (s.

Kapitel 2.1.6) und hat damit die gleichen Probleme. Insgesamt lernt das System kein wirklich neues Wissen für eine Problemlösung, sondern, wie die einzelnen Aktionen basierend auf Sensormessungen in Form von eingenommenen Relationen nacheinander ausgeführt werden müssen. Problematisch ist die Forderung nach völliger Übereinstimmung zweier Zustände und zweier Bewegungen bei der Graphenduktion. Hier wäre eine Erweiterung unter Berücksichtigung induktiver Operatoren notwendig.[1]

In NODDY [And85] wird als Beispiel das Lernen von Roboterprogrammen aus Benutzervorführungen für einfache Bewegungen eines punktförmigen mobilen Roboters untersucht. Andreae geht es sehr viel stärker um echte Generalisierungsoperatoren als Dufay und Latombe und vor allem um eine Rechtfertigung für einen Generalisierungsschritt. Aber auch in diesem Ansatz besteht das Lernproblem im wesentlichen in einer Zusammenfassung verschiedener linearer Traces zu einer gemeinsamen Graphstruktur mit verallgemeinerten Bedingungen und Aktionen. Typische Probleme der Robotik wie die Behandlung von Unsicherheiten oder die Notwendigkeit von Reaktionen werden ausgeklammert.

Eine überwachte induktive Verallgemeinerung einer Menge von vorgeführten Bewegungen wurde von Heise im System ETAR vorgeschlagen [Hei89]. Während einer Bewegung des Roboters durch den Benutzer werden die Gelenkwinkelstellungen aufgezeichnet. Durch einen Filter, der „Focus of attention" genannt wird, werden nur solche Gelenkwinkelstellungen gespeichert, bei denen der TCP in der Nähe eines Objektes liegt. Aus diesen wird dann eine Folge von Elementaroperationen in Form einfacher Bewegungsbefehle abgeleitet. In dieser Folge wird nach Schleifen und Verzweigungen gesucht und eine entsprechende Formulierung der Abbruch- bzw. Verzweigungsbedingungen generiert. Die Parameter einer Aktion werden durch funktionale Induktion verallgemeinert. Anschließend werden die Verallgemeinerungsmöglichkeiten mehrerer Lösungen für dieselbe Aufgabe untersucht. Das Induktionsverfahren basiert auf einer Generate-and-Test-Strategie, die sämtliche möglichen funktionalen Zusammenhänge unter Verwendung vorgegebener Basisfunktionen zwischen den Parametern generiert und untersucht, ob sie zutreffen. Dies führt dazu, daß die Beispielbewegungen des Benutzers äußerst exakt und immer wieder auf die gleiche Art durchgeführt werden müssen, damit darin ein Zusammenhang erkannt werden kann. Das Wissen über die Objekte (Position, Orientierung und Parameter wie Höhe, Breite und Länge) wird explizit vorgegeben. Der Ansatz wurde mit einem realen Roboter in einer 3D-Blockwelt getestet. Als Aufgaben wurden das Stapeln und Plazieren von Blöcken betrachtet.

Eine Architektur für ein System zur Programmierung von Robotern durch Vorführungen wird in [MKKD94] vorgeschlagen. Die Arbeit enthält außerdem eine Analyse der Möglichkeiten des Einsatzes von Lernverfahren und der Notwendigkeit von Benutzerinteraktionen. Erste experimentelle Ergebnisse beschreiben die Gewinnung von Makrooperatoren aus der Sequenz eines vorgeführten Beispiels und die entsprechende Repräsentation der Beispiellösung.

Das Problem bei allen Ansätzen, die vollständige Roboterprogramme lernen, ist die bereits in Abschnitt 3.2 angesprochene Starrheit während des Ablaufs und die Annahme, daß

[1] Eine wesentlich erweiterte Untersuchung des Suchproblems bei der Zuordnung von Knoten und Kanten in mehreren Graphen wurde in [Meu88] durchgeführt. Die Fragestellung war dabei der Aufbau von allgemeinen Handlungsplänen aus einer Menge von Beispielprotokollen, die aufgezeichnet wurden, während ein Benutzer mit der wissensbasierten Mensch-Maschine-Schnittstelle AiD [HKT87] arbeitete.

nur wenige Ausnahmen und Sonderbedingungen auftreten dürfen, damit überhaupt ein vollständiges Programm dafür gelernt werden kann. Für eine neue Aufgabe läuft nur noch das gelernte Programm ab, eine eigentliche Planung findet nicht mehr statt.

3.4.2 Lernen von Aktionsplänen

Die Anwendung eines erklärungsbasierten Lernverfahrens zum Lernen von Aktionsplänen beschreibt Segre mit dem System ARMS [Seg88b]. Ziel des Verfahrens ist es, für das Erreichen einer bestimmten physikalischen Verbindung zwischen Objekten eine geeignete Folge von zu erreichenden Relationen zu bestimmen. Neben der vollständigen Modellierung der Objekte sind als Hintergrundwissen auch die Menge der möglichen Verbindungen und eine einfache kinematische Theorie vorgegeben. Mit diesem Wissen kann eine vom Benutzer textuell vorgegebene Aufgabenlösung erklärt werden; dazu wird ein kausales Modell aufgebaut. Eine entsprechend generalisierte Operatorfolge wird dann als Lösung zur Realisierung der gegebenen physikalischen Verbindung abgespeichert. ARMS wurde in einer erweiterten Blockwelt eingesetzt. Allerdings wurde kein reales System angesteuert, sondern eine Simulation als Ausführungskomponente verwendet. Diese Simulation basierte außerdem auf stark vereinfachenden Annahmen. Es wurde von einer idealen Sensorik ausgegangen, d.h. der Weltzustand war vollständig und fehlerfrei bekannt. Daneben wurden eine Reihe von Annahmen über die Manipulationsmöglichkeiten getroffen, die nicht haltbar sind. Beispielsweise wurde davon ausgegangen, daß die notwendige Operatorfolge nur vom Zielzustand, nicht jedoch vom Ausgangszustand abhängt.

In [Tan90, TS90a, TS90b] wird das Lernsystem CSL beschrieben, das beim Aufbau von Entscheidungsbäumen nach Art von ID3 (s. Abschnitt 2.2.4) auch Kosten und verrauschte Daten berücksichtigen kann. Es wurde mit einem realen Roboter eingesetzt, um mit Ultraschallsensoren Messungen an Objekten durchzuführen und sich danach für eine geeignete Greifprozedur zu entscheiden. Dabei wird berücksichtigt, daß Meß-, Bewegungs- und Greifoperationen bestimmte Kosten verursachen. In einer Trainingsphase wird jedem Objekt eine von drei Greifstrategien zugeordnet, und das System soll geeignete Sensormessungen (4 Varianten) finden, um eine Objektbeschreibung abzuleiten, die zu der gewünschten Greifstrategie führt. In der Ausführungsphase wird der Weltzustand als Eingabe betrachtet und ein Entscheidungsbaum aufgebaut, der zu der kostengünstigsten Strategie führt, das Objekt zu greifen. Zu den Greifstrategien und Sensoroperationen sind die notwendigen Vorbedingungen, die Kosten, die Merkmale und die erwarteten Fehler bekannt. Bei der Objekterkennung wurde von einigen stark vereinfachenden Annahmen ausgegangen, insbesondere davon, daß die Klassifikation bereits auf der Basis einiger weniger Merkmale möglich ist. Ein Problem des Ansatzes ist die strikte Trennung in Trainings- und Ausführungsphase. Dies führt dazu, daß neue Erfahrungen, die bei der Ausführung einer Aufgabe gemacht werden, nicht in das bereits aufgebaute Wissen einfließen können. Dazu zählt auch, daß keine Auswertung von gescheiterten Ausführungen vorgenommen werden kann.

Der Ansatz von Dufay und Latombe [DL84] ist bis zu einem Zwischenschritt auch ein Lernen von Aktionsplänen. Danach werden diese jedoch, wie oben beschrieben, in ein festes Programm transformiert und nur dieses zur Ausführung verwendet.

In [FI87] wird der Einsatz von analogem Schließen und Lernen auf Fragestellungen der Robotik konzeptuell untersucht. Aktionspläne für ähnliche Situationsbeschreibungen sollen zusammengefaßt und bei Vorliegen einer Situationsklasse entsprechend angewendet werden. Das vorgeschlagene Lernverfahren ist ähnlich zur Transformational Analogy, die von Carbonell zur Problemlösung durch Analogien, siehe Kapitel 2.2.6, diskutiert wurde.

3.4.3 Lernen von Aktionsregeln

In der integrierten Systemarchitektur ICARUS [LTI+89] ist für die Planerstellung die Teilkomponente DÆDALUS zuständig. Wie im gesamten System wird als einheitliches unüberwachtes Lernverfahren CLASSIT eingesetzt (s. Kapitel 2.2.4). Im Bereich Planen geht es dabei vor allem um die Zuordnung eines auszuwählenden Operators zu einer Menge von Differenzen zwischen aktuellem Zustand und gewünschtem Zielzustand. Die möglichen Operatoren sind mit Vor- und Nachbedingungen in STRIPS-Form gegeben. Die Differenzbeschreibungen aus konkreten Planungsaufgaben werden in einer Hierarchie angeordnet und für jeden Operator wird die Wahrscheinlichkeit dafür bestimmt, daß er ausgewählt wird. Die einschränkenden Bedingungen, wie Beschreibung durch Vor- und Nachbedingungen, Vorliegen eines einzelnen Agenten, die Closed world assumption (d.h. die Welt ändert sich nur so, wie es der Operator angibt), und die Trennung in Planung und Planausführung, sind allerdings so restriktiv, daß eine Übertragung auf einen Planer für ein reales autonomes System nur schwer möglich erscheint.

Christiansen untersucht in [MCM89, CTM90, Chr92a, Chr92b] den Einsatz von Lernverfahren - speziell Auswendiglernen - in stochastischen Planern vor allem am Beispiel des Tray-Tilting. Aufgabe ist es, ein Objekt auf einer in jede Richtung kippbaren Platte von einem Ausgangspunkt zu einem bestimmten Zielpunkt zu bringen. Das System hat dazu initial keine Informationen über physikalische Gesetze. Ziel des Lernens ist es, durch Beobachtung von Experimentresultaten Wissen darüber aufzubauen, welche Kippung am wahrscheinlichsten zu welcher Bewegung führt. Die Experimente geben jeweils an, in welchem Anfangs- und Endzustand sich das Objekt vor bzw. nach einer bestimmten Kippung befand. Entsprechend werden für dieses Tripel die Wahrscheinlichkeiten erhöht. Bei der Ausführung wird dann im Graph aller möglichen Kippfolgen der Pfad gesucht, der die höchste Wahrscheinlichkeit auf Erfolg verspricht. Die ausführliche experimentelle Erprobung des Ansatzes hat gezeigt, daß die Auswahl der Trainingsbeispiele eine wichtige Rolle spielt. Die besten Ergebnisse lieferte eine Kombination von zufälliger Auswahl von Aktionen und nachfolgender Untersuchung ihrer Zuverlässigkeit. Die Arbeit macht zwei wichtige Einschränkungen: Erstens, die Anzahl der möglichen Zustände ist sehr beschränkt – das Objekt konnte nur in einem von 9 bzw. 16 Feldern der Platte liegen – und zweitens, die Anzahl möglicher Aktionen ist diskret – Kippungen konnten in jeweils 1 Grad Schritten im Vollkreis erfolgen. Nur durch diese Einschränkungen war die gewählte Repräsentation und das Vorgehen bei Lernen und Planung möglich. Das prinzipielle Vorgehen zur Durchführung von Experimenten ist in einem solchen Suchraum relativ einfach.

Einen Ansatz zum Aufbau eines einfachen Aktionsmodells aus Beispielen, die ebenfalls durch einfache Experimente eines Roboters erreicht werden, beschreibt [ZM93]. Der Einsatz von Lernen durch Experimentieren zur Korrektur und Erweiterung von Planungsoperatoren in STRIPS-Notation wird in [CG87, CG90] im Rahmen der umfassenden Planungs-

systemarchitektur Prodigy untersucht (s. Kapitel 2.2.6). Eine Korrektur von falschen oder unvollständigen Vorbedingungen für STRIPS-Operatoren wird auch in [SL89] behandelt.

In [Tal88] wird nicht ein Fehler in der Vorbedingung einer Produktion, sondern in der Berechnungsvorschrift für diese Vorbedingung betrachtet. Entspricht eine Vorbedingung einer bestimmten Relation in der Welt, so kann das zur Bestimmung der Gültigkeit definierte Verfahren fehlerhaft sein. Der Lernvorgang besteht im wesentlichen aus einer Verschiebung (Vergrößerung oder Verkleinerung) von Schwellwerten, die in der Beschreibung vorkommen.

Die Anwendung von AQ-Verfahren auf das Lernen von einfachen Regeln zur Bahnplanung wurde in [Che86] beschrieben und für einen simulierten Roboter mit zwei Gelenken getestet. Ein überwachtes, induktives Lernverfahren wird in DIFFY-S [Kad88] eingesetzt. Ziel des Systems ist es, sogenannte Roboter-Operator-Schemata zu lernen, die angeben, wie der Weltzustand durch Anwendung des Operators verändert wird. Aus mehreren Beschreibungen des Weltzustandes vor und nach der Anwendung des Operators soll der funktionale Zusammenhang abgeleitet werden. Dazu ist eine Menge von primitiven Funktionen vorgegeben, die zur Erklärung des Übergangs dienen können. Bisher wurde DIFFY-S lediglich auf einfache 2D-Blockwelt-Probleme und auf das Erlernen erlaubter Schachzüge angewendet. In diesen Beispielen sind die starken Voraussetzungen erfüllt, und die entwickelte Methode scheint anwendbar. Ob dieses Verfahren jedoch auch auf komplexere Aufgabenstellungen anwendbar ist, muß erst noch gezeigt werden.

Im System GRASPER [Ben89, Ben90] werden erklärungsbasierte Methoden eingesetzt, um Planungsregeln und Objektbeschreibungen zu korrigieren. Dazu besitzt das System umfangreiches Hintergrundwissen in Form von Regeln zum Erreichen einer gewissen Bedingung, in Form von qualitativen Regeln, die eine Aussage über den Einfluß von Größen aufeinander, auf die quantitative Größe eines Prädikates und auf die Wahrscheinlichkeit des Erfolgs eines Prädikates angeben. Das System verwendet Daten- und Regelapproximationen zur Behandlung von Unterschieden zwischen dem internen Modell und der externen realen Welt. Bennett unterscheidet in Kontroll-, Beschränkungs- und Gewichtungsregeln. Die Kontrollregeln führen direkt zu einem Wert für eine Größe der realen Welt, z.B. für die Größe, die angibt, wie weit der Greifer für bestimmte Greifflächen geöffnet werden soll. Eine Beschränkungsregel beschränkt die Auswahl aus einer Menge von möglichen Kandidaten, z.B. dürfen nur solche Greifflächen ausgewählt werden, bei denen eine zulässige Greiferöffnung vorliegt. Die Gewichtungsregeln beeinflussen schießlich das Zusammenspiel verschiedener Beschränkungsregeln. Für den Ablauf einer konkreten Aufgabe wird auf der Basis dieses Hintergrundwissens eine Aktion bestimmt, diese in der realen Welt ausgeführt und mit vorgegebenen Monitorregeln überwacht. Sobald bei der Ausführung eine Abweichung festgestellt wird, werden auf der Basis qualitativer Regeln über die Zusammenhänge zwischen Parametern und Prädikatenwahrscheinlichkeiten Kandidaten dafür bestimmt, wie der aufgetretene Fehler bei zukünftigen Aufgaben vermieden werden kann. Auf dieser Basis werden dann die Daten- und Regelapproximationen entsprechend aktualisiert. In der Anwendung sollte ein Scara-Roboter auf der Basis eines aufgenommenen 2D-Bildes ein Objekt greifen. Dabei konnten als Fehler das Berühren des Objektes mit einer Greiferbacke und das horizontale Herausrutschen des Objektes aus dem Greifer analysiert werden. Die potentielle Lösung für das Problem war dem System bereits bekannt, d.h. Lernziel war ein Tuning der Approximation, so daß das Objekt sinnvoll

gegriffen wurde. Fehler, die nicht im Domänenwissen repräsentiert waren, konnten wegen des rein deduktiven Ansatzes nicht korrigiert werden.

Ein Vergleich dieses von Bennett *permissive planning* genannten Ansatzes mit dem oben beschriebenen stochastischen Planen von Christiansen wurde in [BD91] durchgeführt. Als Anwendungsgebiet wurde das Tray-Tilting untersucht. Dabei stellte sich heraus, daß der Ansatz von Bennett für Aufgaben mit einer einzigen Aktion wesentlich bessere Resultate lieferte, bei Aufgaben mit drei Aktionen nacheinander aber ähnliche Erfolgsraten bei beiden Ansätzen erzielt wurden.

3.4.4 Lernen von reaktivem Verhalten

In [BI87] wird ein Verfahren vorgestellt, wie vom Benutzer vorgegebene Roboterprogramme, die die tatsächliche Lösung einer Aufgabe nur approximieren, automatisch verbessert werden können. Die Programme müssen dazu in Form von Situations-Aktions-Regeln beschrieben werden können, wobei die linke Seite einer konjunktiven Verknüpfung von Sensorbedingungen entspricht und die rechte Seite die zugehörige Aktion des Roboters angibt. In einem solchen Programm können nur zwei Fehler auftreten: 1. Das Programm führt eine Aktion aus, die nicht ausgeführt werden sollte, oder 2. Eine Aktion, die hätte ausgeführt werden sollen, wird nicht ausgeführt. Dementsprechend muß die linke Seite der entsprechenden Regel im ersten Fall spezialisiert und im zweiten Fall generalisiert werden. Als Beispielanwendungen wurden die Erkennung einer festen Menge von Objekten aus 2D-Bildern untersucht, die Optimierung der Bahn eines simulierten planaren Manipulators mit zwei Freiheitsgraden und die Navigation von Landmarken in simulierter statischer Umgebung mit exakter Hinderniserkennung. Offen bleibt, wie mit diesem rein reaktiven Verhalten - es werden immer direkt Roboteraktionen abgeleitet - ein taktisches Vorgehen erreicht werden kann. Eine starke Forderung des Ansatzes ist es, daß ein „Orakel" existiert, das sofort eine Aktion des Roboters als falsch klassifiziert. In der Regel wird diese Information erst nach einer Folge von Regelanwendungen vorliegen. Dann stellt sich aber sofort das Problem, wie die tatsächlich fehlerhafte Regel bestimmt werden kann. In der Arbeit wird keine Angabe gemacht, wie eine Regel zur Generalisierung ausgewählt werden kann, wenn mehrere Regeln zu der gewünschten Aktion führen.

Eine Anwendung des Auswendiglernens bei aktiven Experimenten eines Roboters wurde in [VS91] beschrieben. Für einen Fügevorgang wird eine Abbildung eines gemessenen Zustandes auf die beste Korrekturbewegung gesucht, die ein möglichst schnelles Einfügen erlaubt. Untersucht werden x- und z-Position des TCP, sowie die Drehung O um die y-Achse. Als Sensorwerte werden die Kräfte in x- und z-Richtung sowie das Moment um die O-Achse, sowie die Änderung der Kraft in x bzw. des Moments um O bei einer inkrementellen Änderung in z und die z-Position aufgezeichnet. Die Wertebereiche der sechs Größen werden in jeweils sechs Intervalle unterteilt. In den Experimenten zeigte sich, daß nur 300 der 6^6 möglichen Zustände tatsächlich für die Aufgabenlösung relevant waren. Für den Lernvorgang wird der Fügevorgang in z-Richtung in kleinen Schritten durchgeführt und jeweils die Kräfte gemessen. Wird eine Kraftgrenze überschritten, wird zufällig eine Bewegung in z- oder O-Richtung durchgeführt; sind bereits Experimentergebnisse vorhanden, geschieht diese Bewegung auf deren Basis. Nach Erreichen des Ziels werden die erreichten Zwischenzustände rückwärts untersucht und die Zustands-Aktions-Zuordnungen entsprechend modifiziert.

Im THEO-System [BM89, Mit89, Mit90] werden Ansätze untersucht, wie durch deduktive Lernverfahren reaktive Regeln aus Planungsregeln abgeleitet werden können. Eine genauere Beschreibung der Gesamtarchitektur und der realisierten Lernvorgänge folgt in Abschnitt 3.5.3.

Den Einsatz von Reinforcement Learning und Teachverfahren zur Programmierung von Robotern beschreibt [Lin91]. Das Roboterprogramm besteht aus einer Zuordnung von Aktionen zu einer Situation. Aufgrund von „Belohnungen" oder „Bestrafungen" am Ende einer Aktionskette wird durch ein Rückrechnungsverfahren jede einzelne ausgewählte Aktion bewertet und damit entsprechend das Verhalten des Systems bei zukünftigen Aufgaben angepaßt. Ziel des Lernvorgangs ist es, die Steuerungstaktik so zu optimieren, daß das größtmögliche Reinforcement erreicht wird. Lin kommt bei seinen Versuchen zu dem Resultat, daß ein rein unüberwachtes Lernen selbst für einfache Aufgaben nicht ausreichend effizient ist. Als Anwendungsbeispiel wurde die Steuerung eines mobilen Fahrzeugs untersucht, das im wesentlichen drei elementare Verhalten erlernen sollte: 1. Bewegung entlang einer Wand, 2. Durchfahrt durch eine Tür, 3. Andocken an eine Batteriestation. Die Resultate einer Simulation konnten dabei als Basis für die Steuerung des realen Fahrzeugs Hero benutzt werden. Dabei wird normalerweise eine Markov-Umgebung (engl. markovian environment) angenommen, d.h. alle Informationen zur Entscheidung für eine optimale Aktion sind im aktuellen Zustand enthalten [LM92]. Lin wählt drei bekannte Netztypen, um die Q-Funktion durch ein neuronales Netz zu approximieren, stellt jedoch die Einsetzbarkeit bzw. die Lernbarkeit der Q-Funktion für alle drei Ansätze in Frage und entwickelt daher seinen OAON-Netztyp (one action one network). Bei diesen wird für jede der möglichen diskreten Aktionen ein eigenes Netz eingelernt. Leider wurde auch dieser Ansatz nur für die relativ einfache Steuerungsaufgabe, einen Stab zu balancieren, untersucht. Die tatsächlich möglichen Aktionen wurden nicht aufgeführt.

Die wichtigsten Probleme derartiger Ansätze sind, daß erstens unterstellt wird, daß das System während der Lernphase eine beliebige Aktion in einem Zustand auswählen kann, z.B. gemäß einer Boltzmann-Verteilung [LM92]. Dadurch sollen alle möglichen Aktionsvarianten getestet werden. Dieses Vorgehen ist bei der Manipulation von Objekten, womöglich unter Verwendung eines realen Roboters, nicht praktikabel bzw. nicht zulässig. Zweitens wird davon ausgegangen, daß es eine endliche Menge von möglichen Aktionen gibt. Parameter werden in den Aktionstyp fest eincodiert.

3.4.5 Lernen von elementaren Abbildungen

Die Teilkomponente MÆANDER des ICARUS-Systems (s. Abschnitt 3.5.1) untersucht die Akquisition und Verbesserung von motorischen Fähigkeiten. Sind ein bestimmter Anfangs- und Endzustand, eine Pfadbeschränkung sowie ein Operator zur Änderung der Welt gegeben, so ist ein Programm gesucht, das die Positionen und Geschwindigkeiten der einzelnen Armelemente über die Zeit angibt, um den Zielpunkt zu erreichen. Das konkrete Vorgehen wird in der Arbeit nicht genau beschrieben. Die Motorprogramme sollen nicht im Langzeitwissen abgelegt werden, sondern werden bei jeder Ausführung wieder neu erzeugt. Der Ansatz wurde beispielsweise eingesetzt, um ein gerades Bahnstück abzufahren.

Der Einsatz von neuronalen Netzen zur Regelung von Robotern besitzt eine große Ähnlichkeit mit der adaptiven Regelung. In [YY90] werden daher Eigenschaften und Charak-

teristika von adaptiver Regelung und NN-Regelung gegenübergestellt. Während bei einer adaptiven Regelung einer gewünschten Bahn in einem Regelprozeß gefolgt wird, muß bei einem NN eine Bahn erst mehrfach trainiert werden, bevor sie exakt nachgefahren werden kann.

Ein Einsatz von NN zur Lösung des inversen kinematischen Problems wird in [AG90] beschrieben. Die entwickelte Methode ist roboterunabhängig und wurde an einem simulativen planaren Roboter mit drei Freiheitsgraden, sowie einem PUMA 560 mit 6 Freiheitsgraden eingesetzt. Ahmad erweitert die reinen Ansätze mit NN, die für praktische Anwendungen nicht präzise genug sind, indem er ein hybrides Verfahren entwickelt. In einem ersten Schritt wird von einem NN eine plausible Anfangslösung gefunden, die dann in einem iterativen Verfahren verbessert wird, bis die nötige Genauigkeit erreicht ist.

Simulative Untersuchungen zur Hand-Auge-Koordination wurden von Pabon [PG88] durchgeführt. Mit Hilfe eines hierarchischen NN werden die drei Subsysteme „Auge", „Kopf" und „Arm" koordiniert. Dazu enthält das NN eine Reihe von adaptiven Teilnetzwerken zur Verarbeitung von Sensorinformationen.

[Mil87] setzt das von Albus entwickelte Lernverfahren CMAC zum Aufbau der Beziehung zwischen Sensormessung und den Systemkommandos ein. Die gelernte Information wird verwendet, um die notwendigen Kommandos vorherzusagen, um gewünschte Änderungen in den Sensorausgaben zu erreichen. Mit Hilfe des Verfahrens wurde beispielsweise eine Roboterhand, basierend auf Daten einer an der Roboterhand montierten Videokamera, in drei Dimensionen relativ zu Objekten auf einem Tisch positioniert. In einem anderen Experiment wurde ein CMAC-Speicher eingelernt, um einem Objekt auf einem Fließband zu folgen. Zur Initialisierung wurde ein fester Feedback Controller eingesetzt, auf dem aufbauend gelernt wurde.

3.4.6 Lernen von Klassifikatoren (Lernen zur Perzeption)

Ein Ansatz von Winston et al. beschreibt das Lernen von Objektbeschreibungen aus funktionalen Angaben über das Objekt [WBKL83, WBKL84]. Neben der funktionalen Beschreibung erhält das System ein konkretes physikalisches Beispiel und eine Menge von Präzedenzfällen, d.h. Beispielbeschreibungen. Auf diese Beschreibungen wird dann das von Winston entwickelte ANALOGY-Verfahren [Win80, Win81] zum Lernen durch Analogien angewendet. Die Modellierung der einzelnen Objekte geschieht durch verallgemeinerte Zylinder, wie sie im ACRONYM-System verwendet wurden. In [Seg88a] wird das Lernen und die Klassifikation von 2D-Objekten auf der Basis der äußeren Form untersucht.

In der Teilkomponente LABYRINTH des ICARUS-Systems (s. Abschnitt 3.5.1) wird das unüberwachte Lernverfahren CLASSIT zur Klassifikation von Objekten eingesetzt. Die Unterscheidung der einzelnen Objekte wird gemäß einer Attribut-Wert-Liste mit einer festen Menge von Attributen durchgeführt. Mit dem Verfahren sollen auch strukturierte Objekte behandelt werden können, d.h. Objekte, bei denen ein Attribut selbst wieder durch eine Attribut-Wert-Liste beschrieben werden kann. Eine detailliertere Untersuchung wird allerdings nicht angeführt.

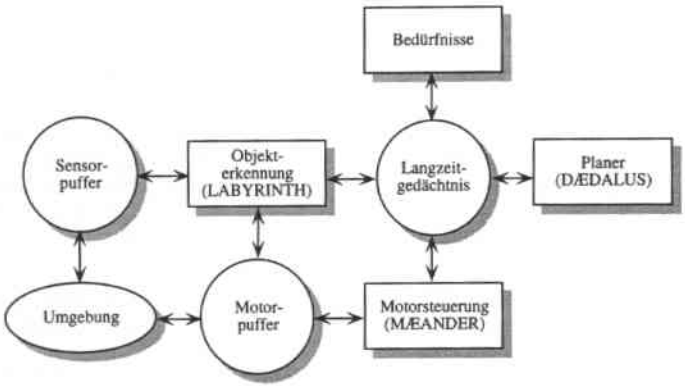

Abbildung 3.2: Systemarchitektur von ICARUS

3.4.7 Lernen zur Fehlerdiagnose

In [ZD91] wird ein Konzept für ein erklärungsbasiertes Lernsystem vorgestellt, welches Fehler in einem Telerobotiksystem erkennen und beseitigen soll. Ziel des Ansatzes ist es, daß das System aus der Analyse einer vorgegebenen Aktionsfolge zur Behebung eines Fehlers eine allgemeingültige Regel ableitet. Falls das System den Fehler nicht durch vorgegebene Fehlerbehebungsschemata auflösen kann, soll eine mögliche Lösung vom Benutzer vorgegeben werden, wobei aber bereits vollständiges und korrektes Wissen zur Interpretation einer beliebigen Benutzervorführung vorhanden sein soll. Die Idee entspricht prinzipiell dem Vorgehen bei ARMS (s. Abschnitt 3.4.2), allerdings wurden keine weiterführenden Untersuchungen angegeben.

3.5 Integrierte Systemarchitekturen

In diesem Abschnitt werden vier Arbeiten näher analysiert und bewertet, die im Gegensatz zu den obigen Arbeiten eine Integration mehrerer Lernverfahren in einer Gesamtsystemarchitektur für ein Robotersystem untersuchen. Einige rein konzeptuelle Ideen über Komponenten und mögliche Lernvorgänge in einer simulierten Roboterumgebung werden in [SH86] beschrieben. Über Ideen hinaus sind aber keine Untersuchungen angeführt, weshalb hier nicht näher auf dieses System eingegangen wird.

3.5.1 ICARUS

Ein Ansatz für eine integrierte kognitive Architektur für autonome Agenten wird in [LTI+89] vorgestellt. Der Ansatz basiert auf einer hierarchischen, probabilistischen Repräsentation des gesamten Wissens und dem Einsatz des unüberwachten Lernverfahrens CLASSIT (s.

Kapitel 2.2.4). Die entworfene Architektur besteht im wesentlichen aus drei Komponenten (s. Abbildung 3.2): LABYRINTH zur Klassifikation von Objekten, DÆDALUS zur Erstellung von Plänen und MÆANDER zur Ausführung von aktorischen Primitiven.

Langley stellt sechs Anforderungen an einen intelligenten autonomen Agenten: 1. Interaktion mit der Umgebung: Konstruktion eines Modells aus Sensoreingaben, 2. Basierte Symbole (engl. grounded symbols): Alle Symbole basieren letztendlich auf konkreten Sensorinformationen oder einer motorischen Aktion, 3. Lernfähigkeit, 4. Gute Strukturierung des Wissens, 5. Psychologische Aspekte menschlichen Denkens sollten in die Entwurfsentscheidungen einfließen, 6. Integrierte Architektur: Möglichst dieselben Langzeitwissensstrukturen, um bestmöglich von alten Erfahrungen zu profitieren.

Die Teilkomponenten des Systems wurden bereits in den entsprechenden Teilkapiteln behandelt und diskutiert: LABYRINTH (3.4.6), DÆDALUS (3.4.3) und MÆANDER (3.4.5). Die einzelnen Komponenten wurden implementiert, eine Integration wurde jedoch nicht durchgeführt. Als Beispielanwendungen für Objekte dienten in CLASSIT [GLF89] einfache Attribut-Wert-Paare und einfach strukturierte Objekte mit einer weiteren Strukturierungsebene. Eine Betrachtung, wie diese Informationen aus der Sensorik gewonnen werden können, findet nicht statt. Die Planungsprobleme, die von DÆDALUS untersucht wurden, beschränken sich auf typische Blockweltaufgaben, die off-line ohne Betrachtung von Sensorik gelöst werden. Damit ist die Übertragung auf reale Robotikprobleme an vielen Stellen noch offen. Eine Untersuchung, wie die einzelnen Komponenten zusammenspielen und sich gegenseitig beeinflussen, wurde nicht durchgeführt.

3.5.2 SOAR

In [LHYT90, LR90] wird die Erweiterung und Anwendung des SOAR-Systems [LNR87] auf Problemstellungen beschrieben, die eine Interaktion mit einer externen Welt benötigen. Soar ist ein System zur integrierten Realisierung von Planung, Planausführung und Lernen. Probleme werden gelöst, indem eine geeignete Folge von Operatoren gesucht wird. Diese Folge kann auch deduktiv generalisiert werden (chunking) und im Langzeitgedächtnis für zukünftige Anwendungen abgespeichert werden. Soar besitzt drei Ebenen zur Planung und Ausführung eines Operators: Die niedrigste und schnellste Ebene bestimmt sofort einen auszuführenden Operator durch eine Produktion (reflexives Verhalten). Auf der mittleren Ebene kann das System aufgrund seines Wissens einen Operator bestimmen. Die Auswahl kann dabei auch durch Vergleich mit alten Planfragmenten geschehen. Die höchste und zeitaufwendigste Ebene ist die Planungsebene. Hier kann außer dem internen Wissen auch externes Wissen von Lehrern oder anderen Agenten genutzt werden.

Bei der Ausführung wird ein Operator auf der Basis von Präferenzen im Langzeitgedächtnis ausgewählt, der vom gegenwärtigen Zustand ausgehend in Richtung Ziel führt. Das System bildet diese Präferenzregeln automatisch, basierend auf einer Auswahl, die vom Benutzer vorgegeben wird. Falls keine Präferenz vorliegt und der Benutzer nicht eingreift, wird ein Planungsvorgang angestoßen. Eine Reaktion auf ein externes Ereignis kann sowohl während des Planens als auch während der Planausführung erreicht werden, indem sofort eine Präferenz für eine geeignete Reaktion erzeugt wird. Soar wurde sowohl für einfache Manipulationsaufgaben (Robo-Soar, Stapeln von Blöcken und Reaktion auf ein Notsignal) als auch für Mobilitätsaufgaben (Hero-Soar) eingesetzt.

Soar stellt einen der ersten Ansätze dar, Planung, Ausführung und Lernen in einer Architektur zu integrieren. Allerdings werden nur Präferenzen für bereits vorhandenes Wissen abgeleitet, die Akquisition von neuem Wissen findet nicht statt.

3.5.3 THEO

Mitchell und Blythe beschreiben in [BM89, Mit90] einen mobilen Agenten (Theo-Agent), der mit Hilfe der allgemeinen Systemarchitektur THEO [Mit89] gesteuert werden kann. Ziel des Systems ist es, Wissen, das in möglicherweise aufwendigen Planungsschritten abgeleitet wurde, durch erklärungsbasiertes Lernen für zukünftige Situationen bereitzuhalten, um damit mit der Zeit ein möglichst reaktives Verhalten des Systems zu erreichen. Nur wenn die aufgebauten reaktiven Regeln auf die durch Sensoren gemessene Situation nicht anwendbar sind, soll wieder ein Planungsvorgang durchgeführt werden.

Das gesamte Wissen des Systems wird in Frames abgespeichert. Das Lernen in Theo besteht zum einen im Abspeichern (engl. Caching) von Slotwerten eines Frames, die einmal berechnet wurden, und zum anderen in der Ableitung von neuen Regeln auf der Basis der Erklärungsstruktur einer früheren Anfrage. Das Caching wird dadurch möglich, daß jeder Slot in der verwendeten Framerepräsentation von Theo zugeordnete Prozeduren besitzt, die angeben, wie er berechnet werden kann. Der dabei bestimmte Wert wird für zukünftige Abfragen inklusive einer Erklärung der Ableitung gespeichert. Erst wenn sich ein Wert der Erklärung ändert, wird auch der abgespeicherte Wert gelöscht.

Die reaktiven Regeln werden durch Anwendung von EBG auf den Ableitungsbaum eines Wertes erzeugt. Auf der Basis des Ableitungsbaumes wird die schwächste Vorbedingung abgeleitet, unter der die Erklärung noch gültig ist, und eine reaktive Regel erzeugt, die unter diesen Vorbedingungen die ausgewählte Aktion festlegt.

Der Theo-Agent wurde auf einem mobilen System Hero 2000 mit einfachem Greifarm getestet, das mit einem rotierenden Ultraschallsensor, einem Ultraschallsensor in der Hand und einem Batteriesensor ausgestattet war. Für jedes Ziel, das der Roboter erreichen konnte, gab es genaue Angaben dafür, wann es weiterverfolgt werden sollte. In den untersuchten Beispielen konnte die Reaktionszeit von einigen Minuten auf unter eine Sekunde verkürzt werden. Insgesamt wurden zwischen 5 und 15 reaktive Regeln gelernt.

Die Erwerb und die Korrektur des Planungswissens an sich und der daraus in früheren Schritten gewonnenen reaktiven Regeln wird nicht untersucht. Prinzipielles Problem der deduktiven Ansätze ist die Bestimmung der Nützlichkeit einer zusätzlich erzeugten reaktiven Regel, d.h. die Entscheidung, ob eine zusätzliche Regel den Planungsaufwand verkürzt – dadurch, daß sie direkt angewendet werden kann – oder verlängert, da zusätzliche Regeln betrachtet werden müssen.

3.5.4 WISMO

W. Simon beschreibt in [SM90, Sim90, Sim91] das regelbasierte Expertensystem WISMO zur intelligenten Ablaufsteuerung bei Füge- und Handhabungsaufgaben eines Roboters (s. Abbildung 3.3). Dazu werden in der Wissensbasis Strategie- und Ausführungsregeln festgelegt, die den nominalen Ablauf des Montagevorgangs steuern. Diese können vom

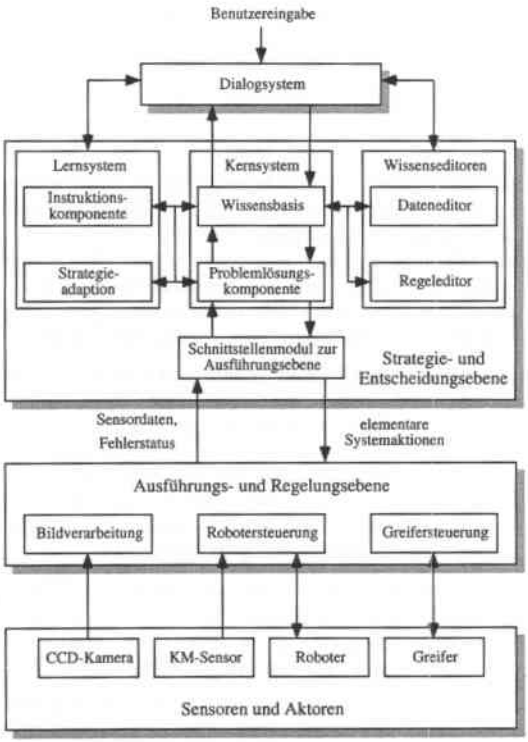

Abbildung 3.3: Systemarchitektur von WISMO

Anwender interaktiv erstellt und durch Lernen aus Anwenderinstruktionen in geeigneterer Form in der Wissensbasis abgelegt werden. Daneben muß der Benutzer die Bauteile modellieren und den Montageplan vorgeben.

Während der Ausführung einer Aufgabe verarbeitet ein Inferenzmechanismus das Regelwissen in drei Phasen: 1. In der Vergleichs- und Bindungsphase werden alle anwendbaren Regeln des momentan aktiven Teilregelsatzes ermittelt. 2. In der Auswahlphase wird auf der Basis von Regelprioritäten aus dem Satz von anwendbaren Regeln die Regel selektiert, deren Konklusionsteil ausgeführt werden soll. 3. In der Anwendungsphase wird der Konklusionsteil der selektierten Regel ausgeführt und anschließend die Datenbasis aktualisiert.

WISMO enthält zwei Lernkomponenten: Eine Komponente zur Akquisition von Strategie- und Ausführungswissen durch Anwenderinstruktion und eine Komponente zur Bestimmung von möglichst optimalen Ausführungsstrategien in teilweise unsicheren Umgebungen. Die erste Lernkomponente soll eingesetzt werden, um eine neue Montageaufgabe möglichst anwenderfreundlich programmieren zu können. Dazu kann der Anwender durch

eine interaktive Benutzerschnittstelle die Bauteile modellieren, den Montageplan vorgeben und aufgabenspezifisches Strategie- und Ausführungswissen definieren. Ein echter (z.B. induktiver) Lernvorgang findet aber nicht statt. Die Arbeit macht auch keine Aussage darüber, wie neues Wissen integriert werden kann, das zumindest teilweise mit bereits bestehendem Wissen kollidiert.

Der nominale Montageablauf wird durch die Aktionsplanungs- und Parameterregeln genau festgelegt. Treten aber Unsicherheiten in der Umwelt auf, z.B. Toleranzen bei der Fertigung der Teile, so können diese zu Fehlersituationen führen. In der zweiten Lernkomponente wird versucht, aufgrund einer Analyse verschiedener möglicher Montagevorgänge einen möglichst optimalen Montagevorgang für eine Aufgabe zu bestimmen. Der entstehende Montagegraph soll dann möglichst viele potentielle Fehlerzustände - eventuell durch zusätzliche Aktionen - vermeiden.

In der anwendungstechnischen Erprobung soll WISMO verschiedene Füge- und Handhabungsaufgaben in einer mit Unsicherheiten behafteten, realen Arbeitsumgebung zuverlässig ausgeführt haben.

3.6 Fortgeschrittene Teachtechniken

In diesem Abschnitt werden Techniken erläutert, die ein benutzergestütztes Trainieren eines Roboters ermöglichen. In den meisten existierenden Arbeiten wird zwar nicht explizit ein Lernvorgang untersucht, aber das entstehende Wissen könnte zumindest in der Form des Auswendiglernens abgespeichert werden. Für zukünftige Anwendungen autonomer oder kooperierender Systeme ist das Programmieren durch Vorführen [SB92, MKKD94] eines der wichtigsten Programmierparadigmen. Aus diesem Grund ist die Integration derartiger Vorführungen in das Systemwissen durch Lernverfahren eine wichtige Forschungsaufgabe.

Herkömmliche Teach-In-Verfahren basieren auf der Record-Playback-Methode und erlauben es, Punkt-zu-Punkt-Bewegungen ohne Entscheidungsmöglichkeiten vorzuführen [ISMK86]. Eine Einbeziehung von Kraftregelungen ist dabei nicht möglich. Sind die Positionen von Objekten bei der angewandten Ausführung eines geteachten Programms aber nicht exakt bekannt, so ist eine hybride Positions-/Kraftregelung notwendig. Das Teach-In kann prinzipiell folgendermaßen durchgeführt werden: 1. Durch Verwendung einer Teachbox, 2. Durch Führen des Roboterarms oder eines Modells davon oder 3. Durch simulative Verfahren.

Eine Erweiterung der Teach-In-Verfahren zur Berücksichtigung von Kräften und Momenten wurde in [HH83] vorgestellt. In diesem Ansatz ist es auch möglich, die Leistung des Roboters durch weiteres Teachen durch den Benutzer oder automatische Anpassung während der Ausführung zu verbessern.

Einen Ansatz zum Teachen von einfachen positions- und kraftgeregelten Bewegungen liefert [AY89, AI89]. Die Positionen und Kräfte bei der Vorführung einer Aufgabe durch den Benutzer werden aufgezeichnet und analysiert. Daraus werden Bewegungsabschnitte extrahiert, die in bestimmten Richtungen positions- und in anderen kraftgeregelt sind.

Vereinfachend wird angenommen, daß die Begrenzungsflächen aufeinander senkrecht stehende Ebenen sind und die Aufgabe mit drei translatorischen Freiheitsgraden gelöst werden kann. Im Experiment wurde die dreidimensionale Positionierung eines Zylinders in der Ecke einer Box untersucht.

In [Asa90] wird ein auf neuronalen Netzen basierendes Verfahren vorgestellt, das bei der Behandlung von einfachen linearen und nicht-linearen Compliance-Aufgaben wie dem Palettieren eines Objektes in einer 2D-Box oder dem Fügen eines Stiftes in ein Loch eingesetzt wird. Das gewünschte Compliance-Verhalten kann durch Einteachen von gewünschten Ein-/Ausgabepaaren aufgebaut werden. Basierend auf einer bestimmten Kraftmessung liefert das NN eine entsprechende Korrekturbewegung. Eine Erweiterung auf drei Dimensionen wurde unter folgenden Einschränkungen untersucht: 1. Es liegen nur Punktkontakte vor, 2. Es tritt keine Reibung auf, und 3. Es sind nur sehr kleine Bewegungen auszuführen. Alle Compliance-Aufgaben werden als lokal betrachtet, d.h. beim Stift-in-Loch-Problem muß bereits ein Kontakt zwischen Stift und Loch bestehen, um eine sinnvolle Bewegung zu erhalten.

Eine Erweiterung auf das Einteachen von allgemeineren Compliance-Strategien führt Buckley durch [Buc89, Buc88]. Das Verfahren basiert auf der Idee der Pre-Images von Erdmann, d.h. Regionen, von denen aus das gewünschte Ziel unter Berücksichtigung von Unsicherheiten bzgl. der Startkonfiguration, des Roboterarms und der eingesetzten Sensorik sicher erreicht wird. Gefordert wird eine feste Schranke aller auftretenden Unsicherheiten. Eventuelle Modellierungsfehler der beteiligten Objekte können nicht berücksichtigt werden. Das System baut aus mehreren Eingaben des Benutzers (per Tastatur) eine Fügestrategie auf, indem ausgehend vom Zielbereich interaktiv Bewegungen zu Zwischenzielen angegeben werden. Deren Pre-Images werden berechnet und dann sukzessive die nächste Bewegung angegeben, bis die gesamte Startregion erfaßt ist. Dabei wird eine Tabelle aufgebaut, die einer Region von gemessenen Positionen und zugehörigen Kraftorientierungen ein Kommando zuordnet, wie es vom Benutzer vorgeschlagen wurde.

Der Einsatz von Teach-In-Methoden in der Telerobotik wird in [SH87, HS88, HS89, HS90] beschrieben. Das Telerobotiksystem MEISTER ist in der Lage, Bewegungen des Bedieners in gewisser Weise zu interpretieren (z.B. der Greifer nähert sich einem Objekt) und dementsprechend bestimmte Teiloperationen (z.B. das Greifen des Objektes), die dem System bereits vorgeführt wurden, selbständig auszuführen. Der Arbeitsraum des Roboters wird in einzelne Zellen unterteilt, die auch Verweise auf die darin befindlichen Objekte verwalten. Während der Bedieneraktion werden dann die Regeln aktiviert, die für die Objekte der aktuellen Zelle anwendbar wären. Auf die tatsächliche Auswahl einer Regel wird nicht genauer eingegangen. Falls neue, unbekannte Objekte auftreten, kann ein neues Objektmodell erzeugt werden. Dieses Telerobotiksystem mit Lernkomponenten wurde bereits in realen Anwendungen getestet. Dabei tritt ständig eine Interaktion zwischen Benutzer und System auf, wenn das System eine begonnene Aufgabe selbständig fortsetzen will, oder es bei der Ausführung einen Fehler begeht.

Einen weiteren Ansatz zum Programmieren von Robotern durch Vorführungen untersucht Ikeuchi an der CMU. In seinen Arbeiten [IS91, IS92, IKS93] untersucht er, wie aus der Aufgabenlösung, die ein Mensch ohne Verwendung des Roboters durchführt, auf die entsprechende Manipulationsfolge geschlossen werden kann, um diese Lösung mit einem Manipulator auch zu erreichen. Auf der Basis von Abstandsbildern und zugrundeliegen-

den Objektmodellen wird der Zustand der Welt vor und nach einer Aktion des Benutzers untersucht und darin die gültigen Relationen bestimmt. Basierend darauf werden vorhandene Aufgabenschemata mit den vorliegenden Objekten entsprechend instanziiert. Diese Aufgabenschemata geben exakt an, wie der Übergang von einer Relation zur nächsten erreicht werden kann. Ein echter Lernvorgang findet nicht statt.

Ein ähnliches Vorgehen untersucht [KII92]. Auch dort wird aus den Bewegungen eines Menschen bei Pick-and-Place-Operationen eine symbolische Beschreibung für die ausgeführte Handlungssequenz abgeleitet und diese basierend auf vorhandenem Ausführungswissen für den Roboter angewandt. Die symbolische Beschreibung soll zwar auf ähnliche Aufgaben mit beliebigem initialen Zustand angewandt werden können, tatsächlich darf aber nur die Position der einzelnen Objekte unterschiedlich sein, und es wird lediglich eine Transformation der bei der PICK-Operation anzufahrenden Position durchgeführt. Der Zielzustand muß außerdem identisch bleiben.

Ein Problem der zuvor genannten Ansätze, bei denen der Benutzer bei der Aufgabenlösung beobachtet wird, ist die Tatsache, daß nicht sichergestellt ist, ob der Manipulator selbst die Aufgabe prinzipiell tatsächlich durchführen kann. Außerdem wird nicht berücksichtigt, daß der Mensch bei der Vorführung der Aufgabe eventuell umfangreiche eigene Sensorik einsetzt, die dem Manipulator bei seiner Ausführung nicht zur Verfügung steht. Bewegungen, die von Sensormessungen zur Laufzeit abhängig sind, können auf diese Weise auch nicht trainiert werden. Im Grunde dienen diese Vorführtechniken also nur dazu, die Folge der Relationen für diesen einen Aufgabentyp anzugeben.

Eine zukünftige Weiterentwicklung wäre, Techniken wie sie im Bereich des Virtual Reality untersucht werden als Möglichkeit zur Vorführung einer Aufgabe einzusetzen. Erste Untersuchungen dazu werden in [TO92] aufgeführt. Ein großes zukünftiges Anwendungsgebiet für das Programmieren durch Vorführen ist die Telerobotik. Durch die Lösungen, die vom Benutzer durchgeführt werden, entsteht eine große Zahl von Beispiellösungen, die das System auswerten könnte, um so für bestimmte Teilbereiche ein autonomes Verhalten zu ermöglichen.

3.7 Kritische Bewertung

In diesem Kapitel wurde die Anwendbarkeit von Lernverfahren in Robotersystemen untersucht. Obwohl eine ganze Reihe von Lernzielen identifiziert werden konnten, ist die Anzahl der dazu veröffentlichten Arbeiten relativ gering. Die Gründe dafür liegen vor allem an den unterschiedlichen Annahmen bzw. Voraussetzungen für typische Anwendungen des Maschinellen Lernens und der Robotik. Die Analyse der veröffentlichten Arbeiten hat zum einen gezeigt, daß sehr spezielle Teilprobleme eines Robotersteuerungssystems oder sehr vereinfachte Aufgabenstellungen untersucht werden. Zum anderen sind nur in wenigen Fällen Standardverfahren des ML eingesetzt worden. Fast immer wurden eigenentwickelte Lernverfahren verwendet, die selten ausreichend detailliert dargestellt wurden. Bisher wurden vier Systemarchitekturen vorgeschlagen, die bis zu einem gewissen Grad die Integration von Lernverfahren ermöglichen. ICARUS enthält zwar die größte Anzahl von Lernzielen, betrachtet aber nur sehr eingeschränkt wirkliche Anforderungen der Robotik. Das System SOAR bildet eine gute Grundlage für den Einsatz von deduktiven Lernverfahren in einem Planungssystem, eine echte induktive Wissensakquisition wird aber nicht

untersucht. Ebenfalls rein deduktive Verfahren zur Steigerung der Reaktivität eines mobilen Systems mit Greifer werden in THEO angewendet. Die am stärksten an der Robotik orientierte Architektur bietet WISMO. Dafür werden allerdings nur sehr eingeschränkt Lernverfahren untersucht. Aus diesen Gründen wird im nachfolgenden Kapitel eine Analyse der Anforderungen für ein lernendes Robotersystem durchgeführt. Von diesen sollen durch den nachfolgenden Systementwurf möglichst viele erfüllt werden.

Kapitel 4

Analyse der Anforderungen an die Systemarchitektur

Dieses Kapitel führt eine Analyse der Anforderungen durch, die an die zu entwickelnde Systemarchitektur gestellt werden. Dazu werden auf der Basis der in den Kapiteln 2 und 3 analysierten Punkte die wichtigsten Eigenschaften eines lernenden autonomen Robotersystems herausgearbeitet. Für die Bereiche Architektur, Wissensrepräsentation, Planung und Ausführung, Lernen und Sensorik und Aktorik werden die Anforderungen und die daraus folgenden Konsequenzen herausgestellt. Ebenso wird dadurch eine Abgrenzung zu anderen Untersuchungsgegenständen vorgenommen bzw. die mögliche Integration aufgezeigt. Eine Detaillierung der einzelnen Anforderungen wird zu Beginn der einzelnen nachfolgenden Kapitel durchgeführt.

4.1 Architektur

Die zu entwerfende Architektur muß vor allem die in den anderen Abschnitten dieses Kapitels aufgestellten Forderungen unterstützen. Darüberhinaus ist eine wichtige Anforderung die Integration von Planung, Ausführung, Perzeption und Lernen in einem möglichst homogenen Ablauf. Eine explizite Trennung in eine Lernphase, eine Planungsphase und eine nachfolgende Ausführungsphase ist nicht adäquat. Zum einen muß ein Weiterlernen auch während der Anwendung des Wissens möglich sein, zum anderen müssen Planung und Ausführung überlappen, um den Anforderungen einer teilweise unbekannten Umwelt gerecht zu werden, in der unerwartete Ereignisse auftreten können. Die Architektur muß den Umgang mit derartigen Unsicherheiten ermöglichen.

Im Gegensatz zu rein verhaltensorientierten Architekturen wird hier angestrebt, daß das System eine Aufgabenstellung eines Benutzers lösen kann. Aus diesem Grund muß ein zielgerichtetes Vorgehen möglich sein. Eine Kopplung von kognitiven und reaktiven Komponenten ist für die erfolgreiche Bearbeitung der Aufgaben in den beschriebenen Umwelten aber wichtig. Das System sollte also zum einen zielorientiert handeln, zum anderen aber ereignisgesteuert auf bestimmte Signale und Informationen reagieren können. Die Architektur sollte ähnlich wie z.B. ACT* [And89] einen sehr natürlichen Ablauf un-

terstützen, explizit in Kurz- und Langzeitwissen unterscheiden, durch entsprechende Verarbeitungseinheiten sukzessive Informationen verarbeiten und neues Wissen ableiten, und durch Lernvorgänge das langfristige Systemwissen verbessern und erweitern. Das Wissen muß dazu fokussiert werden, d.h. in bestimmte Teilkomponenten aufgegliedert werden, damit bei der Ausführung einer Aufgabe oder bei der Durchführung eines Lernschritts eine Beschränkung auf das jeweils relevante Wissen vorgenommen werden kann. Diese Lernvorgänge sollen nicht nur intern ablaufen, sondern ähnlich wie bei menschlichen Vorgängen auch durch Beobachtung eines Menschen oder eines Systems stattfinden, der bzw. das eine bestimmte Teilaufgabe bereits lösen kann.

4.2 Wissensrepräsentation

An das Wissen, das im Rahmen dieser Arbeit verwendet wird, bzw. die zugrundeliegende Wissensrepräsentation werden zwei wichtige Forderungen gestellt: Erstens die Kommunikationsfähigkeit und zweitens die Manipulierbarkeit. Das Wissen, das vom System aufgebaut wird, soll teilweise durch Interaktion mit einem Benutzer akquiriert werden. Daher müssen wesentliche Inhalte dem Benutzer erklärt werden können und die Entscheidungsstruktur für ihn nachvollziehbar sein. Eine Repräsentation wie sie bei Neuronalen Netzen verwendet wird, und die entsprechenden Verfahren sind auf elementaren Steuerungs- bzw. Interpretationsebenen einsetzbar, für höhere Entscheidungsebenen erfüllen sie diese Anforderungen jedoch nicht. Das repräsentierte Wissen soll daneben durch Lernvorgänge ergänzt und korrigiert werden. Aus diesem Grund ist es wichtig, daß das Wissen tatsächlich manipuliert werden kann. Eine Realisierung der planenden Komponenten in Form von Programmcode ist nur sehr bedingt einsetzbar.

Der zu entwickelnde Wissensrepräsentationsformalismus sollte eine möglichst homogene Darstellung der unterschiedlichen Wissensinhalte des Systems ermöglichen. Damit sollten die zugehörigen Inferenzmechanismen und die Verfahren zur initialen Akquisition und sukzessiven Erweiterung und Verbesserung des Wissens möglichst gleichartig eingesetzt werden können. Allerdings müssen zusätzlich andere Repräsentationsformen und Verarbeitungsmechanismen entwickelt werden, wenn die Darstellung bestimmter Wissensinhalte in der „Standarddarstellung" nicht adäquat ist. Die gemeinsame Nutzung verschiedener Repräsentationsformen und Verfahren muß durch den Aufbau der Architektur und den Ablauf unterstützt werden.

Eine häufig gestellte Forderung selbst an autonome Systeme ist, daß für alle Objekte der Umwelt, in der agiert wird, ein exaktes CAD-Modell zur Verfügung steht und – eventuell nach einer initialen Positionsbestimmung der Objekte – ein Umweltmodell existiert. Diese Annahme ist jedoch in natürlichen Umgebungen und bei Aufgaben, in denen teilweise unbekannte Objekte in der Manipulationsumgebung vorliegen können bzw. durch externe Einflüsse in sie gelangen, nicht zulässig. Häufig ist aber eine vollständige Modellierung aller Objekte auch weder möglich noch notwendig. Dies kann sowohl ökonomische Gründe haben, d.h. daß sich der Einsatz eines Roboters dann für eine kleine Losgröße nicht lohnt, oder auch praktische, da ein Objekt zwar eventuell a priori modelliert wurde, aber durch Verarbeitung, Verschmutzung oder Zerbrechen nicht mehr in der modellierten Form vorliegt. Aus diesem Grund soll auf eine exakte Modellbildung aller beteiligten

Objekte verzichtet werden und, falls nötig, auf prototypische, generische Beschreibungen zurückgegriffen werden.

Ansätze, die eine Rekonstruktion der beteiligten Objekte aus Abstands- oder Stereodaten ermöglichen, sind prinzipiell eine sinnvolle Ergänzung des Ansatzes, aber auch dort muß der jeweilige Aufwand in Relation zur Notwendigkeit für die Aufgabenstellung gesetzt werden.

4.3 Integration von Planung und Ausführung

In traditionellen deduktiven Problemlösungsansätzen werden Annahmen gemacht, die für ein autonomes System keinesfalls aufrecht erhalten werden können. Nach [Jan88] zählen zu den wichtigsten dieser Annahmen: 1. Eine neue Situation kann nur durch eine Aktion entstehen, 2. Der Problemlöser ist der einzige Agent im System, d.h. alle Aktionen in der Welt sind Aktionen des Problemlösers, 3. Eine Aktion führt sofort zu der neuen Situation, 4. Die Situationen enthalten keinerlei Informationen über die Historie des Problemlösevorgangs und 5. Die Welt, in der die Problemlösung stattfindet, sowie die Effekte der Aktionen sind vollständig beschrieben. Sämtliche Annahmen müssen hier fallengelassen werden. Dann ist aber, wie bereits oben erwähnt, die Überlappung von Planung und Ausführung eine der wichtigsten Anforderungen. Dadurch soll die Planung in Abhängigkeit von dem jeweils erreichten Zustand aus fortgeführt werden, d.h. ein opportunistisches Verhalten ist wichtig. Durch dieses Vorgehen wird eine kurz- oder langfristige Umorientierung des Systems während der Lösung eines Problems möglich. Allerdings muß die Problemlösung dazu in kleinen Schritten ablaufen, so daß eine Reaktion auf neue Informationen überhaupt möglich ist. Weder eine abgekapselte Abarbeitung größerer Einheiten noch die Vorsehung aller Varianten in einem vollständigen Plan sind dazu geeignete Methoden. Andererseits ist die vollständige Aufzählung aller möglichen Situationen und aller dann möglichen Aktionen unmöglich.

Parallel zu den reaktiven Eigenschaften muß die Planung aber auch strategisch orientiert sein, d.h. die Zielvorgaben des Benutzers müssen konsequent weiterverfolgt werden. Sowohl durch Benutzereingriff als auch durch die Notwendigkeit einer Reaktion kann es sein, daß gleichzeitig mehrere Ziele existieren, die das System verfolgen soll – diese können sogar widersprüchlich sein. Die relative Wichtigkeit eines Ziels ist dabei abhängig von der aktuell vorliegenden Situation. Das System sollte also mit mehreren unterschiedlich wichtigen und zeitkritischen Zielen umgehen können.

4.4 Lernen

Eine der wichtigsten Anforderungen an das System ist die Lernfähigkeit, d.h. die Fähigkeit, initial Wissen zu akquirieren und es sukzessive zu modifizieren und zu erweitern. Dieser Lernprozeß soll zum einen durch systeminterne Analysen, zum anderen durch Interaktion mit einem Benutzer stattfinden. Als Lernverfahren sind – neben anderen – induktive Methoden einzusetzen, da neues Wissen aufgebaut werden soll.

Das System kann das gesamte notwendige Wissen nicht völlig selbständig aufbauen. Dazu ist die Vielfalt, d.h. der Zustandsraum, zu groß, die automatische Bewertung eines Erfolgs,

wie bei Reinforcementverfahren, für den allgemeinen Aufgabenfall zu schwierig, wenn nicht unmöglich und die Menge der dabei erhaltenen positiven Beispiele viel zu gering. Die Untersuchungen von Lin [Lin91] haben gezeigt, daß der unüberwachte Erwerb einer Strategie durch Reinforcement Lernen selbst in einfachen Spielzeugwelten zu zeitaufwendig ist. Aus diesem Grund basiert der verfolgte Ansatz darauf, daß ein Benutzer als Lehrer agiert und bestimmte Vorgehensweisen vorführt. Das aufgebaute Wissen besteht dabei nicht aus der exakten Trajektorie, die vorgeführt wurde – wie bei Teach-In Verfahren –, sondern aus bestimmten Teilzielen, die der Benutzer aufgrund der jeweils vorliegenden Situationen eingenommen hat. Diese Teilziele sind dann auf neue Aufgabenstellungen übertragbar und werden weiterverfolgt, wenn zwischenzeitlich andere Ziele Vorrang hatten. Es muß also eine Abstraktion von der konkreten Bewegung zu einer symbolischen Beschreibung vorgenommen werden.

Das dabei entstehende Wissen soll aber eine möglichst kleine Granularität haben, damit es auf möglichst viele ähnliche Aufgabenstellungen übertragbar ist – die Darstellung als ein monolithisches Programm ist nicht sinnvoll. Die Benutzervorführung soll parallel zum Systemablauf verarbeitet werden, damit nachvollzogen werden kann, welche Situationsmerkmale für die Wahl einer Teilaktion relevant gewesen sein könnten. Die Kausalitäten der Benutzervorführung sollen also explizit modelliert werden, es geht nicht nur um die Zuordnung einer festen Teilaktion zu einer bestimmten Situationsklasse. Andererseits könnte durch das parallele Vorgehen eine Prädiktion der nächsten Benutzeraktion vorgenommen werden, um so die Interpretation zu erleichtern. Der Benutzer sollte auf möglichst natürliche Art die Vorführung der Aufgabenlösung durchführen können. Langfristig sollte dieses Vorgehen auch für einen Laien möglich sein.

4.5 Sensorik und Aktorik

Brooks [Bro85] stellt drei wichtige Anforderungen an ein autonomes System bezüglich der Sensorik und der Aktorik: Robustheit, Erweiterbarkeit und Verarbeitung mehrerer Sensoren. Beim Ausfall von Sensoren oder Prozessoreinheiten oder bei einer drastischen Änderung der Umgebung sollte das System immer noch sinnvoll agieren, d.h. robust sein. Es muß um neue Sensoren und Fähigkeiten erweiterbar sein – inklusive der dazu nötigen höheren Verarbeitungskapazität. Die Messungen der verschiedenen Sensoren des Systems müssen integriert werden, um so fehlerhafte oder verrauschte Daten entsprechend kompensieren zu können.

Für die Aktorik ist wichtig, daß auf bestimmten elementaren Operationen aufgebaut wird. Dies kann von einer direkten Ansteuerung auf Gelenkwinkelebenen bis hin zu relativ hohen Primitiven mit sensorgestützten Teilstrategien reichen [HST91b, HST91a, Sch94]. Höhere Elementaroperationen können dabei selbst durch Lernmethoden aufgebaut werden. Insbesondere für Regelungsaufgaben sind subsymbolische Verfahren erfolgreich eingesetzt worden. Ein allgemeines Vorgehen für ein geeignetes initiales Training, z.B. auch aus Benutzervorführungen, ist noch nicht gefunden.

Auf unterster Ebene der Aktorik muß eine Kollisionsvermeidung bezüglich der Umweltobjekte und anderer Agenten durchgeführt werden. Diese kann nicht – wie in den meisten Arbeiten zu diesem Thema angenommen – off-line geschehen. Da a priori nicht alle Objekte bekannt sind, bzw. während der Ausführung einer Bewegung Objekte hinzukommen

können bzw. ihre Position verändern, muß auch hier ein reaktives Verhalten erreicht werden. Eine solche Vorgehensweise ist im Bereich der mobilen Robotersysteme schon seit langem üblich und erfolgreich eingesetzt worden [DKW93]. Bei mobilen Systemen kann das Problem sehr viel einfacher gelöst werden, weil ein solches System sich nur in einer Ebene bewegen kann und damit das Problem im wesentlichen auf ein zweidimensionales reduziert werden kann. Als Sensorik ist dann ein Ring von Ultraschallsensoren am Rand eines Fahrzeuges ausreichend. Ein ähnliches Vorgehen müßte langfristig auch an einem Roboterarm durchgeführt werden. Falls Abstandssensoren an mehreren Punkten des Manipulators angebracht sind (sensitive Haut, engl. sensitive skin), sollten Erweiterungen der Methoden für mobile Systeme einsetzbar sein. Erste Arbeiten dazu wurden von Cheung und Lumelsky veröffentlicht [CL88, CL89, LC91]. Die modellbasierte Kollisionsvermeidung ist prinzipiell für die Kollisionsvermeidung zwischen steuerbaren Einheiten möglich, die ihre jeweilige Position ausreichend schnell kommunizieren können. Für die Kollisionsvermeidung mit passiven Objekten oder einem Manipulator, der sich auf einem relativ bewegten mobilen System befindet, ist das Vorgehen jedoch nicht sinnvoll.

4.6 Zusammenfassung

In diesem Kapitel wurden die wichtigsten Anforderungen an eine Systemarchitektur und Teilkomponenten für ein lernendes, autonomes Robotersystem herausgestellt. Bisher entwickelte Ansätze erfüllen diese Anforderungen nur zu einem geringen Teil. Insbesondere die Möglichkeit zur Integration von Lernverfahren muß beim Entwurf der Architektur als Ganzes berücksichtigt werden. Die in den folgenden Kapiteln dargestellte Architektur und die entwickelten Teilkomponenten sollen diese Integrationsmöglichkeit von Lernverfahren schaffen und gleichzeitig die anderen aufgestellten Anforderungen erreichen bzw. deren Integration ermöglichen. Eine Detaillierung der einzelnen Anforderungen erfolgt jeweils zu Beginn der nachfolgenden Kapitel.

Kapitel 5

Systemarchitektur

Unter Berücksichtigung der in Kapitel 4 gestellten Anforderungen wird in diesem Kapitel eine entsprechende Systemarchitektur entwickelt (siehe auch [Kre92, Hau92, KH93a, KH93b, Bre93]). Zunächst werden die einzelnen Elemente der Wissensrepräsentation und der Architektur definiert. Anschließend wird der dynamische Ablauf einer Problemlösung innerhalb des Systems beschrieben (Kap. 5.3). Dazu gehört insbesondere die Auswahl geeigneter Komponenten des Systems und der Wissensbasen. Nachfolgend wird das planende Verhalten des Systems analysiert (Kap. 5.4), das aus dem entwickelten Ablaufmechanismus resultiert. Jeder Planschritt führt mittelfristig zur Erarbeitung eines einzunehmenden Teilziels, das als Relation zur Umwelt beschrieben wird. Das Vorgehen zur Erreichung eines derartigen Teilziels ist Gegenstand des nachfolgenden Abschnitts (Kap. 5.5).

Kapitel 5.6 beschreibt kurz die Teilkomponente zur Behandlung von Objekten, da das gesamte Kapitel 7 diesen Teilbereich behandelt. Die Teilkomponenten zur Analyse von Vorführungen des Benutzers (Kap. 5.7) und zur Ankopplung von Sensorik und Aktorik (Kap. 5.8) sind Gegenstand der nächsten Abschnitte. Im letzten Teil dieses Kapitels wird eine kurze Übersicht über die integrierten Lernkomponenten gegeben (Kap. 5.9), die in den nachfolgenden Hauptkapiteln ausführlich diskutiert werden.

5.1 Einleitung

Ein autonomes System wird hier als System betrachtet, das in seiner Umwelt selbständig agieren muß. Durch Sensoren oder Kommunikation mit einem Benutzer nimmt es Informationen über die Umwelt und die Aufgaben auf. Es versucht daraufhin, sowohl die Sensorinformationen zu verarbeiten als auch aus den Aufgabenstellungen entsprechende Handlungen abzuleiten. Grundlegende Idee bei der Verarbeitung von Daten der realen Welt ist, daß in einem ersten Schritt die Daten in einen bestimmten Kontext eingeordnet werden, der als Grundlage für die Interpretation dient. Im zweiten Schritt werden die Daten interpretiert und mögliche Schlußfolgerungen gezogen. Während dieses Vorgangs können ständig neue Informationen in das System gelangen, die eventuell das Systemverhalten beeinflussen, oder das System kann durch eine konkrete Aktion aktiv die Umwelt verändern (s. Abbildung 5.1).

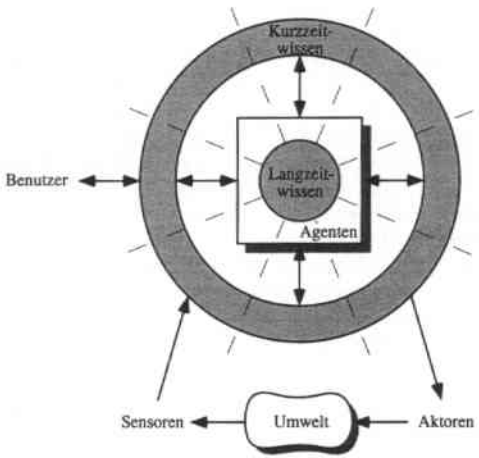

Abbildung 5.1: Anbindung des Systems an die Umwelt

Die aufgenommenen oder verarbeiteten Informationen sind im allgemeinen verschiedenen Einheiten von Wissen zuzuordnen, wie z.B. Wissen über Objekte, über Pläne oder über Teilziele. Das Wissen jeder dieser Wissenseinheiten wird in einer speziellen Form interpretiert und weiterverarbeitet. Die Komponenten zur Verarbeitung des Wissens werden später als Agenten bezeichnet.

Das Wissen, das nur während einer Anwendung des Systems, z.B. nur während einer Aufgabenlösung aufgebaut und verwendet wird, wird als Kurzzeitwissen (KZW) bezeichnet. Es repräsentiert Informationen über die Umgebung, Objekte, die zu manipulieren sind, zu erreichende Teilziele usw. Zur Verarbeitung des Kurzzeitwissens, besitzen die Agenten langfristig gültiges Wissen, das als Langzeitwissen (LZW) bezeichnet wird. Dieses gibt beispielsweise an, wie aus dem vorhandenen kurzfristigen Wissen weitere Informationen abgeleitet werden können.

Auf dem Kurzzeitwissen findet kein Lernvorgang statt, es wird als Menge akquirierter oder abgeleiteter Informationen über den Weltzustand oder den internen Systemzustand aufgefaßt. Dagegen soll das Langzeitwissen der einzelnen Agenten durch eigene Erfahrungen und durch Kommunikation mit dem Benutzer sukzessive erweitert und verbessert werden.

Bei der Bearbeitung einer Aufgabe oder neu eintreffender Sensorwerte muß zwischen den einzelnen Einheiten Wissen weitergegeben werden, d.h. die einzelnen Einheiten müssen kommunizieren können. Dazu werden die Wissenseinheiten über sogenannte Kanäle miteinander verbunden. Beispielsweise können Objektinformationen den Planablauf beeinflussen, neue Sensorwerte zu einer Aktualisierung von Objektinformationen und zu Veränderungen bei der Einnahme von Teilzielen führen.

Die mögliche Repräsentation des Wissens und die Definition der Verarbeitungseinheiten (Inferenz- und Lernverfahren) der einzelnen Komponenten der Architektur sind durch

zwei gegensätzliche Ansätze gekennzeichnet: 1. Verwendung eines einheitlichen Wissensrepräsentationsformalismus und der gleichen Verarbeitungseinheiten für alle Agenten. Damit würde im Extremfall eine einzige Wissenseinheit und ein einziger Agent ausreichen, der für die eine Wissenseinheit im System zuständig ist. Diese Form entspricht der in vielen ML-Arbeiten zugrundegelegten. 2. Spezialisierte Wissensrepräsentationsformen und Verarbeitungsmechanismen für die einzelnen Klassen von Wissenseinheiten, d.h. spezialisierte Definition der einzelnen Agenten. Jeder Agent stellt sein Wissen in einer eigenen Repräsentation dar und verarbeitet es auf seine Art. Vorteil des ersten Ansatzes wäre die völlige Homogenität und damit die Verwendung *einer* Wissensrepräsentation, *eines* Inferenzmechanismus zur Ableitung neuen Wissens und *eines* Lernverfahrens zur Manipulation des Wissens. Der Nachteil dieser und Vorteil der zweiten Methode ist allerdings, daß ein einziges Verfahren im allgemeinen nicht adäquat ist, die unterschiedlichen auftretenden Probleme bei realen Objektbeschreibungen, Plänen, Teilzielen etc. zu lösen. Die Darstellung des Langzeitwissens in einer einzigen Repräsentation ist nicht sinnvoll, und die Fokussierung von Wissen speziell für die Lernvorgänge wird nicht direkt unterstützt. Nachteil des zweiten Ansatzes ist, daß für jeden Agenten eigene Repräsentationsformalismen und Verarbeitungsverfahren entwickelt werden müßten und eine Kommunikation der Agenten untereinander nicht direkt möglich ist.

Aus diesem Grund verwendet der in dieser Arbeit verfolgte Ansatz einen Mittelweg, die Repräsentation des Kurzzeitwissens ist für alle Wissenseinheiten gleich, die Agenten besitzen einen Teil ihres Wissens und der zugehörigen Verarbeitungs- und Lernmechanismen in einer agentenunabhängigen Form, die für alle Agenten gleich verwendet wird. An Stellen, an denen aber andere Repräsentationsmechanismen sinnvoller oder notwendig sind, können diese eingesetzt werden. Allerdings sind dann auch entsprechende Verarbeitungs- und Lernmechanismen zu entwickeln.

Soweit wie möglich sollte also eine homogene Struktur für das ganze System gefunden werden, um die zu entwickelnden Methoden für die allgemeinen Verarbeitungs- und Lernkomponenten an möglichst vielen Stellen einsetzen zu können. Ist die allgemeine Repräsentation nicht adäquat oder das Lernverfahren ungeeignet, müssen entsprechende Speziallösungen gefunden werden.

Als allgemeine Wissensrepräsentation sind Produktionen bereits in vielen Systemen erfolgreich eingesetzt worden. Oft werden Produktionen in KI-Planungssystemen oder Expertensystemen eingesetzt, um off-line durch Suchverfahren einen Pfad zu einem bestimmten Zielpunkt zu bestimmen, z.B. eine Problemlösung oder eine Diagnose. Im Rahmen des vorliegenden Systementwurfs steht jedoch im Vordergrund, daß die Regeln nicht in einem Schritt zu Beginn angewendet werden, sondern sukzessive während des Ablaufs des Systems ausgewertet werden müssen.

Das Ziel, Lernverfahren innerhalb der Architektur einzusetzen, führt dazu, daß bestimmte Definitionen der Systemkomponenten relativ restriktiv vorgenommen werden. Würde das System nur mit der Absicht entwickelt, ein Steuerungssystem zu definieren, dessen Wissen vom Benutzer vorgegeben wird, so könnten an einigen Stellen auch komfortablere Beschreibungsmöglichkeiten vorgesehen werden. Es ist also bei allen Repräsentationen zu berücksichtigen, daß sie modifizierbar und das darin repräsentierte Wissen lernbar ist.

Im nachfolgenden Abschnitt werden die einzelnen Komponenten des Systems definiert. Das darauf basierende dynamische Verhalten, d.h. der Ablauf einer Problemlösung, ist

dann Gegenstand von Abschnitt 5.3.

5.2 Komponenten der Systemarchitektur

In diesem Abschnitt werden die Basiskomponenten der Systemarchitektur vollständig definiert. Dazu gehört zunächst die Definition des Kurzzeitwissens in Form von Hypothesen, die zu Wissenseinheiten zusammengefaßt werden. Anschließend werden die Verbindungen zwischen den Wissenseinheiten zur Kommunikation eingeführt, die Kanäle genannt werden. Die Menge aller Wissenseinheiten zusammen mit der Menge der Kanäle bildet den aktuellen Systemzustand. Im zweiten Teil des Abschnitts erfolgt die Definition der Repräsentation des Langzeitwissens. Die verwendeten Produktionsregeln bestehen aus einem Bedingungsteil, der aus einer Menge elementarer Bedingungen aufgebaut wird, und einer Aktion, die den Systemzustand verändert. Die Komponente zur Speicherung, Interpretation und Modifikation des Langzeitwissens wird als Agent definiert. Für die spezielle Anwendung in der Robotik werden anschließend die notwendigen Agenten und ihre Aufgaben beschrieben.

Zunächst wird der Begriff der Hypothese definiert. Jede Art von Information über den externen Weltzustand oder den internen Systemzustand wird als *Hypothese* bezeichnet, z.B. eine Information über einen Teilplan, ein Objekt oder eine Aufgabe. Eine Hypothese ist eine Aussage, die nur mit einer gewissen Sicherheit wahr ist. Zum einen ist die Information, die von der Sensorik und eventuell auch vom Benutzer initial geliefert wird unsicher, zum anderen ist das Wissen, das zur Ableitung neuer Informationen aus bestehenden verwendet wird, nicht notwendigerweise korrekt. Die Menge aller Hypothesen im System stellt das *Kurzzeitwissen* dar[1]. Die Begriffe der Wissenseinheit, des Typs einer Wissenseinheit und eines Pfades werden nachfolgend definiert.

Definition 5.1 (Hypothese) *Eine **Hypothese** ist ein Quintupel (T,E,P,N,W) und stellt eine elementare Information, eine Aussage, über die Umwelt oder über den Systemzustand dar. Dabei ist T der Typ der erzeugenden Wissenseinheit E, P ein Pfad (s. Def. 5.3), N der Name des Attributs über das eine Aussage gemacht wird und W der zugehörige Wert des Attributs. Der Wert eines Attributes kann sein:*

1. *Ein numerischer oder symbolischer Wert* w_i
2. *Ein numerisches Intervall* $[w_1, w_2]$
3. *Eine Menge von diskreten Werten* $\{w_1, \ldots, w_n\}$
4. *Eine Aufzählung von Werten entsprechend 1 bis 4* $(w_1, \ldots, w_n)$

*Jede Hypothese besitzt eine **Stärke** $S(H) \geq 0$, die ein Maß für die Relevanz und Sicherheit der Hypothese ist.*

[1]Der Begriff Hypothese ist hier nicht zu verwechseln mit einer induktiven Hypothese, die bei einem induktiven Lernverfahren eine abgeleitete, potentiell gültige Gesetzmäßigkeit o.ä. beschreibt.

Hypothesen können zu Gruppen (Wissenseinheiten) zusammengefaßt werden, die Aussagen in einem gemeinsamen Kontext darstellen, z.B. alle Hypothesen über ein Objekt, über ein Teilziel oder eine einzunehmende Relation. Von der Anwendungsdomäne her gesehen gibt es bestimmte Klassen von Gruppierungen, die betrachtet werden müssen, wie z.B. die Typen der Objektwissenseinheiten oder der Planwissenseinheiten.

Definition 5.2 (Wissenseinheit) *Eine **Wissenseinheit** WE ist eine Menge von Hypothesen $\{H_1, \ldots, H_n\}$, die mit dem gleichen Bezug interpretiert werden. Jeder Wissenseinheit wird ein **Typ** zugeordnet. Die Abbildung Typ : WE $\rightarrow$ {Typen von WE} erlaubt eine domänenabhängige Partitionierung der Wissenseinheiten in bestimmte Klassen.*

Der Pfad zu einer bestimmten Aussage wird eingeführt, um eine hierarchische Strukturierung des Wissens zu unterstützen. So müssen beispielsweise wenn eine Wissenseinheit sämtliche Aussagen über ein Objekt zusammenfaßt, Aussagen über eine Kante, wie ihre Länge oder Krümmung, entsprechend dieser Kante zugeordnet werden – und eventuell die Kante einer Fläche etc.

Definition 5.3 (Pfad) *Eine Liste $(P_1, \ldots, P_n)$ heißt Pfad, wenn gilt: $\forall i = 1, \ldots, n : P_i$ ist ein Tupel (N_i, W_i) und N_i ist der Name eines Attributs mit diskretem Wertebereich, W_i ist ein Attributwert. Für alle $P_i = (N_i, W_i)$ ist $N(P_i)$ definiert als N_i und $W(P_i)$ als W_i. Als Spezialfall wird auch der leere Pfad $\emptyset_P$ als Pfad bezeichnet.*

Beispiele für Pfade sind: ((Kante 3)); ((Fläche 2), (Kante 1)); ((Koordinate x)); $\emptyset_P$.

Zu einer Wissenseinheit gibt es Verarbeitungskomponenten, die die Menge der Hypothesen einer Wissenseinheit aktiv verändern können. Diese Komponente, die später als Agent definiert wird, ist implizit enthalten, wenn von einer Wissenseinheit gesprochen wird. Damit erhält die Wissenseinheit einen aktiven Charakter.

Zu jeder Hypothese gehört die Information, von wem sie erzeugt wurde (T und E), d.h. auf welcher Wissenseinheit sie z.B. durch Anwendung einer Regel oder z.B. durch externe Prozesse entstanden ist. Die Erzeugerinformation einer Hypothese ist wichtig, wenn sie auf andere Wissenseinheiten übertragen wird, da nur so eine Unterscheidung der Herkunft einer Hypothese möglich wird. Daneben sind für alle Wissenseinheiten prinzipiell nur Hypothesen bestimmter Typen von Wissenseinheiten interessant. Die Aufnahme von $T = Typ(E)$ als Teil der Definition einer Hypothese ist redundant. Sie ist dadurch begründet, daß eine Analogie zwischen einer Hypothese und einer elementaren Bedingung (s. Definition 5.7) bestehen soll.

Schreibweise: Eine Hypothese (T, E, P, N, W), mit $P = ((N_1, W_1), \ldots, (N_p, W_p))$ wird in den Beispielen in folgender Form geschrieben: $((T\,E)(N_1\,W_1)\ldots(N_p\,W_p)(N\,W))$. Falls $P = \emptyset_P$ ergibt sich die Form $((T\,E)\,(N\,W))$.

Beispiele für Hypothesen:

```
((Objekt-WE 6) (Höhe 100.0))
((Objekt-WE 6) (Typ Zylinder))
```

```
((Objekt-WE 6) (Kante 2) (Länge 20.0))
                Pfad
((Objekt-WE 6) (Kante 2) (Typ gekrümmt))
                Pfad
((Relations-WE 5) (Koordinate x) (Zielbereich [-10.0 10.0]))
                   Pfad
((Relations-WE 5) (Aktuelle-Relation (Oberhalb (Basisobjekt))))
((Relations-WE 7) (Aktuelle-Relation (Zwischen (Kante1 Kante2))))
```

Der wesentliche Unterschied einer Wissenseinheit zu einer Repräsentation wie z.B. den Frames von Minsky [Min75] ist der, daß in einer Wissenseinheit nicht a priori festgelegte Attribute existieren, die dann ausgefüllt werden müssen. Vielmehr ist eine Wissenseinheit eine Menge von Aussagen über möglicherweise strukturierte Attribute, deren Kardinalität jedoch im Laufe des Einsatzes unterschiedlich sein kann. Außerdem kann a priori nicht bestimmt werden, für welche Attribute tatsächlich eine Aussage getroffen bzw. benötigt wird. Diese Information entsteht erst sukzessive durch die Lernvorgänge.

Die Hypothesen werden in folgende Typen unterschieden, um einzelnen Hypothesen eine bestimmte Semantik zuordnen zu können. Auf die unterschiedliche Verarbeitung bei der Anwendung einer Produktion wird in Abschnitt 5.3.3 eingegangen.

- Statische Hypothesen: Die Aussage über ein bestimmtes Attribut wird einmal festgelegt und bleibt dann unverändert. Dies ist der Standardfall für den Typ einer Hypothese. Beispiel: `((Objekt-WE 6) (Kante 2) (Länge 20.0))`
- Veränderliche Hypothesen: Die Attributbelegung kann sich ändern, es bleibt aber die gleiche Hypothese, z.B. eine Aussage über die TCP-Position während der Ausführung einer Aktion. Beispiel: `((Sensor-WE 3) (Koordinate x) (TCP 100.0))`
- Anfragehypothesen: Sind für eine Produktion bereits mehrere Hypothesen bekannt, die für ihre Anwendung wichtig sind, einige fehlen aber, so kann eine Anfragehypothese erzeugt werden. Diese gibt an, daß es vermutlich wichtig wäre, für das in ihr angegebene Attribut eine Aussage zu erreichen. Der Wert des Anfrageattributs ist unbekannt und wird daher mit „?" gekennzeichnet. Beispiel: `((Plan-WE 5) (Höhe ?))`, `((Plan-WE 5) (Kante 2) (Länge ?))`
- Rollenhypothesen: Während der Lösung einer Aufgabe sind im allgemeinen Objekte der Umwelt wichtig für die Beschreibung einer Teilaufgabe oder eines Teilziels. Ein Objekt spielt eine bestimmte *Rolle* bei der Lösung, z.B. es ist das Füge- oder Basisobjekt, eine Ausrichtkante oder ähnliches. Diese Rolle wird dem Objekt in Form einer Rollenhypothese zugewiesen. Diese Hypothesenform ist nur für die planenden Komponenten des Systems relevant. Beispiel: `((Plan-WE 5) (Greifobjekt (Objekt-WE 2)))`, d.h. das Objekt, das durch Objekt-WE 2 beschrieben wird, ist in der Teilaufgabe, die durch Plan-WE 5 repräsentiert wird, das zu greifende Objekt.
- Nachrichtenhypothesen: Normalerweise werden Hypothesen nach einem festen Verfahren (s. Abschnitt 5.3) an andere Komponenten verteilt. Der Benutzer ist dabei

völlig homogen zu anderen Systemkomponenten eingebunden. Möchte er eine Information gezielt an eine bestimmte Komponente schicken, verwendet er dazu Nachrichtenhypothesen. Beispiel: ((`Benutzer-WE 1`) (`Message` *Nachricht*)). Der Inhalt von *Nachricht* ist eine Hypothese.

Die Wissenseinheiten sind über sogenannte Kanäle miteinander verbunden, über die sie miteinander kommunizieren können. Beispielsweise können Hypothesen, die auf einer Objekt-WE abgeleitet wurden, für eine Plan-WE interessant sein, die eine Operation bezüglich dieses Objektes plant. Die Wissenseinheiten zusammen mit den Kanälen definieren einen Graphen, der die Verbindungsstruktur der einzelnen Wissenseinheiten angibt.

Definition 5.4 (Kanal) *Eine gerichtete Verbindung $k = (WE_1, WE_2)$ zwischen zwei Wissenseinheiten WE_1 und WE_2 heißt* ***Kanal****. Über diese Verbindung können Hypothesen von der Wissenseinheit WE_1 auf WE_2 übertragen werden.*

Die Menge der Hypothesen aller Wissenseinheiten, die zu einem Zeitpunkt in einem System vorliegen, beschreibt das Wissen des Systems über den aktuellen Weltzustand und den internen Systemzustand. Zum aktuellen Weltzustand gehören z.B. eingenommene/gültige Relationen, Objektmerkmale und Aufgabenstellungen. Zum internen Systemzustand gehört z.B., ob für ein Teilobjekt schon eine Wissenseinheit erzeugt wurde, wie die Aufgabe bisher in Teilpläne zerlegt ist oder ob für ein Objekt schon eine Rollenhypothese existiert. Die Menge der Wissenseinheiten zusammen mit der auf ihnen definierten Verbindungsstruktur wird als Systemzustand bezeichnet.

Definition 5.5 (Systemzustand) *Die Menge $\mathcal{WE}$ aller Wissenseinheiten, die an einer Problemlösung beteiligt sind, zusammen mit der Menge $\mathcal{K}$ der zwischen ihnen definierten Kanäle heißt* ***Systemzustand*** *$Z = (\mathcal{WE}, \mathcal{K})$. Die* ***Kanäle einer Wissenseinheit*** *$WE \in \mathcal{WE}$ sind dann definiert als: $Kanäle(WE) := \{k \in \mathcal{K} \mid k = (WE, w), w \in \mathcal{WE}\}$. Die* ***verbundenen Wissenseinheiten*** *einer Wissenseinheit WE sind definiert als die Menge aller Wissenseinheiten, zu denen WE über einen Kanal Hypothesen weiterleiten kann, d.h. $VerbundeneWissenseinheiten(WE) := \{w \in \mathcal{WE} \mid \exists k \in \mathcal{K} : k = (WE, w)\}$.*

Für die Ausführung werden die Typen von Wissenseinheiten in statische und dynamische unterschieden. Während statische Wissenseinheiten nur bei der Systeminitialisierung initiiert werden, ist es möglich, während der Problemlösung bei Bedarf neue Wissenseinheiten eines dynamischen Typs zu erzeugen. Aus diesem Grund werden später für die dynamischen Typen von Wissenseinheiten Verbindungsstrukturen definiert, die die jeweilige Eingliederung in den aktuellen Systemzustand beschreiben.

Definition 5.6 (dynamische und statische Wissenseinheit) *Der Typ einer Wissenseinheit WE heißt* ***dynamisch****, wenn ein beliebiger Systemzustand mehr als eine Wissenseinheit dieses Typs enthalten kann. Andernfalls heißt sie* ***statisch****.*

Die Entscheidungen des Systems basieren auf Langzeitwissen, das über mehrere Problemlösungen hinweg Gültigkeit besitzt und durch interne und externe Lernvorgänge erweitert und korrigiert werden soll. Im folgenden werden die einzelnen Elemente des Langzeitwissens und die darauf basierenden Verarbeitungseinheiten definiert. Als Repräsentationsform, die soweit wie möglich durchgehend verwendet wird, werden Produktionsregeln

eingesetzt. Der Bedingungsteil einer Produktion bestehend aus einer Menge von elementaren Bedingungen stellt Anforderungen an bestimmte Hypothesen, die vorliegen müssen, damit die Produktion anwendbar ist. Der Aktionsteil führt zu einer gezielten Veränderung des Systemzustandes.

Definition 5.7 (Elementare Bedingung) *Eine **elementare Bedingung** ist - analog zu einer Hypothese - definiert als Quintupel (T, E, P, N, W). Dabei ist N wieder ein Attributname, T und W sind definiert wie bei einer Hypothese als der Typ der erzeugenden Wissenseinheit bzw. der Wert des Attributes N, T und W können aber zusätzlich noch die Ausprägung „beliebig" annehmen. P ist ein Pfad, in dem jede Ausprägung der Werte auch „beliebig" sein kann. E ist eine Bedingung bezüglich des Erzeugers. Diese ist entweder eine Variable, eine Rollenvariable oder „beliebig".*

Eine elementare Bedingung kann durch eine Hypothese erfüllt werden, wenn die Anforderungen, die in der Bedingung spezifiziert werden, für die Hypothese zutreffen. Die Schreibweise der Bedingungen in den Beispielen entspricht der oben eingeführten Schreibweise für Hypothesen. Beispiele für elementare Bedingungen ohne Verwendung von Variablen sind: `((Objekt-WE 'beliebig') (Objekttyp Zylinder))`, `((Objekt-WE 'beliebig') (Kante 2) (Länge [20,40]))`, `((Objekt-WE 'beliebig') (Kante 'beliebig') (Typ gekrümmt))`.

Innerhalb des Bedingungsteils einer Produktion (s. Definition 5.10) werden mehrere elementare Bedingungen spezifiziert, die alle gleichzeitig gültig sein müssen, damit der Bedingungsteil der Produktion als erfüllt gilt. Um innerhalb dieser Menge von elementaren Bedingungen auszudrücken, daß die erzeugende Wissenseinheit für bestimmte Hypothesen dieselbe sein muß, werden Variablen eingeführt, die bei der Überprüfung der Erfülltheit des Bedingungsteils einer Produktion konkreten Wissenseinheiten zugeordnet werden.

Definition 5.8 (Variable, Rollenvariable) *Die Menge der **Variablen** sei definiert als {Var:<name> | <name> ist eine Zeichenfolge}. Die Menge der **Rollenvariablen** sei definiert als {Rolle:<name> | <name> ist eine Zeichenfolge}.*

Beispiel: Eine Produktion besitzt im Bedingungsteil elementare Bedingungen an zwei Attribute von Objektwissenseinheiten. Die Aussagen über die Attribute $Attribut_1$ und $Attribut_2$ müssen dabei von derselben Wissenseinheit stammen, die durch die Variable `Var:` $Objekt_1$ repräsentiert wird.

$$\begin{aligned}\text{Bedingung} = {} & (\texttt{(Objekt-WE Var:}\,Objekt_1) \ \ldots (Attribut_1\ Wert_1)) \\ & \wedge(\texttt{(Objekt-WE Var:}\,Objekt_1) \ \ldots (Attribut_2\ Wert_2)) \\ & \wedge(\texttt{(Objekt-WE Var:}\,Objekt_2) \ \ldots (Attribut_3\ Wert_3))\end{aligned}$$

Die Zuordnung einer Variable zu einer bestimmten Wissenseinheit ist nur lokal für *eine* Produktion gültig. Sollen dagegen für mehrere Produktionen Hypothesen von derselben Wissenseinheit stammen, so werden Rollenvariablen eingesetzt. Dieser Fall tritt insbesondere im Zusammenspiel von Plan- und Objektagent auf. Im Laufe einer Aufgabe hat ein Objekt eine bestimmte Rolle, z.B. ist es Fügeobjekt oder Basisobjekt. Die Rolle einer

Wissenseinheit wird selbst wieder durch Produktionen zugewiesen und liegt als Hypothese auf einer der Wissenseinheiten vor. Damit kann auch die Zuordnung von Rollen durch die verwendeten Lernverfahren korrigiert bzw. erweitert werden.

Beispiel: Die Aussage über $Attribut_1$ im Bedingungsteil der Produktion P_1 und über $Attribut_2$ im Bedingungsteil der Produktion P_2 müssen von derselben Wissenseinheit stammen und zwar der, der die Rolle des Fügeobjektes zugeordnet wurde.

Bedingung von Produktion P_1 = ((...`Rolle:Fügeobjekt`) ... ($Attribut_1$ $Wert_1$))
Bedingung von Produktion P_2 = ((...`Rolle:Fügeobjekt`) ... ($Attribut_2$ $Wert_2$))

Während der Bearbeitung einer Aufgabe muß der Systemzustand durch entsprechende Aktionen verändert werden, um der Lösung näher zu kommen. Eine Aktion kann dabei entweder eine neue Hypothese oder eine neue Wissenseinheit erzeugen. Eine neue Wissenseinheit stellt z.B. ein neu zu erreichendes Teilziel oder Informationen über ein neu wahrgenommenes Objekt dar.

Definition 5.9 (Aktion) *Eine **Aktion** A stellt einen Übergang von einem Systemzustand auf einen neuen Systemzustand dar. Dieser besteht entweder darin, daß eine neue Hypothese abgeleitet und zu einer Wissenseinheit hinzugefügt wird, oder daß eine neue Instanz einer dynamischen Wissenseinheit erzeugt und in das System eingegliedert wird.*

Mögliche Varianten für Aktionen sind:

- Erzeugen einer neuen Hypothese H, die fest in der Aktion codiert ist, $WE := WE \cup \{H\}$. Beispiele: ((`Bester-Griff Gesamtobjekt`)), ((`Rolle Rolle:Greifobjekt`)).
- Erzeugen einer neuen Hypothese durch Anwendung einer Funktion f, $WE := WE \cup \{f(WE)\}$. In den Beispielen wird der zu berechnende Attributwert durch „?“ dargestellt. Beispiele: ((`Koord x`) (`Im-Bereich ?`)) berechnet durch `Bereichsüberprüfung`, ((`Koord x`) (`Änderung-Relativ ?`)) berechnet durch `Berechne-Koord-Relativ`.
- Erzeugen einer Wissenseinheit eines bestimmten Typs: Sei $SZ = (\mathcal{WE}, \mathcal{K})$, dann $SZ' = (\mathcal{WE} \cup \{W\}, \mathcal{K} \cup \mathcal{K}_W)$, dabei ist W eine Wissenseinheit des gewünschten Typs und $\mathcal{K}_W$ die Verbindungsstruktur der Wissenseinheit, d.h. die Menge der neu aufzubauenden Kanäle. Beispiele: Neue Relations-WE `Oberhalb` mit Bezug zu (`Rolle:Greifobjekt`), Neue Objekt-WE `Ausrichtkante1` mit Bezug zu (`Rolle:Greifobjekt`). Dabei sind `Oberhalb` und `Ausrichtkante1` die Namen der jeweils neu zu erzeugenden Wissenseinheiten und (`Rolle:Greifobjekt`) die Liste der Wissenseinheiten, zu denen zusätzlich ein Kanal zur Kommunikation aufgebaut werden soll.
- Übertragen einer Nachricht: Sei H eine Nachricht für WE', dann $WE' = WE' \cup \{H\}$
- Ausführen eines Seiteneffektes: Es wird nicht explizit eine neue Hypothese oder Wissenseinheit erzeugt, sondern eine Funktion angewendet, die als Seiteneffekt auch die Erzeugung einer neuen Hypothese bewirken kann. Dies wird insbesondere eingesetzt, um die agentenspezifischen Komponenten in den normalen Ablauf einzubinden.

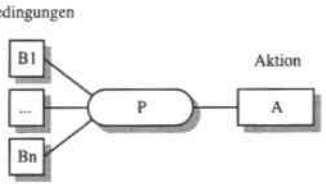

Abbildung 5.2: Graphische Darstellung einer Produktion

Aus der Kombination einer Menge von elementaren Bedingungen und einer Aktion kann nun eine Produktion gebildet werden. Diese Produktionen werden zur Repräsentation des Wissens in agentenunabhängiger Form eingesetzt. Die Begriffe Produktion, Regel und Produktionsregel werden im folgenden synonym verwendet.

Definition 5.10 (Produktion) *Eine* ***Produktion*** *ist ein Tupel der Form* (B, A)*. Dabei ist* B *eine Menge von elementaren Bedingungen und* A *eine Aktion. Sind in einer Situation alle Bedingungen einer Produktion erfüllt, so kann diese Produktion angewendet werden, d.h. ihre Aktion kann ausgeführt werden. Jeder Produktion ist eine* ***Stärke*** *zugeordnet,* $Stärke(P)$*, die die globale Relevanz und Sicherheit der Produktion widerspiegelt.*

Schreibweise: Sei $P = (\{B_1, \ldots, B_n\}, A)$, dann wird die Produktion geschrieben als WENN $B_1 \wedge \ldots \wedge B_n$ DANN A. In den nachfolgenden Graphiken wird P wie in Abbildung 5.2 gezeigt dargestellt.

Beispiel für eine Produktion:

```
WENN
  ((Relations-WE (Var:R1)) (Aktuelle-Relation (Oberhalb (Greifobjekt))))
  ∧ ((Relations-WE (Var:R2)) (Aktuelle-Relation (Greifer-Offen)))
  ∧ ((Plan-WE 'beliebig') (Planname Greife-Objekt))
  ∧ ((Objekt-WE (Rolle:Greifobjekt)) (Bester-Griff Gesamtobjekt))
DANN
  Neue Relation Direkt-Oberhalb mit Bezug zu (Rolle:Greifobjekt)
```

Ein Agent als elementare Verarbeitungseinheit innerhalb des Systems basiert seine Ableitungsschritte auf Wissen, das als eine Menge von Produktionen dargestellt ist. Zusätzlich kann ein Agent Wissen besitzen, das in der für ihn spezifischen Wissensrepräsentationsform dargestellt ist. Die agentenunabhängige Ableitung und das Lernen neuen Wissens ist für alle Agenten einheitlich definiert und basiert auf den in den folgenden Abschnitten definierten Verfahren. Die Verarbeitung des Wissens in agentenspezifischer Form erfordert spezielle Inferenz- und Lernmechanismen.

Definition 5.11 (Agent) *Ein Quadrupel* (WB, WB_A, IV_A, LV_A) *heißt* ***Agent****. Dabei ist* WB *eine Menge von Produktionen, die den Teil des Wissens des Agenten angibt, der in*

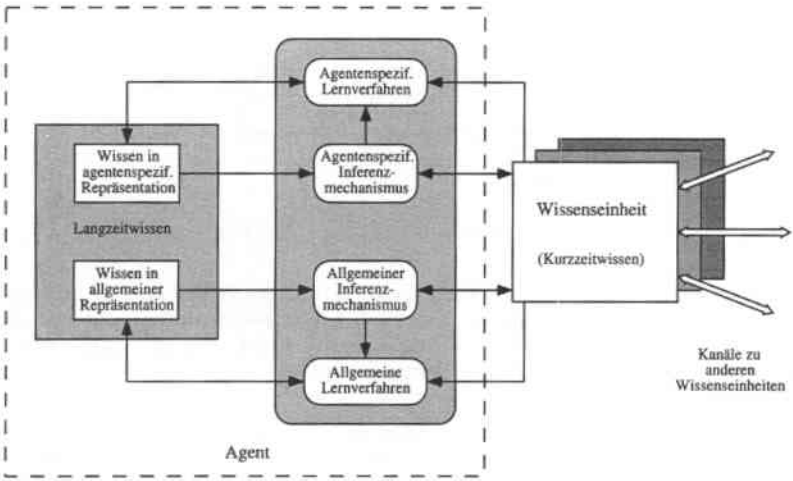

Abbildung 5.3: Aufbau eines Agenten mit Wissenseinheiten

allgemeiner Repräsentation dargestellt werden kann, WB_A ist der Teil des Wissens des Agenten der in agentenspezifischer Repräsentation dargestellt wird. IV_A und LV_A sind das agentenspezifische Inferenz- bzw. Lernverfahren, das der Agent speziell zur Interpretation und zum Aufbau von WB_A besitzt. Falls ein Agent kein Wissen in agentenspezifischer Repräsentation benötigt, können WB_A, IV_A und LV_A auch leer sein. Die Menge aller Agenten sei $\mathcal{AG}$.

Damit ergibt sich ein Baustein der Architektur wie in Abbildung 5.3 gezeigt. Er besteht aus einer oder mehreren Wissenseinheiten eines Typs und dem zugehörigen Agenten, dessen Aktionen auf Langzeitwissen basieren. Die vertikale Unterteilung innerhalb des Agenten entspricht der Unterscheidung von Langzeitwissen, Inferenz- und Lernmechanismus in einen agentenunabhängigen und einen agentenabhängigen Teil.

Definition 5.12 (Zu einer Wissenseinheit gehörige Agent) *Die Funktion $A : \mathcal{WE} \rightarrow \mathcal{AG}$ weist jeder Wissenseinheit den zu ihrem Typ gehörenden Agenten zu.*

Konvention: Wenn von einer Wissenseinheit die Rede ist, ist dabei häufig der zugehörige Agent mit einbezogen. Sei eine Wissenseinheit WE gegeben mit $AG = A(WE)$ und $AG = (WB, WB_A, IV_A, LV_A)$, dann sei $Prods(WE) = WB$ die Menge der Produktionen des zur Wissenseinheit gehörenden Agenten.

Im folgenden werden die Typen von Wissenseinheiten definiert, die für den Anwendungsbereich der Robotik notwendig sind. Die sich daraus ergebende Architektur ist in Abbildung 5.4 dargestellt. Als statische Wissenseinheiten liegen die Aktor-, Sensor-, Benutzer- und

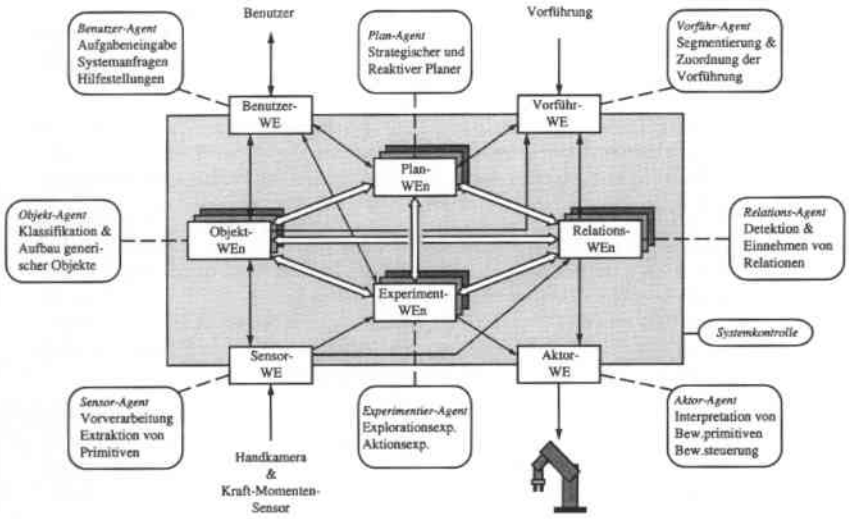

Abbildung 5.4: Systemarchitektur

Vorführwissenseinheit vor. Dynamische Wissenseinheiten sind Objekt-, Plan-, Relations- und Experimentierwissenseinheiten. Aufgrund der vorliegenden Definitionen ist es leicht möglich, weitere Komponenten in das System zu integrieren oder diese Architektur für andere ähnliche Aufgabenstellungen einzusetzen.

1. Die *Aktorwissenseinheit* ist zuständig für die tatsächliche Ansteuerung des realen Roboters. Sie erhält Hypothesen von Relationswissenseinheiten über durchzuführende Elementarbewegungen und setzt diese in entsprechende Primitive der Robotersteuerung um.

2. Die *Sensorwissenseinheit* stellt die Kopplung externer oder interner Sensorik an das System dar. Die angekoppelten Sensoren liefern – eventuell vorverarbeitete – Informationen an die Sensorwissenseinheit, die sie in entsprechende Hypothesen umsetzt.

3. Die Schnittstelle zur normalen Kommunikation zwischen Benutzer und System bildet die *Benutzerwissenseinheit*. Über sie werden Anfragehypothesen an den Benutzer geleitet, die er durch entsprechende Hypothesen beantworten kann. Daneben ermöglicht die Benutzerwissenseinheit die Formulierung von Aufgaben an das System – ebenfalls in Hypothesenform.

4. Die *Vorführwissenseinheit* ist für den Abgleich einer aufgenommenen Benutzervorführung mit den Aktionen des Systems zuständig und liefert, wenn nötig, entsprechende Informationen über aufgetretene Differenzen.

5. Sämtliche planenden Aktivitäten des Systems laufen auf *Planwissenseinheiten* ab, die untereinander zu Hierarchien angeordnet werden können. Durch die Anwendung des Wissens des Planagenten muß sowohl ein reaktives Verhalten als auch die strategische Verfolgung der Aufgabenziele erreicht werden.

6. Die *Objektwissenseinheiten* enthalten sämtliche Informationen über Objekte, die durch Sensorik oder Anwendung von Wissen bekannt sind. Zur Klassifikation werden generische Objektmodelle eingesetzt. Ein Objekt kann mit anderen Objektwissenseinheiten verbunden sein, die ein Teilobjekt von ihm darstellen.

7. Die Abläufe auf den Planwissenseinheiten führen zu Teilzielen, die durch *Relationswissenseinheiten* beschrieben werden. Jedes Teilziel wird durch eine Relation zu Objektwissenseinheiten beschrieben, die punktuell oder über einen bestimmten Zeitraum gültig sein muß. Durch Übertragung von Hypothesen an die Aktorwissenseinheit und den Erhalt von Hypothesen von der Sensor- oder den Objektwissenseinheiten wird versucht, die geforderte Relation einzunehmen.

8. Die *Experimentierwissenseinheiten* repräsentieren Experimente, die das System durchführen soll, um bestimmte Hypothesen über Objekteigenschaften erzeugen und verifizieren zu können. Dazu gehören Explorationsexperimente zur Ableitung von statischen Eigenschaften und Aktionsexperimente zur Bestimmung des dynamischen Verhaltens des Objektes bei einer Aktionsart.

Neben der Verwendung von Produktionen zur Repräsentation von Langzeitwissen können auch andere Repräsentationsformen zum Einsatz kommen. Die Repräsentation durch Produktionen wird als agentenunabhängig bezeichnet, weil sie für alle Komponenten des Systems in gleicher Weise eingesetzt werden kann. Daneben gibt es agentenspezifische Repräsentationsformen, z.B. die Objekthierarchie und Darstellung von generischen Objektmodellen beim Objektagenten (s. Kapitel 7), die Repräsentation von Gültigkeitsbereichen beim Relationsagenten (s. Abschnitt 5.5) und die Repräsentation von Experimentergebnissen beim Experimentieragenten (s. Kapitel 8).

Komplexe Funktionen, die zur funktionalen Bestimmung neuer Hypothesen verwendet werden, können ebenfalls zum agentenspezifischen Wissen gezählt werden. In den angegebenen Kapiteln werden ebenfalls die spezifisch entwickelten Lern- und Inferenzmethoden untersucht.

Für die kommende Beschreibung der Architektur als Ganzes können diese spezifischen Teilkomponenten zunächst so betrachtet werden, als würden sie die notwendigen Hypothesen auf die entsprechende Wissenseinheit liefern. Mit welchen Verfahren und auf welcher Repräsentation das geschieht und wie dabei ein Lernvorgang aussieht spielt hier zunächst keine Rolle.

5.3 Dynamischer Ablauf

Aufbauend auf den Definitionen der einzelnen Komponenten der Systemarchitektur wird in diesem Abschnitt das dynamische Verhalten des Systems erläutert, d.h. der Ablauf

während einer Problemlösung. Dazu gehören die Definition der Anwendbarkeit einer Produktion, die Auswahl einer geeigneten anwendbaren Produktion, die Ausführung der Aktion und die Verarbeitung des durch die Aktion erreichten Ergebnisses.

Die Produktionen eines Agenten sollen auf die Hypothesen einer zugehörigen Wissenseinheit angewendet werden, um neue Hypothesen oder Wissenseinheiten abzuleiten. Dazu muß definiert werden, wann eine Produktion auf eine Menge von Hypothesen anwendbar ist. Die Definition baut auf der Erfüllung einer Teilbedingung und dem Enthaltensein von Pfad und Wert einer Hypothese in einer Bedingung auf.

Definition 5.13 (Enthaltensein von Attributpfaden) *Ein Attributpfad* $Q = (Q_1, \ldots, Q_n)$ ***enthält*** *einen Attributpfad* $R = (R_1, \ldots, R_m)$, $R \subseteq_P Q$, *wenn 1.* $n = m$ *und 2.* $\forall i = 1, \ldots, n : N(Q_i) = N(R_i) \wedge ((W(Q_i) = \text{'beliebig'}) \vee (W(Q_i) = W(R_i)))$. *Für leere Pfade gilt:* $\forall Pfade\, Q : \emptyset_P \subseteq_P Q$, $\exists Pfad\, Q \neq \emptyset_P : Q \subseteq_P \emptyset_P$

Beispiele:

P_1 = `(Teilobjekt` O_1`) (Kante` K_2`)`
$P_1 \subseteq_P$ `(Teilobjekt` O_1`) (Kante 'beliebig')`, $P_1 \subseteq P_1$,
$P_1 \not\subseteq_P$ `(Teilobjekt` O_1`) (Kante` K_3`)`

Definition 5.14 (Enthaltensein von Attributwerten) *Eine Angabe für einen Attributwert* B ***enthält*** *die Angabe für einen Attributwert* H, $H \subseteq_W B$, *genau dann, wenn einer der folgenden Fälle zutrifft:*

1. B *und* H *sind einfache numerische Werte und* $B = H$
2. B *und* H *sind einfache symbolische Werte und* $B = H$ *oder es existiert eine Taxonomie für die Menge aller möglichen Werte und* H *ist in dieser Taxonomie eine Teilmenge von* B.
3. $B = [b_1, b_2]$ *ist ein Intervall,* H *ist einfacher numerischer Wert und* $b_1 \leq H \leq b_2$
4. $B = \{b_1, \ldots, b_n\}$ *ist eine Menge,* H *ist einfacher symbolischer Wert und für ein* $b_i \in B$ *gilt Fall 2*
5. $B = [b_1, b_2]$ *und* $H = [h_1, h_2]$ *sind Intervalle und* $b_1 \leq h_1 \leq h_2 \leq b_2$
6. $B = (b_1, \ldots, b_n)$ *und* $H = (h_1, \ldots, h_n)$ *sind Aufzählungen und es gilt* $\forall i : 1 \leq i \leq n$: b_i *enthält* h_i

Beispiele:

$10 \subseteq_W 10, 10 \subseteq_W [0, 20], 10 \subseteq_W \{0, 10, 20\}$
$[10, 20] \subseteq_W [0, 20], (10, 20, [20, 30]) \subseteq_W (10, \{10, 20, 30\}, [0, 40])$

Aufbauend auf diesen Definitionen kann nun definiert werden, wann eine Hypothese eine elementare Bedingung erfüllt. Dazu muß der Typ der Erzeugerwissenseinheit von Hypothese und Elementarbedingung übereinstimmen und, falls eine Variable oder Rollenvariable verwendet wird, muß diese zum Erzeuger der Hypothese zugeordnet sein. Der Attributpfad muß enthalten sein, der Attributname muß gleich sein und der Attributwert muß enthalten sein.

Definition 5.15 (Zuordnung für Variablen) *Eine Abbildung z von der Menge der Variablen einer Produktion P, $Var(P)$, auf eine Menge von Wissenseinheiten $\mathcal{WE}$ heißt* ***Zuordnung.***

Definition 5.16 (erfüllt) *Eine Hypothese $H = (T_H, E_H, P_H, N_H, W_H)$* ***erfüllt*** *eine elementare Bedingung $B = (T_B, E_B, P_B, N_B, W_B)$ unter Berücksichtigung einer Zuordnung z und der Menge von Rollenhypothesen $\mathcal{R}$ wenn folgende Aussagen gültig sind:*

1. *Die Erzeugerbeschreibung von H ist in der Erzeugerbeschreibung von B enthalten, d.h. $T_H = T_B$ und $E_B =$ 'beliebig', oder E_B ist Variable v, und $z(v) = E_H$, d.h. die Hypothese wurde von der Wissenseinheit erzeugt, die der Variable in der Bedingung zugeordnet wurde oder E_B ist Rollenvariable r und $\exists$ Rollenhypothese Rolle $\in \mathcal{R}$: $Rolle(E_H) = r$.*
2. *Der Attributpfad von H ist im Attributpfad von B enthalten: $P_H \subseteq_P P_B$.*
3. *Der Attributname von H ist identisch zum Attributnamen von B: $N_H = N_B$.*
4. *Der Attributwert von H ist im Attributwert von B enthalten: $W_H \subseteq_W W_B$.*

Definition 5.17 (anwendbar) *Eine Produktion $P = (B, A)$ heißt* ***anwendbar*** *auf eine Wissenseinheit WE im Systemzustand $S = (\mathcal{WE}, \mathcal{K})$, wenn entweder $B = \{\}$ oder $\exists$ injektive Zuordnung $z : Var(P) \rightarrow \mathcal{WE} : \forall b \in B : b$ ist erfüllt durch Hypothesen in $\mathcal{WE}$ unter Berücksichtigung der Zuordnung z und der Menge der Rollenhypothesen $\mathcal{R} \subseteq \mathcal{WE}$.*

Durch die Forderung der Injektivität wird gewährleistet, daß unterschiedliche Variablen auf unterschiedliche Wissenseinheiten abgebildet werden. Die Definition der Anwendbarkeit einer Produktion kann konstruktiv durch einen Algorithmus formuliert werden.

Beispiel:

Sei eine Produktion P gegeben als:

```
WENN
  ((Objekt-WE (Var:A)) (Breite [10,100]))
  ∧ ((Objekt-WE (Var:A)) (Länge [20,50]))
  ∧ ((Plan-WE (Var:B)) (Planname Greife-Objekt))
DANN
  Neue Hypothese (Bester-Griff Gesamtobjekt)
```

und sei eine Wissenseinheit WE mit folgenden Hypothesen gegeben:

```
{ ((Objekt-WE 6) (Breite 120)), ((Objekt-WE 7) (Breite 50)),
  ((Objekt-WE 7) (Länge 50)), ((Plan-WE 5) (Planname Greife-Objekt)) },
```

dann ist P mit der Zuordnung $z = \{$Var:A $\rightarrow$ Objekt-WE 7, Var:B $\rightarrow$ Plan-WE 5$\}$ auf WE anwendbar.

Nachdem nun festgelegt ist, wann eine Produktion eines Agenten überhaupt prinzipiell anwendbar ist, muß ein Verfahren definiert werden, welche Produktion von normalerweise

mehreren anwendbaren tatsächlich ausgewählt werden soll. Dabei ist zu beachten, daß in diesem System das Regelwissen im Gegensatz zu klassischen Planungssystemen nicht in einem Schritt a priori vor der eigentlichen Ausführung eingesetzt wird. In klassischen Planungssystemen ist das Planungsproblem im wesentlichen ein Suchproblem, bei dem in einem vorwärts- oder rückwärtsverketteten Suchraum ein Weg zum gewünschten Zielzustand gefunden werden soll. In der hier beschriebenen Architektur muß in jedem Schritt, basierend auf der aktuellen Situation, die nächste Produktion ausgewählt und angewendet werden. Das schließt allerdings eine gleichzeitige Vorausplanung nicht aus. Die Auswahl der nächsten Produktion muß eine Reaktion auf aktuelle Sensorwerte ermöglichen, gleichzeitig aber ein strategisches Verhalten zur Erreichung des Zieles realisieren.

Als prinzipielle Möglichkeiten ergeben sich a) ein einstufiges Vorgehen, bei dem eine Produktion in der Menge aller anwendbaren Produktionen aller Wissenseinheiten bestimmt wird, oder b) ein zweistufiges Vorgehen, bei dem in jedem Zyklus zwei Entscheidungen getroffen werden,

1. Welche Wissenseinheit soll als nächste aktiviert werden?
2. Welche Produktion dieser Wissenseinheit soll als nächste angewendet werden?

Idealerweise sollte die Produktion angewendet werden, die insgesamt gesehen für das Erreichen einer Lösung am günstigsten ist. Natürlich ist diese Anforderung nicht exakt formalisierbar, wenn von einer On-line-Entscheidung ausgegangen wird. Aus diesem Grund müssen entsprechende Heuristiken definiert werden, die eine möglichst gute Bewertung in diese Richtung liefern.

Probleme, die beachtet werden müssen, sind zum einen, daß alternative Problemlösungen im System vorliegen können, da eine Entscheidung für eine Alternative opportunistisch zum spätest möglichen Zeitpunkt getroffen werden soll. Dabei ist aber sicherzustellen, daß nicht beide Alternativen so weit verfolgt werden, daß sie sich gegenseitig stören, z.B. durch die Ausführung einer Aktion. Dies muß durch eine entsprechend geeignete Auswahl einer Wissenseinheit geschehen. Zum anderen darf eine Produktion erst dann wieder angewendet werden, wenn sich die Hypothesen, auf denen die Vorbedingungen basieren, geändert haben. Für die Aufgabenlösung trägt eine Produktion im allgemeinen sonst nichts Neues bei. Allerdings muß aufgrund der geforderten Reaktivität eine Regel erneut anwendbar werden, wenn z.B. durch unerwartete Ereignisse eine notwendige bereits eingenommene Relation plötzlich nicht mehr gültig ist. In dieser Arbeit wird ein zweistufiges Verfahren verwendet.

5.3.1 Auswahl einer Wissenseinheit

Mögliche Definitionen für die Auswahl einer Wissenseinheit, die im aktuellen Systemzustand als nächste aktiviert werden soll, sind:

- Wähle die Wissenseinheit, auf der mit einer Produktion die Hypothese größter Stärke abgeleitet/erreicht werden kann. Dies führt im Grunde zu dem oben angegebenen einstufigen Verfahren. Nachteil: Sehr aufwendig, da alle Produktionen aller Wissenseinheiten unter Berücksichtigung aller möglichen Zuordnungen getestet werden müssen.

- Vereinfachung über die Stärke einer Hypothese: Wähle die Wissenseinheit als nächste aus, die die Hypothese maximaler Stärke besitzt und auf der eine Produktion anwendbar ist. Nachteile: 1. Es ist nicht sicher, daß auch eine Produktion angewendet werden kann, die tatsächlich die stärkste Hypothese berücksichtigt. 2. Durch das Verteilen von Hypothesen im System kann es sein, daß die stärkste Hypothese auf mehreren Wissenseinheiten vorliegt und dadurch keine Entscheidung möglich ist, 3. Eine überlappende und dabei störende Aktivierung zweier Plan- oder Relationswissenseinheiten kann durch dieses Vorgehen nicht verhindert werden, da nicht sichergestellt ist, daß eine Wissenseinheit zunächst aktiv bleibt, solange sie neue Informationen ableiten kann.

- Aktivierungspotential: Eine dritte Variante der Auswahl ist es, jeder Wissenseinheit bei der Erzeugung eine Stärke in Abhängigkeit von der sie erzeugenden Produktion und den bei der Anwendung verwendeten Hypothesen zu definieren. Diese Lösungsmöglichkeit wird im folgenden durch das Aktivierungspotential einer Wissenseinheit verfolgt.

Definition 5.18 (Aktivierungspotential einer Wissenseinheit) *Das **Aktivierungspotential einer Wissenseinheit WE** ist definiert als*

$$AP(WE) = \frac{\sum_{i=1}^{n} Stärke(H_i)}{n} \cdot (1 + \frac{Stärke(Prod)}{\max_{p \in Prods(WE)} Stärke(p)})$$

Dabei ist Prod die Produktion, die auf der Basis der Hypothesen $H_1, \ldots, H_n$ die neue Wissenseinheit erzeugt. Statische Wissenseinheiten erhalten eine feste Stärke als Aktivierungspotential zugewiesen.

Der erste Faktor entspricht der durchschnittlichen Stärke der erzeugenden Hypothesen, der zweite Faktor gibt die relative Wichtigkeit der erzeugenden Produktion an. Durch diese Definition wird die einmal ausgewählte Teilplan- oder Relationswissenseinheit mit höchster Stärke bis zum Ende abgearbeitet - vorausgesetzt, es sind immer Produktionen anwendbar, bevor eine Alternative ausgewählt wird. Eine weitergehende Definition des Aktivierungspotentials einer Wissenseinheit wird in [Tu94] untersucht. In die Definition gehen dabei die Aktualität, die Aktivität und wie oben die Stärke einer Wissenseinheit ein.

5.3.2 Auswahl einer Produktion

Auch bei der Auswahl einer Produktion auf der ausgewählten Wissenseinheit gibt es wieder mehrere Möglichkeiten, eine Bewertung der Güte einer Produktion vorzunehmen:

- Definition über die Stärke einer Produktion, d.h. Einführung eines statischen Maßes für eine Produktion. Dieses wird entweder a priori festgelegt oder auch bei der

Ausführung gelernt – angewandte Produktionen, die nachfolgend nicht zu fehlerhaften Bewegungen führen, erhalten eine höhere Bewertung. Nachteil: 1. Eine Produktion, die häufiger benötigt und daher angewandt wird, erhält kontinuierlich eine höhere Stärke, auch wenn sie relativ unwichtig ist. 2. Bei zwei Produktionen mit gleichen Bedingungsteilen, die anfangs ungefähr gleiche Stärke haben, wird nachfolgend immer die angewandt, die zu Beginn stärker war.

- Definition eines geeigneten Aktivierungspotentials: Dabei muß der dynamische Ablauf eingehen, d.h. die Wichtigkeit einer Produktion muß von dem aktuellen Zustand innerhalb der Aufgabenlösung abhängen. Darauf basiert die Flexibilität des Systems und das inkrementelle und reaktive Vorgehen.

Das Aktivierungspotential einer Produktion setzt sich zusammen aus einem fixen Anteil, der die (globale) Stärke einer Produktion unabhängig von einem konkreten Zustand innerhalb der Aufgabenlösung angibt, und einem variablen Anteil, der die lokale Stärke der Produktion abhängig vom Zustand der Aufgabenlösung bewertet.

Definition 5.19 (Aktivierungspotential einer Produktion) *Das **Aktivierungspotential einer Produktion** $P = (\{B_1, \ldots, B_n\}, A)$ für eine Menge von erfüllenden Hypothesen $\{H_1, \ldots, H_n\}$ ist definiert als*

$$AP(P) = Stärke(P) \cdot Lokale\ Stärke(P)$$

mit

$$Lokale\ Stärke(P) = \frac{\sum_{i=1}^{n} (S(H_i) \cdot Aktuell(B_i, H_i))}{\sum_{i=1}^{n} S(H_i)} \leq 1$$

Dabei ist $S(H_i)$ die Stärke der Hypothese H_i, die die i-te Teilbedingung B_i erfüllt. Der Faktor Aktuell gibt an, ob die Produktion mit genau der Hypothese bereits angewendet wurde, d.h.

$$Aktuell(B_i, H_i) := \begin{cases} 1, & \textit{falls die Produktion } P \textit{ nicht bereits unter Verwendung von } H_i \\ & \textit{für } B_i \textit{ angewendet werden konnte.} \\ 0, & \textit{sonst} \end{cases}$$

Durch die obige Definition der lokalen Stärke einer Produktion ist sie auf den Bereich $[0, 1]$ eingeschränkt. Ist keine Bedingung einer Produktion erfüllbar oder alle Hypothesen waren bereits existent, dann ist die lokale Stärke und damit auch das Aktivierungspotential gleich Null. Mit jeder neu hinzukommenden Hypothese, die den Bedingungsteil erfüllt, steigt die lokale Stärke und damit das Aktivierungspotential. Wird die Produktion angewendet, fällt das Aktivierungspotential wieder auf Null und der Prozeß beginnt erneut. Damit werden die Produktionen bevorzugt, die möglichst aktuelle Vorbedingungen und die höchste Stärke besitzen.

5.3.3 Anwendung des Agentenwissens

Im folgenden wird nun der gesamte Auswahlprozeß einer geeigneten Produktion und die Verarbeitung des Ergebnisses untersucht.

Definition 5.20 (zulässiges Resultat) *Ein Resultat, d.h. eine neue Hypothese oder eine neue Wissenseinheit heißt* ***zulässig****, wenn keine dem Resultat entsprechende Hypothese bzw. Wissenseinheit im System existiert, die auf der Basis der gleichen Ausgangshypothesen und der gleichen Produktion abgeleitet wurde.*

Über die Definition eines zulässigen Resultates wird verhindert, daß dieselbe Produktion mehrfach hintereinander angewendet wird, ohne daß sich eine Änderung in ihren Vorbedingungen ergeben hat. Die Möglichkeit, daß das auftritt, wird durch die Definition des Aktivierungspotentials ohnehin sehr stark eingeschränkt.

Algorithmus 5.1 (Aktivieren einer Wissenseinheit)

Eingabe: · Eine Wissenseinheit $WE = \{H_1, \ldots, H_m\}$, auf der eine Produktion angewendet werden soll

Ausgabe: · Als *Resultat* wird das Ergebnis der Ausführung der Aktion der Produktion mit dem höchsten Aktivierungspotential zurückgeliefert, das gleichzeitig auch zulässig ist

funct *Aktiviere*(*WE*)
  Resultat := *nil*
  for $P \in$ *Prods*(*WE*) **do**
    Bestimme maximales Aktivierungspotential für P für alle Zuordnungen von Variablen zu Wissenseinheiten und für alle Möglichkeiten der Zuordnung von Bedingungen zu erfüllten Hypothesen
    for $P = (B, A) \in$ *Prods*(*WE*) sortiert nach Aktivierungspotential **do**
      if P ist anwendbar auf *WE* **then**
        Resultat := Führe A aus unter Berücksichtigung der Zuordnung und der verwendeten Hypothesen
      if *Resultat* ist zulässig **then exit** Schleife über *Prods*(*WE*)
  return (*Resultat*)

Eine deutliche Effizienzsteigerung wird dadurch erreicht, daß das Aktivierungspotential nicht, wie hier vereinfacht angegeben, in jedem Zyklus für alle Produktionen berechnet wird, sondern daß es hypothesengesteuert bei der Hinzunahme einer neuen Hypothese zu einer Wissenseinheit aktualisiert wird.

Nachdem mit den oben erläuterten Methoden entschieden ist, welche Produktion im aktuellen Systemzustand angewendet werden soll, muß nun untersucht werden, welche Auswirkungen die unterschiedlichen Typen von Aktionen auf den Systemzustand haben. Eine Aktion kann gemäß Definition 5.9 entweder eine neue Hypothese direkt oder durch Funktionsanwendung erzeugen, eine neue dynamische Wissenseinheit in das System einbinden oder eine Nachricht übermitteln.

Erzeugen neuer Hypothesen

Für den Fall, daß eine Hypothese durch eine Funktion bestimmt werden muß, wird diese unter Berücksichtigung der zugeordneten Hypothese angewendet, und dadurch der Attributwert für die Hypothese bestimmt. Sowohl im Fall der direkten als auch der funktionsgestützten Berechnung des Attributwertes wird die neue Hypothese zur Wissenseinheit hinzugenommen und anschließend verteilt. Die Stärke der neuen Hypothese ist definiert als $S(H_{neu}) := Stärke(P) \cdot \frac{\sum_{i=1}^{n} S(H_i)}{n}$, d.h. sie ist abhängig von der durchschnittlichen Stärke der zugrundeliegenden Hypothesen und der Stärke der angewendeten Produktion. Auf die unterschiedliche Semantik bei der Bearbeitung der verschiedenen Hypothesen- und Aktionstypen wird hier nicht eingegangen. Es werden nur die prinzipiellen Fälle des Erzeugens einer Hypothese und einer Wissenseinheit genauer behandelt.

In Kapitel 5.2 wurde in Form der Kanäle eine Kommunikationsstruktur zwischen den einzelnen Wissenseinheiten definiert, über die Hypothesen von einer Wissenseinheit zu einer anderen gelangen können. Nach der Hinzunahme der neuen Hypothese zu der betrachteten Wissenseinheit wird eine Weiterverteilung an benachbarte Wissenseinheiten vorgenommen. Würde eine neue Hypothese wahllos an alle verbundenen Wissenseinheiten weitergegeben, wäre die Fokussierung des Wissens, die durch die Wissenseinheiten erreicht werden soll, verloren und es wäre ebenso möglich, eine globale Datenbasis zu verwenden. Tatsächlich können aber Einschränkungen formuliert werden, wann eine Hypothese für einen bestimmten Typ von Wissenseinheit prinzipiell interessant ist. Dazu wird der Weg der Übertragung einer Hypothese betrachtet.

Definition 5.21 (Übertragungsweg) *Der Übertragungsweg $W(H)$ einer Hypothese $H \in$ WE gibt die Liste der Wissenseinheiten an, über die sie von der erzeugenden Wissenseinheit aus auf WE übertragen wurde.*

Für jeden Typ von Wissenseinheit ist eine Menge von Übertragungsbedingungen definiert. Diese geben an, welche Hypothesen eventuell für die Wissenseinheit interessant sind. Tabelle 5.1 gibt eine Übersicht über die im System definierten Übertragungsbedingungen.

Definition 5.22 (Übertragungsbedingung) *Ein Prädikat $\ddot{U}B$: Typen von Wissenseinheiten $\times$ Typen von Wissenseinheiten $\rightarrow \{wahr, falsch\}$ heißt **Übertragungsbedingung** für einen Wissenseinheitstyp T. Eine Hypothese H mit dem Übertragungsweg $W(H)$ **erfüllt** dann die Übertragungsbedingung $\ddot{U}B$, wenn $\ddot{U}B(W(H), T) = wahr$.*

Die Definition der Übertragungsbedingung geht ein in die Formulierung, wann eine Hypothese für eine Wissenseinheit interessant ist. Um die unnötige Ausbreitung von unwichtigen Hypothesen zu ermöglichen, wird neben der Erfüllung einer Übertragungsbedingung auch gefordert, daß die Stärke der aufzunehmenden Hypothese im Vergleich zu der Stärke und der Anzahl der bereits auf der Wissenseinheit vorhandenen Hypothesen einen bestimmten Schwellwert überschreitet. Hypothesen, die eine höhere Stärke besitzen, werden damit weiter im System verteilt. Der unten angegebene Zusammenhang ist heuristisch festgelegt und könnte anders definiert werden, um mehr oder weniger Hypothesen aufzunehmen. Neben diesem „blindem Verteilen", d.h. einem Verteilen, das nicht bedarfsgesteuert abläuft,

Typ der WE	Übertragungsbedingung	Erläuterung
Plan	Objekt → Plan	Hypothesen über Objekte oder Teilobjekte
	Relation → Plan	Hypothesen über eingenommene Relationen
	Benutzer → Plan	Aufgabenstellung, Hilfestellung, Beeinflussung der Pläne
	Plan → Plan	Information von über- oder untergeordneten Planwissenseinheiten
Relation	Objekt → Relation	Hypothesen über die Objekte, zu denen eine Relation eingenommen werden soll
	Sensor → Relation	Sensormessungen vor, während und nach Einnahme der Relation
	Benutzer → Relation	Angaben des Benutzers über eine neu zu lernende Relation
Objekt	Plan → Objekt	Anforderung von Objektinformationen
	Sensor → Objekt	Sensorwerte, die Aussagen über ein Objekt darstellen
	Objekt → Objekt	Austausch von Informationen zwischen über- und untergeordneten Objekten
Experiment	Plan → Experiment	Anforderungen gezielter Experimente zur Durchführung einer Teilaufgabe
	Objekt → Experiment	Anforderung gezielter Experimente zur Bestimmung von Objektparametern
Sensor	Objekt → Sensor	Gezielte Anfragen über bestimmte Sensorwerte
Aktor	Relation → Aktor	Bewegungshypothesen zum Erreichen eines Teilziels
Benutzer	* → Benutzer	Übertragen aller im System entstehenden Anfragen an die Benutzer-WE, damit der Benutzer eventuell unterstützen kann
Vorführung	Aktor → Vorführung	Übertragung der abgeleiteten Bewegungsbefehle, um Unterschiede zwischen System- und Benutzerverhalten detektieren zu können

Tabelle 5.1: Übertragungsbedingungen

gibt es das anfrageorientierte Verteilen. Eine Hypothese, die eine Anfragehypothese beantworten kann, gilt immer als interessant. Andererseits wird eine Anfragehypothese nur dann aufgenommen, wenn der zugehörige Agent sie potentiell beantworten kann, d.h. der Agent besitzt eine Produktion, die im Aktionsteil eine Hypothese bezüglich des angefragten Attributs ableitet.

Definition 5.23 (interessante Hypothese) *Eine Hypothese H heißt für eine Wissenseinheit WE* ***interessant****, gdw. einer der folgenden Fälle zutrifft.*

1. *Blindes Verteilen:* $S(H) > \min_{h \in WE} S(h) - \frac{\sum_{h \in WE} S(h)}{|WE|^2}$ *oder* $|\ WE\ | = 0$ *und zusätzlich ist mindestens eine der Übertragungsbedingungen des Typs der Wissenseinheit WE für die Hypothese H erfüllt.*

2. *Es liegt eine Anfrage für die Hypothese auf der Wissenseinheit vor, d.h.* $\exists h \in WE : h$ *ist eine Anfrage für die Belegung des Attributes, das in* H *angegeben ist.*

3. H *ist eine Anfragehypothese und es existiert eine Produktion für den zu* WE *gehörenden Agenten, deren Aktionsteil eine Aussage über das gewünschte Attribut generiert.*

Eine Anfragehypothese wird von einer Wissenseinheit dann generiert, wenn vorhandene Hypothesen einen Teil der Elementarbedingungen einer Produktion erfüllen, für eine andere Elementarbedingung aber noch keine Aussage vorliegt. Eine Anfragehypothese enthält den Attributpfad und den Attributnamen, über den eine Aussage gewünscht wird. Ein reines Verteilen auf Anfrage wäre ausreichend, wenn das Wissen jedes Agenten als korrekt und vollständig angenommen würde. Da hier jedoch speziell das Lernen dieses Wissens angestrebt wird, muß auch berücksichtigt werden, daß zu einem Zeitpunkt noch nicht bekannt ist, welches Wissen eigentlich tatsächlich auf einer Wissenseinheit benötigt wird. Aus diesem Grund wird eine (blinde) Verteilung vorgenommen, die nicht von den Regeln einer Wissenseinheit abhängt.

Die Hypothesen werden nach der Erzeugung rekursiv an andere Wissenseinheiten weitergegeben, die mit der erzeugenden Wissenseinheit über einen Kanal verbunden sind. Bei der Aufnahme einer Hypothese auf eine Wissenseinheit werden die verschiedenen Typen von Hypothesen unterschiedlich behandelt. Statische Hypothesen werden additiv zur Wissenseinheit hinzugenommen. Für eine veränderliche Hypothese wird zunächst, falls vorhanden, die Hypothese auf der Wissenseinheit gelöscht, die bis auf den Attributwert zur neuen Hypothese identisch ist und anschließend die neue Hypothese aufgenommen. Anfrage-, Rollen- und Nachrichtenhypothesen werden bei der Aufnahme wie statische Hypothesen behandelt.

Algorithmus 5.2 (Verteilen von Hypothesen)

Eingabe: · Neue Hypothese H
· Erzeugende Wissenseinheit WE
Ausgabe: · Als Seiteneffekt wird die Hypothese H rekursiv in alle mit WE verbundenen Wissenseinheiten, für die H interessant ist, verteilt

funct *Verteile*(H, WE)
  Schlange := *Verbundene Wissenseinheiten*(WE)
  while *Schlange* $\neq$ *nil* **do**
    W := Nimm erstes Element von *Schlange*
    if W bei der Verteilung noch nicht berücksichtigt **then**
      if *Interessant*(H, W) **then**
        $W := W \cup \{H\}$ (*s. Text*)
        Schlange := Hänge *Verbundene Wissenseinheiten*(W) an Schlange an

Erzeugen neuer Wissenseinheiten

Während des Ablaufs der Problemlösung werden neue Teilplanwissenseinheiten, neue Teilziele – also Relationswissenseinheiten, neue Objekt- und eventuell Experimentwissensein-

heiten erzeugt. Diese müssen in den vorliegenden Systemzustand insbesondere durch den Aufbau neuer Kanäle eingebunden werden. Eine Aktion, die eine neue Wissenseinheit erzeugt, wird daher in folgenden Schritten abgearbeitet:

1. Erzeuge eine neue Instanz des entsprechenden Typs einer dynamischen Wissenseinheit mit zunächst leerer Hypothesenmenge.
2. Binde die neue Instanz in das aktuelle System ein.
3. Speichere zu der Wissenseinheit ähnlich wie bei einer Hypothese die Nummer der erzeugenden Produktion, deren Stärke und die relevanten Hypothesen.

Die unter Punkt drei zusätzlichen Informationen werden für den Lernschritt benötigt, da es sein kann, daß die Produktion auf der Basis falscher Hypothesen angewendet wurde oder sogar selbst fehlerhaft ist.

Für das Einbinden einer neuen dynamischen Wissenseinheit in das System ist für jeden Typ von Wissenseinheit definiert, welche Verbindungsstruktur, d.h. welche Kanäle, zu bereits bestehenden Wissenseinheiten aufgebaut werden soll.

Definition 5.24 (Verbindungsstruktur einer Wissenseinheit) *Die Verbindungsstruktur $\mathcal{V}$ eines dynamischen Typs von Wissenseinheit ist eine Menge von Kanälen, die bei der Erzeugung einer Wissenseinheit dieses Typs zu anderen Wissenseinheiten des aktuellen Systemzustands aufgebaut werden müssen.*

Innerhalb der Verbindungsstruktur für eine Wissenseinheit existieren zum einen Kanäle zu Wissenseinheiten statischen Typs, die als feststehende Verbindungen bezeichnet werden, und zum anderen Kanäle zu Wissenseinheiten dynamischen Typs, den sogenannten zuordnungsabhängigen Verbindungen. Die Kanäle der zweiten Art sind deshalb zuordnungsabhängig, weil erst zur Laufzeit bestimmt werden kann, zu welcher im Systemzustand existierenden Wissenseinheit dieser Kanal aufgebaut werden muß. Beispielsweise wird eine neue Planwissenseinheit immer mit der sie erzeugenden Planwissenseinheit verbunden. Eine Relationswissenseinheit wird mit den Objektwissenseinheiten verbunden, zu denen die Relation eingenommen werden soll.

Das Einbinden einer neuen Wissenseinheit in das System geschieht in drei Schritten: 1. Erzeugen der feststehenden Verbindungen, 2. Erzeugen der zuordnungsabhängigen Verbindungen und 3. Initiale Übertragung der interessanten Hypothesen.

Im folgenden wird nun für die vier konkreten Typen dynamischer Wissenseinheiten angegeben, welche feststehenden und zuordnungsabhängigen Verbindungen jeweils aufgebaut werden müssen.

Eine neue Plan-WE wird zum einen initial von der Benutzer-WE erzeugt, um einen Problemlösevorgang zu aktivieren. Zum anderen kann eine Plan-WE weitere Planwissenseinheiten erzeugen, die dann Teilziele, d.h. Unterziele, dieses Planes darstellen. Zu diesem Zeitpunkt werden keine feststehenden Verbindungen erzeugt, sondern lediglich ein Kanal von und zur erzeugenden Planwissenseinheit.

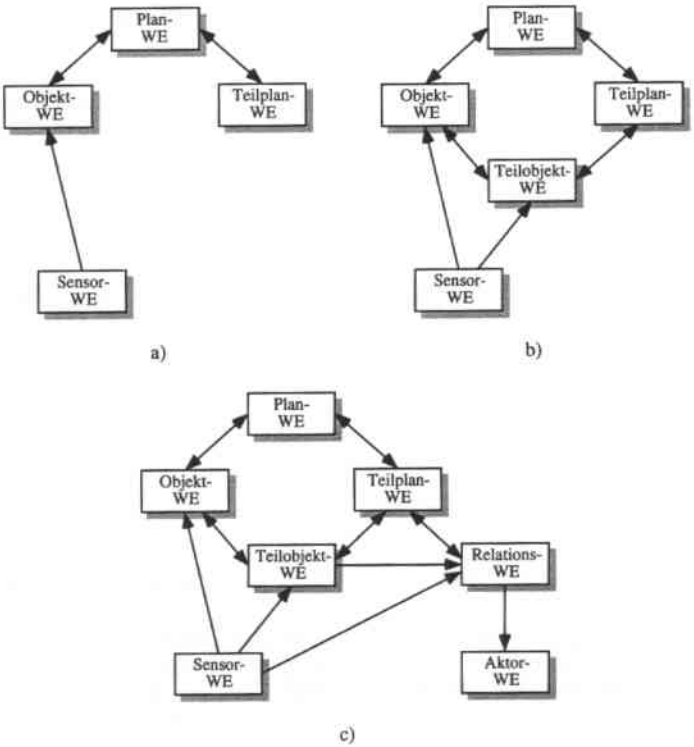

Abbildung 5.5: Systemzustand während des Beispiels

Beispiel: In einer Aufgabe sei als erste Teilaufgabe ein vorliegendes Objekt zu greifen. Der Systemzustand besteht zunächst nur aus den statischen Wissenseinheiten, einer Planwissenseinheit für die Aufgabe und einer Objektwissenseinheit für das Objekt. In der aktuellen Situation wird eine Produktion anwendbar, die der Formulierung einer Teilaufgabe und damit der Einrichtung eines Teilplanes entspricht: **WENN** ... **DANN** Neue Plan-WE **Gegriffen**. Nach der Anwendung der Regel entsteht der in Abbildung 5.5 im Teilbild a) vereinfacht dargestellte Systemzustand – die Benutzer-, Vorführ- und Aktorwissenseinheiten sind zunächst nicht explizit aufgezeigt.

Für eine neue Objektwissenseinheit werden die folgenden feststehenden Verbindungen erzeugt:

- Ein Kanal von und zur aktuellen Planwissenseinheit. Über diese Kanäle werden Hypothesen ausgetauscht, mit denen sich Planausführung und Objektagent gegenseitig beeinflussen können.

- Ein Kanal von der Sensor-WE zur Übertragung weiterer aktualisierter Objektinformationen.

Eine neue Objektwissenseinheit kann sowohl von der Sensor-WE aus erzeugt werden als auch von einer Plan-WE. Der zweite Fall entspricht einer aufgabenbedingten Hierarchisierung des Objektes, d.h. speziell benötigte Teilobjekte werden explizit repräsentiert. In diesem Fall werden zusätzlich zu den feststehenden Verbindungen Kanäle von und zur entsprechenden Plan-WE generiert.

Fortsetzung des Beispiels: Die bereits vorhandene Objektwissenseinheit repräsentiert das Objekt als Ganzes. Für die Lösung der Teilaufgabe ist die parallele Ausrichtung an einem Teilobjekt wichtig. Dazu hat der Planagent eine Regel der Form: `WENN ... DANN` Neue Objekt-WE `Ausrichtkante` mit Bezug zu `Rolle:Greifobjekt`. Nach der Anwendung der Regel ergibt sich der in Abbildung 5.5 im Teilbild b) gezeigte Systemzustand.

Eine neue Relationswissenseinheit wird als zu erreichendes Zwischenziel von einer Plan-WE erzeugt. Dazu werden die folgenden feststehenden Verbindungen aufgebaut:

- Ein Kanal zur Aktor-WE zur Übermittlung von Bewegungshypothesen
- Ein Kanal von der Sensor-WE für die Übertragung aktueller Sensorwerte
- Ein Kanal von und zur Benutzer-WE zur Unterstützung des Systems bei der Korrektur von Relationsbeschreibungen bzw. bei der Generierung neuer Relationen

Als zuordnungsabhängige Verbindungen wird ein Kanal zur aktuellen Planwissenseinheit erzeugt, um das erfolgreiche Einnehmen der Relation zu melden, und Kanäle von allen an der Relation beteiligten Objekten und Teilobjekten. Über diese werden neu aufgebaute und abgeleitete Objektinformationen weitergeleitet. Eine weitere Möglichkeit zur Erzeugung einer Relationswissenseinheit ist die Detektion von Relationen aus Informationen der Sensor- und Objektwissenseinheiten.

Fortsetzung des Beispiels: In der gegebenen Situation soll nun die parallele Ausrichtung des Greifers an der Ausrichtkante erfolgen: `WENN ... DANN` Neue Relations-WE `Ausgerichtet` mit Bezug zu `Rolle:Ausrichtkante`. Es ergibt sich die in Abbildung 5.5 im Teilbild c) gezeigte Struktur.

Eine Experimentierwissenseinheit kann von einer Objektwissenseinheit aus instanziiert werden. Feststehende Verbindungen sind analog zu einer Relationswissenseinheit Kanäle zur Aktor-WE und Benutzer-WE und von der Sensor-WE und der Benutzer-WE. Als zuordnungsabhängige Verbindungen treten Kanäle von und zu den relevanten Objektwissenseinheiten auf.

Im Anschluß an den Aufbau einer Verbindung wird jeweils untersucht, welche Hypothesen von den verbundenen Wissenseinheiten aus auf die neue Wissenseinheit übertragen worden wären, wenn sie schon vorher existiert hätte. Diese interessanten Hypothesen werden auf die neue Wissenseinheit aufgenommen.

Algorithmus 5.3 (Erzeugen einer Wissenseinheit)

Eingabe: · *Typ* der neu zu erzeugenden Wissenseinheit
· Aktueller *Systemzustand* = $(\mathcal{WE}, \mathcal{K})$
· Zuordnungsabhängige *Verbindungen*
Ausgabe: · Der Systemzustand, der aus dem aktuellen Systemzustand durch Einbringen einer neuen Wissenseinheit des angegebenen Typs entsteht.

funct *Erzeuge_Wissenseinheit*(*Typ*, *Systemzustand*, *Verbindungen*)
  $WE := \{\}$
  $\mathcal{K} := \mathcal{K} \cup \{$Feststehende Verbindungen zu WE für *Typ*$\}$
  $\mathcal{K} := \mathcal{K} \cup \{$Zuordnungsabhängige Verbindungen zu WE gemäß *Verbindungen*
  for $W \in \{w \in \mathcal{WE} \mid \exists$ Kanal $k \in \mathcal{K} : k = (w, WE)\}$ **do**
    for $H \in W$ **do**
      if *Interessant*(H, WE) **then**
        $WE := WE \cup \{H\}$
  for $H \in WE$ **do**
    Verteile(H, WE)
  $\mathcal{WE} := \mathcal{WE} \cup \{WE\}$
  return (*Systemzustand*)

5.3.4 Gesamtablauf des Systems

Mit den bisher definierten Teilkomponenten kann der Gesamtablauf des Systems beschrieben werden. Das System führt nacheinander Problemlöseschritte aus. Jeder einzelne besteht zum einen aus der Auswahl einer Wissenseinheit, die in diesem Zyklus aktiviert wird, und zum zweiten aus der Auswahl einer Produktion, die auf der Basis der vorliegenden Hypothesen angewendet werden kann. Das Ergebnis der Anwendung ändert entsprechend den aktuellen Systemzustand. In Abbildung 5.6 ist dieser Zyklus dargestellt, in den die Komponenten zum internen und externen Lernen, die in Kapitel 6 beschrieben werden, bereits aufgenommen wurden.

Konkurrenz, wie sie beim opportunistischen Planen vorliegt, bedeutet in dieser Architektur die Konkurrenz zwischen verschiedenen Hypothesen bzw. der Produktionen auf eventuell verschiedenen Wissenseinheiten. Der Wettbewerb kann zum einen darin bestehen, daß von einer bestimmten Wissenseinheit, z.B. einer Objekt-WE, noch weitere Informationen abgeleitet werden müssen, oder die Aufgabenlösung, z.B. auf einer Plan-WE, schon ohne diese Information weitergeführt werden kann. Zum anderen kann der Wettbewerb z.B. zwischen verschiedenen Varianten einer Aufgabenlösung oder allgemein zwischen verschiedenen abzuleitenden Aussagen bestehen. Mittelfristig soll sich die Wissenseinheit durchsetzen, die von bereits bestehenden oder neu hinzukommenden Hypothesen stärker unterstützt wird.

Bei Systemstart werden Instanzen der statischen Wissenseinheiten erzeugt und zusätzlich eine initiale Plan-WE. Zwischen diesen Wissenseinheiten wird eine Reihe von Kanälen zur Kommunikation aufgebaut. Dies sind im einzelnen:

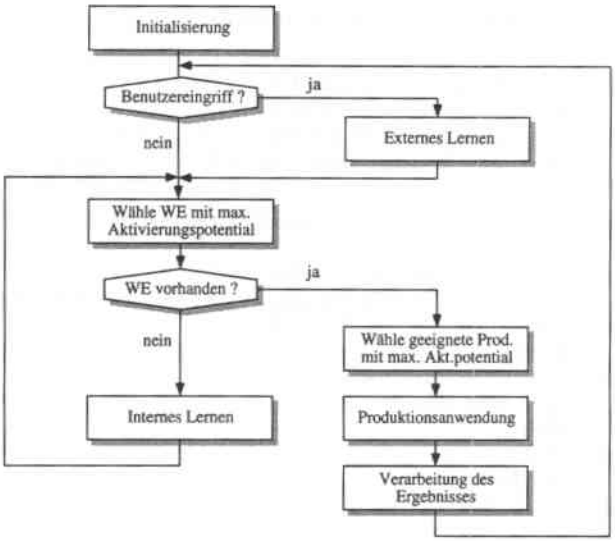

Abbildung 5.6: Systemablauf

- Ein Kanal von der Benutzer-WE zur initialen Plan-WE, um die Aufgabenstellung zu übertragen.
- Ein Kanal von der Sensor-WE zur Vorführungs-WE, um Werte der internen Sensorik zur Analyse der Benutzervorführung verwenden zu können.
- Ein Kanal von der Aktor-WE zur Vorführungs-WE zum Vergleich von System- und Benutzerverhalten bei einer Aufgabenlösung.

Außerdem werden alle Wissenseinheiten durch Kanäle mit der Benutzerwissenseinheit verbunden, um insbesondere Anfragehypothesen an den Benutzer weiterleiten zu können und Hilfestellungen oder Nachrichten übermitteln zu können.

Algorithmus 5.4 (Systemablauf)

Eingabe, Ausgabe:	· Das System nimmt Informationen aus der Umwelt und vom Benutzer auf, wendet das verfügbare Wissen an, und führt Aktionen durch.
Variablen:	· *Resultat* ist das Ergebnis der ausgewählten Produktion · *Systemzustand* ist die jeweils aktuelle Beschreibung aller Wissenseinheiten und Kanäle · *Aktive-WE* ist die Wissenseinheit, die ausgewählt wurde, um eine Produktion des zugehörigen Agenten anzuwenden

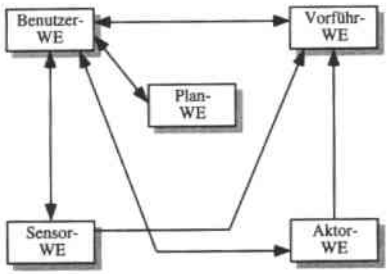

Abbildung 5.7: Initiale Verbindungsstruktur

funct *Systemablauf*()
  Systemzustand := Erzeuge die statischen Wissenseinheiten und eine initiale Plan-WE
  while true **do**
    if Benutzerinteraktion **then**
      Externes_Lernen (Art der Benutzerinteraktion, Gewünschter Zielpunkt)
    Wähle *Aktive-WE* so, daß ihr Aktivierungspotential maximal und nicht bereits bekannt ist, daß keine Produktion des zugehörigen Agenten angewendbar ist
    if *Aktive-WE* = *nil* **then**
      Generalisiere (*Prods* (aktuelle Plan-WE), aktuelle Plan-WE)
    else
      Resultat := *Aktiviere* (*Aktive-WE*)
      if *Resultat* ist eine Hypothese **then**
        Aktive-WE := *Aktive-WE* ∪ {*Resultat*}
        Verteile (*Resultat*, *Aktive-WE*)
      elsif *Resultat* ist eine Wissenseinheit **then**
        Erzeuge_Wissenseinheit (*Resultat*, *Systemzustand*)

Damit ist die Definition der grundlegenden Systemarchitektur und des prinzipiellen Ablaufs abgeschlossen. Für die einzelnen Agenten wird nun in den folgenden Abschnitten darauf eingegangen, welche speziellen Abläufe durch Produktionsschemata oder durch agentenspezifische Verarbeitungsschritte erreicht werden müssen.

5.4 Planendes Verhalten des Systems (Planagent)

Auf der Basis der Komponenten der Systemarchitektur und des prinzipiellen Ablaufs bei einer Problemlösung soll in diesem Abschnitt erläutert werden, wie ein planendes Verhalten des Systems erreicht wird.

Ausgehend von einer Aufgabenstellung, die vom Benutzer an das System gestellt wird, oder einer prinzipiell vorhandenen Verhaltensvorgabe wird in hierarchischer Form auf der Basis des Wissens des Planagenten eine Menge von Teilplänen abgeleitet und diese inkrementell weiterverfolgt. Planung und Ausführung überlappen vollständig, d.h. es

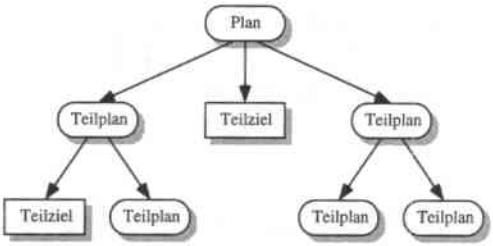

Abbildung 5.8: Partieller Lösungsbaum

wird nicht versucht, ausgehend von der Ausgangssituation einen möglichen Plan zu entwickeln. Während der Lösung der Aufgabe entsteht als Teil des Systemzustandes ein partieller Lösungsbaum in Form miteinander verbundener Plan- und Relationswissenseinheiten, der die noch durchzuführenden Teilpläne und Teilziele beschreibt (s. Abbildung 5.8). Jeder Teilplan wird durch eine eigene Wissenseinheit dargestellt, die für die Durchführung zuständig ist. Aufgrund der vorgegebenen Wissensrepräsentation durch Regeln und der zyklischen Auswahl einer geeigneten Wissenseinheit und Produktion kann prinzipiell an jeder Stelle mit der Lösung fortgefahren werden.

Zeitliche Reihenfolgebeschränkungen, die festlegen, daß ein Teilziel vor einem anderen erfolgreich abgearbeitet sein muß, gehen in die Vorbedingungen der einzelnen Regeln ein. Die Zerlegung eines Planes muß bis zu einer Menge von elementaren Teilzielen erfolgen. Diese Elementarziele werden als Relationen bezeichnet, da sie in Form von punktuellen oder Während-Relationen als geometrische oder Kraft-Beschreibungen vorliegen. Ein elementarer Schritt des Systems besteht dann darin, den Roboter so zu bewegen, daß schließlich die gewünschte Relation eingenommen wird. Im folgenden werden vereinfacht auch die Teilpläne Teilziele genannt, da sie einen ähnlichen Charakter haben, allerdings nicht zu direkten Auswirkungen auf die Umwelt führen.

Eine detaillierte Beschreibung des Relationsagenten und der möglichen Alternativen folgt im nächsten Abschnitt. An dieser Stelle kann zunächst davon ausgegangen werden, daß ein Agent im System ein solches Elementarziel erreichen kann. In dieser Arbeit steht das inkrementelle Planen on-line aus den oben genannten Gründen im Vordergrund. Dies schließt allerdings eine parallel ablaufende vorausschauende Planung nicht aus. Während das System kurzfristig den nächsten Teilplan abarbeitet, könnte langfristig das zukünftige Vorgehen geplant werden. Allerdings wäre dann eine Modellierung der exakten Auswirkungen einer Aktion auf die Umwelt notwendig.

5.4.1 Zeitliche Verkettung

Betrachtet man die zeitliche Abfolge der Einnahme von Teilzielen, so lassen sich prinzipiell folgende Abhängigkeiten unterscheiden, die sich mit Hilfe der verwendeten Regelrepräsentation im Planagenten wie angegeben darstellen lassen (s. Abbildung 5.9):

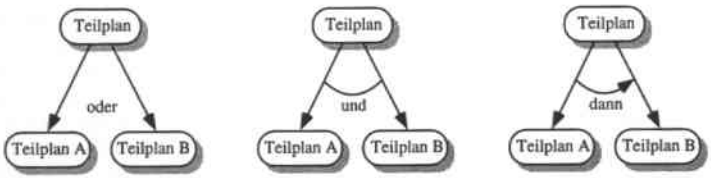

Abbildung 5.9: Alternativen der zeitlichen Verkettung

1. Alternative Lösungsmöglichkeiten als Teilziele:
   WENN ... **DANN** Erzeuge Teilziel A
   WENN ... **DANN** Erzeuge Teilziel B
   WENN Eingenommen(A) **DANN** ...
   WENN Eingenommen(B) **DANN** ...

2. Zeitlich nicht geordnete vollständige Zerlegung eines Zieles:
   WENN ... **DANN** Erzeuge Teilziel A
   WENN ... **DANN** Erzeuge Teilziel B
   WENN Eingenommen(A) $\wedge$ Eingenommen(B) **DANN** ...

3. Zeitlich geordnete vollständige Zerlegung eines Zieles:
   WENN ... **DANN** Erzeuge Teilziel A
   WENN Eingenommen(A) **DANN** Erzeuge Teilziel B
   WENN Eingenommen(B) **DANN** ...

Dabei ist jedoch immer zu berücksichtigen, daß es sich bei diesen Regeln um Regeln handelt, die in dieser Reihenfolge angewandt werden *können*, es aber immer möglich ist, daß auf der Basis neuer aufgenommener oder abgeleiteter Informationen andere Teilziele verfolgt werden. Lediglich im „Normalfall" läuft die Problemlösung tatsächlich ab wie vorgesehen. Oben wurde bereits erläutert, daß es sich bei dem Ablauf um ein opportunistisches Planen handelt, d.h. die tatsächliche Entscheidung für die eine oder andere Alternative wird nicht bei der Zerlegung des übergeordneten Teilziels auf der Basis der oben aufgeführten Regelschemata getroffen, sondern so spät wie möglich. Das führt dann dazu, daß alle Alternativen zunächst abgeleitet und weiterverfolgt werden. Eine Präferenzentscheidung muß jedoch spätestens getroffen werden, bevor als Elementarziel eine Relation eingenommen werden soll, da auf dieser Ebene eine überlappende Ausführung zu Störungen führen kann. Dieser Auswahlprozeß wird durch die oben erläuterte Methode eines Aktivierungspotentials für jede Wissenseinheit realisiert.

5.4.2 Reaktivität

Das System soll in der Lage sein, auf in der Umwelt auftretende unerwartete Ereignisse oder unvorhergesehene Situationen entsprechend zu reagieren. Dazu ist parallel zu dem strategischen Planen der Teilziele auch ein reaktives Planen notwendig. Das reaktive Verhalten des Gesamtsystems wird durch das inkrementelle Vorgehen und die jeweils neue

Untersuchung des Aktivierungspotentials bei der Problemlösung erreicht. Aufgrund neuer Sensorinformationen, neuer Aufgaben oder daraus abgeleiteter Informationen können neue Teilziele oder neue Produktionen aktiv werden, oder auch alte wieder reaktiviert werden (Fokuswechsel). Die Reaktivität bezieht sich damit, wie in Kapitel 2.1.6 ausgeführt, nicht nur auf ein einziges Teilziel, das unter ständiger Reaktion auf aktuelle Sensorwerte versucht wird zu erreichen, sondern auf den gesamten Zustand der Problemlösung. Neben einer reinen Reaktion verfolgt das System aber ständig seine vorgegebenen langfristigen Ziele.

Das System kann in den meisten Fällen nicht unterscheiden, ob eine Abweichung vom „Normplan/-ablauf" durch einen eigenen Fehler oder durch externe Einwirkung auftritt. Aus diesem Grund macht es, wie bereits in Kapitel 2.1.7 ausgeführt, keinen Sinn, explizit Fehler zu untersuchen oder zu behandeln. Vielmehr muß das System durch sein reaktives Verhalten in der Lage sein, auf einen solchen „Fehlerzustand" entsprechend zu reagieren. D.h. das System besitzt zwar keine explizite eigene Fehlerbehandlung aber Wissen darüber, wie es sich in entsprechenden Situationen verhalten soll.

Beispiel: Eine Relation, die bereits erfolgreich eingenommen wurde, wird durch eigenes Verschulden oder durch einen externen Einfluß wieder ungültig. Ist sie für die nachfolgenden Schritte der Aufgabenlösung notwendig, so wird erneut ein Teilziel zur Einnahme dieser Relation erzeugt. Bedingung dafür, daß eine Regel, die bereits angewendet wurde, erneut angewendet werden kann, ist, daß sich die Hypothesen, die die Vorbedingungen erfüllt haben, geändert haben.

5.4.3 Beispiele für Produktionstypen des Planagenten

Anhand einiger typischer Produktionsschemata des Planagenten soll die Repräsentation verschiedener Teilaufgaben des Planagenten dargestellt werden.

1. Zuweisung einer Rolle an ein bestimmtes Objekt: Wie oben erläutert wird einer Objektwissenseinheit im allgemeinen eine Rolle zugewiesen, die das zugehörige Objekt bei der Lösung eines Teilziels spielt.

```
WENN
  ((Plan-WE (Var:P1)) (Planname Planname))
  ∧ ((Plan-WE (Var:P1)) (Attribut11 Wert11))
                  ⋮
  ∧ ((Objekt-WE (Var:/Rolle:O1)) (Attribut21 Wert21))
                  ⋮
  ∧ ((Objekt-WE (Var:/Rolle:O2)) (Attribut31 Wert31))
                  ⋮
  ∧ ((Relations-WE (Var:R1)) (Aktuelle-Relation Relation1))
  ∧ ((Relations-WE (Var:R2)) (Aktuelle-Relation Relation2))
                  ⋮
DANN
  Ordne Rolle:Rolle dem Objekt O1 zu
```

2. Erzeugung einer neuen Wissenseinheit für einen Teilplan: Als Bedingungen für die Verfolgung eines neuen Teilplans können z.B. die Aufgabenstellung, Rollenzuweisungen, Objekteigenschaften und bereits eingenommene Relationen eingehen.

   WENN
   ((Plan-WE (Var:*P1*)) (Planname *Planname*))
   ∧ ((Plan-WE (Var:*P1*)) (*Rollenname* 'beliebig'))
   ∧ ((Plan-WE (Var:*P1*)) (*Attribut*$_{11}$ *Wert*$_{11}$))
   ⋮
   ∧ ((Objekt-WE (Var:/Rolle:*O1*)) (*Attribut*$_{21}$ *Wert*$_{21}$))
   ⋮
   ∧ ((Objekt-WE (Var:/Rolle:*O2*)) (*Attribut*$_{31}$ *Wert*$_{31}$))
   ⋮
   ∧ ((Relations-WE (Var:*R1*)) (Aktuelle-Relation *Relation*$_1$))
   ∧ ((Relations-WE (Var:*R2*)) (Aktuelle-Relation *Relation*$_2$))
   ⋮
   DANN
   Neue Plan-WE *Name* **mit Bezug zu Aktuelle Plan-WE**

3. Erzeugung einer neuen Wissenseinheit für die Einnahme einer Relation: Elementarziele, die bei jeder Zerlegung einer Aufgabe entstehen, werden durch Relationen beschrieben, die einzunehmen sind. Die entsprechenden Wissenseinheiten werden ausgehend von einer Planeinheit erzeugt. Die Abhängigkeiten sind analog zu den für eine neue Planwissenseinheit angegebenen. Die Aktion ist dann entsprechend **Neue Relations-WE** *Relationsname* **mit Bezug zu** (*Rollenname*)

4. Erzeugung einer neuen Wissenseinheit für ein Teilobjekt: Abhängig von bestimmten Aufgabeneigenschaften, Objekteigenschaften eines oder mehrerer Objekte und eventuell eingenommenen Relationen ist es notwendig, ein bekanntes Teilobjekt eines der bekannten Objekte explizit als eigene Wissenseinheit zu erzeugen. Das Produktionsschema sieht analog zu Punkt 1 aus, mit dem Unterschied, daß zusätzlich Bedingungen über eine **Teilobjekt**-Beziehung notwendig sein können und die Aktion lautet: **Neue Objekt-WE** *Teilobjektname* **mit Bezug zu** (*O1*)

5.5 Einnehmen von Relationen (Relationsagent)

Der Relationsagent ist zuständig für das Erreichen oder Einhalten bestimmter Relationen für eine Menge von sensorisch zu erfassenden oder abgeleiteten Merkmalen zu einer Menge von Objektmerkmalen. Die wichtigste Unterscheidung ist, ob es sich bei einer Relation um eine punktuelle oder eine Während-Relation handelt. Eine punktuelle Relation gibt an, daß eine Bewegung des Roboters durchgeführt werden muß, so daß zu einem bestimmten Zeitpunkt eine bestimmte Relation in der Welt erreicht wird. Eine Während-Relation muß über einen bestimmten Zeitraum eingehalten werden.

Der Agent der Relationswissenseinheiten benötigt für das erfolgreiche Erreichen einer punktuellen Relation drei Voraussetzungen:

1. Aussagen über die für die Bezugsobjekte relevanten Attribute
2. Ein Maß für die Gültigkeit einer Relation, d.h. dafür, ob eine Relation bereits eingenommen ist oder nicht oder wie „stark" sie eingenommen ist
3. Eine Methode, um sich der Einnahme einer Relation zu nähern

Für eine Während-Relation werden dagegen benötigt:

1. Aussagen über die für die Bezugsobjekte relevanten Attribute
2. Eine Methode, um eine bereits gültige Relation im nächsten Schritt beizubehalten.

Die in beiden Varianten angesprochene Methode entspricht einem Regler, der aus den relevanten Attributen und den sensorisch erfaßten oder abgeleiteten Größen einen Bewegungsbefehl erzeugt und an die Aktorwissenseinheit schickt. Für die im folgenden speziell untersuchten geometrischen punktuellen Relationen kann das normale Ablaufverfahren der Architektur eingesetzt werden. Für die anderen Relationsarten, insbesondere die Während-Relationen, müssen aber agentenspezifische Inferenz- und Lernverfahren entwickelt werden. Derartige Regler - auch lokale Regler oder Elementaroperationen genannt – lassen sich beispielsweise durch neuronale Netze oder Fuzzy-Regler beschreiben. Für den automatischen Aufbau dieser Regler, d.h. das Lernen der Regelungsfunktion, wurden bereits einige Arbeiten veröffentlicht, die in Kapitel 3.4 zitiert wurden.

Im folgenden wird die Betrachtung der Relationstypen im wesentlichen auf geometrische, punktuelle Relationen eingeschränkt. Die Aussagen über die an der Relation beteiligten Objekte gelangen von deren Objektwissenseinheiten über die Kanäle auf die Relationswissenseinheit. Eine Relation ist ein elementares Ziel, das während einer Aufgabenbearbeitung zu lösen ist. Dabei kann es sein, daß mehrere Relationen gleichzeitig eingenommen werden müssen. Formal wird eine Relation wie folgt definiert:

Definition 5.25 (Relation(sbeschreibung)) *Eine Verknüpfung von Gültigkeitsbedingungen, $G_1, \ldots, G_r$, die den erlaubten Wertebereich einer Menge sensorisch zu erfassender oder daraus abzuleitender Größen $S = \{s_1, \ldots, s_n\}$ angeben, heißt* ***Relation(sbeschreibung)*** *$R = (S, \{G_1(s_1, \ldots, s_n, o_1, \ldots, o_m), \ldots, G_r(s_1, \ldots, s_n, o_1, \ldots, o_m)\})$, mit $O = \{o_1, \ldots, o_m\}$ sind alle möglichen Objektmerkmale, auf die sich Relationsbeschreibungen beziehen können. Eine* ***Gültigkeitsbedingung*** *G_i ist dabei eine Abbildung $S \times O \rightarrow \{wahr, falsch\}$*

Die prinzipiell möglichen Formen der Gültigkeitsbedingungen wie Intervalle, die scharfe Schranken darstellen, oder Potentialfelder, die auch darstellen können, wie stark eine Relation bereits eingenommen ist, sind zwar vielfältig, es muß aber immer berücksichtigt werden, daß sowohl die Gültigkeit einer Relation berechnet werden kann als auch eine Methode existieren muß, um sich der Einnahme einer Relation zu nähern, wenn sie ungültig ist. Schließlich sollte die Relationsbeschreibung auch modifizierbar sein, so daß durch Lernvorgänge mit der Zeit eine möglichst korrekte und vollständige Relationsbeschreibung erreicht wird.

Aus diesem Grund wird im folgenden eine Beschreibung der Relationen durch Hyperintervalle verwandt, wobei jede Gültigkeitsbedingung eine Intervallbeschreibung für einen sensorisch zu erfassenden oder daraus abzuleitenden Meßwert liefert, d.h. $G_i = [g_{i_1}(o_1, \ldots, o_m), g_{i_2}(o_1, \ldots, o_m)]$, mit g_{i_1}, g_{i_2} sind Funktionen über der Menge möglicher Objektmerkmale $o_1, \ldots, o_m$. Für jeden möglichen Sensorwert $s_1, \ldots, s_n$ gibt es eine zugehörige Gültigkeitsbedingung $G_1, \ldots, G_r$, also $n = r$.

Definition 5.26 (Relationstypen) *Eine **geometrische Relation** wird definiert als eine Relation $R_G = (S, \{G_1, \ldots, G_r\})$, für die gilt: $S \subseteq$ {Merkmale, die sensorisch erfaßt oder daraus abgeleitet werden, die in irgendeiner Form eine Aussage über Positionen, Orientierungen, Abstände etc. machen}. Eine **Kraftrelation** ist eine Relation $R_K = (S, \{G_1, \ldots, G_r\})$, für die gilt: $S \subseteq$ {Merkmale, die sensorisch erfaßt oder daraus abgeleitet werden, die in irgendeiner Form eine Aussage über Kräfte oder Momente machen}.*

Die Definition, wann ein Meßwert o.ä. eine Relationsbeschreibung erfüllt, ist natürlich abhängig von der gewählten Repräsentationsform, d.h. sie ist spezifisch für Intervalle, Potentialfelder, etc. Hier werden die oben angegebenen Hyperintervalle verwendet, so daß sich folgende Definition für die Erfüllung einer Relation ergibt.

Definition 5.27 (Erfüllen einer Relation) *Ein Sensorwert oder abgeleiteter Wert $s \in S$ **erfüllt** die Teilbedingung $G_i = [g_1, g_2], 1 \leq i \leq r$ einer Relationsbeschreibung $(S, \{G_1, \ldots, G_r\})$, wenn gilt $g_1 \leq s \leq g_2$, d.h. $G_i(s, g_1, g_2) =$ wahr. Schreibweise: $Erfüllt(s, G_i)$.*

Definition 5.28 (Einnahme einer Relation) *Eine **punktuelle Relation** $R_P = (\{s_1, \ldots, s_r\}, \{G_1, \ldots, G_r\})$ gilt zu einem Zeitpunkt t als **eingenommen**, wenn zum Zeitpunkt t gilt: $Erfüllt(s_1, G_1) \wedge \ldots \wedge Erfüllt(s_r, G_r)$*

*Eine **Während-Relation** $R_W = (\{s_1, \ldots, s_r\}, \{G_1, \ldots, G_r\})$ gilt während eines Zeitintervalls $[t_1, t_2]$ als **eingenommen**, wenn zu jedem Zeitpunkt $t \in [t_1, t_2]$ gilt: $Erfüllt(s_1, G_1) \wedge \ldots \wedge Erfüllt(s_r, G_r)$*

Im allgemeinen tritt eine Während-Relation immer zusammen mit einer punktuellen Relation auf. Die Während-Relation gibt die Nebenbedingungen an, die eingehalten werden sollen, während eine Bewegung durchgeführt wird, um die punktuelle Relation zu erreichen. Zusammenfassend ergibt sich also die in Abbildung 5.10 aufgeführte Taxonomie von Relationstypen. Beispiele für die einzelnen Relationstypen sind:

1. Punktuell, geometrisch: Fahre eine Position oberhalb eines Objektes an
2. Punktuell, Kraft: Fahre in z-Richtung bis Kraft F erreicht
3. Während, geometrisch: Fahre parallel entlang Kante
4. Während, Kraft, (geometrisch): Fahre mit konstanter Kraft in z-Richtung (entlang Kante)

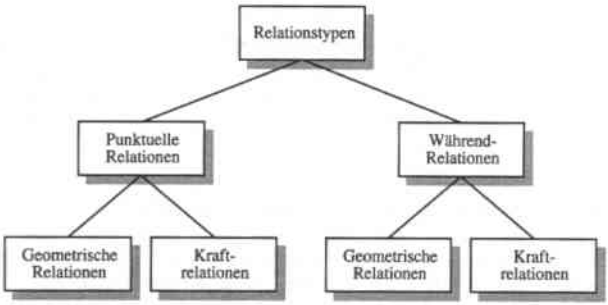

Abbildung 5.10: Taxonomie von Relationstypen

Neben diesen Elementarrelationen sind auch Kombinationen verschiedener Relationen möglich, z.B. eine Während-Kraftrelation, wobei gleichzeitig eine bestimmte Orientierung zu einer Fläche eingehalten werden soll. Der Ablauf im System zur Einnahme einer Relation als elementares Teilziel geschieht nach dem in Abbildung 5.11 gezeigten Schema. Die Definition eines Gültigkeitsbereichs einer Relation hängt im allgemeinen von einer Menge von Objektmerkmalen eines oder mehrerer (Teil-)Objekte und eventuell der konkreten Aufgabenstellung ab. Aus diesem Grund ist eine Relationswissenseinheit über Kanäle mit den beteiligten (Teil-)Objektwissenseinheiten und der erzeugenden Planwissenseinheit verbunden.

Die Gültigkeitsbedingungen in Form von Hyperintervallen sind funktionale Beschreibungen, die abhängig von Objektmerkmalen definiert werden.

Beispiel: Oberhalb: $x \in [Objekt_{x_{min}}, Objekt_{x_{max}}]$,
$y \in [Objekt_{y_{min}}, Objekt_{y_{max}}]$,
$z \in [Objekt_{z_{max}} + 10, Objekt_{z_{max}} + 30]$

Auf die Akquisition derartiger Beschreibungen durch funktionale Induktion aus Beispielen wird in Kapitel 6.3.2 eingegangen. Für jede Koordinate werden die konkreten Werte für die Begrenzungen evaluiert und getestet, ob die jeweiligen Ist-Werte im Gültigkeitsbereich liegen. Ist das für alle Koordinaten der Fall, gilt die Relation als eingenommen und eine entsprechende Hypothese wird erzeugt. Andernfalls muß der Unterschied zwischen Soll-Bereich und Ist-Wert zu einer Aktion des Manipulators führen. Dazu kann z.B. der Mittelpunkt des Gültigkeitsbereichs oder der nächste Punkt am Rand des Gültigkeitsbereiches als Zielpunkt angegeben werden. Diese Analyse führt zu einer Reihe von Bewegungshypothesen, die auf die Aktorwissenseinheit übertragen werden, und dort zu einer echten Ansteuerung des Manipulators führen können. Die Information über tatsächlich ausgeführte Bewegungen wird über die Sensorwissenseinheit und eventuell Objektwissenseinheiten wieder an die Relationswissenseinheit geliefert.

Die Relationswissenseinheit bleibt auch nach Einnahme der Relation bestehen und ist verantwortlich für die Kontrolle, ob die Relation noch gültig ist. Dazu wird bei jeder Aktivierung der Relationswissenseinheit ein Test auf Gültigkeit der Relation durchgeführt.

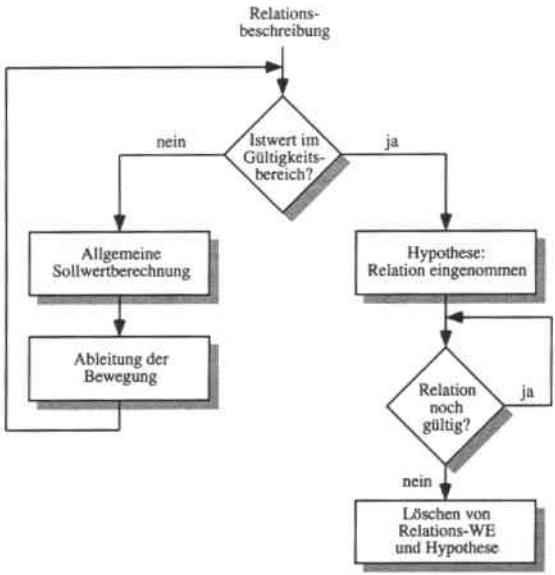

Abbildung 5.11: Ablauf der Relationseinnahme

Sobald dieser nach entsprechenden Aktionen des Roboters fehlschlägt, wird die Relationswissenseinheit und die Hypothese über die Gültigkeit der Relation aus dem System gelöscht.

Durch dieses Vorgehen können weitere Teilziele eingenommen werden, die vom Vorliegen einer bestimmten Relation abhängen. Nachdem eine Relation eingenommen wurde darf im Unterschied zur eigentlichen Einnahme keine Bewegung mehr abgeleitet werden, da im allgemeinen die Einnahme einer neuen Relation zur Zerstörung bereits bestehender Relationen führt.

5.5.1 Beispiele für Produktionstypen des Relationsagenten

Auch für den Relationsagenten soll anhand einiger typischer Produktionsschemata die Repräsentation verschiedener Teilaufgaben gezeigt werden:

1. Definition des Gültigkeitsbereiches einer Relation für eine Koordinate eventuell in Abhängigkeit von Typ und Merkmalen beteiligter Objekte:

```
WENN
  ((Relations-WE (Var:R)) (Relationsname Relationsname))
  ∧ ((Relations-WE (Var:R)) (Attribut11 Wert11))
```

```
                    ⋮
  ∧ ((Objekt-WE (Var:O1)) (Objekttyp Objekttyp1))
  ∧ ((Objekt-WE (Var:O1)) (Attribut21 Wert21))
                    ⋮
  ∧ ((Objekt-WE (Var:O2)) (Objekttyp Objekttyp2))
  ∧ ((Objekt-WE (Var:O2)) (Attribut31 Wert31))
                    ⋮
DANN
  Berechne Gültigkeitsbereich
    Koord muß im Intervall von Funktion1 bis Funktion2 liegen
```

2. Gültigkeitstest: In einer nachfolgenden Produktion wird die Gültigkeit der Relation basierend auf den tatsächlich vorliegenden Werten und dem in 1. berechneten Gültigkeitsbereich bestimmt. Die Hypothese über den Sollbereich wird durch die Aktion der Regel in Punkt 1 abgeleitet.

```
WENN
  ((Relations-WE (Var:R)) (Koord Koordinate) (Soll 'beliebig'))
  ∧ ((Sensor-WE (Var:R)) (Koord Koordinate) (Ist 'beliebig'))
DANN
  (Berechne veränderliche Hypothese
    ((Koord Koordinate) (Im-Bereich ?)) durch Funktionsname))
```

Bei einer Darstellung der Gültigkeitsbereiche als Hyperintervall besteht die referenzierte Funktion lediglich aus einem Intervalltest.

3. Erzeugung von Bewegungshypothesen: Solange ein Koordinatenwert noch nicht im berechneten Gültigkeitsintervall liegt, müssen Bewegungshypothesen abgeleitet werden, die an die Aktorwissenseinheit weitergeleitet werden. Setzt sich eine Bewegungshypothese durch, wird sie auch an die Vorführwissenseinheit übertragen – zum Abgleich mit Benutzervorführungen.

```
WENN
  ((Relations-WE (Var:R)) (Koord Koordinate) (Soll 'beliebig'))
  ∧ ((Sensor-WE (Var:R)) (Koord Koordinate) (Ist 'beliebig'))
  ∧ ((Relations-WE (Var:R)) (Koord Koordinate) (Im-Bereich NIL)))
DANN
  Berechne Hypothese
    ((Koord Koordinate) (Änderung ?)) durch Funktionsname))
```

Bei Verwendung von Intervalldarstellungen kann die referenzierte Funktion z.B. eine relative Bewegung zum Mittelpunkt des Gültigkeitsbereichs ableiten.

5.6 Behandlung von Objekten (Objektagent)

Ein wesentlicher Bestandteil der internen Hypothesen über die reale Welt sind die Informationen über vorliegende Objekte. Aufgabe des Objektagenten ist die Verarbeitung sämtlicher objektbezogener Informationen des Systems. Dazu gehört zum einen die Integration der aufgenommenen visuellen und taktilen Sensordaten, die Ableitung zusätzlicher Informationen - z.B. einer Klassenzuordnung - auf der Basis vorhandener generischer Objektmodelle und der Aufbau bzw. die Verfeinerung dieser Objektmodelle. Die Langzeitrepräsentation der generischen Objektmodelle wird in Kapitel 7 ausführlich beschrieben, ebenso die Lernverfahren und die Klassifikation von Objekten. An dieser Stelle wird nur die Einbindung und die Verarbeitung im Gesamtsystem besprochen.

Objektwissenseinheiten werden zunächst durch entsprechende Verarbeitungsschritte des Sensoragenten erzeugt. Hier wird davon ausgegangen, daß der Objektagent für die einzelnen Objektwissenseinheiten die entsprechenden Hypothesen der Sensorwissenseinheit herausfiltern kann, d.h. daß durch den Sensoragent eine Segmentierung vorgenommen wird und vom Objektagent die Zuordnung von Informationen, die während der Problemlösung anfallen, zu Objektwissenseinheiten durchgeführt werden kann. Für dieses Problem wurde jedoch noch keine Lösung entwickelt. Daneben können neue Objektwissenseinheiten auch durch Planwissenseinheiten erzeugt werden. Sie repräsentieren im wesentlichen Teilobjekte, die für die Problemlösung als eigenständiges Merkmal wichtig sind (s. Abschnitt 5.4.3), z.B. Teilkomponenten eines Objektes oder eine Fläche oder Kante. Auf die Wissenseinheit eines Teilobjektes gelangen dann nur die Hypothesen, die eine Aussage über das Teilobjekt machen. Außerdem ist es auch möglich, daß eine Planwissenseinheit eine Objektwissenseinheit als Erwartungswert erzeugt, d.h. wenn für eine gegebene Teilaufgabe bestimmte Objekte eingesetzt werden müssen, können vom Plan aus entsprechende Wissenseinheiten „vorhergesagt" werden.

5.7 Analyse einer Benutzervorführung (Vorführagent)

Aufgabe des Vorführagenten ist es, eine vollständige Benutzervorführung aufzuzeichnen, diese in entsprechende Teilaktionen/Teilziele zu zerlegen und an entsprechender Stelle in den Systemablauf einfließen zu lassen. Es ergibt sich also die in Abbildung 5.12 aufgezeigte Struktur.

In Kapitel 3.6 wurden die verschiedenen Möglichkeiten aufgezeigt, eine Vorführung als Beispiel für ein System durchzuführen. Die wichtigsten Varianten sind zum einen die direkte Verwendung des Manipulators oder einer Telemanipulatorsteuerung zur Demonstration einer Aufgabenlösung und zum anderen die Vorführung durch eigene Manipulation des Benutzers ohne Einbeziehung eines technischen Hilfsmittels. Im zweiten Fall ist eine Beobachtung des Benutzerverhaltens, z.B. durch Kamerasysteme, und die Umsetzung in geeignete Roboteraktionen notwendig.

Soll das System nicht nur eine reine Aktionsfolge, sondern die Abhängigkeit bestimmter Aktionen von Sensormessungen lernen, so ist das zweite Vorgehen nicht sinnvoll. Dem Benutzer sollte nach Möglichkeit nur die Sensorik zur Verfügung stehen, auf die auch das

Abbildung 5.12: Ablauf Vorführagent

System zugreifen kann. Andernfalls ist eine automatische adäquate Modellierung der Entscheidungsstrukturen nicht möglich. Die erste Variante ermöglicht z.B. auch die Ableitung von nötigen Aktionen, um bestimmte Informationen zu beschaffen, die auch der Benutzer benötigt, um eine Aufgabe zu lösen (z.B. Bewegung zu bestimmten Punkten, um dort ein Kamerabild aufzunehmen).

In den Teilaufgaben Vorverarbeitung, Segmentierung und Zuordnung ist sowohl ein automatisches Vorgehen als auch ein interaktives Vorgehen zusammen mit dem Benutzer denkbar. In dieser Arbeit wurde ein interaktives Vorgehen angestrebt und dazu ein grafisches Werkzeug entwickelt (s. Anhang A), das eine Visualisierung der Vorführung und die Segmentierung ermöglicht [Lew93]. Die resultierenden Teilziele werden dann während der Problemlösung durch das System vom Benutzer zugeordnet.

Ziel der Segmentierung ist eine Zerlegung der Aufzeichnung in Abschnitte, nach denen Teilziele erreicht, d.h. Relationen eingenommen wurden. Kandidaten dafür sind Endpunkte von Bewegungsfolgen, starke Änderungen des Bewegungsvektors oder allgemein signifikante Änderungen in einem der aufgezeichneten Sensorverläufe. Eine automatische Analyse wird in [KGN94] erstmals untersucht.

In der Zuordnung müssen die extrahierten Teilziele mit den Teilzielen, die das System selbst verfolgt, in Zusammenhang gebracht werden. Diese Teilziele entsprechen Relationen,

die das System versucht einzunehmen, bzw. Relationen, die das System einnehmen sollte. Im nachfolgenden Schritt führen die zugeordneten Teilziele zu einem externen Lernvorgang, wie er in den Abschnitten 6.3 und 6.4 beschrieben wird. Dies resultiert entweder in einer Generalisierung oder einer Spezialisierung des Systemwissens oder in der Formulierung neuen Systemwissens.

Erhält die Aktor-WE eine Hypothese, die eine Bewegung verursachen würde, so wird diese auch an die Vorführ-WE übergeben. Dort wird untersucht, ob der Benutzer eine nächste Aktion vorgeführt hat. Ist dies der Fall und Benutzer- und Systemverhalten sind identisch, wird die Aktion tatsächlich ausgeführt. Ansonsten muß ein Lernschritt durchgeführt werden.

5.8 Aktor- und Sensoragenten

Die Aktor-WE und die Sensor-WE sind jeweils an das reale System gekoppelt. Dazu wurden zwei Prozesse definiert, die Bewegungshypothesen auf der Aktor-WE in eine Ansteuerung des Roboters mit Hilfe von Elementaroperationen umsetzen bzw. die eine Umwandlung von verarbeiteten Sensordaten in entsprechende Hypothesen auf der Sensor-WE vornehmen (siehe auch [Fan93]).

Die Aktorwissenseinheit erhält Bewegungshypothesen von Relationswissenseinheiten, die versuchen – eventuell parallel – bestimmte Teilziele einzunehmen. Ein solches Teilziel, das dauerhaft bestehen kann, ist die Vermeidung von Kollisionen – eine genauere Betrachtung dieser Problematik wurde im Rahmen dieser Arbeit allerdings nicht durchgeführt. Der Aktoragent wählt die stärkste Bewegungshypothese aus und generiert daraus einen elementaren Bewegungsbefehl der zugrundeliegenden Robotersteuerung. In Anhang A wird eine kurze Übersicht über die verfügbaren Steuerungsbefehle gegeben.

Der Sensoragent nimmt vorverarbeitete Meßsignale von internen und externen Sensoren auf und überträgt sie in entsprechende Hypothesenform, die dann in das System eingebracht und verteilt werden.

5.9 Lernkomponenten

Eines der Hauptziele beim Entwurf der Architektur war es, eine Möglichkeit zu schaffen, durch Lernvorgänge das Systemwissen zu erweitern und zu korrigieren. Diese Anforderung führte insbesondere zu den strikten Definitionen für die einzelnen Komponenten der Systemarchitektur und des Systemablaufs. Prinzipiell sind alle Lernaufgaben, die in Kapitel 3.2 aufgezeigt wurden, von Interesse und sollten im Rahmen der Architektur realisiert werden. In dieser Arbeit wurden einige dieser Lernaufgaben untersucht und geeignete Methoden für deren Realisierung entwickelt. Dazu zählen das

1. Lernen von Regelwissen: Für die Produktionenrepräsentation aller Agenten soll die Möglichkeit geschaffen werden, das bestehende Wissen aufgrund neuer Erfahrungen zu korrigieren und zu erweitern. Dazu zählt insbesondere die Kopplung eines Systemzustandes und eines nächsten einzunehmenden Teilziels, d.h. der Aufbau von

Planungsregeln, und die Zuordnung von Rollen zu bestimmten vorliegenden Objektmerkmalen. Für die Teilziele soll eine entsprechende Beschreibung gelernt werden, die die Gültigkeitsbedingungen für die gewählte Hyperintervalldarstellung angeben.

2. Lernen von Objektbeschreibungen: Eine wichtige Information, die der Objektagent für eine Objektwissenseinheit ableiten können muß, ist die Klasse/der Typ des durch die Wissenseinheit repräsentierten Objektes. Um diese Klassifikation durchführen zu können, muß der Objektagent zunächst Objektbeschreibungen (Modelle) lernen.

3. Lernen von Objekteigenschaften: Bestimmte Eigenschaften eines Objektes lassen sich nur durch aktive Experimente des Roboters ableiten. Dazu müssen entsprechende Experimente geplant und ausgewertet, und eine Integration dieser Ergebnisse in die Objektbeschreibung vorgenommen werden.

Eine detaillierte Beschreibung der entwickelten Verfahren folgt in den nächsten Kapiteln.

5.10 Zusammenfassung

In diesem Kapitel wurde eine Systemarchitektur für ein autonomes Robotersystem vorgestellt, die zum einen die Ansprüche an die Flexibilität eines solchen Systems erfüllt, zum anderen die Integration von Lernverfahren ermöglicht, wie in den nächsten Kapitel gezeigt werden wird. Mit dem definierten Vorgehen für den Systemablauf ist eine integrierte Betrachtung von Planung, Ausführung und Perzeption möglich. Darüberhinaus können sich reaktive Verhaltensweisen und strategisches Vorgehen überlagern und gleichzeitig betrachtet werden.

Kapitel 6

Lernen von Regelwissen

In diesem Kapitel wird nun untersucht, wie das Wissen der Agenten, das in der agentenunabhängigen Repräsentationsform, d.h. als Produktionsregeln, dargestellt ist, durch induktive Lernvorgänge akquiriert und verbessert werden kann (siehe auch [Kre92, Hau92, KH93a, KH93b, Bre93]). Dazu werden sowohl Generalisierungs- als auch Spezialisierungsoperatoren definiert und ein Vorzugskriterium für den besten induktiven Operator entwickelt. Unterschieden werden interne, d.h. automatische, und externe, d.h. durch den Benutzer initiierte, Lernvorgänge. In beiden Fällen wird formal eine Bedingung dafür definiert, wann ein derartiger Lernvorgang stattfindet und was jeweils das genaue Lernziel ist. Für beide Varianten wird dann detailliert das entwickelte Vorgehen beschrieben.

Das zugrundeliegende Wissen der einzelnen Agenten, das hier betrachtet wird, liegt in Form von Produktionsregeln vor. Deshalb muß zunächst analysiert werden, was ein induktiver Lernschritt in Form von Generalisierung oder Spezialisierung auf einer Menge von Produktionen überhaupt bedeutet. Für das Begriffslernen wurden in Kapitel 2.2.2 bereits Generalisierung und Spezialisierung definiert als die Vergrößerung bzw. Verkleinerung der Menge, für die der Begriff gültig ist. Während es für die Repräsentation eines Begriffs durch logische Formeln oder z.B. auch durch eine Regel einfach ist, sich eine Verallgemeinerung oder Einschränkung vorzustellen, ist das bei einer Menge von Regeln, die nacheinander angewendet werden müssen, um eine Aufgabe zu lösen, zunächst nicht derart offensichtlich.

Wird die Menge der korrekt lösbaren Aufgaben betrachtet, so sollte diese Menge sowohl bei der Generalisierung als auch bei der Spezialisierung größer werden. Keine der Entscheidungen für die Anwendung einer Produktionen während der Aufgabenlösung darf zu einem Fehler führen. Andererseits muß immer mindestens eine Produktion existieren, die das System der Aufgabenlösung näher bringt. Es ist daher sinnvoll, die induktiven Operatoren auf eine einzelne Produktion zurückzuführen, die generalisiert oder spezialisiert wird. Für eine Produktion P werden diese Operatoren über den Begriff der Anwendbarkeit definiert, d.h. bei der Generalisierung von P wird die Menge $\mathcal{S} = \{$ Situation $s \mid P$ ist anwendbar in $s\}$ vergrößert, bei der Spezialisierung wird $\mathcal{S}$ eingeschränkt. Eine Generalisierung bzw. Spezialisierung einer Menge von Produktionen, d.h. der Wissensbasis eines Agenten, liegt dann vor, wenn aus dieser Menge eine Produktion generalisiert bzw. spezialisiert wird.

Das Hauptproblem ist nun zu entscheiden, wann es zu einem solchen induktiven Schritt kommen soll und welche Produktion(en) dabei modifiziert werden sollen. In bisherigen

Ansätzen zum Regellernen, die in Abschnitt 2.2.4 aufgeführt und analysiert wurden, wird zumeist davon ausgegangen, daß zur Korrektur des Regelwissens ein „allwissendes Orakel" zur Verfügung steht, das entweder den korrekten Problemlösevorgang als Ganzes liefert oder an jeder Stelle die richtige Entscheidung treffen kann. Daraufhin wird in der aktuellen Situation eine Spezialisierung aller anwendbaren, aber nicht gewünschten Produktionen und eine Generalisierung der gewünschten, aber nicht anwendbaren Produktionen durchgeführt.

Die aufgestellte Forderung nach einem derartigen Orakel ist in der Realität nicht haltbar. Daher müssen andere Möglichkeiten gefunden werden, wie das System feststellen kann, daß eine Veränderung des Systemwissens notwendig ist. Unterscheiden lassen sich dann

- interne Lernvorgänge, die durch systeminterne Untersuchung von Zuständen und Problemlösungen initiiert werden, und
- externe Lernvorgänge, die durch externe Einflußnahme zustande kommen.

Für interne Lernvorgänge muß dann definiert werden, in welchen Fällen ein induktiver Schritt sinnvoll oder notwendig ist. Eine Generalisierung könnte beispielsweise stattfinden, wenn in einem Zustand keine Produktion mehr anwendbar ist. Das System versucht dann, eine Produktion so zu generalisieren, daß sie anwendbar wird. Eine Spezialisierung könnte dann initiiert werden, wenn das System einen Fehler bei der Ausführung oder der Ableitung einer Hypothese feststellt. Die erste Form der Initiierung einer Spezialisierung ist nur schwer zu automatisieren, da es nicht möglich ist, ein allgemeingültiges Fehlererkennungsmodul zu definieren (s. Kapitel 2.1.7). Ein tatsächlicher Fehler würde nur dann auftreten, wenn es zu einer Kollision käme. Die zweite Form der Initiierung – die Betrachtung fehlerhafter Hypothesen – könnte dann eingesetzt werden, wenn eine Hypothese abgeleitet wird, von der gezeigt werden kann, daß sie im Widerspruch zu bereits existierenden sicheren Hypothesen steht. In dieser Arbeit wird als interner Lernvorgang die Generalisierung untersucht.

Ein extern initiierter Lernvorgang entsteht durch Kommunikation mit einem externen Experten, der in der Regel ein menschlicher Benutzer/Lehrer des Systems ist (→ Programmieren durch Vorführen), der aber auch aus einem anderen System bestehen kann, das bereits über geeignetes Wissen verfügt. Dabei ist aber zu berücksichtigen, daß die Kommunikation normalerweise nicht auf der systeminternen Repräsentation aufbauen kann, sondern daß eine gemeinsame Sprache existieren muß, um beispielsweise nur einzunehmende Teilziele mitzuteilen. Dementsprechend muß dann eine Auswirkung des Lernschritts auf alle an der Erarbeitung eines Teilziels beteiligten Produktionen untersucht werden.

Neben der Modifikation bereits bestehender Produktionen ist auch die Akquisition neuer Produktionen wichtig, entweder initial oder dann, wenn keine sinnvolle Generalisierung oder Spezialisierung einer bestehenden Regel möglich ist. In diesem Fall müssen bestimmte Produktionsschemata instanziiert werden, die die Ableitung einer Aktion/Information auf der Basis bestimmter Elemente der vorliegenden Situation ermöglichen. Zusammengefaßt werden also die in Abbildung 6.1 gezeigten Varianten der Initiierung eines induktiven Lernens von Regeln untersucht.

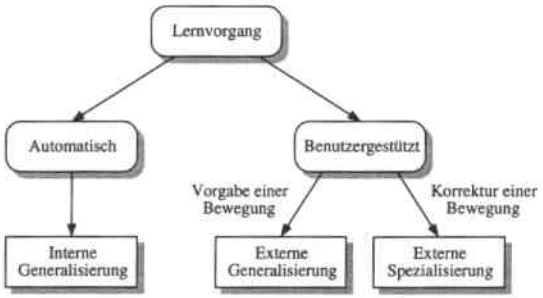

Abbildung 6.1: Klassifikation von Lernvorgängen

6.1 Grundlagen

Zunächst wird die Generalisierung einer Elementarbedingung definiert, auf der dann die Generalisierung einer Produktion aufbaut. Die Operatoren für die Spezialisierung werden in Abschnitt 6.4 eingeführt. Dabei ist wichtig, daß die Generalisierung nicht beliebig vorgenommen wird, sondern immer unter Berücksichtigung einer bestimmten Situation, für die die Anwendbarkeit einer Produktion erreicht werden soll. Aus diesem Grund geht in die Definition der Generalisierung einer Elementarbedingung die Belegung einer Hypothese des aktuellen Zustandes ein. Der Übersichtlichkeit halber wird die Zuordnung von Variablen zu Wissenseinheiten nicht explizit aufgeführt.

Definition 6.1 (Generalisierung einer Elementarbedingung) *Sei* $B = (T_B, E_B, P_B, N_B, W_B)$ *eine Elementarbedingung,* $H = (T_H, E_H, P_H, N_H, W_H)$ *eine Hypothese, und die Erzeugerbeschränkung, der Pfad und der Attributname von* H *sei in den entsprechenden Elementen von* B *enthalten. Eine Elementarbedingung* $G = (T_G, E_G, P_G, N_G, W_G)$ *heißt dann* ***Generalisierung*** *der Elementarbedingung* B *unter Berücksichtigung der Hypothese* H*, wenn eine der folgenden Aussagen gilt:*

1. W_H *beschreibt einen einfachen symbolischen Wert* h*,* W_B *beschreibt einen einfachen symbolischen Wert* b*, dann* $W_G := \{b, h\}$
2. W_H *beschreibt einen einfachen symbolischen Wert* h*,* W_B *beschreibt eine Menge* $\mathcal{B}$*, dann* $W_G := \mathcal{B} \cup \{h\}$
3. W_H *beschreibt einen einfachen numerischen Wert* h*,* W_B *beschreibt einen einfachen numerischen Wert* b*, dann* $W_G := [b, h]$*, falls* $b \leq h, [h, b]$*, sonst*
4. W_H *beschreibt einen einfachen numerischen Wert* h*,* W_B *beschreibt ein numerisches Intervall* $[b_1, b_2]$*, dann* $W_G := [\min(b_1, h), \max(b_2, h)]$
5. W_H *beschreibt ein numerisches Intervall* $[h_1, h_2]$*,* W_B *beschreibt ein numerisches Intervall* $[b_1, b_2]$*, dann* $W_G := [\min(h_1, b_1), \max(h_2, b_2)]$

6. *W_H beschreibt eine Aufzählung $(h_1, \ldots, h_n)$, W_B beschreibt eine Aufzählung $(b_1, \ldots, b_n)$. Dann $W_G := (Generalisierung(b_1, h_1), \ldots, Generalisierung(b_n, h_n))$*

7. *Aufstieg in einer Hierarchie (Taxonomie), die eine Aussage über die möglichen Attributwerte macht. Sei T eine Taxonomie, die als Baum repräsentiert ist, mit der Wurzel als Ebene 0. Sei W_B ein Element des Baumes auf Ebene e_B, und der Pfad von der Wurzel sei $(k_0, k_1, \ldots, k_{e_B} = W_B)$. Sei analog W_H ein Element auf Ebene e_H und der Pfad $(k'_0, k'_1, \ldots, k'_{e_H} = W_H)$. Sei e_G so gewählt, daß $\forall i : 0 \leq i \leq e_G : k_i = k'_i$ und $k_{e_G+1} \neq k'_{e_G+1}$. Dann ist W_G das Element des Baumes, das k_{e_G} entspricht. W_G ist stets definiert, da immer $k_0 = k'_0$.*

8. *Falls W_B = „beliebig", dann W_G = „beliebig".*

Aufbauend auf der Generalisierung einer Elementarbedingung kann eine Produktion, wie sie in Definition 5.10 formuliert wurde, nun auf verschiedene Arten generalisiert werden: 1. Durch Löschen einer Elementarbedingung oder 2. Durch Generalisierung einer Elementarbedingung. Eine weitere mögliche Generalisierung wäre die Generalisierung der Einschränkung an den Erzeuger, die im folgenden aber nicht betrachtet wird.

Definition 6.2 (Generalisierung einer Produktion) *Eine Produktion $P = (\mathcal{B}, A)$ kann auf zwei Arten, die gemischt eingesetzt werden können, generalisiert werden, um eine Menge von Hypothesen $\mathcal{H}$ zu erfüllen:*

1. *Lösche eine oder mehrere Elementarbedingungen von $\mathcal{B}$, die von den Hypothesen in H nicht erfüllt werden.*

2. *Generalisiere gemäß Definition 6.1 den Wertebereich einer oder mehrerer der Elementarbedingungen, so daß sie jeweils durch eine Hypothese in $\mathcal{H}$ erfüllt werden.*

Aufgrund der Definition einer Generalisierung einer Produktion gibt es im allgemeinen eine Vielzahl verschiedener Generalisierungsmöglichkeiten. Darüberhinaus muß innerhalb einer Menge von Produktionen eventuell auch die Produktion bestimmt werden, die generalisiert werden soll. Zu diesem Zweck ist die Definition eines Vorzugskriteriums (engl. bias) für die induktiven Schlüsse notwendig (s. Definition 2.7), durch das eine vergleichende Bewertung der unterschiedlichen Generalisierungsvarianten möglich wird. In Kapitel 2.2.4 wurden bereits einige mögliche Vorzugskriterien genannt, die aber fast alle für das Problem des Begriffslernens definiert wurden. Beim Regellernen ist zu berücksichtigen, daß während der Ausführung die nächste anzuwendende Produktion ausgewählt werden muß. Aus diesem Grund ist es wichtig zu bestimmen, wie plausibel eine Generalisierung im Kontext der bereits existierenden Produktionen ist. Um diese Plausibilität zu formalisieren, wird untersucht, wie wichtig, d.h. diskriminierend, eine bestimmte Elementarbedingung einer Produktion ist. Diese Wichtigkeit wird als Signifikanz bezeichnet.

Definition 6.3 (Signifikanz einer Elementarbedingung) *Die **Signifikanz** einer Elementarbedingung $B = (T, E, P, N, W)$ ist wie folgt definiert:*

$$Signifikanz(B) = \frac{\sum_{P \in Prods_A^-(WE)} Stärke(P)}{\sum_{P \in Prods_A^+(WE)} Stärke(P)}$$

Dabei sei A das Attribut bezüglich dessen B eine Bedingung definiert, $Prods_A(WE)$ die Menge der Produktionsregeln, deren Bedingungsteil einen Test auf A enthält, $Prods_A^+(WE) \subseteq Prods_A(WE)$ die Menge der Produktionen, für die dieser Test immer dann erfüllt ist, wenn auch B erfüllt ist, und schließlich $Prods_A^-(WE) := Prods_A(WE) \setminus Prods_A^+(WE)$. Falls $\sum_{P \in Prods_A^+(WE)} Stärke(P) = 0$, dann sei $Signifikanz(B) = +\infty$. Falls W = „beliebig", dann sei $Signifikanz(B) = 0$.

Eine Elementarbedingung, d.h. deren Belegung eines Attributes, ist also dann besonders signifikant, wenn sie im Bedingungsteil von möglichst wenigen anderen Produktionen enthalten ist, d.h. wenn die spezielle Ausprägung der Elementarbedingung möglichst selten zur Unterscheidung verwendet wird. Dann ist sie nämlich andererseits ein gutes Unterscheidungskriterium für eine der Produktionen. Die Produktionen gehen dabei nicht gleichgewichtet in die Formel ein, sondern mit ihrer globalen Stärke.

Beispiele für die Signifikanz von Elementarbedingungen:

Sei die Menge der Produktionen eines Agenten $Prods(WE)$ gegeben durch:

```
{ WENN (...(Länge [10,50])) DANN ...,
  WENN (...(Länge [20,60])) DANN ...,
  WENN (...(Länge [20,40])) DANN ...,
  WENN (...(Länge [100,150])) DANN ...,
  WENN (...(Höhe [10,100])) DANN ...,
  WENN (...(Länge 60)) ∧ (...(Höhe 50)) DANN ... }
```

Die ersten Auslassungszeichen repräsentieren die Erzeugerinformation, die zweiten die auszuführende Aktion. Sei die Stärke aller Produktionen gleich 10.0. Dann ist die Signifikanz der folgenden Bedingungen wie angegeben:

`B = (...(Länge 30))`,	$Signifikanz(B) = 20,0/30,0 = 0,66$
`B = (...(Länge 50))`,	$Signifikanz(B) = 30,0/20,0 = 1,5$
`B = (...(Länge 70))`,	$Signifikanz(B) = \infty$
`B = (...(Länge [20,40]))`,	$Signifikanz(B) = 20,0/30,0 = 0,66$
`B = (...(Länge [10,50]))`,	$Signifikanz(B) = 30,0/20,0 = 1,5$
`B = (...(Breite 30))`,	$Signifikanz(B) = \infty$

Als Kosten einer Generalisierung werden nun der Verlust an Signifikanz betrachtet, der durch Wegfall der Elementarbedingung oder durch Generalisierung des Wertebereichs entsteht.

Definition 6.4 (Kosten der Generalisierung einer Elementarbedingung) *Die **Kosten** der Generalisierung einer Elementarbedingung B zur Elementarbedingung G sind definiert als*

$$Kosten(B, G) = Signifikanz(B) - Signifikanz(G)$$

Falls $Signifikanz(B) = Signifikanz(G) = +\infty$, *dann* $Kosten(B, G) = 0$. *Entsprechend sind die anderen Kombinationen definiert.*

Auf dieser Basis wird nun ein Maß für die Kosten der Generalisierung einer Produktion definiert, das als Vorzugskriterium verwendet wird.

Definition 6.5 (Kosten der Generalisierung einer Produktion) *Die* ***Kosten*** *der Generalisierung einer Produktion* $P = (\{B_1, \ldots, B_n\}, A)$ *zur Produktion* $Q = (\{G_1, \ldots, G_n\}, A)$ *sind definiert als die Summe der Kosten der einzelnen Generalisierungen. Dabei sei* $\mathcal{B}_1$ *definiert als die Menge aller* B_i, *die gelöscht werden, d.h. zu „wahr" generalisiert werden, und* $\mathcal{B}_2$ *die Menge aller generalisierten Elementarbedingungen.*

$$Kosten(P, Q) = \sum_{B_i \in \mathcal{B}_1} Signifikanz(B_i) + \sum_{B_i \in \mathcal{B}_2} Kosten(B_i, G_i)$$

Für die Berechnung der Kosten ist $\infty + \infty$ definiert als ∞. In der konkreten Realisierung wird ∞ durch *maxreal* dargestellt. Die Kosten sind nicht auf den positiven Zahlenbereich normiert, d.h. die Kosten einer Generalisierung können negativ sein. Dieses Maß wird nur zum Vergleich zwischen verschiedenen Generalisierungsmöglichkeiten eingesetzt. Ein negatives Maß ist also nicht derart zu interpretieren, daß die Qualität der Wissensbasis besser wird, wenn generalisiert wird. Bevorzugt wird die notwendige Generalisierung mit den geringsten Kosten.

Beispiele für Kosten von Produktionsgeneralisierungen:

Sei wieder die oben angegebene Menge von Produktionen $Prods(WE)$ gegeben. Die Kosten der Generalisierung von $P =$ `WENN (...(Länge 60)) ∧ (...(Höhe 50)) DANN ...` zu verschiedenen Produktionen sind wie unten angegeben:

$Q =$ `WENN (...(Länge [10,60])) ∧ (...(Höhe 50)) DANN ...`,
$Kosten(P, Q) = (1,5 - 0,25) + (0,0 - 0,0) = 1,25$

$Q =$ `WENN (...(Länge [20,60])) ∧ (...(Höhe 50)) DANN ...`,
$Kosten(P, Q) = (1,5 - 0,66) + (0,0 - 0,0) = 0,83$

$Q =$ `WENN (...(Länge [30,60])) ∧ (...(Höhe [0 50])) DANN ...`,
$Kosten(P, Q) = (1,5 - 1,5) + (0,0 - 1,0) = -1,0$

6.2 Internes Lernen

Im vorigen Abschnitt wurden bereits die beiden Varianten des Lernens von Regelwissen eingeführt: Internes Lernen und externes Lernen. Ziel des internen Lernens ist die automatische Korrektur des Systemwissens. Dazu muß definiert werden, wann tatsächlich ein solcher Lernvorgang nötig ist. In der Einleitung zu diesem Kapitel wurde bereits erläutert,

warum eine interne Spezialisierung hier nicht betrachtet wird. Dagegen läßt sich definieren, wann eine interne Generalisierung notwendig ist, und zwar, wenn in einem Systemzustand keine weitere Produktion mehr anwendbar ist, die Aufgabe aber noch nicht als gelöst gilt. Mit den Definitionen aus Kapitel 5 läßt sich der Zustand, in dem eine interne Generalisierung untersucht werden soll, damit wie folgt formal darstellen:

Definition 6.6 (Notwendigkeit einer internen Generalisierung) *Eine interne Generalisierung wird dann initiiert, wenn auf keiner der Wissenseinheiten eine Produktion anwendbar ist, d.h.*

$$\neg\exists WE \in System : \exists P \in Prods(WE) : Anwendbar(P, WE)$$

und die Aufgabe noch nicht gelöst wurde.

Beispiel: In einer Problemlösung sei ein Zustand erreicht, in dem keine Produktion anwendbar ist. Für den Planagenten seien zwei Regeln gegeben, in denen jeweils eine Elementarbedingung nicht erfüllt ist: Für Regel 1 die Elementarbedingung (`Objekthöhe [40,80]`), für Regel 2 die Elementarbedingung (`Objektbreite [10,70]`). Für die Objekthöhe sei ein Wert von 85 bekannt, die Objektbreite wurde bisher nicht abgeleitet, es existiert allerdings für den Objektagenten eine Produktion zur Ableitung einer Hypothese darüber, die nicht anwendbar ist. Eine interne Generalisierung kann dann entweder die Elementarbedingung von Regel 1 so generalisieren, daß sie erfüllt wird, oder die Regel des Objektagenten generalisieren, so daß eine Hypothese erzeugt wird, die die Elementarbedingung von Regel 2 erfüllt.

Es wird nun die kostengünstigste Generalisierung einer Produktion gesucht, um wieder eine anwendbare Produktion zu erreichen, d.h. die Lernaufgabe bei der internen Generalisierung ist die folgende:

Definition 6.7 (Lernaufgabe der internen Generalisierung) *Finde die generalisierte Produktion P' von P für den Agenten einer Wissenseinheit $WE \in System$, die die kostengünstigste interne Generalisierung aller Produktionen darstellt und anwendbar ist, d.h. für die gilt: $Anwendbar(P', WE)$.*

Die notwendige Suche im Raum der möglichen Generalisierungen wird rekursiv durchgeführt und beginnt wie in Kapitel 5.3 gezeigt auf der aktuellen Planwissenseinheit, weil angenommen wird, daß eine Generalisierung dort den größten Fortschritt für die Aufgabenlösung bringt. Die rekursive Betrachtung findet statt, weil die Anwendbarkeit einer Produktion durch zwei Möglichkeiten erreicht werden kann:

1. Durch Generalisierung der Vorbedingungen der betrachteten Produktion selbst.
2. Durch Generalisierung von Produktionen desselben oder eines anderen Agenten, die bei ihrer Anwendung Hypothesen erzeugen würden, die die Vorbedingungen der betrachteten Produktion erfüllen.

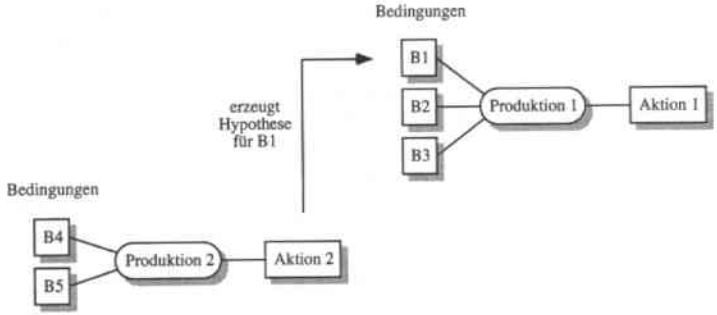

Abbildung 6.2: Erzeugende Produktionen

Beispiel: In Abbildung 6.2 seien die Bedingungen B2 und B3 der Produktion 1 erfüllt, B1 sei nicht erfüllt. Eine Generalisierung der Produktion ist möglich, indem die Bedingung B1 verallgemeinert wird. Alternativ kann die Produktion 2 so verallgemeinert werden, daß sie anwendbar wird, da sie als Aktion eine Hypothese erzeugt, die B1 erfüllt.

Definition 6.8 (Erzeuger für eine Bedingung) *Eine Produktion P heißt* ***Erzeuger*** *für die Bedingung B, wenn der Aktionsteil von P eine Hypothese erzeugt, die B erfüllt.*

Durch diese Einschränkung muß keine vollständige Suche der besten Generalisierung einer Produktion durchgeführt werden, sondern es werden nur Produktionen untersucht, die die Problemlösung weiterführen könnten, d.h. die zu einer Aussage führen, die als Vorbedingung auf der untersuchten Wissenseinheit benötigt wird. Die Generalisierung einer erzeugenden Produktion ist nur sinnvoll, wenn das Ergebnis einer Produktionsanwendung tatsächlich der untersuchten Wissenseinheit zur Verfügung gestellt werden kann. Aus diesem Grund wird bei der Rekursion die aktuelle Verbindungsstruktur im System berücksichtigt. Im folgenden werden stufenweise die Verfahren angegeben, wie die beste Generalisierung einer Bedingung, einer Produktion, einer erzeugenden Produktion und einer Menge von Produktionen bestimmt werden.

Die beste Generalisierung für eine Bedingung wird dadurch bestimmt, daß zunächst für jede vorliegende Hypothese untersucht wird, ob eine Generalisierung der Bedingung möglich ist, so daß sie durch die Hypothese erfüllt wird, und falls ja, welche Kosten diese Generalisierung verursachen würde. In der so entstehenden Alternativenmenge wird die günstigste Generalisierung bestimmt. Kann die Bedingung durch keine Hypothese erfüllbar gemacht werden, wird als Generalisierungsmöglichkeit das Löschen der Bedingung betrachtet.

Konvention: In den folgenden Algorithmen werden Variablen mit Index benutzt, z.B. G_{Kosten}, G_{Bed}. Bei einer Zuweisung der Variablen ohne Index z.B. $BG := G$, werden per Definition auch die einzelnen indizierten Komponenten zugewiesen.

Algorithmus 6.1 (Finde beste Generalisierung für eine Bedingung)

Eingabe: · B ist die Bedingung, die unter Berücksichtigung von $\mathcal{H}$ generalisiert werden soll
· $\mathcal{H}$ ist eine Menge von Hypothesen
· Z ist eine Zuordnung
Ausgabe: · Die beste Generalisierung der Elementarbedingung B bei Betrachtung der Hypothesenmenge $\mathcal{H}$
Variablen: · BG ist die jeweils beste Generalisierung von B mit den Kosten BG_{Kosten} und der generalisierten Bedingung BG_{Bed}

funct *Finde_beste_Generalisierung_für_Bedingung*$(B, \mathcal{H}, Z)$
  $BG_{Bed} := nil$; $BG_{Kosten} := maxreal$
  for $H \in \mathcal{H}$ **do**
    G_{Bed} := Generalisiere Elementarbedingung B bezüglich H und Z laut Def. 6.1
    if $G_{Bed} \neq nil$ **then** (∗Eine Generalisierung ist möglich∗)
      $G_{Kosten} := Kosten(B, G_{Bed})$
      if $G_{Kosten} \leq BG_{Kosten}$ **then** $BG := G$
  if $BG_{Bed} = nil$ **then** $BG_{Kosten} := Signifikanz(B)$
  return (BG)

Aufbauend auf der Generalisierung einer Bedingung wird die beste Generalisierung einer Produktion durch sukzessives Betrachten der einzelnen Bedingungsteile bestimmt. Falls ein Bedingungsteil schon von in $\mathcal{H}$ vorliegenden Hypothesen erfüllt wird, wird er ungeändert übernommen. Andernfalls wird die direkte Generalisierung der Bedingung untersucht und alternativ die Generalisierung von Produktionen, die eine Hypothese erzeugen würden, die diese Bedingung erfüllt. Nach diesen erzeugenden Produktionen wird auf der betrachteten Wissenseinheit selbst und zusätzlich auf allen verbundenen Wissenseinheiten rekursiv gesucht.

Aufgrund der unterschiedlichen Semantik der verschiedenen Produktionstypen müssen für einige Produktionen zusätzliche Bedingungen erfüllt sein, damit eine Generalisierung der Produktion zulässig ist. Bei Produktionen, die eine Wissenseinheit erzeugen, müssen alle (Rollen-)Variablen für die zuordnungsabhängigen Verbindungen im Bedingungsteil der Produktion vorkommen. Für die Produktionen, die einen Gültigkeitsbereich für eine Relation angeben, müssen alle Merkmale, die in die Funktionsberechnungen eingehen, auch in den Vorbedingungen enthalten sein.

Algorithmus 6.2 (Finde beste Generalisierung für eine Produktion)

Eingabe:
- $P = (\mathcal{B}, A)$ ist die Produktion, die generalisiert werden soll
- WE ist die Wissenseinheit, für die eine Generalisierung von P bestimmt werden soll
- $\mathcal{P}_{mark}$ ist die Menge von Produktionen, die bereits bei der Suche nach einer besten Erzeugergeneralisierung berücksichtigt wurden
- *Limit* ist eine vorgegebene Schranke für die maximalen Kosten, die bei einer Generalisierung anfallen dürfen

Ausgabe:
- Die beste Generalisierung BG der Produktion P ausgehend von der Wissenseinheit WE, bestehend aus der generalisierten Produktion BG_{Prod}, den Kosten für die Generalisierung BG_{Kosten} und den Generalisierungen für Erzeugerproduktionen $BG_{Erz.Prod}$

Variablen:
- $\mathcal{V}$ ist die Menge der neuen Vorbedingungen
- *Kosten* sind die akkumulierten Kosten der Produktionsgeneralisierung (laut Def. 6.4 und 6.5)
- *Erz.Prod* ist die Menge der Produktionen, die generalisiert werden müßten, wenn die unerfüllten Vorbedingungen von P, die nicht generalisiert werden, erfüllt werden sollen

funct *Finde_beste_Generalisierung_für_Produktion*$(P, WE, \mathcal{P}_{mark})$
  if $\exists$ Zuordnung $Z : \forall B \in \mathcal{B} : \exists H \in WE :$ *Erfüllt*(H, B, Z) **then**
    $BG_{Prod} := P$; $BG_{Kosten} := 0$; $BG_{Erz.Prod} := \{\}$
    return (BG)
  elsif P wurde gerade vorher spezialisiert oder die Produktion P darf nicht generalisiert werden **then return** (nil)
  else
    $BG_{Prod} := nil$; $BG_{Kosten} := maxreal$; $BG_{Erz.Prod} := \{\}$
    for $Z \in$ alle möglichen Zuordnungen von Variablen in P zu Wissenseinheiten **do**
      $\mathcal{V} := \{\}$; $Kosten := 0$; $Erz.Prod := \{\}$
      for $B \in \mathcal{B}$ **do**
        if $\exists H \in WE :$ *Erfüllt*(H, B, Z) **then** $\mathcal{V} := \mathcal{V} \cup \{B\}$
        else
          $G :=$ *Finde_beste_Generalisierung_für_Bedingung*(B, WE, Z)
          $EG :=$ *Finde_beste_Generalisierung_für_Erzeuger*$(B, WE, Z, \mathcal{P}_{mark})$
          if $(G_{Bed} \neq nil) \wedge (G_{Kosten} \leq EG_{Kosten}) \wedge (G_{Kosten} \leq Limit)$ **then**
            $\mathcal{V} := \mathcal{V} \cup \{G_{Bed}\}$; $Kosten := Kosten + G_{Kosten}$
          elsif $(EG_{Bed} \neq nil) \wedge (EG_{Kosten} \leq G_{Kosten}) \wedge (EG_{Kosten} \leq Limit)$ **then**
            $\mathcal{V} := \mathcal{V} \cup \{B\}$; $Kosten := Kosten + EG_{Kosten}$
            $Erz.Prod := Erz.Prod \cup EG_{Erz.Prod}$
          else (* Vorbedingung wird gelöscht *)
            $Kosten := Kosten +$ *Signifikanz*(B)
    $NP :=$ Neue Produktion$(\mathcal{V}, A)$
    if NP ist eine zulässige Generalisierung $\wedge \neg\exists Prod \in Prods(WE) : Prod = NP$ **then**
      if $Kosten < BG_{Kosten}$ **then**
        $BG_{Prod} := NP$; $BG_{Kosten} := Kosten$; $BG_{Erz.Prod} := Erz.Prod$
      return (BG)

Zur Bestimmung einer Generalisierung für eine Produktion, die eine bestimmte Hypothese für eine Bedingung erzeugen würde, werden alle Produktionen der aktuellen Wissenseinheit und aller damit verbundenen Wissenseinheiten untersucht. Für jede dieser Produktionen wird durch obiges Verfahren die beste Generalisierungsmöglichkeit berechnet.

Algorithmus 6.3 (Finde beste Generalisierung für Erzeuger)

Eingabe:
- B ist die Bedingung, für die nach erzeugenden Produktionen gesucht wird.
- WE ist die Wissenseinheit, für die eine Generalisierung von P bestimmt werden soll
- Z ist eine Zuordnung
- $\mathcal{P}_{mark}$ ist die Menge von Produktionen, die bereits bei der Suche nach einer besten Erzeugergeneralisierung berücksichtigt wurden

Ausgabe:
- Die beste Erzeugergeneralisierung BEG der Produktion P ausgehend von der Wissenseinheit WE

funct *Finde_beste_Generalisierung_für_Erzeuger*$(B, WE, Z, \mathcal{P}_{mark})$
  $BEG_{Prod} := nil$; $BEG_{Kosten} := maxreal$; $BEG_{Erz.Prod} := \{\}$
  for $W \in \{WE\} \cup \{w \mid WE \in$ *Verbundene Wissenseinheiten*$(w)\}$ **do**
    for $EP \in$ *Prods*(WE)
      if $EP \notin \mathcal{P}_{mark} \wedge EP$ kann B erzeugen **then**
        $EG :=$ *Finde_beste_Generalisierung_für_Produktion*$(EP, W, \mathcal{P}_{mark} \cup \{EP\})$
        if $EG_{Kosten} \leq BEG_{Kosten}$ **then** $BEG := EG$
  if $BEG_{Kosten} = 0$ **then**
    (* es gab schon Erzeugerproduktion, die aber noch nicht angewendet wurde *)
    $BEG_{Erz.Prod} := \{\}$
  else $BEG_{Erz.Prod} := BEG_{Erz.Prod} \cup \{BEG_{Prod}\}$
  return (BEG)

Aufbauend auf dem bisher formulierten Vorgehen zur Generalisierung einer Produktion kann nun definiert werden, wie in einer Menge von Produktionen, die z.B. alle eine bestimmte Aktion ausführen, oder allgemeiner, die prinzipiell die Aufgabenlösung weiterbringen könnten, die beste Generalisierung bestimmt werden kann.

Algorithmus 6.4 ((Interne) Generalisierung)

Eingabe:
- $\mathcal{P}$ ist eine Menge von Produktionen, für die eine Generalisierung gesucht wird
- WE ist die Wissenseinheit, auf der nach einer Generalisierung gesucht werden soll
- *Limit* ist eine vorgegebene Schranke für die maximalen Kosten, die bei einer Generalisierung anfallen dürfen

Ausgabe:
- Einbindung der besten Generalisierungsvariante in das Wissen der einzelnen Agenten des Systems

Variablen:
- G ist die beste gefundene Generalisierungsvariante einer Produktion, BG ist die beste Generalisierung aller Produktionen $\mathcal{P}$

funct *Generalisiere*$(\mathcal{P}, WE)$
  (∗Finde die beste Generalisierung für die Produktionen $\mathcal{P}$∗)
  $BG_{Prod} := nil$; $BG_{Kosten} := maxreal$; $BG_{Erz.Prod} := nil$
  for $P \in \mathcal{P}$ **do**
    $G := Finde\_beste\_Generalisierung\_für\_Produktion(P, WE, \{\})$
    if $(G_{Prod} \neq nil) \wedge (G_{Kosten} < BG_{Kosten})$ **then** $BG := G$
  if $0 < BG_{Kosten} < Limit$ **then**
    Berechne Stärke der neuen Produktion gemäß (6.1)
    $Prods(WE) := Prods(WE) \cup BG_{Prod}$
    for $EG \in BG_{Erz.Prod}$ **do**
      Berechne Stärke der neuen Produktion gemäß (6.1)
      $Prods(EG_{WE}) := Prods(EG_{WE}) \cup EG_{Prod}$
  else $BG_{Prod} := nil$
  return (BG)

Wenn die Kosten für eine Generalisierung unterhalb einer vorgegebenen Höchstgrenze liegen, wird die generalisierte Regel zusätzlich in die Wissensbasis des entsprechenden Agenten aufgenommen und der Problemlösezyklus fortgesetzt. Die Stärke der neuen Regel wird auf die Stärke der Ausgangsregel und die Kosten der Generalisierung zurückgeführt.

$$Stärke(G) := Stärke(P) - \min\left(\frac{Kosten(P,G)}{Stärke(P)}, 0\right) \tag{6.1}$$

Das heißt je stärker generalisiert werden muß, um so weniger sicher wird die neue Produktion betrachtet. Die Ausgangsregel bleibt ebenfalls in der Wissensbasis des Agenten gespeichert, so daß in einem zukünftigen Problemlöseschritt beide Regeln in Konkurrenz stehen können. Aufgrund der anfänglich höheren Stärke der Ausgangsregel wird sich diese, wann immer sie anwendbar ist, zunächst durchsetzen.

Sollten bei der Suche nach einer möglichen Generalisierung keine Produktionen gefunden werden, deren Kosten unterhalb der vorgegebenen Höchstgrenze liegen, so ist eine Hilfe durch den Benutzer notwendig. Dazu teilt der Benutzer das nächste zu erreichende Teilziel mit, und vom System wird ein Satz neuer Regeln erzeugt. Das genaue Vorgehen ist Teil der im folgenden Abschnitt beschriebenen Behandlung einer externen Generalisierung. Liegt keine Hilfe vom Benutzer vor, so gilt das vorliegende Problem als gegenwärtig nicht lösbar.

6.3 Externes Lernen

Falls eine selbständige Generalisierung des Systemwissens nicht möglich ist oder der Benutzer des Systems als Lehrer auftreten will, wird ein extern initiierter Lernvorgang durchgeführt. Zu diesem Zweck findet eine Kommunikation mit dem Benutzer statt, während der er Systemziele vorgeben, korrigieren oder erweitern kann. Als Systemziele werden die einzelnen Teilziele in Form von einzunehmenden Relationen betrachtet.

Die prinzipiellen Varianten einer Vorführung an sich wurden bereits in den Abschnitten 3.6 und 5.7 analysiert. Hier wird davon ausgegangen, daß eine Vorführung a priori oder inkrementell vorverarbeitet und segmentiert wird und als Folge von einzunehmenden Teilzielen vorliegt. Prinzipiell können dann vier wesentliche Möglichkeiten unterschieden werden, wie der Benutzer die Wissensakquisition und -korrektur unterstützt:

1. Demonstration einer vollständig neuen Aufgabe
2. Demonstration einer Variante einer bekannten Aufgabe
3. Demonstration des nächsten zu erreichenden Teilziels während einer Aufgabenlösung
4. Korrektur des zuletzt eingenommenen Teilziels während einer Aufgabenlösung

Die Fälle 1. und 2. werden durch sukzessive Anwendung von 3. und 4. erreicht, d.h. das System lernt während einer Aufgabenausführung aus dem Vergleich mit einer Benutzervorführung.

Für den Aufbau einer neuen Produktion liegen zum einen die aktuellen Hypothesen des Systems und zum anderen die Aktion des Benutzers vor. Eine direkte Kopplung des Zustandes mit der durchgeführten Aktion des Benutzers über eine neue Produktion soll nicht durchgeführt werden, da sie zu einem äußerst inflexiblen System führen würde. Daher wird versucht, ein neues Teilziel, d.h. eine Relation, zu identifizieren, die der Benutzer bei seiner Bewegung angestrebt hat. Aus diesem Grund ist ein Zusammenhang zwischen Hypothesen, die im System den aktuellen Zustand charakterisieren, und der vom Benutzer eingenommenen Relation herzustellen.

Zur Interpretation der Vorführung eines Benutzers, d.h. zur Integration dieser Informationen in das Systemwissen, müssen drei prinzipielle Teilprobleme gelöst werden:

1. Lernen des Zusammenhangs zwischen Situationsbeschreibung und Teilzielauswahl, d.h. Lernen von Produktionen, die im aktuellen Systemzustand eine passende Relationswissenseinheit erzeugen.
2. Lernen der Relevanz und der Rollen einzelner (Teil-)Objekte für ein Teilziel in Abhängigkeit von ihren Merkmalen und von Aufgaben- und Situationsmerkmalen.
3. Lernen der Beschreibung (Gültigkeitsbedingungen) eines Teilzieles, d.h. Lernen der Benutzersemantik für die Gültigkeit eines Teilziels.

Zunächst wird für alle drei Teilprobleme versucht, ob das bereits existierende Wissen des Systems so modifiziert werden kann, daß auch die neue Situation abgedeckt wird. Dazu wird eine Generalisierung untersucht, wie sie in Abschnitt 6.2 für die interne Generalisierung beschrieben wurde. Allerdings werden als Ausgangsproduktionen nicht alle Produktionen einer Wissenseinheit betrachtet, sondern nur die, die eine passende Aktion bewirken (Erzeugen einer Relationswissenseinheit, Zuweisen einer Rolle, Berechnung eines Gültigkeitsbereichs). Für die Teilprobleme, für die die Kosten aller möglichen Generalisierungen oberhalb einer bestimmten Grenze liegen, wird ein neuer Satz von Produktionen erzeugt.

Dies gilt insbesondere dann, wenn für einen Teilzieltyp oder für eine Rolle noch keine Produktionen vorliegen. Die neu erzeugten Produktionen sollen es in einem zukünftigen ähnlichen Systemzustand ermöglichen, daß die Problemlösung selbständig weitergeführt wird.

Die beiden ersten genannten Lernschritte wirken sich primär auf die Wissensbasis des Planagenten aus, weil dieser sowohl Teilziele als auch neue für einen Plan relevante Objektwissenseinheiten erzeugt und letzteren eventuell Rollen zuweist. Das dritte Lernziel liegt vollständig im Aufgabenbereich des Relationsagenten.

Im folgenden werden nun zunächst die Lernverfahren für den Planagenten beschrieben. Anschließend folgt die detaillierte Untersuchung der möglichen Fälle der Lernverfahren für den Relationsagenten. Damit sind dann die Voraussetzungen geschaffen, um im letzten Schritt das gesamte Vorgehen beim externen Lernen zu definieren. Das Vorgehen bei der Spezialisierung wird separat im nächsten Abschnitt erläutert.

Definition 6.9 (Notwendigkeit einer externen Generalisierung) *Eine externe Generalisierung ist notwendig entweder, wenn 1. Eine interne Generalisierung notwendig ist und zusätzlich eine Zielvorgabe des Benutzers vorliegt oder, wenn 2. Eine Wissenseinheit zur Erreichung eines Teilziels erzeugt wurde, dieses eingenommen wird, und der Benutzer ein anderes Teilziel vorgibt.*

6.3.1 Lernverfahren für Planagenten

Die Wissensbasis des Planagenten muß durch die externe Generalisierung so verändert werden, daß 1. Eine Produktion anwendbar wird, die eine Wissenseinheit für die gewünschte Relation erzeugt, 2. Jedes Objekt, das an der Relation beteiligt ist, durch eine eigene Wissenseinheit repräsentiert wird, d.h. es muß eine Produktion anwendbar sein, die eine entsprechende Teilobjektwissenseinheit erzeugt und 3. Jedem Objekt, das an der Relation beteiligt ist, eine bestimmte Rolle zugewiesen ist, d.h. eine Produktion zur Generierung einer Rollenhypothese muß anwendbar sein. Alle drei Lernziele können in der Systemarchitektur gleichartig behandelt werden. Exemplarisch wird im folgenden die Erzeugung von Objektwissenseinheiten detailliert beschrieben. Für die beiden anderen Lernziele werden nur die Unterschiede dargestellt. Alle drei Verfahren werden in den folgenden Algorithmen gemeinsam durch *Lerne_Regeln_für_Planagent* aufgerufen.

Erzeugung von Objektwissenseinheiten

Zunächst müssen für eine Relation die Bezugsobjekte festgestellt werden, zu denen die Relation eingenommen wurde. Diese können entweder vom Benutzer erfragt werden, oder alle vorhandenen Objektwissenseinheiten werden als potentielles Bezugsobjekt behandelt und jeweils untersucht, wie eine Relationsbeschreibung dafür generiert werden könnte (s. Abschnitt 6.3.2).

Für alle Bezugsobjekte müssen – falls noch nicht geschehen – eigene Objektwissenseinheiten erzeugt werden. Dies können auch Teilobjekte einer bestehenden Objektwissenseinheit sein, die für die Relation relevant sind. Diese Wissenseinheiten müssen bei zukünftigen

Aufgabenstellungen selbständig erzeugt werden. Für die Integration des Beispiels in die Wissensbasis können drei Fälle unterschieden werden:

1. Es gibt eine Produktion, die die Objektwissenseinheit erzeugen würde und anwendbar ist, aber sie wurde noch nicht angewendet (→ keine Modifikation)
2. Es gibt Produktionen, die die Objektwissenseinheit erzeugen würden, aber die Bedingungen sind nicht erfüllt (→ Generalisierung wie bei der internen Generalisierung)
3. Es gibt keine Produktionen, die die gewünschten Objektwissenseinheiten erzeugen würden (→ Neue Definition einer Produktion)

Zunächst wird untersucht, ob Fall 1 vorliegt, oder ob gemäß Fall 2 eine Generalisierung möglich ist. Falls die Kosten für eine Generalisierung bestehender Produktionen zu hoch sind, wird eine neue Produktion eingeführt. Diese Produktion erhält als Vorbedingungen die konjunktive Verknüpfung von Elementarbedingungen, die genau die aktuellen Hypothesen der Planwissenseinheit abdecken. Dadurch entsteht im Normalfall eine Produktion, die zu speziell ist. Sie kann aber in nachfolgenden Problemlösungsschritten durch interne oder erneute externe Generalisierung entsprechend verallgemeinert werden.

Algorithmus 6.5 (Lerne Erzeugung einer Objektwissenseinheit)

Eingabe: · Name des Teilobjektes T der Objektwissenseinheit O
Ausgabe: · Einbindung der besten Generalisierung bzw. der neuen Produktion in das Wissen des Planagenten
Variablen: · G ist die beste Generalisierungsmöglichkeit bereits bestehender Produktionen

funct *Lerne_Erzeugung_einer_Objektwissenseinheit*(T, O)
  $\mathcal{P}$:= {Produktionen, die das Teilobjekt T von O erzeugen würden}
  G := *Generalisiere*$(\mathcal{P}$, Aktuelle Plan-WE$)$
  if $G_{Prod} \neq nil$ **then**
    (* entweder existiert bereits passende Produktion oder Generalisierung war möglich *)
    Wende die Produktion G_{Prod} und alle Produktionen in $G_{Erz.Prod}$ an
  else (* Neue Produktion erzeugen *)
    $\mathcal{B} := \{\}$
    A := 'Neue Objekt-WE T mit Bezug zu O'
    for $H \in$ Aktuelle Plan-WE **do**
      if ($Typ(H) \neq$ 'Anfrage') $\wedge$ ($Typ(H) \neq$ 'Nachricht') **then**
        $NB := H$ (* $NB = (T, E, P, N, W)$ *)
        if $\exists R \in$ Aktuelle Plan-WE : R ist Rollenhypothese für den Erzeuger von NB **then** E := 'Rolle:<R>' mit diesem Rollennamen
        else
          E := 'Var:<Name>' mit neuem Namen
          (* die Variable wird für alle Hypothesen derselben WE verwendet *)
      $\mathcal{B} := \mathcal{B} \cup \{NB\}$
    $Prods(Planagent) := Prods(Planagent) \cup \{(\mathcal{B}, A)\}$

Zuweisung von Rollen

Alle an der Relation beteiligten Objekte müssen eine Rolle zugewiesen bekommen. Entweder wurde dem Objekt bereits eine Rolle zugeordnet, es wird eine Rolle vom System automatisch erzeugt, oder es wird ihm durch Benutzerinteraktion eine bekannte oder unbekannte Rolle zugewiesen.

Wurde dem Objekt bereits eine Rolle zugeordnet, ist an dieser Stelle keine Modifikation von Wissen notwendig. Ist noch keine Rolle zugeordnet, werden wie bei der Erzeugung einer Objektwissenseinheit sowohl eine Generalisierung der bestehenden Produktionen als auch die Definition einer neuen Produktion untersucht. Wurde dem Objekt automatisch oder durch Benutzerinteraktion eine neue, unbekannte Rolle zugewiesen, wird in jedem Fall eine neue Produktion erzeugt. In die Vorbedingung einer neuen Produktion gehen alle Hypothesen ein, die Aussagen über aktuelle Objekte, Relationen oder den Namen der Aufgaben machen. Aktion der neuen Produktion ist die Zuweisung der gewünschten Rolle.

Das tatsächliche Vorgehen ist dann analog zu den entsprechenden Fällen zur Erzeugung oder Generalisierung von Produktionen für Teilobjekte; die dadurch definierte Funktion heißt *Lerne_Zuweisung_einer_Rolle*.

Erzeugen einer gewünschten Relation als Teilziel

Das Vorgehen ist analog zu den beiden anderen Lernzielen, wobei die gewünschte Aktion die Erzeugung einer Relationswissenseinheit für die vorgegebene Relation ist. Auch hierfür lassen sich die drei Fälle unterscheiden, daß bereits eine Produktion zur Erzeugung vorhanden ist, daß eine Generalisierung möglich ist oder daß eine neue Produktion erzeugt werden muß. Die entsprechende Funktion heißt *Lerne_Erzeugung_eines_Teilziels*.

Beispiel für eine neu generierte Produktion zur Erzeugung eines Teilziels:

```
WENN
  (((Plan-WE (Var:3784)) (Plan-Name Greife-Objekt))
  ∧ ((Plan-WE (Var:3784)) (Relation3786-Obj3787 'beliebig'))
  ∧ ((Relations-WE (Var:3785)) (Aktuelle-Relation (Oberhalb (Greifobj.))))
  ∧ ((Objekt-WE (Var:3787)) (Länge 50.0))
  ∧ ((Objekt-WE (Var:3787)) (Objektname Objekt1))
  ∧ ((Objekt-WE (Var:3787)) (Min-Y 80.0))
  ∧ ((Objekt-WE (Var:3787)) (Max-Y 140.0))
  ∧ ((Objekt-WE (Var:3787)) (Min-X 60.0))
  ∧ ...
DANN
  Neue Relations-WE Relation3786 mit Bezug zu (Relation3786-Obj3787)
```

Dabei ist `Relation3786-Obj3787` eine Rolle, die als neue Rollenhypothese für die vorliegende Objektwissenseinheit erzeugt wird.

6.3.2 Lernverfahren für Relationsbeschreibungen (Relationsagent)

Ein Teilproblem bei der Integration einer Benutzervorführung in das Systemwissen ist, die Semantik der vom Benutzer verwendeten Teilzielbeschreibungen (Relationen) zu erlernen. Bei verschiedenen Vorführungen kommt ein Teilziel (z.B. oberhalb eines Objekts) in unterschiedlichsten Ausprägungen vor. Zunächst ist ein Teilziel im allgemeinen nicht nur in einem einzigen Punkt, sondern in einem bestimmten Bereich gültig, aus dem sukzessive Punkte als Beispiele vorgegeben werden. Daneben ist eine Relationsbeschreibung aber wesentlich abhängig von bestimmten Objektmerkmalen, die von Beispiel zu Beispiel unterschiedlich sein können. Das heißt mit der Zeit muß für diese verschiedenen Fälle die notwendige Zahl von Beschreibungen entwickelt werden.

In allgemeinster Form läßt sich das Problem wie folgt beschreiben: Gegeben die Menge aller möglichen Objektmerkmale $\{M_1, M_2, \ldots, M_n\}$, die in der Realität erst sukzessive bekannt wird, sowie eine Menge von n Beispielen für einen Zielpunkt und die vorliegenden Objektmerkmale: $\{(M_{i_{11}}, M_{i_{12}}, \ldots, M_{i_{1m_1}}, Z_1), (M_{i_{21}}, M_{i_{22}}, \ldots, M_{i_{2m_2}}, Z_2), \ldots\}$. Gesucht sind eine oder mehrere Beschreibungen für den Zielpunkt oder einen Bereich für die Zielpunkte, d.h. beispielsweise im Idealfall die Funktion $f(M_1, \ldots, M_n)$, die aus den Objektmerkmalen das entsprechende Z ableitet. Im allgemeinen Fall ist der Zielpunkt ein Vektor, z.B. bestehend aus den verschiedenen Koordinaten, und für jedes Element des Vektors muß eine Beschreibung abgeleitet werden.

Normalerweise ist es nicht möglich, eine exakte Funktion abzuleiten, da die Beispiele aus Vorführungen des Benutzers stammen und oft tatsächlich ein Gültigkeitsbereich vorliegt. Aus diesem Grund müssen die Methoden zur Beschreibung von Gültigkeitsbereichen, wie sie in Kapitel 5.5 erläutert wurden, entsprechend eingesetzt werden. Im folgenden wird die Darstellung durch Gültigkeitsintervalle untersucht. Zwar basieren die entwickelten Verfahren im Kern jeweils auf der Intervalldarstellung, die prinzipiellen Probleme und die behandelten unterschiedenen Varianten treten aber bei allen Darstellungsformen gleichermaßen auf.

Die Lernaufgabe des Relationsagenten besteht darin, eine Abhängigkeit der Parameter des Gültigkeitsbereichs (hier Intervallgrenzen) von einer Menge von Objektmerkmalen zu bestimmen. Formal läßt sich also das hier untersuchte Lernen von Relationsbeschreibungen wie folgt definieren:

Definition 6.10 (Lernen von Relationsbeschreibungen) *Gegeben sei eine Menge von Objektmerkmalen $\mathcal{M} = \{M_1, \ldots, M_n\}$ und eine Menge von Beispielen $\mathcal{B} = \{(M_{i_{11}}, M_{i_{12}}, \ldots, M_{i_{1m_1}}, Z_1), (M_{i_{21}}, M_{i_{22}}, \ldots, M_{i_{2m_2}}, Z_2), \ldots\}$. Aufgabe des* ***Lernens von Relationsbeschreibungen*** *ist die Ableitung einer expliziten funktionalen Beziehung für jede Koordinate x_k des Zielpunktes der Form $x_k \in [f_k(M_1, \ldots, M_n), g_k(M_1, \ldots, M_n)]$, so daß für möglichst alle Beispiele die vorgegebenen Zielpunkte innerhalb des dadurch definierten Bereichs liegen und der Bereich gleichzeitig möglichst klein ist.*

Als Lösung wurde ein nicht-inkrementelles Verfahren entwickelt, d.h. alle Beispiele für eine bestimmte Relation werden erneut betrachtet, wenn eine neue Beschreibung für diese Relation abgeleitet werden soll. Das Verfahren basiert auf einer Generate-and-Test-Strategie

(vgl. [Hei89]), d.h. daß eine Aufzählung der möglichen Varianten vorgenommen und jeweils deren Güte bestimmt wird. Aus der Literatur bekannte Verfahren zur funktionalen Induktion wie BACON oder ABACUS (s. Seite 34) setzen eine vollständige Tabellierung von Beispielen voraus, um durch vollständige Suche oder durch Gradientenverfahren – bei gegebener Funktionsstruktur – eine geeignete Beschreibung abzuleiten, diese vollständige Tabellierung ist hier nicht gegeben.

Zur Bestimmung der besten Beschreibung der Intervallgrenzen werden alle vorliegenden Objektmerkmale mit einer Menge von Skalierungsfaktoren, z.B. $\{-2,-1,-0.5,0.5,1,2\}$, multipliziert und die Ergebnisse additiv verknüpft. Damit ist die Menge der exakt ableitbaren Funktionen auf Linearkombinationen der Objektmerkmale eingeschränkt: $f = s_1M_1 + s_2M_2 + \ldots + s_nM_n$, wobei s_i ein Skalierungsfaktor ist. Aus praktischen Gründen kann die maximale Anzahl von additiven Komponenten dabei begrenzt werden. Jede dieser Funktionen wird durch eine Bewertungsfunktion evaluiert und die günstigste als Beschreibung gewählt. Als Bewertungsfunktion kann beispielsweise der Abstand von den Intervallgrenzen in allen Beispielen verwendet werden. Allerdings wäre dann neben der Betrachtung aller Merkmalskombinationen auch noch die Untersuchung aller möglicher Intervallgrenzenkombinationen notwendig.[1] Aus diesem Grund wird hier von einer separaten Berechnung der beiden Grenzen ausgegangen und das Distanzmaß für die untere Intervallgrenze wie folgt definiert:

$$Distanz(f,\mathcal{B}) = \frac{\sum_{B\in\mathcal{B}} d(B)}{\mid \mathcal{B} \mid}, \text{ mit } d(B) = \begin{cases} \mid Ziel(B) - f(B) \mid & , \text{ falls } f(B) \leq Ziel(B) \\ \infty & , \text{ sonst} \end{cases}$$

mit $\mathcal{B}$ ist die Menge der Beispiele, Ziel(B) war der in B angegebene Zielpunkt und f ist die zu bewertende Intervallgrenze. Die Berechnung für die obere Intervallgrenze erfolgt analog mit der Unterscheidung ob $g(b) \geq Ziel(b)$. Bei Winkelangaben muß zusätzlich modulo 360° gerechnet werden. Mit dieser Bewertungsfunktion werden Funktionsbeschreibungen vermieden, bei denen nicht alle Beispiele innerhalb des aufgebauten Intervalls liegen. Ebenso wäre alternativ eine Bewertung denkbar, die eine weniger negative Bewertung für Ausreißer vornimmt.

Algorithmus 6.6 (Erzeuge Intervallgrenze)

Eingabe: · Objektmerkmale $\mathcal{O}$, die berücksichtigt werden sollen
· Beispiele $\mathcal{B}$
· A ist die Art der Schranke ('untere'/'obere')
Ausgabe: · Funktionsbeschreibung für die Schranke
Variablen: · $\mathcal{K}$ ist die Menge aller Kombinationen von skalierten Objektmerkmalen
· IG ist ein konkreter Wert für eine Intervallgrenze

[1] Für den Test der Gültigkeit einer Relation wird in Definition 6.11 ein stark erweitertes Maß eingesetzt, da in diesem Fall eine sehr viel kleinere Menge von bereits bekannten Relationen untersucht und nicht die Kombination verschiedener Intervallgrenzen betrachtet werden muß.

funct *Erzeuge_Intervallgrenze*$(\mathcal{O}, \mathcal{B}, A)$
$BI_{Bew} := maxreal$; $BI_{Funk} := nil$
$\mathcal{K}$:= {Kombinationen von skalierten Objektmerkmalen $\mathcal{O}$ der Länge $\leq$ Komplexitätsbeschränkung unter Verwendung der gegebenen Skalierungsfaktoren}
for $k \in \mathcal{K}$ **do**
Bewertung := 0
for $b \in \mathcal{B}$ **do** (* über alle Beispiele *)
$IG := k(b)$
if $(A =$ 'untere' $\wedge\ IG \leq Ziel(b)) \vee (A =$ 'obere' $\wedge\ IG \geq Ziel(b))$ **then**
Bewertung := *Bewertung*+ | *Ziel(b)* − *IG* |
else
Bewertung := *maxreal*
exit Schleife über b
if *Bewertung* < BI_{Bew} **then**
BI_{Bew} := *Bewertung*; $BI_{Funk} := k$
return (*BI*)

Beispiel für eine Funktionsinduktion:

Gegeben sind für jedes Objekt Hypothesen über Min-x, Max-x, Min-y, Max-y, Höhe sowie die drei Koordinaten des gewünschten Zielpunktes. Funktionen, die daraus induziert werden, sind:

Beispiele für Oberhalb: x = [Min-x; Max-x], y analog, z = [1,0·Höhe; 2,0·Höhe]
Beispiele für Rechts neben: x = [1,0·Max-x; 1,0·Max-x + 1,0·Höhe],
y = [0,5·Min-y + 0,5·Max-y; 0,5·Min-y + 0,5·Max-y],
z = [0,5·Höhe - 0,5·Min-y; 1,0·Höhe]

Das angegebene Verfahren zur funktionalen Induktion läßt sich in zwei Richtungen verbessern, zum einen über die Betrachtung von komplexeren Funktionsstrukturen, d.h. neben Addition und Subtraktion von skalierten Objektmerkmalen auch die Verarbeitung von Konstanten, allgemeinen Multiplikationen, trigonometrischen Funktionen und nichtlinearen Funktionen wie Minimum- oder Maximumbildung. Zum anderen wäre ein deutlicher Effizienzgewinn möglich, wenn durch ein Verfahren der Suchraum eingeschränkt werden könnte. Dazu sind sowohl automatische Verfahren (z.B. Betrachtung von Monotonieeigenschaften) aber vor allem auch eine Interaktion mit dem Benutzer möglich. Dieser könnte bestimmte Merkmale oder bereits Teilfunktionen auswählen, aus denen die Intervallbeschreibung zusammengesetzt werden soll.

Wird vom Benutzer ein neues Beispiel für eine bestimmte Relation angegeben, so können, aufbauend auf der beschriebenen Möglichkeit, eine funktionale Gültigkeitsbedingung aus einer Menge von Beispielen zu induzieren, verschiedene Fälle unterschieden werden, wie bestehende Produktionen modifiziert oder neue erzeugt werden müssen.

Eine Produktion zur Berechnung eines Gültigkeitsbereichs hat wie in Kapitel 5.5 gezeigt folgende Struktur:

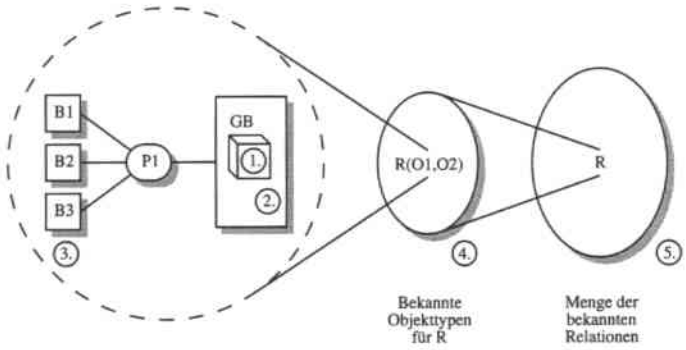

Abbildung 6.3: Varianten für das Lernen einer Relationsbeschreibung

```
WENN
  ((Relations-WE (Var:R)) (Relationsname Relationsname))
  ∧ ((Relations-WE (Var:R)) (Attribut11 Wert11))
                  ⋮
  ∧ ((Objekt-WE (Var:O1)) (Objekttyp Objekttyp1))
  ∧ ((Objekt-WE (Var:O1)) (Attribut21 Wert21))
                  ⋮
  ∧ ((Objekt-WE (Var:O2)) (Objekttyp Objekttyp2))
  ∧ ((Objekt-WE (Var:O2)) (Attribut31 Wert31))
                  ⋮
DANN
  Berechne Gültigkeitsbereich
    Koord muß im Intervall von Funktion1 bis Funktion2 liegen
```

Damit müssen fünf Varianten für mögliche Lernschritte behandelt werden, die in Abbildung 6.3 im Überblick gezeigt und nachfolgend einzeln untersucht werden.

1. Der Zielpunkt liegt innerhalb des Gültigkeitsbereichs der Relationsproduktion, die in der aktuellen Situation angewendet würde: In diesem Fall ist keine Modifikation des Wissens des Relationsagenten notwendig.

2. Der Zielpunkt liegt außerhalb des Gültigkeitsbereichs der Relationsproduktion, die in der aktuellen Situation angewendet würde: In diesem Fall wird eine Intervallgrenze unter- bzw. überschritten. Für diese Grenze muß eine neue Funktionsbeschreibung bestimmt werden. Zunächst wird versucht, eine neue Beschreibung mit denselben Objektmerkmalen zu finden. Ist dies nicht möglich, wird die Schnittmenge aller Merkmale jedes Beispiels bestimmt und versucht, damit eine Beschreibung zu generieren. Dabei müssen eventuell die Vorbedingungen der Produktion zur Berechnung des Gültigkeitsbereichs angepaßt werden. Ist auch dies nicht möglich, wird eine alternative Relationsbeschreibung für das neue Beispiel eingeführt.

Algorithmus 6.7 (Lerne Gültigkeitsbereich für ungültige Koordinate)

Eingabe: · Relation R
· Ein neues Beispiel B für die Relation R
Ausgabe: · Anpassung der Wissensbasis des Relationsagenten (Korrektur einer existierenden Produktion bzw. Hinzufügen einer neuen)
Variablen: · NB ist eine neue Beschreibung für eine Bereichsgrenze
· K, $\mathcal{P}$, P, $\mathcal{O}$, $\mathcal{B}$, *Schnitt*, NB^u, NB^o wie unten angegeben

funct *Lerne_Relation_bei_ungültiger_Koordinate* (R, B)
for $K \in$ {ungültige Koordinaten von R bzgl. B} **do**
$\mathcal{P}$:= {Produktion $\in$ Relationsagent | Berechnung für Koordinate K von R}
P := Stärkste Produktion aus $\mathcal{P}$, die anwendbar ist
$\mathcal{O}$:= {Objektmerkmale, die in P verwendet werden}
$\mathcal{B}$:= {Beispiele für P} $\cup$ {B}
NB := *Erzeuge_Intervallgrenze* $(\mathcal{O}, \mathcal{B}$, ungültige Grenze von P für $B)$
if $(NB_{Funk} \neq nil) \wedge (NB_{Bew} <$ Max. Abstand) **then**
Ersetze die ungültige Grenze von P durch NB_{Funk}
else
Schnitt := Bilde die Schnittmenge über die Merkmale aus allen $b \in \mathcal{B}$
NB := *Erzeuge_Intervallgrenze* (*Schnitt*, $\mathcal{B}$, ungültige Grenze von P für B)
if $(NB_{Funk} \neq nil) \wedge (NB_{Bew} <$ Max. Abstand) **then**
Ersetze die Produktion P durch entsprechende Produktion mit NB_{Funk} für die verletzte Grenze und den dadurch notwendigen Vorbedingungen
else
$NB^{u/o}$:= *Erzeuge_Intervallgrenze* (Objektmerkmale von B, B, 'untere'/'obere' Grenze von P für B)
Erzeuge neue Produktion mit entsprechender Vorbedingungsmenge, der Beispielmenge $\{B\}$ und den Grenzen NB^u_{Funk} und NB^o_{Funk}

3. Der Gültigkeitsbereich ist nicht berechenbar, da die Produktion nicht anwendbar ist: Dieser Fall tritt dann auf, wenn entweder Teile der Vorbedingung der entsprechenden Produktion nicht durch die aktuellen Hypothesen erfüllt werden oder Merkmale nicht bekannt sind, die in der Funktionsbeschreibung verwendet werden.

   Zunächst wird versucht, ob durch eine Generalisierung der Vorbedingungen einer Produktion die Koordinate berechnet werden kann. Dabei wird die Generalisierungsmöglichkeit ausgeschlossen, die zum Wegfall einer Elementarbedingung führt, die eine Aussage über ein Objektmerkmal macht, das in den Funktionsbeschreibungen verwendet wird.

   Ist eine solche Generalisierung nicht möglich, wird versucht, eine völlig neue Beschreibung zu bestimmen, die sowohl das neue Beispiel als auch alle alten Beispiele abdeckt. Für beide Intervallgrenzen wird unter Berücksichtigung der Schnittmenge aller vorkommenden Merkmale eine entsprechende Funktionsbeschreibung gesucht. Ist auch dies nicht möglich, wird wieder eine alternative Relationsbeschreibung ausschließlich für das neue Beispiel eingeführt.

Algorithmus 6.8 (Lerne Gültigkeitsbereich für Koordinate, die nicht berechnet werden kann)

Eingabe: · Relation R
· Ein neues Beispiel B für die Relation R
Ausgabe: · Anpassung der Wissensbasis des Relationsagenten (Korrektur einer Produktion bzw. Hinzufügen einer neuen)
Variablen: · K, $\mathcal{P}$, P, $\mathcal{O}$, $\mathcal{B}$, G, *Schnitt*, UG, OG, NB^u, NB^o wie unten angegeben

funct *Lerne_Relation_bei_nicht_berechenbarer_Koordinate* (R, B)
  for $K \in$ {ungültige Koordinaten von R bzgl. B} **do**
    gleiches Vorgehen wie in *Lerne_Relation_bei_ungültiger_Koordinate*
  for $K \in$ {nicht berechenbare Koordinaten von R bzgl. B} **do**
    $\mathcal{P}$:= {Produktion $\in$ Relationsagent | Berechnung für Koordinate K von R}
    P := Stärkste Produktion aus $\mathcal{P}$, die anwendbar ist
    $\mathcal{B}$:= {Beispiele für P}
    G := *Generalisiere* ($\mathcal{P}$, aktuelle Relationswissenseinheit)
    (* Einschränkung s. Text *)
    if $G_{Prod} \neq nil$ **then**
      Lerne_Relation_bei_ungültiger_Koordinate (R, B)
    else (* Bestimme beste Schnittmenge *)
      Schnitt := Bilde die beste Schnittmenge über die Merkmale aus allen $b \in \mathcal{B}$ bzgl. aller möglicher Zuordnungen der aktuellen Objektwissenseinheiten für das neue Beispiel zu einem der referenzierten Objekte in den alten Beispielen
      $NG^{u/o}$:= *Erzeuge_Intervallgrenze* (*Schnitt*, $\mathcal{B} \cup \{B\}$, 'untere/obere Grenze')
      if $(NG^u_{Funk} \neq nil) \wedge (NG^o_{Funk} \neq nil) \wedge (NG^u_{Bew} <$ Max. Abstand$) \wedge$ $(NG^o_{Bew} <$ Max. Abstand$)$ **then**
        Ersetze die Produktion P durch eine entsprechende Produktion mit NG^u_{Funk} und NG^o_{Funk} als Grenzen und den notwendigen Vorbedingungen
      else
        $NB^{u/o}$:= *Erzeuge_Intervallgrenze* (Objektmerkmale von B, B, 'untere/obere Grenze')
        Erzeuge neue Produktion mit entsprechender Vorbedingungsmenge, der Beispielmenge $\{B\}$ und den Grenzen NB^u_{Funk} und NB^o_{Funk}

4. Bekannte Relation zu neuen Objekttypen: Zunächst wird untersucht, ob durch Generalisierung über den Objekttyp eine der für die Relation vorhandenen Produktionen anwendbar gemacht werden kann. Ist dies nicht möglich, wird für das neue Beispiel eine neue Produktion erzeugt.

Algorithmus 6.9 (Lerne Gültigkeitsbereich für neue Objekttypen)

Eingabe: · Relation R
· Ein neues Beispiel B für die Relation R
Ausgabe: · Anpassung der Wissensbasis der Relationsagenten (Korrektur einer Produktion bzw. Hinzufügen einer neuen)
Variablen: · $\mathcal{P}$, G, NG^u, NG^o wie unten angegeben

funct *Lerne_Relation_zu_neuen_Objekttypen* (R, B)
  for $K \in$ {Koordinaten von R} **do**
    $\mathcal{P}$:= {Produktion $\in$ Relationsagent | Berechnung für Koordinate K von R}
    G := *Generalisiere* ($\mathcal{P}$, aktuelle Relationswissenseinheit)
    if $G_{Prod} \neq nil$ **then**
      Lerne_Relation_bei_nicht_berechenbarer_Koordinate (R, B)
    else
      $NG^{u/o}$:= *Erzeuge_Intervallgrenze* (Objektmerkmale von B, $\{B\}$, 'untere/obere Grenze')
      Erzeuge neue Produktion mit entsprechender Vorbedingungsmenge, der Beispielmenge $\{B\}$ und den Grenzen NG^u_{Funk} und NG^o_{Funk}

5. Unbekannte Relation: Ist eine Relation unbekannt, wird für jede Koordinate eine entsprechende neue Produktion erzeugt (*Lerne_unbekannte_Relation*).

6.3.3 Integration einer Benutzervorführung

Nachdem in den letzten beiden Abschnitten getrennt die Lernverfahren für den Plan- und den Relationsagenten untersucht wurden, die bei der Integration einer Benutzervorführung benötigt werden, soll in diesem Abschnitt das Zusammenspiel und das vollständige Vorgehen bei der externen Generalisierung beschrieben werden.

Die Interaktion mit dem Benutzer beginnt wie in Definition 6.9 erläutert entweder, wenn im aktuellen Systemzustand keine weiteren Produktionen anwendbar sind und der Benutzer ein nächstes Teilziel vorgeben will oder wenn vom System ein Teilziel eingenommen wurde und der Benutzer Korrekturen vornehmen will, indem er den Zielpunkt korrigiert, die zuletzt eingenommene Relation ersetzt oder wenn er die nächste einzunehmende Relation angibt.

Der Benutzer gibt in allen Fällen zunächst nur einen zu erreichenden Zielpunkt an – damit ist auch die Integration eines automatischen Segmentierungswerkzeuges einer vollständigen Vorführung möglich. Aus der Angabe des Zielpunktes muß das System – eventuell durch zusätzliche Kommunikation mit dem Benutzer – die entsprechende Relation ableiten. Für die Vorsortierung oder automatische Auswahl einer Relationsbeschreibung, die am besten auf den aktuellen Zustand paßt, ist eine vergleichende Bewertung der bereits bekannten Relationsbeschreibungen notwendig. Diese Bewertung basiert auf dem Abstand einer Relationsbeschreibung von einem Zielpunkt. In der folgenden Definition werden translatorische und rotatorische Koordinaten unterschiedlich behandelt.

Definition 6.11 (Abstand einer Relationsbeschreibung von einem Zielpunkt) *Der* ***Abstand*** *einer Relationsbeschreibung* $RB = ([f_x, g_x], [f_y, g_y], \ldots, [f_d, g_d])$ *von einem Zielpunkt* $Z = (Z_x, \ldots, Z_d)$ *ist definiert über die Summe der Abstände in den einzelnen Koordinaten:*

$$Abstand(RB, Z) = \sqrt{Abstand_x^2 + \ldots + Abstand_d^2}.$$

Dabei ist $Abstand_K$ *folgendermaßen definiert – die Indizierung von Abstand,* f, g *und* Z *mit der Koordinate* K *ist der Übersichtlichkeit halber weggelassen:*

$$Abstand = \frac{\mid \frac{f(Z)+g(Z)}{2} - Z \mid}{\mid g(Z) - f(Z) \mid}$$

für translatorische Koordinaten und

$$Abstand = \frac{\min(\mid (MP(f(Z), g(Z)) - Z \mid, \mid\mid MP(f(Z), g(Z)) - Z \mid - 360° \mid}{IB(f(Z), g(Z))}$$

für rotatorische Koordinaten, mit dem Mittelpunkt MP *und der Intervallbreite* IB *definiert als*

$$MP(f(Z), g(Z)) = \begin{cases} \frac{f(Z)+g(Z)}{2} & , falls\ f(Z) \leq g(Z) \\ \frac{f(Z)+g(Z)-360°}{2} & , falls\ f(Z) + g(Z) \geq 360° \wedge f(Z) > g(Z) \\ \frac{f(Z)+g(Z)+360°}{2} & , sonst \end{cases}$$

und

$$IB(f(Z), g(Z)) = \begin{cases} g(Z) - f(Z) & , falls\ f(Z) \leq g(Z) \\ 360° - f(Z) + g(Z) & , sonst \end{cases}$$

Für den Spezialfall, daß die Intervallbreite 0 ist, wird der nicht normierte Abstand des gegebenen Punktes vom Mittelpunkt des Intervalls verwendet.

Aufgrund der vorgenommen Auswahl der vorgeführten Relation ergeben sich unter Berücksichtigung des aktuellen Systemzustandes die in Abbildung 6.4 angegebenen Alternativen. Prinzipiell lassen sich die beiden Fälle unterscheiden, daß die einzunehmende Relation dem System bekannt ist oder nicht. Im ersten Fall kann die Einnahme der Relation als nächste oder nachfolgende Relation auch vom System geplant sein, oder sie ist in der vorliegenden Situation nicht vorgesehen. Dazu muß dann überprüft werden, ob die einzunehmende Relation ausschließlich bereits vorhandene Objektwissenseinheiten benötigt, oder ob sie sich auch auf Teilobjekte der Objekte bezieht. Dabei muß jeweils unterschieden werden, ob sich der aktuelle TCP im Gültigkeitsbereich der Relation befindet oder nicht, oder ob der Gültigkeitsbereich mit den vorliegenden Daten noch nicht berechnet werden kann.

Im zweiten Fall der unbekannten Relation läßt sich unterscheiden, ob die Relation vom Namen her bekannt ist, aber noch keine Beschreibung für die aktuellen Objekttypen vorliegt, oder ob die Relation auch vom Namen her unbekannt ist. Für diese verschiedenen Varianten sind entsprechend unterschiedliche Lernvorgänge durchzuführen, die im folgenden aufgeführt werden:

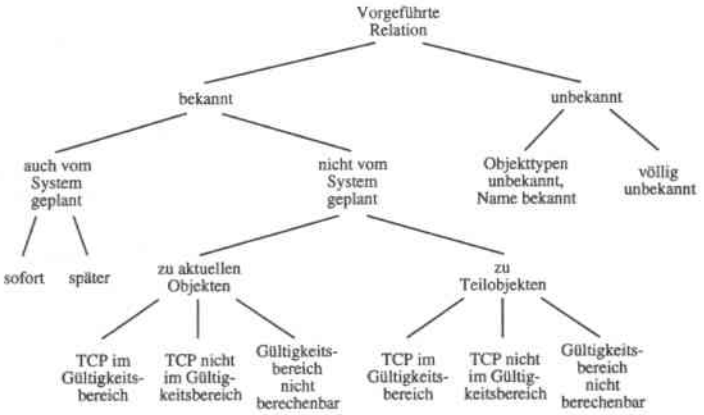

Abbildung 6.4: Alternativen für eine vorgeführte Relation

1. Relation ist bekannt

   1.1 und auch vom System geplant, d.h. es gibt bereits eine Relationswissenseinheit zur Erreichung dieses Teilziels: An dieser Stelle ist kein eigentlicher Lernschritt notwendig, da nicht definitiv gesagt werden kann, welche Relation dann tatsächlich als nächste realisiert wird. Bei der Fortsetzung der Problemlösung können dann aber Relationen eingenommen werden, bei denen es zu einer Korrektur kommen muß.

   1.2 aber nicht vom System geplant

      1.2.1 zu aktuellen Objekten: Für den Planagenten muß das Wissen so verändert werden, daß die gewünschte Relation in der aktuellen Situation eingenommen wird, d.h. es müssen Produktionen zur Zuweisung von Rollen und zur Erzeugung der Relation als Teilziel gelernt werden. Für den Relationsagenten sind die in Abschnitt 6.3.2 aufgeführten Fälle 1 bis 3 möglich:

      - Zielpunkt liegt innerhalb des Gültigkeitsbereichs: Keine Modifikation des Wissens des Relationsagenten notwendig.
      - Zielpunkt liegt außerhalb des Gültigkeitsbereichs: Die Relationsbeschreibung muß entsprechend angepaßt werden.
      - Der Gültigkeitsbereich ist nicht berechenbar: Bestimme anwendbare Produktionen, die einen Gültigkeitsbereich berechnen

      1.2.2 zu Teilobjekten: Für den Planagenten müssen zusätzlich zu den in 1.2.1 genannten Lernschritten auch Produktionen gelernt werden, durch die die entsprechenden Teilobjektwissenseinheiten generiert werden. Für den Relationsagenten wird zunächst, wie in Abschnitt 6.3.2, Fall 4 angegeben, versucht, bereits bestehende Relationsbeschreibungen für die Typen der Teilobjekte zu verallgemeinern. Anschließend wird analog zu den drei Varianten bei aktuellen Objekten ein Lernvorgang durchgeführt.

2. Relation ist unbekannt

   2.1 Relationsname ist zwar bekannt, aber die Objekttypen sind unbekannt. Für den Planagenten wird eine Behandlung wie bei den bekannten Relationen durchgeführt. Für den Relationsagenten wird gemäß Abschnitt 6.3.2, Fall 5 ein Satz neuer Produktionen erzeugt, die diese Objekttypen abdecken.

   2.2 Relationsname ist auch unbekannt: Sowohl für den Plan- als auch für den Relationsagenten müssen alle angegebenen Lernvorgänge durchgeführt werden.

Eine genaue Beschreibung des Ablaufs bei den Lernvorgängen in Abhängigkeit der Art der vorgeführten Relation gibt der folgende Algorithmus:

Algorithmus 6.10 (Lerne vorgeführtes Teilziel)

Eingabe: · Zielpunkt Z
Ausgabe: · Anpassung der Wissensbasen des Plan- und Relationsagenten
Variablen: · Nächste einzunehmende Relation NR
· Menge von Teilobjekten $\mathcal{T}$
· $\mathcal{R}_{möglich}$, $\mathcal{R}_{aktuell}$, $\mathcal{R}_{berechenbar}$, $\mathcal{R}_{gültig}$, $\mathcal{R}_{nicht\ gültig}$ wie angegeben

funct *Lerne_vorgeführtes_Teilziel*(Z)
  $\mathcal{R}_{möglich}$:= Menge aller möglichen Relationen für die im System vorliegenden Objektwissenseinheiten
  $\mathcal{R}_{aktuell}$:= Menge der aktuell eingenommenen gültigen Relationen
  $\mathcal{R}_{berechenbar} := \{R \in \mathcal{R}_{möglich} \setminus \mathcal{R}_{aktuell} \mid \forall$ Koordinaten kann der Gültigkeitsbereich berechnet werden$\}$
  $\mathcal{R}_{gültig} := \{R \in \mathcal{R}_{berechenbar} \mid$ Zielpunkt liegt im Gültigkeitsbereich$\}$
  NR := Ordne $\mathcal{R}_{gültig}$ nach einem Gütekriterium (z.B. Abstand vom Zielpunkt) und wähle eine aus (interaktiv oder die beste)
  if $NR \in \{$Relationswissenseinheit $\in$ System $\mid$ Relation noch nicht eingenommen$\}$ **then return** () (* Relation ist auch vom System geplant *)
  elsif $NR \neq nil$ **then**
    for $O \in \{$Objekte von $NR\}$ **do** *Lerne_Zuweisung_einer_Rolle*(O)
    Lerne_Erzeugung_eines_Teilziels(NR)
  else
    $\mathcal{R}_{nicht\ gültig} := \mathcal{R}_{berechenbar} \setminus \mathcal{R}_{gültig}$
    NR := Ordne $\mathcal{R}_{nicht\ gültig}$ nach einem Gütekriterium und wähle eine aus
    if $NR \neq nil$ **then**
      for $O \in \{$Objekte von $NR\}$ **do** *Lerne_Zuweisung_einer_Rolle*(O)
      Lerne_Erzeugung_eines_Teilziels(NR)
      Lerne_Relation_bei_ungültiger_Koordinate$(NR,$ (Menge der Objektmerkmale, $Z))$
    else
      NR := Ordne $\mathcal{R}_{möglich} \setminus \{\mathcal{R}_{aktuell} \cup \mathcal{R}_{berechenbar}\}$ und wähle eine aus
      if $NR \neq nil$ **then**
        for $O \in \{$Objekte von $NR\}$ **do** *Lerne_Zuweisung_einer_Rolle*(O)
        Lerne_Erzeugung_eines_Teilziels(NR)
        Lerne_Relation_bei_nicht_berechenbarer_Koordinate$(NR,$

(Menge der Objektmerkmale, Z))
else (* Relation zu Teilobjekten? *)
$\mathcal{T}$:= {An der Relation beteiligte Teilobjekte, die noch keine eigene Wissenseinheit besitzen}
for $T \in \mathcal{T}$ **do**
Lerne_Erzeugung_einer_Objektwissenseinheit(T, zugehöriges Objekt)
Bestimme analog zum obigen Vorgehen die Relation NR
if $NR \neq nil$ **then**
Bestimme analog zu oben die Art der Relation und verwende die entsprechenden Lernverfahren
else (* unbekannte Relation *)
Name := Wähle einen bekannten oder neuen Relationsnamen aus
Lerne_Regeln_für_Planagent(*Name*)
if *Name* ist bekannt **then**
Lerne_Relation_zu_neuen_Objekttypen(*Name*,
(Menge aller Objektmerkmale, Z))
else
Lerne_unbekannte_Relation(*Name*,
(Menge aller Objektmerkmale, Z))

Mit den in diesem Abschnitt entwickelten Methoden und zusätzlich der im nächsten Abschnitt behandelten Spezialisierung ist nun das Vorgehen beim externen Lernen als Ganzes beschreibbar. Beim Korrigieren des zuletzt eingenommenen Zielpunktes wird, falls notwendig, eine Korrektur der vorliegenden Relationsbeschreibung vorgenommen. Soll die zuletzt eingenommene Relation ersetzt werden, so muß zunächst eine Spezialisierung durchgeführt werden, damit in zukünftigen Situationen diese Relation nicht abgeleitet wird. Anschließend wird das zu erreichende Teilziel mit den oben angegebenen Verfahren gelernt. Dabei werden sowohl die nötigen Lernschritte für den Planagenten als auch für den Relationsagenten durchgeführt. Ein sehr ähnliches Vorgehen findet bei der Integration einer nachfolgend einzunehmenden Relation statt. Das nächste vom System zu erreichende Teilziel wird abgeleitet und bei einer Abweichung von der Benutzervorführung zunächst eine Spezialisierung durchgeführt und anschließend, wenn nötig, das zu erreichende Teilziel eingelernt.

Algorithmus 6.11 (Externes Lernen)

Eingabe: · *Art* gibt den gewünschten Benutzereingriff an: 'Korrigiere Zielpunkt', 'Ersetze die zuletzt eingenommene Relation' oder 'Lerne nächste Relation'
· Z ist der vom Benutzer eingegebene Zielpunkt
Ausgabe: · Als Seiteneffekt werden die Wissensbasen der einzelnen Agenten des Systems modifiziert
Variablen: · Offene Relationswissenseinheiten *ORB*, d.h. Relationen, die zwar eingenommen werden sollen, aber noch nicht eingenommen sind
· Vom Benutzer gewünschte nächste Relation *GR*

```
funct Externes_Lernen(Art, Z)
  case Art of
    'Korrigiere_Zielpunkt' :
      RB := Zuletzt vollständig eingenommene und noch gültige Relation
      if RB ≠ nil then Lerne_Relation_bei_ungültiger_Koordinate(RB, Z)
    'Ersetze die zuletzt eingenommene Relation' :
      RB := Zuletzt vollständig eingenommene und noch gültige Relation
      if RB ≠ nil then
        P := Produktion, die zur Erzeugung von RB geführt hat
        Spezialisiere(P, erzeugende Hypothesen von RB, Restliche Hypothesen)
        Lerne_vorgeführtes_Teilziel(Z)
    'Lerne nächste Relation' :
      S := Bestimme nächste vom System einzunehmende Relation
      if S ≠ nil then
        GR := Abfrage, ob S die gewünschte Relation ist
        if GR = nil then
          Spezialisiere(Produktion, die zur Erzeugung von S geführt hat,
            Erzeugende Hypothesen, Restliche Hypothesen)
          Lerne_vorgeführtes_Teilziel(Z)
  return ()
```

6.4 Externe Spezialisierung

Eine externe Spezialisierung ist dann nötig, wenn ein signifikanter Unterschied zwischen einer Benutzeraktion und einer Systemaktion festgestellt wird. Dieser Fall tritt entweder auf, wenn während des Systemlaufs vom Benutzer eine eingenommene Relation explizit als falsch gekennzeichnet wird oder wenn das System ein Teilziel, d.h. eine Relation, herleitet, die nicht mit der nächsten Relation der gegebenen Vorführung übereinstimmt. Ausgangspunkt der Suche nach einer geeigneten Spezialisierung ist die Produktion, die die fehlerhafte Aktion hervorgerufen hat. Von da aus wird die kostengünstigste Spezialisierung einer der Produktionen gesucht, so daß in den betrachteten Systemzuständen die fehlerhafte Hypothese oder Wissenseinheit nicht abgeleitet würde. Die Lernaufgabe bei der externen Spezialisierung läßt sich also wie folgt formulieren:

Definition 6.12 (Lernaufgabe der externen Spezialisierung) *Finde die Produktion P eines Agenten $AG \in System$, die die kostengünstigste Spezialisierung aller Produktionen ermöglicht, für die gilt: $\exists$ Zustand in der Folge der durchlaufenen Systemzustände, in dem P angewendet wurde: Anwendbar$(P, Zustand)$ und für die Spezialisierung der Produktion gilt: $\neg$Anwendbar$(Spezialisierung(P), Zustand)$)*

Zunächst werden in Analogie zu den Definitionen über Generalisierungsmöglichkeiten die Spezialisierung einer Elementarbedingung und die Spezialisierung einer Produktion definiert.

Definition 6.13 (Spezialisierung einer Elementarbedingung) *Sei $B = (T_B, E_B, P_B, N_B, W_B)$ eine Elementarbedingung, $H = (T_H, E_H, P_H, N_H, W_H)$ eine Hypothese und die Erzeugerbeschränkung, der Pfad und der Attributname von H sei in den entsprechenden Elementen von B enthalten. Eine Elementarbedingung $S = (T_S, E_S, P_S, N_S, W_S)$ heißt dann* ***Spezialisierung*** *der Elementarbedingung B unter Berücksichtigung der Hypothese H, wenn eine der folgenden Aussagen gilt:*

1. *W_H beschreibt einen einfachen symbolischen Wert h, W_B beschreibt einen einfachen symbolischen Wert b und $h = b$, dann $W_S := \{$Menge aller Werte für das Attribut, die in einer Produktion vorkommen$\} \setminus \{h\}$. Falls kein weiterer Wert für das Attribut mehr vorkommt, ist dieser Fall nicht anwendbar.*

2. *W_H beschreibt einen einfachen symbolischen Wert h, W_B beschreibt eine Menge $\mathcal{B}$ und $h \in \mathcal{B}$, dann $W_S := \mathcal{B} \setminus \{h\}$.*

3. *W_H beschreibt einen einfachen numerischen Wert h, W_B beschreibt ein numerisches Intervall $[b_1, b_2]$ und $b_1 \leq h \leq b_2$, dann $W_S :=$ größtes Intervall in $\{[b_1, h(,)h, b_2]\}$.*

4. *W_H beschreibt ein numerisches Intervall $[h_1, h_2]$, W_B beschreibt ein numerisches Intervall $[b_1, b_2]$ und $b_1 \leq h_1 \leq b_2$ oder $b_1 \leq h_2 \leq b_2$, dann $W_S :=$ Größtes Intervall, das bei Ausschluß von $[h_1, h_2]$ übrig bleibt. Falls $b_1 = h_1$ und $b_2 = h_2$, ist dieser Fall nicht anwendbar.*

5. *W_H beschreibt eine Aufzählung $(h_1, \ldots, h_n)$, W_B beschreibt eine Aufzählung $(b_1, \ldots, b_n)$. Dann $W_S :=$ Mindestens eine der Komponenten der Aufzählung W_B ist spezialisiert.*

6. *Abstieg in einer Hierarchie (Taxonomie), die eine Aussage über die möglichen Attributwerte macht. Falls W_H kein Element der Taxonomie unterhalb von W_B ist, ist dieser Fall nicht anwendbar. Sonst ist W_S die Menge der allgemeinsten Knoten unterhalb von W_B, so daß W_H gerade ausgeschlossen wird.*

7. *Falls $W_B =$ „beliebig", dann wird analog zu Fall 1 eine Wertbelegung für W_S in den bereits bekannten Produktionen gesucht und der Wert W_H ausgeschlossen.*

Definition 6.14 (Spezialisierung einer Produktion) *Eine Produktion $P = (\mathcal{B}, A)$ kann auf zwei Arten spezialisiert werden, damit sie eine Menge von Hypothesen $\mathcal{H}$ nicht mehr erfüllt:*

1. *Nimm neue Elementarbedingungen zu $\mathcal{B}$ hinzu, die eine Attributbelegung beschreiben, für die in $\mathcal{B}$ noch keine Bedingung existiert. Als Belegung wird eine Beschreibung gewählt, die den vorliegenden Attributwert der Hypothese nicht enthält. Diese Belegung kann die Belegung einer Elementarbedingung einer anderer Produktionen sein, die über das Attribut eine Aussage macht, den vorliegenden Wert aber nicht enthält.*

2. *Spezialisiere eine oder mehrere der Elementarbedingungen, so daß sie durch die Hypothesen in $\mathcal{H}$ nicht mehr erfüllt werden, die ihnen bei der Anwendbarkeitsuntersuchung der Produktion zugeordnet wurden.*

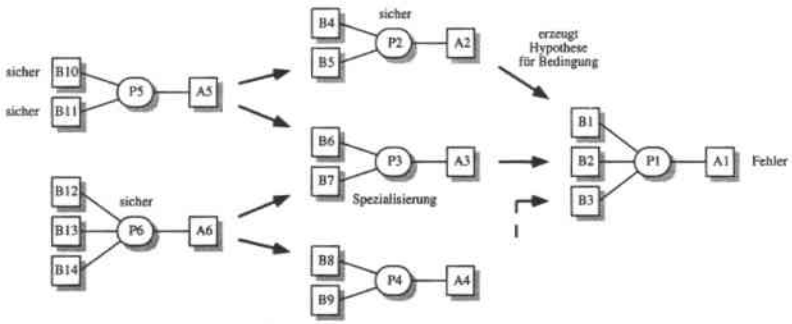

Abbildung 6.5: Ableitungsgraph für Spezialisierung

Definition 6.15 (Kosten der Spezialisierung einer Elementarbedingung) *Die **Kosten** der Spezialisierung einer Elementarbedingung B zur Elementarbedingung S sind gemäß der Spezialisierungsvarianten definiert als*

$$Kosten(B, S) = Signifikanz(S) - Signifikanz(B)$$

Der Fall, daß die Signifikanz von B oder S gleich ∞ ist, wird entsprechend der Generalisierung behandelt.

Definition 6.16 (Kosten der Spezialisierung einer Produktion) *Die **Kosten** der Spezialisierung einer Produktion $P = (\{B_1, \ldots, B_n, B_{n+1} = true, \ldots, B_{n+m} = true\}, A)$ zur Produktion $Q = (\{S_1, \ldots, S_n, S_{n+1}, \ldots, S_{n+m}\}, A)$ sind definiert als die Summe der Kosten der einzelnen Spezialisierungen. Die Bedingungen $S_{n+1}, \ldots, S_{n+m}$ sind dabei die neu hinzugekommenen Elementarbedingungen.*

$$Kosten(P, Q) = \sum_{i=n+1}^{n+m} Signifikanz(S_i) + \sum_{i=1}^{n} Kosten(B_i, S_i)$$

Zunächst muß eine Methode definiert werden, wie die Produktion herausgefunden werden kann, die spezialisiert werden soll. Eine Einschränkung der Komplexität der Lernaufgabe der externen Spezialisierung kann dadurch erreicht werden, daß nicht in der Menge aller Produktionen, die angewandt wurden, alle Spezialisierungsmöglichkeiten betrachtet werden, sondern daß zunächst eine Produktion bestimmt wird, die spezialisiert werden soll und für diese alle Varianten untersucht werden. Im folgenden wird ein derartiges zweistufiges Vorgehen beschrieben, in dem als Heuristik die Produktion im Ableitungsgraphen spezialisiert wird, die die geringste Stärke besitzt. Für die allgemeine Form der externen Spezialisierung muß diese Untersuchung für alle Produktionen des Ableitungsgraphen stattfinden.

Für die Spezialisierung wird für jede Hypothese H zusätzlich gespeichert: 1. Eine laufende Nummer, 2. Die Nummer der erzeugenden Produktion, 3. Die Nummern der Hypothesen,

aus denen die Hypothese unter Verwendung der erzeugenden Produktion abgeleitet wurde (*Hypos*(*H*)) und 4. Eine Referenz auf alle anderen Hypothesen, die zum Anwendungszeitpunkt auch auf der erzeugenden Wissenseinheit vorlagen (*RestlicheHypos*(*H*)). Durch diese Informationen kann ausgehend von der fehlerhaften Hypothese rückwärts der Ableitungsgraph bestimmt werden. Darin wird die Produktion geringster Stärke gesucht, die auf Hypothesen basiert, die nicht als sicher bekannt sind. Eine Hypothese gilt als *sicher*, wenn sie von der Sensor- oder Benutzerwissenseinheit stammt, oder eine Anfragehypothese ist. Eine Produktion kann als *sicher* gekennzeichnet werden, wenn sie bei der Spezialisierung nicht verändert werden darf. In Abbildung 6.5 wird unter der Voraussetzung, daß die Stärke der Produktion P1 größer ist als die von P3 und in der Ableitung für B3 ebenfalls keine schwächere Produktion vorliegt, die Produktion P3 spezialisiert.

Algorithmus 6.12 (Suche schwächste Produktion)

Eingabe: · Hypothese H, von der aus die Suche erfolgen soll
· Produktion P, die unter den bisher betrachteten Produktionen die geringste Stärke hatte
· Hypothese NH, die durch Anwendung von P entstanden ist
Ausgabe: · Die beste Spezialisierungsstelle BS
Variablen: · EP ist die Produktion, die H erzeugt hat

funct *Suche_schwächste_Produktion* (H, P, NH)
  $BS_{Prod} := nil$; $BS_{NH} := \{\}$
  if H bereits betrachtet wurde oder sicher ist **then**
    $BS_{Prod} := P$; $BS_{NH} := NH$
    return (BS)
  else
    $EP :=$ Produktion, die H erzeugt hat
    if $(EP \neq nil \wedge \neg sicher(EP)) \wedge (P = nil \vee (Stärke(EP) < Stärke(P)))$ **then**
      $P := EP$; $NH := H$
    for $h \in Hypos(H)$ **do** (∗Hypothesen, auf denen H basiert∗)
      $S :=$ *Suche_schwächste_Produktion* (h, P, NH)
      if $Stärke(S_{Prod}) < Stärke(BS_{Prod})$ **then** $BS := S$
  return (BS)

Die schwächste Produktion im Ableitungsgraph, die mit Hilfe des obigen Algorithmus gefunden wird, muß anschließend geeignet spezialisiert werden. Die Produktion soll dabei so spezialisiert werden, daß der Bedingungsteil nicht mehr von den Hypothesen erfüllt wird, die zu der Anwendung der Produktion geführt haben, und dabei minimale Kosten entstehen. Im angegebenen Algorithmus wird maximal eine Elementarbedingung spezialisiert oder neu hinzugenommen.

Algorithmus 6.13 (Spezialisiere eine Produktion)

Eingabe: · Produktion $P = (\mathcal{B}, A)$, die spezialisiert werden soll
· Hypothesen $\mathcal{H}$, die zur Anwendung von P geführt haben
· Restliche Hypothesen $\mathcal{R}$, die zum Zeitpunkt der Anwendung von P ebenfalls vorlagen
Ausgabe: · Spezialisierung der Produktion P als Seiteneffekt
Variablen: · BE ist die beste zu spezialisierende Elementarbedingung
· ES ist die beste Spezialisierung durch Einschränkung des Erzeugers einer Hypothese
· BS ist die beste Spezialisierung durch Einschränkung der Belegung einer Hypothese
· BXS ist die beste Spezialisierung durch Hinzunahme einer weiteren Elementarbedingung

funct *Spezialisiere_Produktion* $(P, \mathcal{H}, \mathcal{R})$
$BE_{Kosten} := maxreal$
(* Spezialisierung einer Elementarbedingung, $P = (\mathcal{B}, A)$ *)
for $B \in \mathcal{B}$ **do**
$ES :=$ *Finde_beste_Spezialisierung_für_Erzeuger*(B)
if $ES_{Kosten} < BE_{Kosten}$ **then** $BE := ES$; $BE_{alt} := B$
$BS :=$ *Finde_beste_Spezialisierung_für_Belegung*$(B,$ Hypothese $\in \mathcal{H}$, die B bei der Anwendung zugeordnet würde$)$
if $BS_{Kosten} < BE_{Kosten}$ **then** $BE := BS$; $BE_{alt} := B$
(* Hinzufügen einer weiteren Bedingung *)
$BXS := nil$; $BXS_{Kosten} := maxreal$
for $H \in \mathcal{H}$ **do**
(* Erzeuge eine zu H passende, aber nicht erfüllte Elementarbedingung *)
$XS :=$ Suche in der Menge aller Produktionen eine Elementarbedingung in der Vorbedingung, die eine Aussage über das Attribut von H macht, aber den Wert von H nicht enthält.
if $XS_{Kosten} < BXS_{Kosten}$ **then** $BXS := XS$
if $BXS_{Kosten} < BE_{Kosten}$ **then** $P := (\mathcal{B} \cup \{BXS_{Bed}\}, A)$
elsif $BE_{Kosten} \neq maxreal$ **then** Ersetze die Bedingung BE_{alt} von P durch BE_{Bed}

Bei der Spezialisierung wird im Gegensatz zur Generalisierung die Ausgangsproduktion gelöscht, da sie zu einem fehlerhaften Verhalten führte. Die neue Produktion erhält die initiale Stärke

$$Stärke(S) := Stärke(P) + \min\left(\frac{Kosten(P,S)}{Stärke(P)}, 0\right)$$

Im nächsten Schritt werden alle Hypothesen und alle Wissenseinheiten, die nach Anwendung der schwächsten Regel entstanden sind, gelöscht. Insgesamt ergibt sich dann für die externe Spezialisierung der folgende Ablauf:

Algorithmus 6.14 (Externe Spezialisierung)

Eingabe: · Produktion P, die zur fehlerhaften Aktion führte
· Hypothesen $\mathcal{H}$, die zur Anwendung von P geführt haben
· Restliche Hypothesen $\mathcal{R}$

Ausgabe: · Anpassung der Wissensbasis eines Agenten (Spezialisierung einer Produktion)

Variablen: · BS ist die beste gefundene Spezialisierung

funct *Spezialisiere*$(P, \mathcal{H}, \mathcal{R})$
  $BS_{Prod} := nil$; $BS_{NH} := nil$
  for $H \in \mathcal{H}$ **do**
    $S :=$ *Suche_schwächste_Produktion*(H, nil, nil)
    if *Stärke*$(S_{Prod}) <$ *Stärke*(BS_{Prod}) **then** $BS := S$
  if $BS_{Prod} \neq nil \wedge$ (*Stärke*$(BS_{Prod}) <$ *Stärke*(P)) **then**
    Spezialisiere_Produktion$(BS_{Prod},$ *Hypos*$(BS_{NH}),$ *Restliche Hypos*$(BS_{NH}))$
    Lösche alle aus BS_{NH} nachfolgend abgeleiteten Hypothesen und Wissenseinheiten
  else *Spezialisiere_Produktion*$(P, \mathcal{H}, \mathcal{R})$

Nach der Durchführung eines Spezialisierungsschrittes kann es passieren, daß keine andere Regel mehr im System anwendbar ist. Daher kann dann natürlich wieder die interne Generalisierung angewendet, oder über eine externe Vorführung neues Wissen in das System eingebracht werden. Wird dabei eine Produktion übergeneralisiert und die Anwendung führt zu Fehlern, wird wieder die externe Spezialisierung eingesetzt. Damit eine unendliche Wiederholung von Generalisierung und Spezialisierung verhindert wird, ist innerhalb einer Problemlösung die Anzahl der erlaubten Lernschritte nach oben begrenzt. Ist eine Lösung innerhalb dieser Lernschritte nicht möglich, gilt diese Aufgabe mit dem aktuellen Systemwissen als unlösbar.

Die Frage ist, wie häufig es überhaupt zu derartigen Wiederholungen kommen kann, da durch eine Aktion des Systems oder des Benutzers mit dem Roboter auch der Weltzustand verändert wird, und es daher unwahrscheinlich ist, daß tatsächlich der gleiche Zustand mehrfach hintereinander eingenommen werden kann.

6.5 Zusammenfassung

In diesem Kapitel wurden vollständig die agentenunabhängigen Lernverfahren erläutert, die in einer internen oder externen Generalisierung oder Spezialisierung von Produktionswissen der Agenten bestehen. Die Komplexität des externen Lernverfahrens im Vergleich zu manchen anderen Arbeiten (z.B. [BI87]) ergibt sich dadurch, daß nicht von einem allwissenden Orakel ausgegangen wird, das sofort bei der Anwendung einer beliebigen Produktion angibt, ob sie korrekt ist oder nicht. Vielmehr wird von einer benutzeradäquaten Kommunikation ausgegangen, die im wesentlichen auf Teilzielen basiert. Aus dieser punktuellen Information müssen dann entsprechende Konsequenzen für andere Komponenten der Problemlösung abgeleitet werden. Wäre ein Orakel vorhanden, so könnte das externe Lernen im wesentlichen auf die Basisverfahren zur Generalisierung und Spezialisierung einer vorliegenden Produktion und das Lernen von Relationsbeschreibungen eingeschränkt werden.

Kapitel 7

Lernen von generischen Objektbeschreibungen

Eine wichtige Teilkomponente eines autonomen Systems ist die Komponente zur Verarbeitung und Ableitung von Objektinformationen. Wie bereits in Kapitel 2.1.3 angeführt, wird in den meisten vorhandenen Ansätzen – auch für autonome Systeme – von bereits existierenden CAD-Modellen für alle beteiligten Objekte ausgegangen. Gegen diese Annahme spricht, daß ein autonomes System auch in teilweise unbekannten Umgebungen agieren soll. Aus diesem Grund ist die Verwendung von sogenannten generischen Modellen, d.h. Modellen für eine ganze Klasse von ähnlichen Objekten, notwendig. In diesem Kapitel wird nun für den Objektagenten des Systems untersucht, wie derartige Modellierungen gelernt werden können. Ausgangspunkt sind zweidimensionale Ansichten von konkreten Objekten, aus denen eine Modellhierarchie abgeleitet, d.h. gelernt werden soll, die für zukünftige Klassifikationsaufgaben eingesetzt werden kann (siehe auch [KW94, Wen92, Sto93]). In diesem Ansatz wird kein Hintergrundwissen in Form von a priori gegebenen CAD-Modellen vorausgesetzt.

Während in den bisherigen Kapiteln die agentenunabhängigen Repräsentations- und Lernverfahren behandelt wurden, wird in diesem Kapitel eine agentenspezifische Repräsentationsform für das Wissen des Objektagenten entwickelt, die den Notwendigkeiten der zu behandelnden Strukturen gerecht wird. Entsprechend sind auch die Methoden zum Lernen und die Anwendung des Wissens zur Klassifikation agentenspezifische Verfahren.

Das Zusammenspiel mit den anderen Komponenten des Systems wurde bereits in Kapitel 5 erläutert. Von der Sensorwissenseinheit gelangt die Kanteninformation eines Objektes auf eine entsprechende Objektwissenseinheit. Hauptaufgabe des Objektagenten ist dann die Klassifikation dieser Ansicht bzw. die Integration der Ansicht in die existierende Modellhierarchie (s. Abbildung 7.1).

Das Kapitel beginnt mit einer Einführung in das Gebiet der generischen Objektmodelle und einer kurzen Bewertung der bereits existierenden Ansätze (Kap. 7.1). Im folgenden wird dann zunächst eine Definition der Bestandteile der agentenspezifischen Wissensrepräsentation für Objekte, Metabeschreibungen und Modelle vorgenommen (Kap. 7.2 bis 7.4). Darauf aufbauend wird ein Satz von Lern- und Klassifikationsoperatoren formuliert und ein Maß für die Bewertung der Ballungsnützlichkeit entwickelt. Diese werden dann

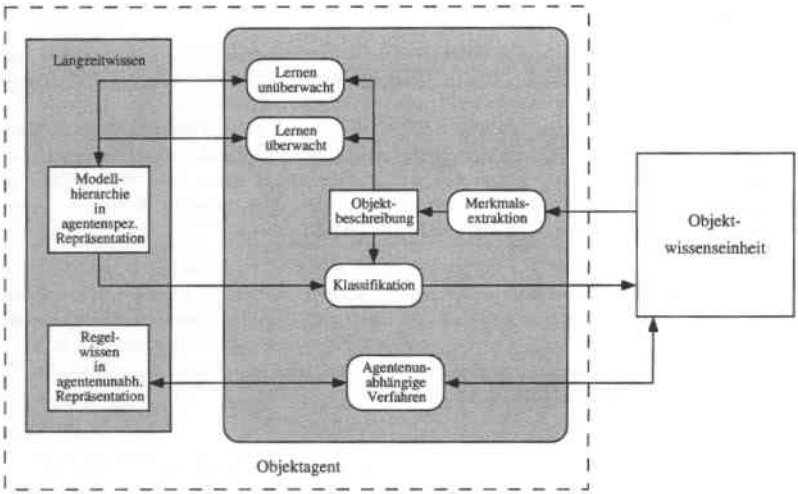

Abbildung 7.1: Struktur des Objektagenten

in den Algorithmen zum unüberwachten und überwachten Lernen sowie zur Klassifikation eingesetzt (Kap. 7.5 und 7.6). Anschließend wird das Zuordnungsverfahren entwickelt, das eine Zuordnung zwischen Elementen eines Objektes und eines Modells vornimmt (Kap. 7.7).

7.1 Grundlagen

Auf dem Gebiet der Objekterkennung für Robotersysteme werden im wesentlichen zwei Klassen von Ansätzen verfolgt: 1. Ansätze zur Objekterkennung basierend auf exakten Modellen, 2. Ansätze zur Objekterkennung basierend auf generischen Modellen. Eine dritte Klasse, die eine Klassifikation lediglich auf der Basis bestimmter globaler Parameter für ein Objekt (z.B. Fläche, Schwerpunkt, Histogrammverteilung) durchführt, soll hier nicht betrachtet werden, da darauf aufbauend keine allgemeinen Manipulationen eines unbekannten Objektes unterstützt werden. Bei der ersten Klasse wird von einem exakten vorliegenden Modell für jedes einzelne zu erkennende Objekt ausgegangen, z.B. einem CAD-Modell. Der Erkennungsvorgang kann dann häufig durch Erzeugung von Merkmalshypothesen aus dem CAD-Modell unterstützt werden. Die Ansätze zur generischen Objekterkennung basieren die Klassifikation zwar auch auf einem Modell, das aber im allgemeinen weniger exakt ist und vor allem für eine ganze Klasse von Objekten gültig ist. Insbesondere ist auch der Mensch problemlos in der Lage, auf der Basis generischer Objektmodelle eine Szene zu interpretieren oder Eindrücke auf diesem Abstraktionsgrad zu kommunizieren und zu interpretieren. Der entscheidende Unterschied zwischen beiden Ansätzen ist, daß der zweite Ansatz Objekte besser behandeln kann, die unerwartet auftreten, für die also

kein exaktes CAD-Modell vorliegt.

Ein Hauptproblem generischer Ansätze ist die Wahl einer geeigneten Repräsentation für die generischen Objektmodelle. Aufgrund der Forderung nach der Gültigkeit des Modells für eine ganze Klasse von Objekten können quantitative geometrische Größen nur eine untergeordnete Rolle spielen. Wichtig sind vor allem qualitative Aussagen über Teilkomponenten der Objekte (z.B. Krümmung, Größenverhältnisse, physikalische Eigenschaften), relationale Beziehungen zwischen diesen (z.B. Parallelität, Kontakt) und eventuell höhere symbolische Informationen wie die Funktionalität einer oder mehrerer Teilkomponenten (z.B. kann als Griff dienen, Klappmechanismus [SB91]).

Bei der Erkennung eines Objektes auf der Basis derartiger generischer Objektbeschreibungen müssen die entsprechenden Merkmale abgeleitet werden und wie bei den auf geometrischen Modellen basierenden Ansätze eine Zuordnung der einzelnen Komponenten stattfinden. Die Modelle müssen in den meisten Fällen von Hand definiert werden. Zukünftige Systeme sollten aber auch in der Lage sein, das erforderliche Wissen selbständig zu akquirieren, d.h. zu lernen [NB87].

7.1.1 Repräsentationstechniken für generische Objektbeschreibungen

Bisherige Arbeiten auf dem Gebiet der generischen Objektbeschreibungen untersuchen meistens nur den Gesichtspunkt der Klassifikation mit Hilfe bekannter generischer Objektbeschreibungen [BGSDM90, Sta88b, Sta90, Sta91, Won91]. Stansfield macht in seinem Ansatz die Annahme, daß die Objekte durch eine feste Menge von Ansichten beschrieben werden können und mit diesen auch eine Klassifikation möglich ist. Konkret betrachtet er die fünf Ansichten von oben, von vorne und hinten und von links und rechts. Um diese Annahmen auch bei der Klassifikation zu erfüllen muß sich das Objekt allerdings in genau der vorher modellierten Position und Orientierung befinden. Sinnvoller ist die objektspezifische Bildung von Äquivalenzklassen von Ansichten. Nur in wenigen Arbeiten, die oft starke Vereinfachungen annehmen, wird auf den automatischen Erwerb solcher Objektbeschreibungen eingegangen, d.h. auf den initialen Aufbau und die inkrementelle Korrektur. Generalisierungsmöglichkeiten für zweidimensionale Objektbeschreibungen behandeln [KLL84, BGGS89, CK91]. Einen Ansatz über die Darstellung von 2D-Primitiven und Relationen als semantisches Netz und den Einsatz einer abgeänderten Version von Winstons ANALOGY-Programm beschreiben [CB87].

Ein Ansatz aus der kognitiven Psychologie, der insbesondere untersucht, wie der Mensch seine Umwelt wahrnimmt und speziell unbekannte Objekte behandeln kann, ist [Bie85]. Biederman entwickelt einen Satz von 36 dreidimensionalen Basisprimitiven (ähnlich den verallgemeinerten Zylindern) und Relationen zwischen diesen und beschreibt, daß Menschen auf der Basis dieser Merkmale eine Zuordnung eines Objektes vornehmen können. Diese Interpretation war in psychologischen Experimenten sogar dann möglich, wenn nur eine zweidimensionale Linienzeichnung vorlag. Eine vereinfachte Version dieser Theorie wurde erstmals in [BL90, BL92] algorithmisch realisiert. Aus einem Kantenbild werden 2D-Primitive und Relationen zwischen diesen abgeleitet, daraus eine Segmentierung in Flächenstücke vorgenommen und dann eine Zuordnung zu einem der Grundprimitive durchgeführt.

[RMLB88] behandelt die Erstellung geometrischer Beschreibungen unbekannter Objekte aus einem Abstandsbild unter Verwendung von einigen einfachen Grundkörpern. Auf der Basis vorgegebener Tabellen werden daraus mögliche Greifaktionen abgeleitet. Zur Durchführung der Manipulation ist daher kein a priori gegebenes exaktes geometrisches Modell des Objektes notwendig. Das Ableiten von Informationen für eine ganze Klasse von Objekten ist nicht vorgesehen. In [HW90] wurde ein Verfahren vorgestellt, um aus Draufsichten eines Objektes und daraus abgeleiteten Primitiven automatisch folgende Dinge zu lernen: 1. Die Menge der Primitive, die die Objekte bestmöglich diskriminieren, 2. Die Filterparameter, mit denen die Primitive extrahiert werden sollen und 3. Eine Strategie zum Bestimmen der Reihenfolge, in der die Primitive zur Klassifikation verwendet werden sollen. Ein echtes Objektmodell wird bei diesem Verfahren jedoch nicht aufgebaut. Die Menge der insgesamt zu unterscheidenden Werkstücke muß a priori bekannt sein, um die verschiedenen Varianten testen zu können.

In dieser Arbeit wird insbesondere der Aspekt der automatischen Akquisition von generischen Objektmodellen untersucht. Ausgangspunkt ist eine Menge von zweidimensionalen Ansichten eines Objektes in Form von Kantenstücken. Hier wird keine genauere Untersuchung der Bildvorverarbeitungsoperationen [NB87, Hus91] durchgeführt, sondern eine vorhandene Hardware eingesetzt (s. Anhang A), die prinzipiell in der Lage ist, derartige Kanteninformationen aus einem Grauwertbild zu extrahieren. Im folgenden wird von einzelnen Objekten ausgegangen, d.h. die Verarbeitung einer Szene ist nicht Gegenstand der Untersuchungen.

7.1.2 Ansätze zur begrifflichen Ballung

Das in dieser Arbeit entwickelte Lernverfahren gehört zum Teilbereich der begrifflichen Ballung (s. Kapitel 2.2.4). Inkrementell sollen unklassifizierte zweidimensionale Ansichten von Objekten verwendet werden, um eine Modellhierarchie zu lernen, die für die Zuordnung weiterer Ansichten zu einer Klasse von Objekten geeignet ist.

Als Grundidee wird der CLASSIT-Algorithmus [GLF89, Gen90] eingesetzt. In CLASSIT werden als Wissensrepräsentation Attribut-Wert-Paare betrachtet, wobei die einzelnen Attribute numerische kontinuierliche Wertebereiche besitzen können. Im Vorgänger COBWEB [Fis87] sind ausschließlich nominale Attribute, d.h. Attribute mit diskreten Wertebereichen, vorgesehen. Die entstehende Modellhierarchie ist als „IS-A"-Hierarchie zu verstehen – die Wurzel ist das allgemeinste Modell, jeder Nachfolger ist eine Spezialisierug davon, die Blätter der Hierarchie sind die speziellsten Modellbeschreibungen. Jeder einzelne Knoten der Hierarchie besitzt eine vollständige Beschreibung der jeweils repräsentierten Klasse, wobei für jeden Attributwert entsprechende Wahrscheinlichkeiten gespeichert werden. Sowohl beim Lern- als auch beim Klassifikationsvorgang wird das neue Objekt ausgehend von der Wurzel sukzessive in die Modellhierarchie tiefer eingeordnet. Die Klassifikation wird beendet, sobald eine ausreichende Ähnlichkeit zwischen dem Modellknoten und dem Objekt festgestellt wird. Beim Lernvorgang werden verschiedene Operatoren untersucht, wie die Modellhierarchie lokal angepaßt werden muß, um unter Berücksichtigung des neuen Objektes eine möglichst gute Hierarchie zu erreichen. Das dadurch realisierte Suchverfahren ist ein inkrementelles Hill-Climbing auf der Menge der möglichen Modellhierarchien.

Eine wichtige Unterscheidung zu den meisten Arbeiten aus der statistischen Ballungsanalyse ist, daß es sich hier um ein divisives Verfahren und nicht um ein agglomeratives handelt, d.h. zunächst wird eine Instanz als bereits zu einer gegebenen Klasse gehörig betrachtet und nur wenn nötig wird eine neue Unterklasse gebildet. Bei den statistischen Verfahren werden dagegen meist alle Beispiele als einzelne Klassen aufgefaßt und diese sukzessive zu Oberklassen zusammengefaßt [DH73, Nag84]. Dies ist auch eine wesentliche Ursache dafür, daß diese Verfahren im Normalfall nicht-inkrementell sind.

Die Analyse von CLASSIT hat zu folgenden Eigenschaften geführt, die dem Verfahren für den hier angestrebten Einsatz zur Erzeugung generischer Beschreibungen realer Objekte fehlen bzw. weiterentwickelt werden müssen. Einige dieser Punkte wurden auch in [GLF89] als Forschungsschwerpunkte von Erweiterungen des CLASSIT-Verfahrens angesprochen. Allen beschreibt in [AT91] für COBWEB$_R$ – eine Erweiterung von COBWEB – einige Ideen zur Integration von relationalen Merkmalen und einer partiellen Zuordnung zur Behandlung von Zustandsbeschreibungen in einer Blockwelt. Im einzelnen sind an das Verfahren die folgenden Anforderungen zu stellen.

1. Neben rein numerischen Größen wie in CLASSIT müssen parallel auch nominale Attribute wie in COBWEB behandelt werden können.[1]

2. Die Repräsentation durch einfache Attribut-Wert-Paare ist nicht mächtig genug. In einer Objektrepräsentation kommen Attribute vor, die als Wert z.B. eine Menge von Kanten, Ecken, etc. besitzen. Die Korrespondenzen zwischen derartigen Mengen können dann nicht mehr nur auf der Basis der Attributnamen bestimmt werden, sondern ein spezielles Zuordnungsverfahren muß entwickelt werden.

3. Für die Zuordnung, aber auch als wichtiger Bestandteil des Objektwissens, sind Beziehungen zwischen den Objektmerkmalen wichtig. Diese müssen ebenfalls als Attribute behandelt werden können.

4. Für die erweiterte Objektrepräsentation ist dann auch die Definition der Bewertungsfunktion entsprechend vorzunehmen.

5. Ein Objektmodell besteht im allgemeinen aus einer Reihe verschiedener Ansichten, die eine interne Struktur jeden Modellknotens darstellen. Neben der Modellhierarchie ist daher eine Ansichtenhierarchie zu unterlagern. Diese muß auch bei der Definition der einzelnen Lern- und Klassifikationsoperatoren berücksichtigt werden.

6. Zusatzinformation über die tatsächliche Zugehörigkeit einer Ansicht zu einem Modell sollte ausgenutzt werden können. Dies kann dadurch geschehen, daß ein Benutzer das Modell explizit angibt, oder daß bekannt ist, daß eine Reihe von Ansichten zu einem Modell gehören. Zur Behandlung ist dann ein teilweise überwachter Lernvorgang notwendig.

Zunächst wird in den drei folgenden Abschnitten die für den Objektagenten spezifische Repräsentation definiert, die es ermöglicht, generische Objektmodelle aufzubauen. Dabei sind drei prinzipielle Repräsentationen zu unterscheiden:

[1] Eine eigene Kombination beider Verfahren namens CLASSWEB wurde bereits in [HKW92] beschrieben.

1. Objektrepräsentation: Mit Hilfe dieser Darstellung werden einzelne konkrete Objekte, die gelernt oder klassifiziert werden sollen, beschrieben. Sie stellt neben den Basismerkmalen, aus denen ein Objekt aufgebaut werden kann, auch strukturelle Beziehungen zwischen den einzelnen Merkmalen dar.

2. Metarepräsentation: Die entwickelte Lernmethodik ist prinzipiell unabhängig vom konkreten Einsatz im Objektagenten. Außerdem kann je nach Vorhandensein entsprechender Hardware oder Verarbeitungsverfahren die Objektrepräsentation weitere Informationen enthalten. Um diese Informationen leicht berücksichtigen zu können, wird in der Metabeschreibung eine Beschreibung der einzelnen Elemente der Objektrepräsentation vorgenommen.

3. Modellrepräsentation: Im Gegensatz zur Objektrepräsentation, die ein einzelnes Objekt darstellt, muß in der Modellrepräsentation eine ganze Klasse von Objekten repräsentiert werden. Dazu sind bei sonst prinzipiell ähnlichem Aufbau wie bei der Objektrepräsentation entsprechende Darstellungsformen für die Merkmale und deren Beziehungen in einem Modell notwendig.

Das System basiert zur Zeit auf 2D-Ansichten eines 3D-Objektes. Dies ist aber keine dem entwickelten Verfahren inhärente Eigenschaft. Vielmehr ist dieser Ansatz auf der Basis von Beschreibungen von Teilkomponenten und relationalen Beziehungen zwischen diesen übertragbar auf entsprechende 3D-Repräsentationen. Sehr vielversprechend für weitergehende Interpretationsaufgaben ist die Repräsentation durch die erwähnten Primitive von Biederman [Bie85].

7.2 Objektbeschreibung

Der gesamte Ablauf der Bildaufnahme, Vorverarbeitung und Merkmalsextraktion zur Ableitung einer Objektbeschreibung ist in Abbildung 7.2 gezeigt. Die ersten drei Schritte wurden in Hardware realisiert (s. Anhang A), die restlichen in Software. Als Ergebnis liefert dieser Ablauf eine zweidimensionale Objektansicht, die den wesentlichen Bestandteil einer Objektbeschreibung darstellt.

Eine 2D-Ansicht eines Objektes wird durch die Basiselemente Ecke, Kante und Fläche sowie Relationen zwischen diesen Merkmalen beschrieben (s. Abbildung 7.3). Zur Darstellung dieser Primitive werden sowohl nominale als auch numerische Attribute verwendet, die wie folgt definiert werden.

Definition 7.1 (Nominales Attribut) *Ein Attribut A heißt* ***nominal****, wenn der Wertebereich von A endlich ist (* $\mid DOM(A) \mid < \infty$ *). Die einzelnen möglichen Werte* $W_1, \ldots, W_n$ *eines nominalen Attributes ergeben den Wertebereich* $\{W_1, \ldots, W_n\}$.

Beispiel: A = Objektname, $DOM(A)$ = {Schraube, Zylinder, ...}
A = Eckentyp, $DOM(A)$ = {Verbindung, T-Verbindung, Kreuz, Pfeil}

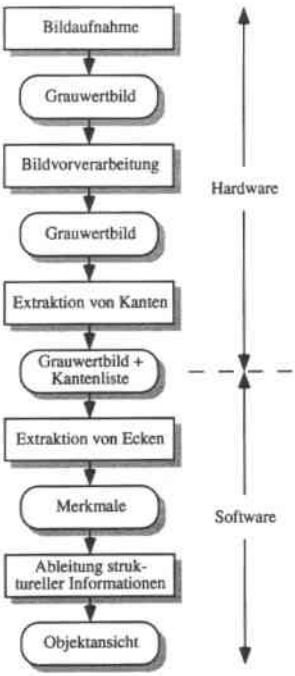

Abbildung 7.2: Erzeugung einer Objektansicht

Eine große Zahl von Lernverfahren behandelt numerische Werte nach „geeigneter" Diskretisierung wie nominale Werte, d.h. der zu erwartende Wertebereich wird in entsprechende Intervalle geteilt. Dabei muß für den einzelnen Fall sowohl die Anzahl als auch die Größe der einzelnen Intervalle bestimmt werden. Sinnvoller ist es, eine adäquate Behandlung im Lernverfahren auch für numerische Attribute zu ermöglichen. Dieser zweite Ansatz wird im entwickelten Verfahren zugrundegelegt.

Definition 7.2 (Numerisches Attribut) *Ein Attribut A heißt* ***numerisch****, wenn sein Wertebereich isomorph zu einer unendlichen Teilmenge der reellen Zahlen ist.*

Beispiel: A = Anzahl Kanten, $DOM(A)$ = natürliche Zahlen
A = Kantenlänge, $DOM(A)$ = positive reelle Zahlen

Aufbauend auf nominalen und numerischen Attributen werden Basiselemente definiert, die durch eine geordnete Menge von Attributen beschrieben werden. Genaugenommen besteht

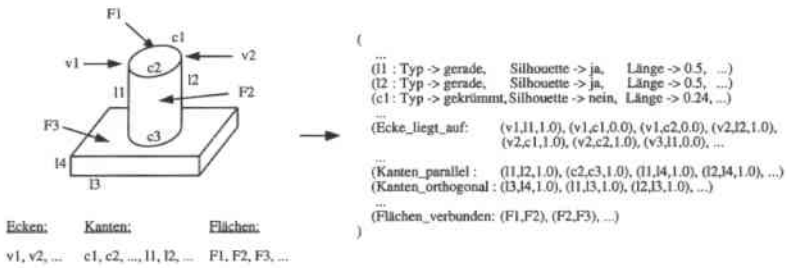

Abbildung 7.3: Beispiel für die Erzeugung einer Objektansicht

auch die Möglichkeit, daß ein Attributwert aus mehreren nominalen und numerischen Werten besteht, z.B. besteht ein 2D-Punkt aus zwei numerischen Werten. Auf diese zusätzliche Strukturierung wird der Übersichtlichkeit halber im folgenden nicht eingegangen.

Definition 7.3 (Basiselement) *Ein **Basiselement** B wird beschrieben durch eine Menge $B = \{A_1, \ldots, A_{nom}\} \cup \{A_{nom+1}, \ldots, A_{nom+num}\}$ von nominalen Attributen $A_1, \ldots, A_{nom}$ und numerischen Attributen $A_{nom+1}, \ldots, A_{nom+num}$.*

Die Definition der Basiselemente Ecke, Kante und Fläche, wie sie im Objektagenten eingesetzt wird, wird in den Tabellen 7.1, 7.2 und 7.3 angegeben. Koordinaten, die aus einer Kameraaufnahme berechnet werden, sind relativ zu der Bounding Box des Objektes angegeben, d.h. des minimalen einschließenden Rechtecks um alle vorhandenen Objektmerkmale. Durch Verwendung der im nächsten Abschnitt definierten Metabeschreibung bei den Lern- und Klassifikationsverfahren ist die Hinzunahme anderer Attribute problemlos möglich. Einige der aufgeführten Attribute stellen dreidimensionale Informationen dar, die nicht auf der Basis einer 2D-Ansicht einer Kamera abgeleitet werden können. Diese Objektinformationen können entweder durch entsprechende Stereo-Verfahren – die hier nicht untersucht wurden – oder aber durch die taktile Exploration des Experimentieragenten (s. Kap. 8) gewonnen werden.

Attribut	Attributtyp	Bedeutung
Name	nominal	Name der Ecke
Typ	nominal	Eckpunkttyp: Verbindung, T-Verbindung, Kreuz, Pfeil, Gabelung, Sonstiger
Punkt	numerisch^2	2D-Koordinaten des Eckpunktes in Bounding-Box Koordinaten
3D-Punkt	numerisch^3	3D-Koordinaten des Eckpunktes in Weltkoordinaten

Tabelle 7.1: Attributdefinitionen des Basiselements *Ecke*

Der Typ einer Ecke ist definiert als eine abstrakte Beschreibung des Zusammentreffens der umliegenden Kanten in der Ecke. Neben der Anzahl der Kanten ist insbesondere die Art des Schnittes und der dabei entstehenden Winkel relevant. Eine Übersicht über

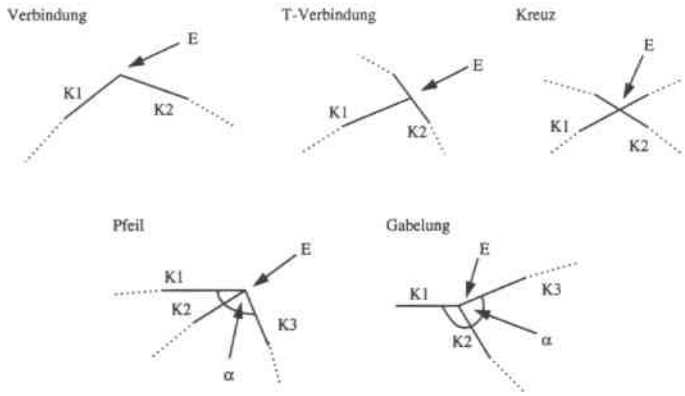

Abbildung 7.4: Eckpunkttypen

die verwendeten Varianten Verbindung, T-Verbindung, Kreuz, Pfeil und Gabelung gibt Abbildung 7.4.

Zwischen den Basiselementen Ecke, Kante und Fläche können nun Relationen definiert werden, wie Orthogonalität zweier Kanten, Zugehörigkeit einer Ecke zu einer Kante, etc. Eine konkrete Relationsausprägung wird als *n-Tupel* bezeichnet. Die Menge aller n-Tupel für eine gegebene Relation wird zu einem *strukturellen* Attribut zusammengefaßt.

Definition 7.4 (n-Tupel) *Ein Tupel $(b_1, \ldots, b_n, r)$, mit $b_1, \ldots, b_n$ sind Basiselemente, wobei $(b_1, \ldots, b_n)$ ein Element einer n-stelligen Relation $\mathcal{R} = \mathcal{B}_1 \times \ldots \times \mathcal{B}_n$ ist, wird als* **n-Tupel** *bezeichnet. Es bezeichnet eine konkrete Ausprägung der Relation $\mathcal{R}$. Das Relationsattribut r läßt eine weitere Charakterisierung der konkreten Relation zu und ist selbst entweder ein nominales oder numerisches Attribut. Das Relationsattribut kann auch wegfallen, wenn keine zusätzliche Charakterisierung der Relation benötigt wird.*

In den folgenden Definitionen wird das Relationsattribut als numerisches Attribut mit Wertebereich $[0.0, 1.0]$ angesehen.

Beispiel: Seien K, L Merkmale vom Typ Kante und als Relationstyp wird *Parallel* betrachtet, dann gibt das n-Tupel $(K, L, 1.0)$ an, daß die Kanten K und L zueinander echt parallel ($r = 1.0$) sind.

Definition 7.5 (Strukturelles Attribut) *Ein Attribut A heißt* **strukturell**, *wenn es eine Menge von strukturellen Beziehungen in Form eines Relationstyps zwischen Einzelkomponenten repräsentiert. Diese werden dargestellt durch eine geordnete Menge von n-Tupeln $\{t_1, \ldots, t_s\}$, t_i ist n-Tupel. $DOM(A) = Potenzmenge(\mathcal{W}_1 \times \ldots \times \mathcal{W}_n \times [0.0, 1.0])$.*

Der Fall $n = 1$ stellt insbesondere eine Möglichkeit dar, Attribute auszudrücken, die einen mengenmäßigen Wertebereich besitzen.

Attribut	Attributtyp	Bedeutung
Name	nominal	Name der Kante
Typ	nominal	Kantentyp: gerade, gekrümmt
Silhouette	nominal	Kante ist eine Außenkante oder nicht
Startpunkt	numerisch^2	2D-Koordinaten des Startpunktes in Bounding Box Koordinaten
Endpunkt	numerisch^2	2D-Koordinaten des Endpunktes in Bounding Box Koordinaten
Länge	numerisch	Länge der Kante relativ zur Bounding Box
Mittelpunktsabstand	numerisch	Abstand der Kante vom Mittelpunkt der Bounding Box
3D-Startpunkt	numerisch^3	3D-Koordinaten des Startpunktes in Weltkoordinaten
3D-Endpunkt	numerisch^3	3D-Koordinaten des Endpunktes in Weltkoordinaten
3D-Länge	numerisch	Länge der Kante
3D-Typ	nominal	3D-Kantentyp: konkav, konvex (unter Berücksichtigung der angrenzenden Flächen)

Tabelle 7.2: Attributdefinitionen des Basiselements *Kante*

Attribut	Attributtyp	Bedeutung
Name	nominal	Name der Fläche
Typ	nominal	Flächentyp: Kreis, Ellipse, Quadrat, Rechteck, n-Eck
Symmetrie	nominal	Symmetrie der Fläche: achsensymmetrisch, punktsymmetrisch, asymmetrisch
Mittelpunkt	numerisch^2	Mittelpunkt der Fläche in Bounding Box Koordinaten
3D-Normale	numerisch^3	Flächennormale in Weltkoordinaten
3D-Krümmung	numerisch	3D-Krümmung der Fläche: konkav, konvex
3D-Typ	nominal	Innenfläche oder Außenfläche

Tabelle 7.3: Attributdefinitionen des Basiselements *Fläche*

Beispiel: A = Kantenparallelität, Ausprägung: $\{(K_1, K_2, 1.0), (K_1, K_4, 0.95) \ldots\}$
A = Kanten, Ausprägung: $\{K_1, K_2, K_3, K_4\}$

Eine Objektansicht repräsentiert eine zweidimensionale Sicht auf ein Objekt von einem beliebigen Punkt des Raumes aus. Sie bildet die wesentliche Darstellung eines Objektes für das Lernen der Modellhierarchie und die nachfolgende Klassifikation neuer Objekte.

Definition 7.6 (Objektansicht) *Eine **Objektansicht** OA ist eine geordnete Menge von nominalen, numerischen und strukturellen Attributen* $\{A_1, \ldots, A_m\}$.

Mit der Definition der Objektansicht ist nun die Basis für eine echte Objektbeschreibung geschaffen. Diese besteht aus einer Menge von Objekt-2D-Ansichten. Darüberhinaus können weitere Attribute vorliegen, die Aussagen über das Objekt treffen. Diese Attribute werden beispielsweise durch Ergebnisse einer taktilen Exploration oder durch Einsatz weiterer Sensorik belegt.

Attribut	Attributtyp	Bedeutung
Name	nominal	Name der Objekt-2D-Ansicht
Anzahl Ecken	numerisch	Anzahl der verschiedenen Eckpunkte
Anzahl Kanten	numerisch	Anzahl der verschiedenen Kanten
Anzahl Flächen	numerisch	Anzahl der verschiedenen Flächen
Bounding Box	numerisch[4]	Einschließendes Rechteck in 2D-Koordinaten
Ecken	strukturell	Liste aller Eckpunkte
Kanten	strukturell	Liste aller Kanten
Flächen	strukturell	Liste aller Flächen
Ecke_liegt_auf	strukturell	Liste aller Eckpunkt-/Kantenbeziehungen
Kanten_parallel	strukturell	Liste aller parallelen Kantentupel
Kanten_orthogonal	strukturell	Liste aller orthogonalen Kantentupel
Kanten_linear	strukturell	Liste aller linearen Kantentupel
Kante_liegt_in	strukturell	Liste aller Kanten-/Flächenbeziehungen
Flächen_symmetrisch	strukturell	Liste aller symmetrischen Flächentupel
Fläche_grenzt_an	strukturell	Liste aller aneinandergrenzenden Flächentupel

Tabelle 7.4: Attributdefinitionen einer *Objekt-2D-Ansicht*

Definition 7.7 (Objektbeschreibung) *Eine **Objektbeschreibung** O ist ein Tupel $(\mathcal{A}, \mathcal{T})$, mit $\mathcal{A}$ ist eine Menge von Objektansichten und $\mathcal{T}$ ist eine Menge von weiteren Attributen für Objekteigenschaften, wie sie z.B. durch taktile Exploration bestimmt werden.*

Attribut	Attributtyp	Bedeutung
Name	nominal	Name der Objektbeschreibung
Anzahl 2D-Ansichten	numerisch	Anzahl der verschiedenen 2D-Ansichten
2D-Ansichten	strukturell	Liste aller 2D-Ansichten
Masse	numerisch	Masse des Objektes
3D-Bounding Box	numerisch[6]	Einschließender Quader in 3D-Koordinaten

Tabelle 7.5: Attributdefinitionen einer *Objektbeschreibung*

Ausgehend von einer vorverarbeiteten visuellen Sensoraufnahme eines Objektes werden die einzelnen Attribute der Komponenten einer Objektbeschreibung durch entsprechende Methoden berechnet. Alle entwickelten Verfahren (s. [Wen92]) sind so ausgelegt, daß sie mit verrauschten Daten und mit Ungenauigkeiten von Sensorik und Berechnung umgehen können. Durch eine Reihe von Zugehörigkeitsfunktionen können die Grenzen der Zuordnung zu einer Kategorie, z.B. einem Eckpunkttyp, entsprechend angepaßt werden.

7.3 Metabeschreibung

Die wesentlichen Teile der oben definierten Wissensrepräsentation sind unabhängig von dem konkreten Einsatz im Objektagenten. Ebenso wurde die im folgenden beschriebene Lernmethodik im Kern unabhängig von der speziellen Anwendung definiert. Aus diesem

Grund wird eine Metabeschreibung eingeführt, über die Informationen über die einzelnen Attribute, deren Wertebereich, etc. an das Lernverfahren übergeben werden.

Definition 7.8 (Metabeschreibung für ein Attribut) *Die Metabeschreibung für ein Attribut mit Namen N ist ein Sechstupel (N, T, W, A, G, S), mit*

- *Typ des Attributs T: nominal, numerisch, strukturell*
- *Wertebereich W (bei nominalen Attributen): Aufzählung der möglichen Attributwerte*
- *Auflösung A (bei numerischen Attributen): Kleinster möglicher Unterschied zwischen zwei Attributwerten (s. 7.5.2)*
- *Gewicht G: Bedeutung des Attributs (für Zuordnungsverfahren, s. Abschnitt 7.7)*
- *Symmetrie S (bei strukturellen Attributen): Angabe bei zweistelligen Relationen, ob das Attribut symmetrisch ist (für Zuordnungsverfahren, s. Abschnitt 7.7)*

Für Basiselemente, 2D-Ansicht und 2D-Beschreibung ist die zugehörige Metabeschreibung definiert als eine Liste von Attribut-Metabeschreibungen für jedes der beteiligten Attribute. Zusätzlich kann für jedes Attribut angegeben werden, ob es bei der Bewertung für die Lern- und Klassifikationsschritte berücksichtigt werden soll oder nicht. Die Funktion *Attrs(Metabeschreibung)* liefert die Menge aller Attributbeschreibungen einer Metabeschreibung.

Beispiel: Meta-Beschreibung für 2D-Ansicht (vgl. Tabelle 7.2):

((Anzahl_Ecken, numerisch, –, 2.0, 0.5, –)
(Ecken, strukturell, –, –, 0.5, ja)
(Kanten_parallel, strukturell, –, –, 2.0, ja) ...)

7.4 Modellbeschreibung

In den bisherigen Abschnitten wurde die Repräsentation und die Berechnung einer einzigen Objektbeschreibung mit einer Menge einzelner Objekt-2D-Ansichten beschrieben. Ziel des Objektagenten ist jedoch die Beschreibung generischer Objektmodelle, d.h. von Modellen, die für eine ganze Klasse von ähnlichen Objekten gültig sind. Ein Modellattribut beschreibt nicht mehr nur eine konkrete Ausprägung der Attributbelegung, vielmehr fließen die Ausprägungen eines Attributes der ganzen Klasse in die Attributbelegung ein. Aus diesem Grund wird im folgenden, aufbauend auf den Repräsentationen für konkrete Objekte, eine entsprechende Repräsentation für Modelle aufgebaut, die probabilistische Beschreibungen verwendet. Dabei werden die Wahrscheinlichkeitsparameter bezüglich der Instanzen einer Modell-2D-Ansicht und nicht bezüglich eines gesamten Modells berechnet.

Die Struktur der Repräsentation bleibt unverändert, d.h. auch ein Modell besteht aus einer Menge von Modell-2D-Ansichten, die ihrerseits aus Modellattributen (inklusive relationaler Beziehungen) bestehen. Die Modellattribute werden wieder in nominale, numerische und strukturelle unterschieden.

Für jedes einzelne Attribut wird anstelle eines konkreten Wertes die Wahrscheinlichkeit für das Auftreten eines Wertes gespeichert. Damit ist die Vorhersagbarkeit eines Attributwertes sofort berechenbar. Auf dieser Größe und der Vorhersagekraft eines Attributwertes, wie sie im folgenden definiert sind, basiert das Bewertungskriterium für die Güte einer Ballung, d.h. die Einordnung eines konkreten Objektes in eines der vorhandenen Modelle.

Definition 7.9 (Vorhersagekraft, Vorhersagbarkeit) *Die **Vorhersagekraft** eines Wertes W eines Attributes A für eine Klasse K ist definiert als die a posteriori Wahrscheinlichkeit, also als die bedingte Wahrscheinlichkeit, daß eine Instanz I Element der Klasse K ist, falls das Attribut A von I den Wert W hat, d.h. $P(Klasse(I) = K \mid Wert(A) = W)$. Die **Vorhersagbarkeit** eines Wertes W eines Attributes A für eine Klasse K ist definiert als die klassenbedingte Wahrscheinlichkeit, also als die bedingte Wahrscheinlichkeit, daß das Attribut A einer Instanz I den Wert W annimmt, falls I Element der Klasse K ist, d.h. $P(Wert(A) = W \mid Klasse(I) = K)$.*

Die Vorhersagekraft kann mit der Bayesregel aus der Vorhersagbarkeit und der Wahrscheinlichkeit jeder Klasse bestimmt werden. Aus diesem Grund werden in der Modellrepräsentation für jedes Attribut Zähler bzw. Verteilungsparameter für die Vorhersagbarkeit gespeichert. Zusätzlich enthält jeder Modellknoten einen Zähler, wieviele Instanzen bereits in ihn einsortiert wurden. Damit ist die Wahrscheinlichkeit jedes Nachfolgers eines Modellknotens in der Modellhierarchie berechenbar.

Definition 7.10 (Nominales Modellattribut) *Sei A ein nominales Attribut mit Wertebereich $\{W_1, \ldots, W_n\}$. Ein korrespondierendes **nominales Modellattribut** M_{nom} ist eine Liste $(m_1, \ldots, m_n)$, wobei das i-te Listenelement der absoluten Häufigkeit des Auftretens des i-ten Attributwerts entspricht. Dabei wird eine Gleichverteilung unter allen insgesamt möglichen Instanzen angenommen.*

Bemerkung: Sei $\mathcal{B}$ eine Menge von Beispielwerten $b_1, \ldots, b_k$ für das Attribut A, dann gilt:

$$\forall i : m_i = |\{b \in \mathcal{B} : b = W_i\}| \text{ und damit } P(A = W_i) = \frac{m_i}{k}$$

Bei einem numerischen Attribut können keine Zähler für jeden einzelnen möglichen Attributwert geführt werden. Aus diesem Grund wird eine Normalverteilung der Attributwerte angenommen und die entsprechenden Parameter Mittelwert und Standardabweichung berechnet.

Definition 7.11 (Numerisches Modellattribut) *Sei A ein numerisches Attribut. Ein korrespondierendes **numerisches Modellattribut** M_{num} ist ein Tupel (μ, σ). Dabei wird eine Normalverteilung $N(\mu, \sigma)$ aller möglichen Beispiele unterstellt (mit μ als Mittelwert und σ als Standardabweichung).*

Bemerkung: Sei $\mathcal{B} = \{b_1, \ldots, b_k\}$ eine Menge von Beispielwerten für das Attribut A, dann können μ und σ wie folgt berechnet werden:

$$\mu = \frac{1}{k} \cdot \sum_{j=1}^{k} b_j$$

$$\sigma = \frac{1}{k} \cdot \sqrt{\sum_{j=1}^{k} (b_j - \mu)^2} = \frac{1}{k} \cdot \sqrt{\sum_{j=1}^{k} (b_j)^2 - \frac{1}{k} \cdot (\sum_{j=1}^{k} b_j)^2}$$

Die zweite Version der Beschreibung der Standardabweichung σ dient der inkrementellen Berechnung der Größe. Intern kann ein numerisches Modellattribut daher durch die Größen $sum_{quad} = \sum_{j=1}^{k} (b_j)^2$ und $sum = \sum_{j=1}^{k} b_j$ repräsentiert werden.

Definition 7.12 (Modell-Basiselement) *Sei B ein Basiselement. Ein korrespondierendes* ***Modell-Basiselement*** *$M_{Basiselem}$ ist eine Liste von nominalen und numerischen Modellattributen, die jeweils mit den einzelnen Attributen von B korrespondieren.*

Definition 7.13 (Modell-n-Tupel) *Sei T ein n-Tupel. Ein korrespondierendes* ***Modell-n-Tupel*** *$M_{n-Tupel}$ ist analog zum n-Tupel (s. Definition 7.4) definiert. Lediglich die einzelnen Elemente des Tupels sind Modell-Basiselemente bzw. ein Modellattribut und keine Basiselemente bzw. kein Attribut.*

Definition 7.14 (Strukturelles Modellattribut) *Sei $A = (t_1, \ldots, t_s)$ ein strukturelles Attribut. Ein korrespondierendes* ***strukturelles Modellattribut*** *$M_{strukt} = (T_1, \ldots, T_s)$ ist eine Liste von Modell-n-Tupeln T_i und deren relativen Häufigkeit in einer Ansicht für den Relationstyp, der durch A beschrieben wird. Falls ein Relationsattribut vorhanden ist, wird dieses wie ein nominales oder numerisches Modellattribut behandelt.*

Bemerkung: Sei $\mathcal{B}$ eine Menge von Beispielen für das strukturelle Attribut A, dann können die Wahrscheinlichkeiten der einzelnen Modell-n-Tupel T_i berechnet werden durch

$$P(A = T_i) = \frac{|\{b \in \mathcal{B} : b = T_i\}|}{|\mathcal{B}|}$$

Innerhalb einer Modell-2D-Ansicht spiegelt $P(A = T_i)$ also die relative Häufigkeit eines n-Tupels, d.h. einer konkreten Relationsausprägung in der Menge der Beispiele für diese Ansicht wider.

Definition 7.15 (Modell-2D-Ansicht) *Eine* ***Modell-2D-Ansicht*** *wird definiert als eine Menge $\{A_1, \ldots, A_n\}$ von nominalen, numerischen und strukturellen Modellattributen.*

Definition 7.16 (Modell(beschreibung)) *Eine* ***Modellbeschreibung*** *oder kurz ein* ***Modell*** *ist ein Tupel $(\mathcal{A}_M, \mathcal{T}_M)$ mit $\mathcal{A}_M$ ist eine Menge von Modell-2D-Ansichten und $\mathcal{T}_M$ ist eine Menge von modellspezifischen Modellattributen.*

Für ein Modell $M = (\mathcal{A}_M, \mathcal{T}_M)$ sei für alle Modell-2D-Ansichten $A \in \mathcal{A}_M$ die Funktion Modell definiert als: $Modell(A) := M$

7.4.1 Erzeugen eines initialen Modellelementes

In den beiden folgenden Abschnitten wird algorithmisch angegeben, wie zunächst ein initial leeres Modellelement, z.B. ein Modellattribut oder eine Modell-2D-Ansicht erzeugt wird und wie in den nachfolgenden Schritten eine Aktualisierung eines Modellelementes vorgenommen wird, um ein neues Objektelement zu integrieren. Die dabei jeweils hinzukommenden Elemente wirken sich auf die Parameter aus, die zur Berechnung der Wahrscheinlichkeiten der Attributwerte benötigt werden. Die Erzeugung eines initialen Modellelements läßt sich wie folgt algorithmisch beschreiben:

Algorithmus 7.1 (Erzeugen eines Modellelementes)

Eingabe: · *Typ* des neu zu erzeugenden Modellelementes
Ausgabe: · Das initial belegte Modellelement *Elem*

funct *Erzeuge*(*Typ*)
  Elem := Neue Instanz des entsprechenden Typs *Typ*
  case *Typ* **of**
    'Nominales Modellattribut' : (∗ d.h. $Elem = (m_1, \ldots, m_n)$ ∗)
      for $i := 1$ **to** n **do** $m_i := 0$
    'Numerisches Modellattribut' :
      $sum := 0$; $sum_{quad} := 0$
    'Modell-Basiselement' : (∗ d.h. $Elem = \{\}$ ∗)
      for $a \in Attrs$(Meta-Beschreibung(*Elem*)) **do**
        Elem := *Elem* ∪ *Erzeuge*(Modellattribut je nach Typ von a)
    'Modell-n-Tupel' : (∗ d.h. $Elem = (b_1, \ldots, b_t, r)$ ∗)
      for $i := 1$ **to** t **do** b_i := *Erzeuge*(Modell-Basiselement)
      r := *Erzeuge*(nominales bzw. numerisches Modellattribut)
    'Strukturelles Modellattribut' : (∗ d.h. $Elem = (t_1, \ldots, t_s)$ ∗)
      for $i := 1$ **to** s **do** t_i := *Erzeuge*(Modell-n-Tupel)
    'Modell-2D-Ansicht' : (∗ d.h. $Elem = \{\}$ ∗)
      for $a \in Attrs$(Meta-Beschreibung(*Elem*)) **do**
        Elem := *Elem* ∪ *Erzeuge*(Modellattribut je nach Typ von a)
    'Modell' : (∗ d.h. $Elem = (\mathcal{A}_M, \mathcal{T}_M)$ ∗)
      $\mathcal{A}_M := \{\}$
      for $a \in Attrs$(Meta-Beschreibung(*Elem*)) **do**
        $\mathcal{T}_M := \mathcal{T}_M \cup$ *Erzeuge*(Modellattribut je nach Typ von a)
  return (*Elem*)

7.4.2 Einbringen eines konkreten Objektes in eine Modellbeschreibung

Das nachfolgende Verfahren beschreibt, wie entweder initial aus einer konkreten Objektbeschreibung und deren Teilattributen eine entsprechende Modellbeschreibung herzuleiten oder später eine weitere konkrete Ausprägung zu einer bereits bestehenden Modellbeschreibung hinzuzunehmen ist.

Dieses Verfahren wird in den folgenden Algorithmen auch durch den Operator $\oplus$ beschrieben, z.B. *Modellansicht* := *Modellansicht* $\oplus$ *NeueAnsicht*. Von außen wird der Algorithmus nur für die Aktualisierung von Modell-2D-Ansichten und von Modellen aufgerufen. Die Zuordnungsliste ist beim Aufruf leer und wird bei der Behandlung einer Modell-2D-Ansicht erstellt. Das Vorgehen zur Zuordnung von Modellelementen zu Objektelementen wird im Anschluß an die Bewertungsfunktion in Abschnitt 7.7 erläutert. Die Funktion $Attr(\mathcal{A}, a)$ liefert innerhalb einer Menge $\mathcal{A}$ von Attributen das Attribut mit Namen a.

Algorithmus 7.2 (Aktualisierung eines Modellelementes)

Eingabe: · *Elem* ist das zu aktualisierende Modellelement
· *W* ist das in *Elem* zu integrierende Objektelement
· *ZL* gibt falls vorhanden eine Zuordnung von einzelnen Objekt- zu Modellelementen an

Ausgabe: · Als Seiteneffekt wird die Beschreibung des zu aktualisierenden Modellelementes und dessen Teilkomponenten geändert

funct *Aktualisiere* (**var** *Elem*, *W*, *ZL*)
  case *Typ* (*Elem*) **of**
    'Nominales Modellattribut' : (* d.h. $Elem = (m_1, \ldots, m_n)$ *)
      Wähle i so, daß Domänenwert(Metabeschreibung(*Elem*),i) = W
      $m_i := m_i + 1$
    'Numerisches Modellattribut' :
      $sum := sum + W$; $sum_{quad} := sum_{quad} + W * W$
    'Modell-Basiselement' :
      for $a \in Attrs$ (Meta-Beschreibung(*Elem*)) **do**
        Aktualisiere (*Attr* (*Elem*, a), *Attr* (*W*, a), *ZL*)
    'Modell-n-Tupel' : (* d.h. $Elem = (b_1, \ldots, b_t, r)$ *)
      for $i := 1$ **to** t **do**
        if Basiselement b_i noch nicht aktualisiert wurde **then**
          Aktualisiere (b_i, zugeordnetes Basiselement in *W*, *ZL*)
      for Basiselemente *belem*, die laut *ZL* neu erzeugt werden müssen
        $= \{b_{t+1}, \ldots, b_{t+t'}\}$ **do**
      if *belem* noch nicht aktualisiert wurde **then**
        belem-neu := *Erzeuge* (Modell-Basiselement)
        Aktualisiere (*belem-neu*, *belem*, *ZL*)
    $Elem := (b_1, \ldots, b_t, b_{t+1}, \ldots, b_{t+t'})$
    if Relationsattribut r vorhanden **then**
      Aktualisiere (r, Relationsattribut von *W*, *ZL*)
  'Stukturelles Modellattribut' : (* d.h. $Elem = (t_1, \ldots, t_s)$ *)
    for $i := 1$ **to** s **do**
      Aktualisiere (t_i, zugeordnetes n-Tupel in *W* laut *ZL*, *ZL*)
    for *n-Tupel*, die laut *ZL* neu erzeugt werden müssen $= \{t_{s+1}, \ldots, t_{s+s'}\}$ **do**
      n-Tupel-neu := *Erzeuge* (Modell-n-Tupel)
      Aktualisiere (*n-Tupel-neu*, *n-Tupel*, *ZL*)
    $Elem := (t_1, \ldots, t_s, t_{s+1}, \ldots, t_{s+s'})$
  'Modell-2D-Ansicht' :

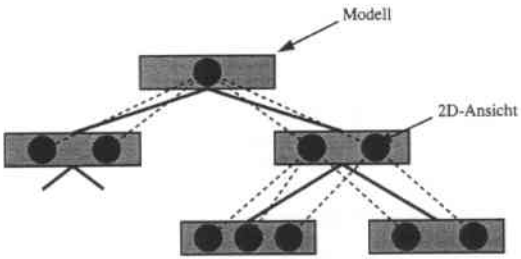

Abbildung 7.5: Modellhierarchie mit überlagerter Ansichtenhierarchie

if $ZL = nil$ **then** $ZL :=$ *Zuordnung*$(Elem, W, nil)$
for $a \in$ *Attrs*(*Metabeschreibung*$(Elem)$) **do**
Aktualisiere(*Attr*$(Elem, a)$, *Attr*(W, a), ZL)
'Modell' : (* d.h. $Elem = (\mathcal{A}_M, \mathcal{T}_M)$ *)
if *ansicht* nicht vorgegeben wurde **then**
ansicht := *Erzeuge*(Modell-2D-Ansicht)
$\mathcal{A}_M := \mathcal{A}_M \cup \{ansicht\}$
anzahl_ansichten := *anzahl_ansichten* + 1
Aktualisiere(*ansicht*, W, ZL)
(* In ein Modell wird immer genau eine neue Objekt-2D-Ansicht eingefügt. *)

7.4.3 Modellhierarchie

Wie bereits in Abschnitt 7.1 erläutert sollen die entstehenden Modelle nicht unabhängig voneinander gelernt und dargestellt werden. Durch das sukzessive Einlernen von Beispielen soll vielmehr eine Hierarchie aufgebaut werden, deren Wurzel der allgemeinsten und deren Blätter den speziellsten Modellbeschreibungen entspricht. Jeder Nachfolger eines Modells in der Modellhierarchie ist eine Spezialisierung dieses Modells. Parallel zu den Modellen können auch die einzelnen Ansichten der Modelle hierarchisch angeordnet werden. Zu diesem Zweck wird der Modellhierarchie eine Ansichtenhierarchie überlagert (s. Abbildung 7.5).

Definition 7.17 (Modellhierarchie) *Eine* ***Modellhierarchie*** *wird definiert als ein Baum* $MH = (\mathcal{M}, \mathcal{MK})$, *mit* $\mathcal{M}$ *ist eine Menge von Modellen und* $\mathcal{MK}$ *ist eine Menge von Tupeln* $(i, j) \in \mathcal{M} \times \mathcal{M}$, *die Kanten repräsentieren.*

Bemerkung: Für jede Kante (i, j) in $\mathcal{MK}$ gilt: i ist allgemeiner als j, d.h. jede Objektbeschreibung, die in j enthalten ist, ist auch in i enthalten.

Definition 7.18 (Ansichtenhierarchie) *Sei eine Modellhierarchie* $MH = (\mathcal{M}, \mathcal{MK})$ *gegeben.* MH *ist eine* ***Ansichtenhierarchie*** $AH = (\mathcal{A}, \mathcal{AK})$ *überlagert, die ihrerseits*

einen Baum darstellt. Es gilt: $\forall A \in \mathcal{A} : \exists_1 M \in \mathcal{M} : M = Modell(A)$. *Außerdem gilt:* $\forall (a,b) \in \mathcal{AK} : (Modell(a), Modell(b)) \in \mathcal{MK}$

Konvention: Im folgenden umfasse die Modellhierarchie stets auch die überlagerte Ansichtenhierarchie, d.h. bei Verwendung einer Modellhierarchie sind auch die Elemente und Funktionen der Ansichtenhierarchie verfügbar.

Definition 7.19 (Nachfolger, Vorgänger) *Sei eine Modellhierarchie* $MH = (\mathcal{M}, \mathcal{MK})$ *gegeben. Für jedes Modell* $M \in \mathcal{M}$ *ist die Menge seiner direkten* ***Nachfolger*** $\mathcal{N}(M)$ *definiert als*

$$\mathcal{N}(M) := \{m \in \mathcal{M} \mid (M,m) \in \mathcal{MK}\}$$

Der direkte ***Vorgänger*** $V(M)$ *für* $M \in \mathcal{M}$ *sei das* $m \in \mathcal{M}$ *mit* $(m,M) \in \mathcal{MK}$, *falls ein solches existiert, sonst nil.*

Bemerkung: $V(M) = nil$ gilt nur für die Wurzel der Modellhierarchie.

Definition 7.20 (Nachfolgeransichten, Vorgängeransicht) *Sei eine Ansichtenhierarchie* $AH = (\mathcal{A}, \mathcal{AK})$ *gegeben. Für jede Ansicht* $A \in \mathcal{A}$ *ist die Menge ihrer direkten* ***Nachfolgeransichten*** $\mathcal{N}_A(A)$ *definiert als*

$$\mathcal{N}_A(A) := \{a \in \mathcal{A} \mid (A,a) \in \mathcal{AK}\}$$

Die direkte ***Vorgängeransicht*** $V_A(A)$ *für* $A \in \mathcal{A}$ *sei das* $a \in \mathcal{A}$ *mit* $(a,A) \in \mathcal{AK}$, *falls ein solches existiert, sonst nil.*

7.5 Lernen von Objektbeschreibungen

Im folgenden Abschnitt wird untersucht, wie auf der Basis der oben definierten Repräsentationen eine Modellhierarchie automatisch gelernt werden kann. Die Lernaufgabe kann dabei nicht-inkrementell oder inkrementell formuliert werden. Die nicht-inkrementelle Version wird jedoch auf entsprechende inkrementelle Schritte abgebildet.

Definition 7.21 (Lernaufgabe des Objektagenten (nicht-inkrementell)) *Gegeben eine Menge von Objektbeschreibungen* $\mathcal{O} = \{O_1, \ldots, O_n\}$. *Gesucht eine Modellhierarchie, in der für jeden Knoten die Modelle eines seiner Nachfolger möglichst ähnlich, die Modelle verschiedener Nachfolger möglichst verschieden sind.*

Definition 7.22 (Lernaufgabe des Objektagenten (inkrementell)) *Gegeben eine Modellhierarchie* M *und eine Objektbeschreibung* O. *Gesucht ist eine Eingliederung von* O *in* M, *so daß weiterhin für jeden Knoten die Modelle eines seiner Nachfolger möglichst ähnlich, die Modelle verschiedener Nachfolger möglichst verschieden sind.*

Die Eingliederung eines neuen Objektes in eine bereits bestehende Modellhierarchie wird von der Wurzel ausgehend vorgenommen. In jedem Schritt werden fünf verschiedene Lernoperatoren zur Manipulation der Hierarchie untersucht. Die ersten drei bilden den Kern für den Aufbau einer Hierarchie: 1. Das Einfügen des Objektes in einen bestehenden Knoten, 2. Das Einfügen in ein bestehendes Blatt und 3. Das Erzeugen eines neuen Blattes in der Modellhierarchie. Sie werden auch bei der Klassifikation in entsprechend angepaßter Form eingesetzt. Zusätzlich werden zwei restrukturierende Operatoren betrachtet: 1. Das Aufsplitten eines Modells und 2. Das Zusammenfassen mehrerer Modelle zu einem gemeinsamen allgemeineren Modell. Diese restrukturierenden Lernoperatoren werden benötigt, um bestimmte Effekte zu eliminieren, die durch die inkrementelle Arbeitsweise des Verfahrens auftreten können. Die entstehende Modellhierarchie ist abhängig von der Reihenfolge der auftretenden Beispiele. Treten die Beispiele der verschiedenen Klassen nicht gleichverteilt auf, so kann ohne die Restrukturierungsoperatoren eine Modellhierarchie entstehen, die insgesamt keine sinnvolle Strukturierung der Klassen darstellt.

Die fünf Lernoperatoren werden zunächst exakt definiert. Nachfolgend wird dann das vom CLASSIT-Verfahren her bekannte Bewertungsmaß für die Güte oder Nützlichkeit einer Ballung, d.h. eines Teils der Modellhierarchie, eingeführt und auf die hier entwickelten Verfahren erweitert. Mit diesem Maß ist die vergleichende Bewertung verschiedener Lernoperatoren möglich, die als Basis für die dann definierten unüberwachten und überwachten Lernverfahren dient.

7.5.1 Lern- und Klassifikationsoperatoren

Für jeden Lern- bzw. Klassifikationsoperator ist sowohl eine exakte algorithmische Beschreibung als auch eine graphische Visualisierung der Auswirkungen der Operatoranwendung angegeben. Jedes Bild besteht aus zwei Teilen: Links vom Pfeil ist der Zustand eines Teils der Modellhierarchie vor, rechts vom Pfeil nach Anwendung des Operators angegeben. Der Knoten, von dem aus der Operator untersucht wird, ist immer der oberste. Die an der Entscheidung beteiligten Knoten sind durch einen schmalen Pfeil markiert. Im rechten Teilbild sind alle Modelle und Modellansichten schraffiert gekennzeichnet, in die die neue Objektansicht integriert wurde.

In den algorithmischen Beschreibungen wird bei der Auswahl der besten Modell-2D-Ansicht bereits auf die Bewertungsfunktion referenziert, die im nächsten Abschnitt definiert wird. Diese Bewertungsfunktion wird ein lokales Maß für die Qualität einer Modellhierarchie darstellen.

Einfügen in einen Knoten

Die neue Objektansicht wird von der Wurzel aus sukzessive tiefer in die Modellhierarchie eingelernt. Der häufigste dabei auftretende Lernoperator ist das Einfügen in einen (inneren) Knoten (s. Abbildung 7.6). Dazu muß eine ausreichend hohe Ähnlichkeit zwischen der Objektansicht und dem Modell, das im Knoten repräsentiert ist, bestehen. Ist das nicht der Fall kann alternativ ein neuer Nachfolgerknoten in Form eines neuen Blattes erzeugt werden (s. Lernoperator Erzeugen eines Blattes)

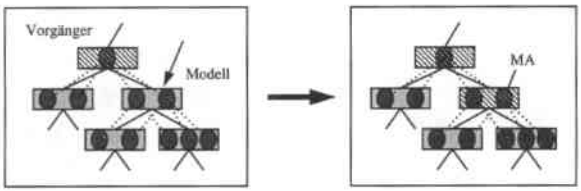

Abbildung 7.6: Lernoperator: Einfügen in einen Knoten

Algorithmus 7.3 (Einfügen in einen Knoten)

Eingabe:
- Ein Modellknoten $Modell = (\mathcal{A}_M, \mathcal{T}_M)$ der Modellhierarchie $MH = (\mathcal{M}, \mathcal{MK})$
- Der Name *Ansichtname* einer Modell-2D-Ansicht, in die eingefügt werden soll, falls die Ansicht vorgegeben ist, sonst 'neu', 'unbekannt' oder nil. Der Unterschied zwischen 'unbekannt' und nil ist der, daß bei nil keine neue Modell-2D-Ansicht erzeugt werden darf, selbst wenn die beste Übereinstimmung schlecht ist (s. Abschnitt 7.5.4).
- Eine Modell-2D-Ansicht *Vorgänger*, mit $V(Modell) = Modell(Vorgänger)$
- Eine Objekt-2D-Ansicht *OA*

Ausgabe:
- Die Modell-2D-Ansicht *MA*, der *OA* in *Modell* zugeordnet wurde

Variablen:
- *MA* ist Modell-2D-Ansicht von *Modell* oder eine neu erzeugte Modell-2D-Ansicht

funct *Füge_in_Knoten_ein* (**var** *MH, Modell, Ansichtname, Vorgänger, OA*)
  $Zähler := Zähler + 1$ (* Aktualisiere Zähler für die Klassenwahrscheinlichkeit *)
  if $Ansichtname =$ 'unbekannt' $\vee$ $Ansichtname = nil$ **then**
    Wähle $MA \in \mathcal{A}_M$ so, daß
      $Eval_A(MA) - Eval_A(MA \oplus OA) = \min_{a \in \mathcal{A}_M} Eval_A(a) - Eval_A(a \oplus OA)$
      $\wedge (V_A(MA) = Vorgänger \vee Vorgänger = nil)$
    if $Ansichtname =$ 'unbekannt' **then**
      if $Eval_A(MA) - Eval_A(MA \oplus OA) >$ 2D-Ansichten-Cutoff **then**
        $MA := Erzeuge$(Modell-2D-Ansicht) $\oplus$ *OA*
        $\mathcal{A} := \mathcal{A} \cup \{MA\}$
        if $Vorgänger \neq nil$ **then** $\mathcal{AK} := \mathcal{AK} \cup \{(Vorgänger, MA)\}$
    $MA := MA \oplus OA$
  elsif $Ansichtname =$ 'neu' **then**
    $MA := Erzeuge$(Modell-2D-Ansicht) $\oplus$ *OA*
    $\mathcal{A} := \mathcal{A} \cup \{MA\}$
    if $Vorgänger \neq nil$ **then** $\mathcal{AK} := \mathcal{AK} \cup \{(Vorgänger, MA)\}$
  else
    Wähle $MA \in \mathcal{A}_M$ so, daß $Name(MA) = Ansichtname$
    if $MA = nil$ **then error** ("Ansicht mit diesem Namen existiert nicht.")
    $MA := MA \oplus OA$
  return (MA)

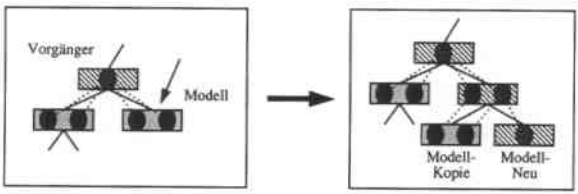

Abbildung 7.7: Lernoperator: Einfügen in ein Blatt

Die verwendete Konstante 2D-Ansichten-Cutoff ist ein relativer Schwellwert für das notwendige minimale Maß der Übereinstimmung der besten Modell-2D-Ansicht und der Objektansicht.

Einfügen in ein Blatt

Beim Abstieg in der Modellhierarchie wird nach mehreren Einfügevorgängen in innere Knoten ein Blatt erreicht. Hier muß kein bestes Nachfolgermodell bestimmt werden. Wie im ursprünglichen CLASSIT-Algorithmus wird auch hier eine Kopie des Blattes erzeugt und als Nachfolgerknoten für den aktuellen Knoten genommen. Die aktuelle Instanz wird als zweiter Nachfolgeknoten eingefügt (s. Abbildung 7.7). Die Funktion $Ansichten(M)$ liefert für ein Modell $M = (\mathcal{A}_M, \mathcal{T}_M)$ die Menge $\mathcal{A}_M$ aller Ansichten.

Algorithmus 7.4 (Einfügen in ein Blatt)

Eingabe: · Ein Modellknoten *Modell* $= (\mathcal{A}_M, \mathcal{T}_M)$ der Modellhierarchie $MH = (\mathcal{M}, \mathcal{MK})$, der ein Blatt darstellt
· Eine Modell-2D-Ansicht *Vorgänger*, mit $V(Modell) = Modell(Vorgänger)$
· Eine Objekt-2D-Ansicht *OA*
Ausgabe: · Die aktualisierte Modellhierarchie *MH*
Variablen: · *MA* ist Modell-2D-Ansicht von *Modell*, d.h. $MA \in \mathcal{A}_M$
· *Modell-Kopie* und *Modell-Neu* sind neue Modellbeschreibungen

funct *Füge_in_Blatt_ein* (**var** *MH, Modell, Vorgänger, OA*)
Wähle $MA \in \mathcal{A}_M$ so, daß
$Eval_A(MA) - Eval_A(MA \oplus OA) = \min_{a \in \mathcal{A}_M} Eval_A(a) - Eval_A(a \oplus OA)$
$\wedge (V_A(MA) = Vorgänger \vee Vorgänger = nil)$
Modell-Neu := *Erzeuge*(Modell) $\oplus$ *OA*
Modell-Kopie := *Modell*
$\mathcal{M} := \mathcal{M} \cup \{Modell\text{-}Neu, Modell\text{-}Kopie\}$
$\mathcal{MK} := \mathcal{MK} \cup \{(Modell, Modell\text{-}Neu), (Modell, Modell\text{-}Kopie)\}$
$\mathcal{A} := \mathcal{A} \cup Ansichten(Modell\text{-}Neu) \cup Ansichten(Modell\text{-}Kopie)$
$\mathcal{AK} := \mathcal{AK} \cup \{(MA, a) \mid a \in Ansichten(Modell\text{-}Neu)\}$
$\cup \{(a, b) \in \mathcal{A}_M \times Ansichten(Modell\text{-}Kopie) \mid b$ ist Kopie von $a\}$
Füge_in_Knoten_ein (*MH, Modell, Name*(*MA*), *Vorgänger, OA*)
(* Aktualisiere die Werte des ursprünglichen Blattes *)

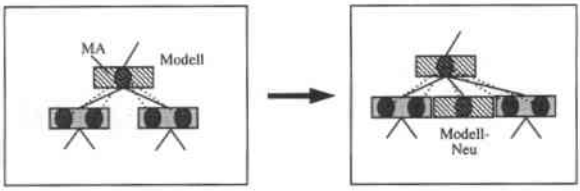

Abbildung 7.8: Lernoperator: Erzeugen eines Blattes

Erzeugen eines Blattes

Dieser Fall tritt dann auf, wenn die Objektansicht keinem der Nachfolgermodelle zugeordnet werden soll, d.h. die neue Objektansicht ist ausreichend unterschiedlich zu den vorhandenen Modellen. Dann wird ein neuer Nachfolgeknoten erzeugt, der ein Modell mit der gegebenen Ansicht als einziger Ansicht darstellt (s. Abbildung 7.8).

Algorithmus 7.5 (Erzeugen eines Blattes)

Eingabe: · Ein Modellknoten $Modell = (\mathcal{A}_M, \mathcal{T}_M)$ der Modellhierarchie $MH = (\mathcal{M}, \mathcal{MK})$
· Eine Modell-2D-Ansicht $MA \in \mathcal{A}_M$, der OA in $Modell$ zugeordnet werden soll
· Eine Objekt-2D-Ansicht OA
Ausgabe: · Die aktualisierte Modellhierarchie MH
Variablen: · *Modell-Neu* ist eine neue Modellbeschreibung

funct *Erzeuge_Blatt* (**var** $MH, Modell, MA, OA$)
$Modell\text{-}Neu := Erzeuge(\text{Modell}) \oplus OA$
$\mathcal{M} := \mathcal{M} \cup \{Modell\text{-}Neu\}$ (* Aktualisierung der Modellhierarchie *)
$\mathcal{MK} := \mathcal{MK} \cup \{(Modell, Modell\text{-}Neu)\}$
$\mathcal{A} := \mathcal{A} \cup Ansichten(Modell\text{-}Neu)$ (* Aktualisierung der Ansichtenhierarchie *)
$\mathcal{AK} := \mathcal{AK} \cup \{(MA, a) \mid a \in Ansichten(Modell\text{-}Neu)\}$

Aufsplitten eines Modells

Das Aufsplitten eines Modells ist der erste der beiden restrukturierenden Lernoperatoren (s. Abbildung 7.9). Dieser Operator sollte dann angewendet werden, wenn eine unnötige Zwischenstufe in der Modellhierarchie entstanden ist. Dazu wird das Nachfolgermodell aus der Hierarchie entfernt, und dessen Nachfolger werden direkte Nachfolger des aktuellen Modells.

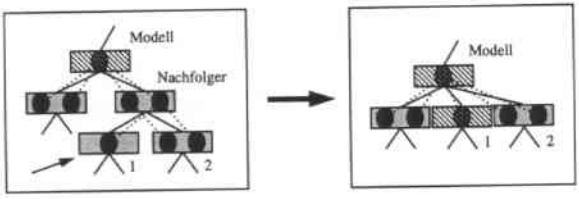

Abbildung 7.9: Lernoperator: Aufsplitten eines Modells

Algorithmus 7.6 (Aufsplitten eines Modells)

Eingabe: · Ein Modellknoten *Modell* der Modellhierarchie $MH = (\mathcal{M}, \mathcal{MK})$
· Ein Nachfolgerknoten *Nachfolger* von *Modell*
Ausgabe: · Die aktualisierte Modellhierarchie *MH*

funct *Splitte_Modell* (**var** *MH, Modell, Nachfolger*)
$\mathcal{M} := \mathcal{M} \setminus \{Nachfolger\}$
$\mathcal{MK}' := \mathcal{MK}$
$\mathcal{MK} := \mathcal{MK} \setminus \{(Modell, Nachfolger)\} \setminus \{(Nachfolger, i) \mid i \in \mathcal{M}\}$
$\cup\{(Modell, i) \mid (Nachfolger, i) \in \mathcal{MK}'\}$
$\mathcal{A} := \mathcal{A} \setminus Ansichten(Nachfolger)$
$\mathcal{AK}' := \mathcal{AK}$
$\mathcal{AK} := \mathcal{AK} \setminus \{(a, b) \mid a \in Ansichten(Modell), b \in Ansichten(Nachfolger)\}$
$\setminus\{(b, c) \mid b \in Ansichten(Nachfolger) \wedge c \in \mathcal{A}\}$
$\cup\{(a, c) \mid a \in Ansichten(Modell) \wedge c \in \mathcal{A} \wedge \exists b \in Ansichten(Nachfolger) :$
$(a, b) \in \mathcal{AK}' \wedge (b, c) \in \mathcal{AK}'\}$

Zusammenfassen von Modellen

Dieser Lernoperator ermöglicht das Zusammenfassen von zwei Modellen einer Stufe zu einem gemeinsamen Modell, das als neue Zwischenklasse eingeführt wird. Die Einschränkung auf zwei Nachfolgermodelle wird aus dem verwendeten Lernalgorithmus deutlich. Würden beliebige Anzahlen von Modellen betrachtet, müßte eine Betrachtung aller Kombinationen von Nachfolgermodellen durchgeführt werden.

Algorithmus 7.7 (Zusammenfassen von Modellen)

Eingabe: · Ein Modellknoten *Modell* der Modellhierarchie $MH = (\mathcal{M}, \mathcal{MK})$
· Zwei Nachfolgerknoten *N1* und *N2* von *Modell*
Ausgabe: · Der neu erzeugte Modellknoten *Modell-Neu*

funct *Vereinige_Modelle* (**var** *MH, Modell, N1, N2*)
Modell-Neu := *Erzeuge*(Modell)

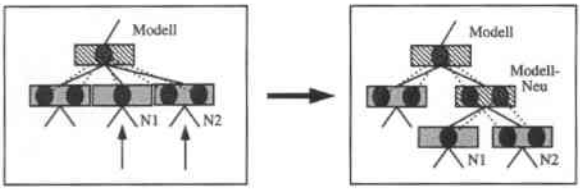

Abbildung 7.10: Lernoperator: Zusammenfassen von Modellen

Fasse die Modell-2D-Ansichten in *N1* und *N2* mit der gleichen Vorgänger-2D-Ansicht aus *Modell* zusammen und füge sie als neue 2D-Ansichten *Modell-Neu* hinzu

$\mathcal{M} := \mathcal{M} \cup \{Modell\text{-}Neu\}$

$\mathcal{MK} := \mathcal{MK} \setminus \{(Modell, N1), (Modell, N2)\}$
$\cup\{(Modell, Modell\text{-}Neu)\} \cup \{(Modell\text{-}Neu, N1), (Modell\text{-}Neu, N2)\}$

$\mathcal{A} := \mathcal{A} \cup Ansichten(Modell\text{-}Neu)$

$\mathcal{AK}' := \mathcal{AK}$

$\mathcal{AK} := \mathcal{AK} \setminus \{(a,b) \mid a \in Ansichten(Modell) \wedge (b \in Ansichten(N1) \vee b \in Ansichten(N2))$
$\cup\{(a,b) \mid a \in Ansichten(Modell) \wedge b \in Ansichten(Modell\text{-}Neu)$
$\wedge \exists c \in Ansichten(N1) \cup Ansichten(N2) : (a,c) \in \mathcal{AK}' \wedge b$ wurde für c erzeugt$\}$
$\cup\{(b,c) \mid b \in Ansichten(Modell\text{-}Neu) \wedge c \in Ansichten(N1)$
$\wedge \exists a \in Ansichten(Modell) : (a,c) \in \mathcal{AK}' \wedge b$ wurde für c erzeugt$\}$
$\cup\{(b,c) \mid b \in Ansichten(Modell\text{-}Neu) \wedge c \in Ansichten(N2)$
$\wedge \exists a \in Ansichten(Modell) : (a,c) \in \mathcal{AK}' \wedge b$ wurde für c erzeugt$\}$

return $(Modell\text{-}Neu)$

7.5.2 Bewertungsfunktion

Zur Auswahl der im vorigen Abschnitt beschriebenen Lern- bzw. Klassifikationsoperatoren wird eine Bewertungsfunktion auf den einzelnen Elementen der Objekt- bzw. Modellrepräsentation definiert. Diese Bewertungsfunktion basiert auf einem Maß aus der kognitiven Psychologie, der *Ballungsnützlichkeit* (engl. category utility), das in [GC85] definiert und in [GLF89, Gen90] leicht verallgemeinert auf die Aufgabe der Begriffsbildung übertragen wurde. Das Maß bevorzugt Ballungen/Klassenbildungen, aus denen potentiell ein Höchstmaß an Informationen abgeleitet werden kann. Dazu wird versucht, die Ähnlichkeiten zwischen den Instanzen einer Klasse (Vorhersagbarkeit) und gleichzeitig die Unterschiede zwischen Instanzen verschiedener Klassen (Vorhersagekraft) zu maximieren. Das Maß geht davon aus, daß die Begriffsbeschreibungen probabilistischer Natur sind, d.h. daß die Beschreibung aus einer Menge von Attributen, deren Werten und zugehörigen Wahrscheinlichkeiten besteht.

Gluck und Corter definieren das Maß, um eine Charakterisierung von menschlichen Kategorien der Elementarebene (engl. basic level categories) zu erhalten. Sie nehmen an, daß die Kategorien am wichtigsten sind, die im Mittel optimal zur Kommunikation von Informationen über die einzelnen Instanzen geeignet sind, d.h. die die Unsicherheit möglichst klein

lassen. Dazu verwenden sie zum einen ein Unsicherheitsmaß aus der Informations- und zum anderen aus der Spieltheorie. In der Informationstheorie ist die Unsicherheit einer Menge von n Nachrichten $f_1, \ldots, f_n$ definiert durch $U(\{f_1, \ldots, f_n\}) = -\sum_{i=1}^{n} P(f_i) \log P(f_i)$. In diesem Fall sind die n Nachrichten die möglichen Attributwerte. Ist zusätzlich die Information gegeben, daß ein Objekt zu einer Klasse k gehört, dann ergibt sich $U(\{f_1, \ldots, f_n\} \mid k) = -\sum_{i=1}^{n} P(f_i \mid k) \log P(f_i \mid k)$. Dabei ist $P(f_i \mid k)$ die Wahrscheinlichkeit, daß ein Element der Klasse k für das untersuchte Attribut den Wert f_i annimmt. Die Ballungsnützlichkeit wird dann definiert als die Abnahme der Unsicherheit ohne bzw. mit Klasseninformation: $BN(k, \{f_1, \ldots, f_n\}) = P(k)(\sum_{i=1}^{n} P(f_i \mid k) \log P(f_i \mid k) - \sum_{i=1}^{n} P(f_i) \log P(f_i))$ Aus der Spieltheorie läßt sich laut [GLF89] eine ähnliche Ballungsnützlichkeit ableiten. Gluck und Corter konnten in psychologischen Experimenten zeigen, daß mit der so definierten Ballungsnützlichkeit tatsächlich die von Menschen verwendete Elementarebene bestimmt werden konnte. Dieses Maß wird im folgenden auf die verwendete Wissensrepräsentation übertragen.

Zur Definition des Maßes werden die oben definierten Begriffe der Vorhersagbarkeit und der Vorhersagekraft verwendet. Für den Fall *nominaler* Attributwerte wird also für jedes Attributwertpaar $A_i = W_{ij}$ und jede Klasse M_k die Vorhersagbarkeit $P(A_i = W_{ij} \mid M_k)$ und die Vorhersagekraft $P(M_k \mid A_i = W_{ij})$ berechnet. Diese werden über alle Attribute, alle Werte und alle Klassen zusammengefaßt, um ein Maß für die gesamte Klasseneinteilung zu erhalten

$$\sum_{k=1}^{K} \sum_{i=1}^{I} \sum_{j=1}^{J(i)} P(A_i = W_{ij}) * P(A_i = W_{ij} \mid M_k) * P(M_k \mid A_i = W_{ij}) \tag{7.1}$$

Das Produkt aus Vorhersagbarkeit und Vorhersagekraft als Trade-Off zwischen den beiden Größen wird gewichtet mit der relativen Häufigkeit $P(A_i = W_{ij})$ des betrachteten Attributwertes W_{ij}. Dadurch sollen Werte, die häufiger auftreten, stärker gewichtet werden als andere. Die Summation läuft über die Anzahl der Unterklassen K, die Anzahl der Attribute I und die Anzahl der Elemente des Wertebereichs des i-ten Attributes $J(i)$. Mit der Formel von Bayes $P(A_i = W_{ij})P(M_k \mid A_i = W_{ij}) = P(M_k)P(A_i = W_{ij} \mid M_k)$ kann (7.1) umgeformt werden zu:

$$\sum_{k=1}^{K} P(M_k) \sum_{i=1}^{I} \sum_{j=1}^{J(i)} P(A_i = W_{ij} \mid M_k)^2 \tag{7.2}$$

Nach Gluck und Corter entspricht der Ausdruck $\sum_{i=1}^{I} \sum_{j=1}^{J(i)} P(A_i = W_{ij} \mid M_k)^2$ der erwarteten Anzahl von Attributwerten, die für ein beliebiges Element der Klasse M_k korrekt geraten werden können. Es wird also angenommen, daß mit der gleichen Wahrscheinlichkeit des Auftretens eines Attributwertes er auch geraten werden kann.

Auf dieser Basis wird nun die Ballungsnützlichkeit definiert (s. [GLF89]), die angibt, wie sinnvoll eine Unterteilung eines Modells in eine Menge von Teilmodellen ist. Die Normierung mit der Anzahl der Teilmodelle K wird durchgeführt, um Einteilungen in unterschiedlich viele Teilmodelle miteinander vergleichen zu können.

Definition 7.23 (Ballungsnützlichkeit für nominale Attribute) *Die **Ballungsnützlichkeit** für nominale Attribute ist definiert als die Zunahme der erwarteten Anzahl korrekt zu ratender Attributwerte bei K gegebenen Klassen im Vergleich zu einer nicht vorhandenen Einteilung in Teilmodelle:*

$$BN(\{M_1,\ldots,M_K\}) = \frac{\sum_{k=1}^{K} P(M_k) \sum_{i=1}^{I} \sum_{j=1}^{J(i)} P(A_i = W_{ij} \mid M_k)^2 - \sum_{i=1}^{I} \sum_{j=1}^{J(i)} P(A_i = W_{ij})^2}{K}$$

Um eine gemeinsame Betrachtung numerischer und nominaler Attribute zu erreichen, sollte ein möglichst äquivalentes Maß für beide Attributtypen verwendet werden. Für numerische Attribute ist eine Anpassung der Ballungsnützlichkeit auf die dort verwendete Repräsentation notwendig. Statt der Summation über die Wahrscheinlichkeiten der einzelnen Attributwerte wird eine Integration über die entsprechenden Werte der Normalverteilung durchgeführt [GLF89], d.h.

$$\sum_{j=1}^{J(i)} P(A_i = W_{ij})^2 \;\hat{=}\; \int \frac{1}{\sigma_i^2 2\pi} * e^{-(\frac{x-\mu_i}{\sigma_i})^2}\, dx \;=\; \frac{1}{\sigma_i 2\sqrt{\pi}} \tag{7.3}$$

Die Ballungsnützlichkeit wird nur verwendet, um verschiedene Klassenbildungen zu vergleichen. Daher könnte bei einer Menge von Attributen, die alle numerisch sind, auch der konstante Faktor $\frac{1}{2\sqrt{\pi}}$ wegfallen. Der zweite Term im Zähler der Ballungsnützlichkeit ist unabhängig von der Klasseneinteilung und entspricht damit der Verteilung im Vorgängerknoten V. Damit erhält man folgende Ergänzung der obigen Definition:

Definition 7.24 (Ballungsnützlichkeit für numerische Attribute) *Die **Ballungsnützlichkeit** für numerische Attribute ist wie folgt definiert:*

$$BN(\{M_1,\ldots,M_K\}) = \frac{\sum_{k=1}^{K} P(M_k) \sum_{i=1}^{I} \frac{1}{\sigma_{ik} \cdot 2\sqrt{\pi}} - \sum_{i=1}^{I} \frac{1}{\sigma_{iV} \cdot 2\sqrt{\pi}}}{K}$$

Ziel der nachfolgenden Abschnitte ist die Definition einer Bewertungsfunktion für die verschiedenen Lernalternativen, die lokal auf einer Ebene verglichen werden müssen. Um diese anzugeben, werden nun sukzessive für die einzelnen Komponenten der Wissensrepräsentation entsprechende Bewertungsfunktionen definiert. Dabei wurde jede Teilbewertungsfunktion *Eval* so definiert, daß der Wertebereich das geschlossene Intervall $[0.0, 1.0]$ ist. Dadurch ist eine Normierung unterschiedlicher Attributwerte, unterschiedlicher Anzahl von n-Tupeln in einem strukturellen Attribut, etc. möglich. Die im folgenden definierten Bewertungsfunktionen werden immer für *ein* Modell ausgewertet, das nicht explizit angegeben ist.

Bewertungsfunktion für Attribute

Nominale Attribute: Ein Attribut entspricht laut Definition 7.1 einer bekannten endlichen Menge diskreter Werte. Sei A_i daher ein nominales Modellattribut der Form $(m_1, \ldots, m_n)$. Es wird eine Gleichverteilung über alle möglichen Instanzen angenommen. Damit kann zur Evaluierung die in (7.2) definierte Anzahl der korrekt zu erratenden Attributwerte verwendet werden.

$$Eval_{Attr}(A_i) = \sum_{j=1}^{n} P(A_i = W_{ij})^2 \tag{7.4}$$

Dabei ist $P(A_i = W_{ij})$ die Wahrscheinlichkeit, daß das Attribut A_i den Wert W_{ij} annimmt, n die Anzahl der möglichen Attributwerte und W_{ij} der j-te Attributwert des Attributs.

Numerische Attribute: Für ein numerisches Attribut, das einen beliebigen numerischen Wert annehmen kann, wird eine Normalverteilung über alle möglichen Instanzen unterstellt. Sei A_i ein numerisches Modellattribut der Form (μ, σ). Dann ergibt sich die Bewertungsfunktion für numerische Attribute gemäß (7.3) zu:

$$Eval_{Attr}(A_i) = \begin{cases} 1 & , \sigma_i 2\sqrt{\pi} < Auflösung_i \\ \frac{Auflösung_i}{\sigma_i 2\sqrt{\pi}} & , \text{sonst} \end{cases} \tag{7.5}$$

Dabei ist σ_i die Standardabweichung von A_i. Besitzt eine Klasse nur einige einzige Instanz, so gilt $\sigma_i = 0$. Aus diesem Grund wird der Parameter $Auflösung$ eingeführt, der eine untere Grenze für die Standardabweichung oder genauer für $\sigma_i 2\sqrt{\pi}$ darstellt. Die Auflösung ist ein Maß für den kleinsten möglichen Unterschied zwischen zwei Attributwerten und kann in der Metabeschreibung für jedes Attribut speziell definiert werden. Dieser Parameter beeinflußt dadurch das Breitenwachstum der Modellhierarchie.

Strukturelle Attribute: Jedes strukturelle Attribut A_i wird laut Definition 7.5 durch eine Menge von n-Tupeln beschrieben $A_i = \{t_1, \ldots, t_s\}$. Die im folgenden dafür definierte Bewertungsfunktion $Eval_N$ dient daher als Grundlage für die Definition der Bewertungsfunktion für strukturelle Attribute:

$$Eval_{Attr}(A_i) = \frac{1}{s} * \sum_{j=1}^{s} (Eval_N(t_j)) \tag{7.6}$$

Bewertungsfunktion für n-Tupel

In die Berechung der Bewertungsfunktion $Eval_N$ für ein n-Tupel $t_j = (b_1, \ldots, b_n, r)$ geht sowohl die relative Häufigkeit ein, mit der das n-Tupel in der Klasse auftritt, als auch die Bewertungsfunktion für das optionale Relationsattribut r. Durch diese Bewertung wird zum einen festgestellt, wie oft ein Merkmal überhaupt in den Instanzen einer Klasse vorkommt, und zum anderen, wie gut dann die Übereinstimmung in diesem Merkmal tatsächlich ist.

$$Eval_N(t_j) = \begin{cases} \frac{P(t_j)}{n} \cdot \sum_{l=1}^{n} (Eval_B(b_l)) & \text{, ohne Relationsattribut} \\ \frac{P(t_j)}{n} \cdot \left(\sum_{l=1}^{n} (Eval_B(b_l)) + Eval_{Attr}(r) \right) & \text{, mit Relationsattribut } r \end{cases} \tag{7.7}$$

$P(t_j)$ ist die Wahrscheinlichkeit, daß das n-Tupel t_j in der Klasse enthalten ist. Die Bewertungsfunktion wird zurückgeführt auf die Bewertungen $Eval_B$ der beteiligten Basiselemente.

Bewertungsfunktion für Basiselemente

Jedes Basiselement der Wissensrepräsentation besteht laut Definition 7.3 aus einer Menge von Attributen. In der Metabeschreibung kann jeweils angegeben werden, welche Attribute tatsächlich beim Lern- oder Klassifikationsvorgang berücksichtigt werden sollen. Hier wird der Übersichtlichkeit halber einfach unterstellt, daß alle Attribute in die Bewertung eingehen.

Die Attribute des Basiselements b_l sind gegeben durch $\mathcal{AT} = Attrs(Metabeschreibung(b_l))$. Der Zugriff auf ein einzelnes Attribut findet wieder mit $Attr$ statt.

$$Eval_B(b_l) = \frac{1}{|\mathcal{AT}|} * \sum_{a \in \mathcal{AT}} Eval_{Attr}(Attr(b_l, a)) \tag{7.8}$$

Bewertungsfunktion für 2D-Ansichten

Eine 2D-Ansicht a_l besteht ähnlich zu den Basiselementen aus einer Menge von Attributen. Diese sind wieder gegeben durch $\mathcal{AT} = Attrs(Metabeschreibung(a_l))$. Neben nominalen und numerischen Attributen werden hier jedoch auch strukturelle Attribute verwendet. Die Bewertungsfunktion stützt sich wie oben auf die bereits bewerteten Teilkomponenten ab:

$$Eval_A(a_l) = \frac{1}{|\mathcal{AT}|} * \sum_{a \in \mathcal{AT}} Eval_{Attr}(Attr(a_l, a)) \tag{7.9}$$

Bewertungsfunktion für ein Modell

Ein Modell faßt verschiedene 2D-Ansichten zusammen und enthält eventuell weitere Attribute, die für das Gesamtobjekt gültig sind. Wie einleitend beschrieben, soll für die Bewertung eine Zuordnung einer konkreten Ansicht zu einem Modell auf der Basis der am besten passenden 2D-Ansicht des Modells vorgenommen werden. Alternativ wäre natürlich auch die Berücksichtigung mehrerer bester Zuordnungen möglich.

$$Eval_M(M_k) = \max_{an \in Ansichten(M_k)} Eval_A(an) \tag{7.10}$$

7.5.3 Bewertungsfunktion für Lernoperatoren

Die gesamten Definitionen für eine Bewertungsfunktion dienen letztendlich dafür, beim Einordnen einer neuen Ansicht in eine bestehende Modellhierarchie zur Klassifikation oder zum Lernen zu bewerten, welche Instanziierung welcher der 3 Klassifikations- bzw. 5 Lernoperatoren als nächstes verwendet werden sollte. Aus diesem Grund wird zum Abschluß basierend auf der bisher verwendeten Bewertungsfunktion definiert, wie eine Bewertung eines Lernoperators vorgenommen werden kann. Dazu wird eine Bewertung des aktuellen Knotens der Modellhierarchie vor und nach einer Anwendung des Operators vorgenommen. Diese entspricht von der Struktur her dem allgemeinen Aufbau der oben definierten Ballungsnützlichkeit und wendet je nach Attributtyp die entsprechenden Teilbewertungsfunktionen an. Bewertet wird die Zunahme korrekt vorhersagbarer Attributwerte im Modellknoten M ohne Unterteilung im Vergleich zur Unterteilung der Modelle in entsprechende Nachfolgeklassen, wie sie bei Anwendung der einzelnen Operatoren entstehen würden.

Das Prinzip der Evaluierungsfunktion für die einzelnen Lernoperatoren wird beispielhaft für den Lernoperator $Eval_{Füge\_in\_Knoten\_ein}$ gezeigt. Die Evaluierungsfunktionen für die restlichen vier Lernoperatoren werden analog definiert.

Sei $MH' = (\mathcal{M}', \mathcal{MK}')$ eine Kopie der Modellhierarchie MH, auf die die oben beschriebene Funktion $Füge\_in\_Knoten\_ein(MH', Modell, Ansichtname, Vorgänger, MA)$ angewendet wurde. Sei $\{M'_1, \ldots, M'_K\} = \mathcal{N}(modell)$ die Menge der Nachfolger des Modells *Modell* in der aktualisierten Modellhierarchie. Dann ist $Eval_{Füge\_in\_Knoten\_ein}$ wie folgt definiert:

$$Eval_{Füge\_in\_Knoten\_ein}(MH, Modell, Ansichtname, Vorgänger, MA) = \frac{\sum_{k=1}^{K} (P(M'_k) * Eval_M(M'_k)) \; - \; Eval_M(Modell)}{K} \tag{7.11}$$

Die Parameter MH, $Ansichtname$, $Vorgänger$ und MA werden für den Aufruf der Funktion $Füge\_in\_Knoten\_ein$ benötigt, durch den die M'_k erzeugt werden. $P(M'_k)$ ist die relative Häufigkeit der ausgewählten Ansicht MA des Nachfolgermodells, d.h. die Anzahl der Beispiele für diese Ansicht in M'_k im Vergleich zur Anzahl für diese Ansicht in *Modell*. Die Normierung über die Anzahl der Untermodelle wird wie oben durchgeführt, um Partitionierungen unterschiedlicher Größen vergleichen zu können.

In den nachfolgend beschriebenen Lernverfahren wird als lokales Optimierungskriterium die Maximierung der obigen Funktionen über alle Nachfolgermodelle und alle Lern- bzw. Klassifikationsoperatoren verwendet. Der Berechnungsaufwand für die Bewertung kann dadurch reduziert werden, daß auf einer tieferen Ebene nur noch solche 2D-Ansichten berücksichtigt werden, die Nachfolger einer 2D-Ansicht sind, die auf höherer Ebene ausgewählt wurde. Zu diesem Zweck wird in den Operatoren der Parameter *Vorgänger* eingesetzt. Außerdem kann der Berechnungsaufwand verringert werden, indem die Zuordnung und die Bewertungsergebnisse für den ausgewählten besten Nachfolger gespeichert werden. Dieser ist nämlich auf der nächsten Stufe selbst das *Modell* und muß dann wieder neu berechnet werden.

7.5.4 Einlernen von Objektansichten

Durch den Einsatz der oben definierten Lernoperatoren ist es möglich, initial eine Modellhierarchie für den Objektagenten zu erzeugen, und diese während des Einsatzes des Gesamtsystems sukzessive zu verbessern. Dabei werden zwei unterschiedliche Lernverfahren betrachtet, die grob als *überwacht* und *unüberwacht* bezeichnet werden können (s. Kapitel 2.2.2).

Unüberwachtes Lernen

Im Falle des unüberwachten Lernens soll das System selbständig eine Unterteilung und Hierarchisierung einer Menge von Objektansichten vornehmen. Diese Aufgabe stellt sich z.B. dann, wenn das Gesamtsystem in einer teilweise unbekannten Umgebung agiert, dabei Objekte visuell wahrnimmt und daran Aktionen ausführen soll. Zu diesem Zweck muß das System selbständig eine Einordnung in ein bereits bestehendes Modell, die Generierung eines neuen Modells und eventuell sogar die Umstrukturierung der Hierarchie vornehmen können. Ein Benutzer wird in diesen Vorgang nicht involviert, daher die Bezeichnung *unüberwacht*.

Das Verfahren führt bei einer Einordnung der neuen Objektansicht von der Wurzel aus eine sukzessive lokale Optimierung der Bewertungsfunktion durch. Gemäß der Definition der Bewertungsfunktion wird damit auf jeder Ebene eine Strukturierung in Untermodelle erreicht, die die Gemeinsamkeiten der Instanzen eines Untermodells und gleichzeitig die Unterschiede zwischen Instanzen unterschiedlicher Untermodelle maximiert.

Algorithmus 7.8 (Unüberwachtes Lernen einer Objektansicht)

Eingabe:
- Aktueller Modellknoten *Modell* der Modellhierarchie $MH = (\mathcal{M}, \mathcal{MK})$
- Die Modell-2D-Ansicht *Vorgänger*, der die Instanz *OA* im Vorgängermodell zugeordnet wurde (nil, falls noch kein Vorgängermodell untersucht wurde)
- Eine Objekt-2D-Ansicht *OA*

Ausgabe:
- Die aktualisierte Modellhierarchie *MH*

Variablen:
- *N1, N2, vereinigung* sind Modelle der Modellhierarchie
- *MA* ist die beste Modell-2D-Ansicht von *Modell* für *OA*
- *F, E, S, V* sind Bewertungen für die verschiedenen Lernoperatoren

funct *Unüberwachtes Lernen* (**var** *MH, Modell, Vorgänger, OA*)
  if $|\mathcal{N}(Modell)| = 0$ **then** (* *Modell* ist ein Blatt *)
    Füge_in_Blatt_ein (*MH, Modell, Vorgänger, OA*)
    return (*Modell*)
  else
    $MA := Füge\_in\_Knoten\_ein(MH, Modell, nil, Vorgänger, OA)$
    Wähle $N1 \in \mathcal{N}(Modell)$ so, daß
      $Eval_{Füge\_in\_Knoten\_ein}(MH, N1, \text{nil}, MA, OA) =$
      $\max_{m \in \mathcal{N}(Modell)} Eval_{Füge\_in\_Knoten\_ein}(MH, m, \text{nil}, MA, OA)$
    $F := Eval_{Füge\_in\_Knoten\_ein}(MH, N1, \text{nil}, MA, OA)$
    Wähle $N2 \in \mathcal{N}(Modell)$ so, daß

$Eval_{Füge\_in\_Knoten\_ein}(MH, N2,$ nil $, MA, OA) =$
$\max_{m \in \mathcal{N}(Modell) \setminus \{N1\}} Eval_{Füge\_in\_Knoten\_ein}(MH, m,$ nil $, MA, OA)$
$E := Eval_{Erzeuge\_Blatt}(MH, Modell, MA, OA)$
$S := Eval_{Splitte\_Modell}(MH, Modell, N1)$
$V := Eval_{Vereinige\_Modelle}(MH, Modell, N1, N2)$
if $max(F, E, S, V) > cutoff$ **then**
  if $F = max(F, E, S, V)$ **then**
    Unüberwachtes_Lernen$(MH, N1, MA, OA)$
  if $E = max(F, E, S, V)$ **then**
    Erzeuge_Blatt$(MH, Modell, MA, OA)$
  if $S = max(F, E, S, V)$ **then**
    Splitte_Modell$(MH, Modell, N1)$
    Unüberwachtes_Lernen$(MH, Modell, Vorgänger, OA)$
  if $V = max(F, E, S, V)$ **then**
    vereinigung := *Vereinige_Modelle*$(MH, Modell, N1, N2)$
    Unüberwachtes_Lernen$(MH,$ *vereinigung*$, MA, OA)$

Der initiale Aufruf des Algorithmus lautet dann: *Unüberwachtes_Lernen*(*Modellhierarchie*, *Wurzel*, *nil*, *Objektansicht*).

Im unüberwachten Fall kann nicht zwischen einer neuen 2D-Ansicht eines bestehenden Modells und einer 2D-Ansicht eines neuen Modells unterschieden werden. Aus diesem Grund wird hier immer ein neues Modell erzeugt, d.h. unüberwachtes Lernen erzeugt nie neue Ansichten für bestehende Modelle, sondern neue Modelle mit einer einzigen Ansicht.

Im Algorithmus wird ein Parameter *cutoff* verwendet, um das Tiefenwachstum der entstehenden Modellhierarchie zu beschränken. Ohne einen solchen Parameter würde jede Objektansicht bis in ein bestehendes Blatt oder ein neues Blatt eingelernt. Durch Verwendung des *cutoff* kann eine Einordnung der Ansicht auch in einem inneren Knoten der Hierarchie beendet werden. Damit wird insbesondere bei verrauschten Daten eine Überanpassung (engl. overfitting) an die vorgelegten Beispiele und eine unnötige Unterscheidung sehr ähnlicher Beispiele (insbesondere bei numerischen Daten) vermieden. Schließlich resultiert aus einem solchen Vorgehen auch ein geringerer Speicherplatzbedarf, da nicht jede Instanz in der Modellhierarchie gespeichert bleibt.

Überwachtes Lernen

Der Fall des überwachten Lernens von Objektansichten tritt dann auf, wenn der Benutzer das System unterstützt und angibt, in welchen Modellknoten und eventuell sogar in welche Modell-2D-Ansicht dieses Knotens eine Objekt-2D-Ansicht eingelernt werden soll. Das Erzeugen neuer Modelle ist in dieser Lernvariante nicht zugelassen. Grund dafür ist, daß die durch überwachte Eingriffe entstehende Hierarchie auch wieder für weitergehendes unüberwachtes Lernen eingesetzt werden soll. Die Ballungsnützlichkeit, die als Maß für das Einlernen und auch Klassifizieren verwendet wird, würde durch Hierarchiemanipulationen durch den Benutzer jedoch gestört werden. Daher ist hier lediglich das weitere Einbringen von Objekt-2D-Ansichten für ein Modell vorgesehen.

Bezüglich der Angabe der Modell-2D-Ansicht für ein Modell sind drei mögliche Fälle zu unterscheiden: 1. Der Benutzer gibt eine bereits existierende Modell-2D-Ansicht vor, 2. Der Benutzer wünscht eine neue Modell-2D-Ansicht oder 3. Die Modell-2D-Ansicht wird nicht angegeben und das System wählt selbständig die am besten passende aus.

Der im folgenden angegebene Algorithmus realisiert einen überwachten Lernvorgang. Da die entstehende Modellhierarchie auch weiterhin zur Klassifikation und zum unüberwachten Lernen verwendet werden soll, darf die neue Objekt-2D-Ansicht nicht nur in das vorgegebene Modell integriert werden, sondern es muß ebenfalls eine Aktualisierung der Knoten auf dem Pfad von der Wurzel zu dem gegebenen Modellknoten durchgeführt werden. Diese Korrektur wird bottom-up, d.h. vom Modellknoten zur Wurzel hin, durchgeführt.

Die unterlagerte 2D-Ansichtenhierarchie muß bei dem Lernschritt ebenfalls aktualisiert werden. Beim Übergang von einem Modell zu dessen Vorgänger ist dann zu unterscheiden, ob zu der verwendeten 2D-Ansicht ein Vorgänger existiert oder nicht. Im ersten Fall, der immer dann auftritt, wenn die verwendete 2D-Ansicht bereits vorhanden war, wird die Vorgänger-2D-Ansicht verwendet. Im zweiten Fall, der vorliegt, wenn eine neue 2D-Ansicht erzeugt wird, wird wieder die am besten passendste Modell-2D-Ansicht bestimmt bzw. eine neue erzeugt und eine entsprechende Aktualisierung der Ansichtenhierarchie vorgenommen. Durch dieses Vorgehen wird auch innerhalb der Ansichten sukzessive von Details abstrahiert.

Algorithmus 7.9 (Überwachtes Lernen einer Objektansicht)

Eingabe:
- Aktueller Modellknoten *Modell* der Modellhierarchie $MH = (\mathcal{M}, \mathcal{MK})$ mit der zugehörigen Ansichtenhierarchie $(\mathcal{A}, \mathcal{AK})$
- Das Modell *Einfügemodell*, in das die neue Objekt-2D-Ansicht eingefügt werden soll
- Der Name *Ansichtname* der Modell-2D-Ansicht, in die *OA* eingefügt werden soll oder 'unbekannt' oder 'neu'
- Eine Objekt-2D-Ansicht *OA*
- *neu* gibt beim Aufstieg in der Rekursion an, ob im tieferen Modell eine neue Ansicht erzeugt wurde oder nicht

Ausgabe:
- Die aktualisierte Modellhierarchie *MH*
- Die ausgewählte Modell-2D-Ansicht *MA*

Variablen:
- *Nachfolger* ist ein Modell der Hierarchie
- *Nachfolger-Ansicht* ist die Modell-2D-Ansicht, in die die neue Ansicht eine Ebene tiefer eingefügt wird

funct *Überwachtes Lernen* (**var** *MH, Modell, Einfügemodell, Ansichtname, OA*, **var** *neu*)
  if *Modell* = *Einfügemodell* **then**
    $\mathcal{A}' := \mathcal{A}$
    MA := *Füge_in_Knoten_ein* (*MH, Modell, Ansichtname, nil, OA*)
    if $MA \notin \mathcal{A}'$ **then**
      neu := *true*
    return (*MA*)
  else
    Wähle *Nachfolger* $\in \mathcal{N}$(*Modell*) so, daß

Nachfolger = *Einfügemodell* $\vee$ (*Nachfolger*, *Einfügemodell*) $\in \mathcal{MK}$
$\vee \exists c_1, \ldots, c_n, n \geq 1 : (Nachfolger, c_1) \in \mathcal{MK} \wedge (c_1, c_2) \in \mathcal{MK} \wedge \ldots$
$\wedge (c_{n-1}, c_n) \in \mathcal{MK}, (c_n, Einfügemodell) \in \mathcal{MK}$
if *Nachfolger* = *nil* **then**
error ('Vorgegebenes Modell nicht in Modellhierarchie vorhanden.')
else
Nachfolger-Ansicht := *Überwachtes Lernen*(*MH*, *Nachfolger*, *Einfügemodell*, *Ansichtname*, *OA*, *neu*)
$\mathcal{A}' := \mathcal{A}$
if *neu* = *true* **then**
MA := *Füge_in_Knoten_ein*(*MH*, *Modell*,'*unbekannt*', *nil*, *OA*)
else
MA := *Füge_in_Knoten_ein*(*MH*, *Modell*, *nil*, *nil*, *OA*)
if *MA* $\notin \mathcal{A}'$ **then**
neu := *true*
$\mathcal{AK} := \mathcal{AK} \cup \{(MA, Nachfolger\text{-}Ansicht)\}$
return (*MA*)

Der Aufruf des überwachten Lernens ist dann *Überwachtes_Lernen*(*Modellhierarchie*, *Wurzel*, *Einfügemodell*, *Ansichtname oder 'unbekannt' oder 'neu'*, *Objektansicht*, *nil*).

7.6 Klassifikation

Die wichtigste Anwendung der durch Lernvorgänge aufgebauten Modellhierarchie ist die Klassifikation einer neuen Objekt-2D-Ansicht. Diese Klassifikation kann nahezu identisch zum Vorgang des unüberwachten Lernens durchgeführt werden. Auch dort wurde ein Modell gesucht, in das die neu einzulernende Ansicht am besten passen würde.

Aus diesem Grund wird eine Teilmenge der Operatoren und die gleiche Bewertungsfunktion wie beim unüberwachten Lernen bei der Klassifikation eingesetzt. Lediglich die restrukturierenden Operatoren *Aufsplitten* und *Zusammenfassen* werden nicht betrachtet. Außerdem wird bei der Klassifikation keine Aktualisierung der einzelnen betrachteten Modellknoten vorgenommen.

Natürlich ist auch ein Modus der Klassifikation möglich, in dem parallel ein Einlernen stattfindet. Dabei muß dann das in Abschnitt 7.7 beschriebene, aufwendigere Zuordnungsverfahren für Lernvorgänge verwendet werden, und die Klassifikation benötigt mehr Zeit. Damit läßt sich bereits der gesamte Klassifikationsalgorithmus formulieren:

Algorithmus 7.10 (Klassifikation einer Objektansicht)

Eingabe: · Aktueller Modellknoten $Modell = (\mathcal{A}_M, \mathcal{T}_M)$ der Modellhierarchie $MH = (\mathcal{M}, \mathcal{MK})$
· Eine Objekt-2D-Ansicht OA
· Die Modell-2D-Ansicht *Vorgänger*, der die Instanz OA im Vorgängermodell zugeordnet wurde (nil, falls noch kein Vorgängermodell untersucht wurde)
Ausgabe: · Das zu OA passende Modell *Modell* in der Modellhierarchie MH
Variablen: · *Bester-Nachfolger* ist Modell der Hierarchie
· MA ist eine Modell-2D-Ansicht
· F, E sind Bewertungen der Klassifikationsalternativen

funct *Klassifikation* (**var** *MH, Modell, OA, Vorgänger*)
  if $|\mathcal{N}(Modell)| = 0$ **then** (* *Modell* ist ein Blatt *)
    return (*Modell*)
  else
    Wähle MA so, daß $Eval_A(MA) - Eval_A(MA \oplus OA) =$
      $\min_{a \in \mathcal{A}_M} Eval_A(a) - Eval_A(a \oplus OA)$
      $\wedge (V_A(MA) = Vorgänger \vee Vorgänger = nil)$
    Wähle $Bester\text{-}Nachfolger \in \mathcal{N}(Modell)$ so, daß
      $Eval_{Füge\_in\_Knoten\_ein}(MH, Bester\text{-}Nachfolger, nil, MA, OA) =$
      $\max_{m \in \mathcal{N}(Modell)} Eval_{Füge\_in\_Knoten\_ein}(MH, m, nil, MA, OA) \wedge$
      $\exists a \in Ansichten(Bester\text{-}Nachfolger) : V_A(a) = MA$
    $F := Eval_{Füge\_in\_Knoten\_ein}(MH, Bester\text{-}Nachfolger,$ nil $, MA, OA)$
    $E := Eval_{Erzeuge\_Blatt}(MH, Modell, MA, OA))$
    if $max(F, E) < Cutoff$ **then return** (*Modell*)
    else
      if $F = max(F, E)$ **then**
        return (*Klassifikation* (*MH, Bester-Nachfolger, OA, MA*))
      if $E = max(F, E)$ **then return** (*Modell*)

7.7 Zuordnungsverfahren

Im letzten Abschnitt dieses Kapitels muß nun noch definiert werden, wie die in den obigen Verfahren verwendete Zuordnung zwischen den Merkmalen einer Objekt-2D-Ansicht und einer Modell-2D-Ansicht durchgeführt werden kann. Das Zuordnungsproblem läßt sich dabei wie folgt formulieren:

Definition 7.25 (Zuordnung einer Objekt- zu einer Modell-2D-Ansicht) *Sei eine Modell-2D-Ansicht MA und eine Objekt-2D-Ansicht OA gegeben. Die Menge aller Basiselemente von MA und OA sei* $\mathcal{MB} = \{MB_1, \ldots, MB_n\}$ *bzw.* $\mathcal{OB} = \{OB_1, \ldots, OB_m\}$. *Die Menge aller n-Tupel sei* $\mathcal{MT} = \{MT_1, \ldots, MT_l\}$ *bzw.* $\mathcal{OT} = \{OT_1, \ldots, OT_k\}$. *Eine*

Zuordnung *Z ist eine Abbildung der einzelnen Mengen auf die jeweils korrespondierenden Mengen.*

$$Z : \begin{cases} \mathcal{MB} \rightarrow \mathcal{OB} \\ Z(MB_i) = OB_j \text{ oder } nil \\ \mathcal{OB} \rightarrow \mathcal{MB} \\ Z(OB_i) = MB_j \text{ oder } nil \end{cases}$$

und entsprechend für $\mathcal{MT}$ und $\mathcal{OT}$. Jedem Modellelement wird nach Möglichkeit ein Objektelement zugeordnet und umgekehrt.

Bei der Zuordnung sind drei Fälle zu unterscheiden: 1. Zu einem Modellbasiselement oder -n-Tupel gibt es genau ein Objektbasiselement oder -n-Tupel. 2. Zu einem Modellbasiselement oder -n-Tupel gibt es kein Objektbasiselement oder -n-Tupel, d.h. die Zähler und Summen zur Berechnung der Wahrscheinlichkeitsparameter bleiben unverändert. 3. Zu einem Objektbasiselement oder -n-Tupel gibt es kein Modellbasiselement oder -n-Tupel. Dann wird ein entsprechendes neues Modellbasiselement oder -n-Tupel angelegt. Bisher werden noch keine Mehrfachzuordnungen zwischen Modell- und Objektelementen berücksichtigt.

Definition 7.26 (Zuordnungsproblem) *Das **Zuordnungsproblem** besteht darin, für eine gegebene Objekt-2D-Ansicht und eine Modell-2D-Ansicht eine Zuordnung zu finden, die eine möglichst maximale Bewertung erhält.*

$$Eval_A(MA \oplus OA) = max$$

Für das geschilderte Problem kann aus Komplexitätsgründen (NP-Vollständigkeit) nur ein heuristisches Suchverfahren eingesetzt werden, da eine vollständige Untersuchung aller Zuordnungen zeitlich nicht praktikabel ist. Aus dem Einsatz des Zuordnungsalgorithmus ergeben sich eine Reihe von Anforderungen:

1. Für die Klassifikation muß die Zuordnung möglichst schnell sein und nicht notwendigerweise optimal. Natürlich sollte auch diese Zuordnung möglichst gut sein, um ein möglichst gutes Zuordnungsergebnis zu erhalten. Für das Zusammenspiel mit den anderen Agenten ist eine mehrstufige Klassifikation vorgesehen, d.h. bei der Aufnahme eines neuen Objektes wird zunächst eine grobe Klassifikation durchgeführt und während der Weiterverarbeitung dieser Information durch die anderen Agenten wird vom Objektagenten eine genauere Klassifikation vorgenommen. Dabei soll auch die Integration weiterer Kameraaufnahmen möglich sein – was bisher jedoch nicht genauer untersucht wurde.

2. In der Lernphase sollte die Zuordnung möglichst gut sein, da hier eine Veränderung der Modellhierarchie auf der Basis der Zuordnung vorgenommen werden soll. Auch diese Lösung wird jedoch aus Zeitgründen nur suboptimal sein.

3. Als Suchverfahren sollte eine heuristische Suche verwendet werden, die die relationale Struktur zwischen den Basiselementen der Wissensrepräsentation ausnutzt.

4. Das Zuordnungsverfahren sollte ebenfalls lernfähig sein, d.h. über mehrere Zuordnungen hinweg sollten sich wichtigere oder leichter zuzuordnende Merkmale herausstellen, die dann entsprechend frühzeitig berücksichtigt werden.

5. Daneben sollte a priori Wissen über die Wichtigkeit bestimmter Merkmale in die Zuordnung integriert werden können.

6. Die Zuordnung kann im allgemeinen keine vollständige Lösung liefern, da Basismerkmale durch Aufnahmefehler oder ähnliches nicht vorhanden sein können.

Der entwickelte Zuordnungsalgorithmus nutzt im wesentlichen die relationale Struktur zwischen den Basiselementen aus, die durch die als n-Tupel dargestellten strukturellen Attribute gegeben ist. Ziel jedes Schrittes des Zuordnungsalgorithmus ist eine weitere Zuordnung eines der n-Tupel und damit indirekt der darin referenzierten Basiselemente.

Während der Zuordnung wird ein Modell-n-Tupel bestimmt, für das als nächstes ein entsprechendes Objekt-n-Tupel gefunden werden soll. Für die Auswahl wird die dynamische Priorität verwendet, die zum einen vom aktuellen Zustand der Zuordnung abhängt, konkret von der Anzahl der Basiselemente eines n-Tupels, die noch nicht zugeordnet wurden, zum anderen von der statischen Priorität. Diese setzt sich zusammen aus einer a priori bekannten Wichtigkeit eines Attributs und der Häufigkeit, mit der ein Merkmal in bisherigen Beispielen aufgetreten ist. Die Wichtigkeit eines Attributs wird in der Metabeschreibung festgelegt (s. Abschnitt 7.3).

Definition 7.27 (statische Priorität) *Die **statische Priorität** eines Modell-n-Tupels MT_i ist definiert als*

$$SP(MT_i) = G(MT_i) \cdot H(MT_i)$$

Sie ist ein Maß für die Wichtigkeit dieses Merkmals basierend auf a-priori Wissen über das Attribut (G) und der Häufigkeit (H), mit der dieses Merkmal in bisherigen Objekt-2D-Ansichten aufgetreten ist.

Definition 7.28 (dynamische Priorität) *Die **dynamische Priorität** eines Modell-n-Tupels MT_i ist definiert als*

$$DP(MT_i) = SP(MT_i) \cdot \frac{1}{1 + OR(MT_i)}$$

Dabei gibt OR die Anzahl der noch offenen Referenzen für ein n-Tupel bei der Zuordnung an, d.h. die Anzahl der noch nicht zugeordneten Basiselemente des n-Tupels.

Bei der Zuordnung des nächsten n-Tupels werden nur solche Korrespondenzen betrachtet, bei denen Modell- und Objekt-n-Tupel zum gleichen strukturellen Attributtyp gehören, d.h. Ausprägung der gleichen Relation sind, und bei denen alle Basiselemente der n-Tupel, die bereits zugeordnet wurden, korrespondieren. Wird diese Bedingung erfüllt, so heißen die n-Tupel syntaktisch passend.

Definition 7.29 (syntaktisch passend) *Ein Objekt-n-Tupel OT heißt* ***syntaktisch passend*** *zu einem Modell-n-Tupel MT genau dann wenn 1. Beide n-Tupel vom gleichen strukturellen Attributtyp sind und 2. Eine Übereinstimmung in den zu diesem Zeitpunkt zugeordneten Basiselementen besteht. OT heißt* ***am besten passend*** *zu MT, wenn es aus der Menge aller noch nicht zugeordneten, syntaktisch passenden Objekt-n-Tupel die Bewertungsfunktion $Eval_N$ auf der Basis der bereits zugeordneten Basiselemente maximiert.*

Auf der Basis einer bestehenden (Teil-)Zuordnung wird ein neues Modell-n-Tupel ausgewählt und diesem das am besten passende Objekt-n-Tupel zugeordnet. Die dabei beteiligten Basiselemente werden mit Hilfe eines Greedy-Verfahrens zugeordnet und das Ergebnis an alle noch nicht zugeordneten n-Tupel weiterpropagiert. Damit ergibt sich der im folgenden angegebene Zuordnungsalgorithmus für den Fall der Klassifikation. Der Aufruf geschieht über $Zuordnung(MA, OA, \{\})$.

Algorithmus 7.11 (Zuordnung einer Objekt-2D-Ansicht zu einer Modell-2D-Ansicht)

Eingabe:
- Eine Modell-2D-Ansicht MA
- Eine Objekt-2D-Ansicht OA
- Die aktuelle Zuordnungsliste ZL

Ausgabe:
- Die Zuordnungsliste ZL – die Zuordnungen der Objektelemente zu den Modellelementen sind implizit enthalten

Variablen:
- $\mathcal{MT}$ ist die Menge der noch nicht zugeordneten Modell-n-Tupel von MA
- $\mathcal{OT}$ ist die Menge der noch nicht zugeordneten Objekt-n-Tupel von OA
- $\mathcal{MH}$ sind die n-Tupel in $\mathcal{MT}$ mit der höchsten dynamischen Priorität, die als nächstes zugeordnet werden sollten.
- MH ist das als nächstes zuzuordnende Modell-n-Tupel
- $\mathcal{OH}$ ist die Menge der Objekt-n-Tupel, die syntaktisch zu MH passen
- OH ist das am besten zu MH passende Objekt-n-Tupel aus $\mathcal{OH}$

funct $Zuordnung(MA, OA, ZL)$
$\mathcal{MT} := \{MT_i \mid MT_i$ nicht in ZLzugeordnet $\wedge\ MT_i$ ist n-Tupel in $MA\}$
$\mathcal{OT} := \{OT_i \mid OT_i$ nicht in ZLzugeordnet $\wedge\ OT_i$ ist n-Tupel in $OA\}$
$\mathcal{MH} := \{MT \in \mathcal{MT} \mid DP(MT) = max\}$
while $\mathcal{MH} \neq \{\}$ **do**
$MH := MT \in \mathcal{MH}$ mit $DP(MT) = max$
$\mathcal{MH} := \mathcal{MH} \setminus \{MH\}$
$\mathcal{OH} := \{OT \in \mathcal{OT} \mid OT$ ist syntaktisch passend zu $MH\}$
if $\mathcal{OH} \neq \{\}$ **then**
$OH :=$ am besten passende $OT \in \mathcal{OH}$
if $OH \neq nil$ **then**
$ZL := ZL \cup \{MH \rightarrow OH\} \cup$
$\{MB_i \rightarrow OB_j \mid MB_i$ ist Basiselement von $MH \wedge OB_j$ ist Basiselement von $OH \wedge Eval_N(MH \oplus OH) = max$ bei Zuordnung von MB_i zu $OB_j\}$
$\mathcal{OT} := \mathcal{OT} \setminus \{OH\}$
$\mathcal{MH} := \mathcal{MH} \cup \{MT \in \mathcal{MT} \mid \exists MB \in MT$, das in MH enthalten ist$\}$
$ZL := ZL \cup \{MT \rightarrow nil \mid MT$ nicht zugeordnet$\} \cup \{MB \rightarrow nil \mid MB$ nicht zugeordnet$\}$

$\cup \{neu \rightarrow OT \mid OT \text{ nicht zugeordnet}\} \cup \{neu \rightarrow OB \mid OB \text{ nicht zugeordnet}\}$
return (ZL)

Die Zuordnungsverfahren für Klassifikation und Lernen unterscheiden sich in der Auswahlstrategie für das nächste Objekt-n-Tupel. Bei der Klassifikation wird ein Greedy-Verfahren angewandt, d.h. es wird immer nur das am besten passende Objekt-n-Tupel berücksichtigt. Beim Lernvorgang wird eine Best-First-Suchstrategie verwendet, um bei jedem Zuordnungsschritt mehrere Alternativen zu berücksichtigen. Als Bewertungskriterium wird die Anzahl der nicht zugeordneten Modell-n-Tupel verwendet. Im entstehenden Suchbaum entspricht jeder Knoten einer möglichen Zuordnung und jede Kante einem syntaktisch passenden Objekt-n-Tupel für das ausgewählte Modell-n-Tupel. Ein Blatt des Suchbaums enthält alle Zuordnungspaare und die Anzahl der nicht zuzuordnenden Modell-n-Tupel. Dieser Wert wird im Baum zurückpropagiert, um so eventuell andere Teilsuchbäume frühzeitig abzuschneiden, weil die Anzahl der nicht zuzuordnenden Merkmale bereits größer ist, als in der bis dahin besten Lösung.

Der oben angegebene Zuordnungsalgorithmus muß für den Einsatz beim Lernen um einen Rekursionsschritt erweitert werden, in dem die verschiedenen Alternativen einer Zuordnung für ein Modell-n-Tupel entsprechend untersucht werden. Beim Lernvorgang werden für alle nicht zugeordneten Modell-n-Tupel und -basiselemente die Summen und Zähler zur Berechnung der Wahrscheinlichkeitsparameter nachfolgend nicht verändert. Für alle nicht zugeordneten Objekt-n-Tupel und -basiselemente werden neue Modell-n-Tupel bzw. -basiselemente in der Modell-2D-Ansicht erzeugt.[2]

7.8 Zusammenfassung

In diesem Kapitel wurden die Repräsentationsformalismen, die Lernverfahren und der Klassifikationsvorgang für den Objektagenten beschrieben. Als Kernidee wurde der CLASSIT-Algorithmus verwendet, der erstmalig auf eine gemeinsame Betrachtung von nominalen, numerischen und strukturellen Attributen erweitert wurde. Für die erweiterte Strukturierung von Objekt- und Modellrepräsentation wurden sowohl die Bewertungsfunktion der Ballungsnützlichkeit definiert, als auch ein geeignetes Zuordnungsverfahren entwickelt. Die Lern- und Klassifikationsoperatoren arbeiten sowohl auf der Modell- als auch der überlagerten Ansichtenhierarchie.

Die meisten neu entwickelten Methoden sind unabhängig vom konkreten Einsatz im Objektagenten auch für andere ähnliche Aufgabenstellungen einsetzbar. Ebenso können die Beschreibungen der Basiselemente und die betrachteten Relationen auf dreidimensionale Betrachtungen erweitert werden, wenn entsprechende Sensorik zur Verfügung steht.

[2] Eine sinnvolle Ergänzung wäre daher, selten zugeordnete Modell-n-Tupel und -basiselemente im Laufe der Zeit wieder zu vergessen, d.h. aus der Modell-2D-Ansicht zu löschen.

Kapitel 8

Akquisition von Objekteigenschaften

In diesem Kapitel wird das Vorgehen des Experimentieragenten zur Planung, Durchführung und Auswertung von Explorations- und Aktionsexperimenten beschrieben. Im folgenden wird das entwickelte Konzept dargestellt. Eine detaillierte Beschreibung der einzelnen Teilkomponenten und der genauen Definition der einzelnen Experimente findet sich in [Die92, Wei93].

8.1 Übersicht

Eine wichtige Teilaufgabe bei einer Manipulation in teilweise unbekannter Umgebung – und damit einer Manipulation teilweise unbekannter Objekte – ist die Durchführung von Experimenten zur Bestimmung von Objekteigenschaften. Diese Eigenschaften können zum einen Informationen über das Objekt sein, die z.B. durch eine 2D-Aufnahme, wie sie der Objektagent verwendet, nicht bestimmt werden können, wie 3D-Informationen, Masse oder Oberflächeneigenschaften. Zum anderen können Experimente durchgeführt werden, die das Verhalten eines Objektes bei einer bestimmten Manipulation untersuchen, so z.B. beim Greifen oder Verschieben, und dadurch die beste Vorgehensweise zur Durchführung dieser Aktion an dem Objekt ermitteln. Soweit möglich sollten diese Informationen wieder für Objekte einer ganzen Klasse gelten und in die in Kapitel 7 aufgebauten generischen Objektbeschreibungen integriert werden. Im folgenden werden zunächst kurz die wichtigsten bereits bekannten Arbeiten zum Einsatz von aktiven Experimenten dargestellt.

Den Einsatz taktiler Experimente für die modellbasierte Objekterkennung beschreibt [All87]. Zunächst werden mit Hilfe eines 3D-Sichtsystems „interessante" Bereiche eines Objektes identifiziert, die nachfolgend durch gezielte taktile Exploration untersucht werden. Dabei sind alle Objekte vollständig a priori modelliert und ein Lernvorgang findet nicht statt. Auch in den Arbeiten von Stansfield [Sta88b, Sta88a, Sta88c, Sta91] wird eine taktile Untersuchung im Anschluß an eine visuelle Aufnahme durchgeführt, um eine Objekterkennung zu ermöglichen und dann eine Greifaktion zu planen.

[BT87, Baj89] beschreiben einen Ansatz zur aktiven Perzeption, in dessen Kontext auch aktive Experimente eines Manipulators durchgeführt werden. Dazu zählen das *Erkunden*

Abbildung 8.1: Ablauf eines Experiments

von Oberflächeneigenschaften der Arbeitsfläche und von Eigenschaften eines zu manipulierenden Objektes wie Gewicht, Größe, etc.

Der Einsatz von Aktionsexperimenten wurde von Christiansen beim Tray-Tilting untersucht [MCM89, CTM90, Chr92a, Chr92b] (s. Kapitel 3.4.3). Zrimec und Mowforth [ZM93] untersuchen den Erwerb eines hierarchischen, qualitativen Modells für das 2D-Rotationsverhalten eines einfachen Objektes. Dabei wurde allerdings weder eine quantitative oder qualitative Objektbeschreibung verwendet noch eine Analyse durchgeführt, wenn Aktion und Vorhersage nicht übereinstimmten.

Einen Ansatz zum Lernen von zweidimensionalen Griffen an Objekten durch Experimente anstatt einer deduktiven geometrischen Analyse beschreiben Dunn und Segen [DS88]. Die Eingabe für die Experimentierkomponente ist die 2D-Kontur des zu greifenden Objekts, auf der die zu testenden Greifpunkte zufällig erzeugt werden. Das Ergebnis des Greifvorgangs wird abgespeichert, so daß in einer späteren gleichen Aufgabe die Erfahrungen weiterverwendet werden können. Eine Verallgemeinerung der Lernergebnisse findet allerdings nicht statt.

Ein abstrakterer Ansatz zur Akquisition von Planungswissen durch Experimente wird in [CG90] vorgestellt. Für das Prodigy-System sollen die Vor- und Nachbedingungen der existierenden Operatoren durch gezielte Experimente korrigiert werden. Dabei wird davon ausgegangen, daß das vorliegende Wissen korrekt, aber eventuell unvollständig ist.

Zunächst wird im nächsten Abschnitt die Struktur des Experimentieragenten im Überblick erläutert. Eine detailliertere Beschreibung der Teilkomponenten zur Durchführung einer Exploration eines Objektes, d.h. zur Bestimmung von Objekteigenschaften, und zur Durchführung von Aktionsexperimenten zur Bestimmung der Reaktion eines Objektes bei einer bestimmten Aktion folgt in den beiden nachfolgenden Abschnitten.

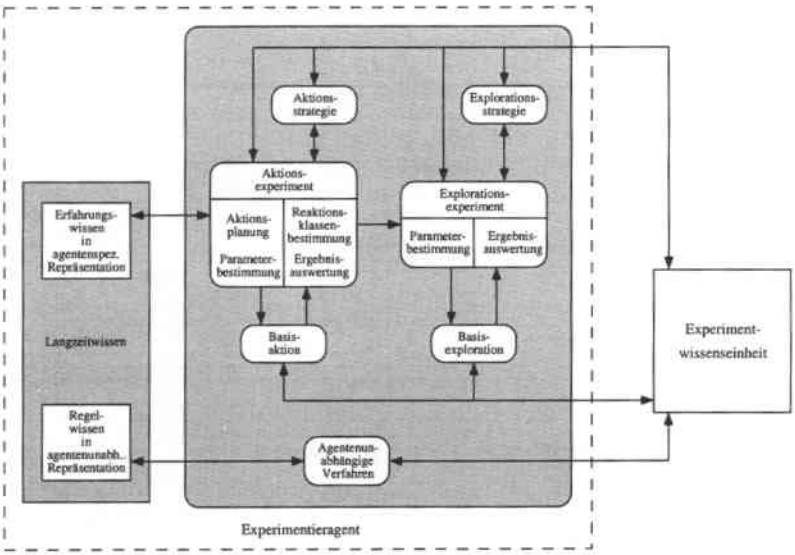

Abbildung 8.2: Struktur des Experimentieragenten

8.2 Struktur des Experimentieragenten

Die Experimentierkomponente der Systemarchitektur plant Experimente, führt diese aus, analysiert die resultierenden Meßwerte und bewirkt entsprechende Einträge in die Objektrepräsentation des Objektagenten (s. Abbildung 8.1). Die Experimente, die durchgeführt werden, sind zum einen Explorationsexperimente zur Bestimmung von Objektinformationen und Objekteigenschaften und zum anderen Aktionsexperimente zur Bestimmung des Verhaltens eines Objektes bei einer bestimmten Aktion. Die Struktur und das Zusammenspiel der einzelnen Komponenten des Experimentieragenten sind in Abbildung 8.2 gezeigt.

Der Teil zur Durchführung einer Exploration eines Objektes umfaßt die Explorationsstrategie, die Explorationsexperimente und die Basisexploration. Eine gezielt angeforderte Information über ein Objektmerkmal wird direkt von einem Explorationsexperiment bearbeitet. Alternativ kann vom System aber auch die Anforderung kommen, das Objekt zu erkunden. Die Explorationsstrategie bestimmt dann einen geeigneten Satz von Explorationsexperimenten. Um den Umfang der Erkundung einzuschränken, werden jeder Exploration Kosten zugeordnet, die den Aufwand für deren Durchführung widerspiegeln. Das Anfangsguthaben für die möglichen Erkundungen wird dem Experimentieragent als Parameter übergeben. Während der Planung eines Explorationsexperiments werden Objektpunkte bestimmt, die durch eine Basisexploration gezielt angetastet werden. Für jedes Explorationsexperiment ist definiert, wie die Ergebnisse der Basisexplorationen ausgewer-

tet werden müssen, um die gewünschte Objekteigenschaft zu bestimmen.

Die Teilkomponente für die Untersuchung von Aktionen ist ebenfalls in drei Module gegliedert. Innerhalb der Aktionsstrategie wird – wenn nötig – untersucht, welche Manipulationsaufgaben in welcher Reihenfolge durchgeführt werden sollen. Innerhalb eines Aktionsexperiments wird Erfahrungswissen bezüglich einer vorgegebenen Zielreaktion des Objektes und einer speziellen Aktionsart erworben. Ein Aktionsexperiment basiert wiederum auf einer Basisaktion, die eine reale Aktionssequenz ableitet. Innerhalb der Aktionsplanung können Explorationsexperimente durchgeführt werden, um weitere Objektinformationen für die Entscheidungsprozesse abzuleiten.

8.3 Explorationsexperimente

Als Explorationsexperimente werden Aktionen bezeichnet, die zur Bestimmung von geometrischen Eigenschaften eines Objektes dienen. Die Aktion besteht darin, daß mit Hilfe einer Spitze im Greifer des Manipulators verschiedene Punkte auf dem Objekt angetastet werden, die gemessenen Werte des Kraft-/Momentensensors als Abbruchkriterium dienen, und anschließend die Koordinaten des angefahrenen Punktes bestimmt werden. Gemäß der Basiselemente, die in Kapitel 7 für die Objektbeschreibung definiert wurden, können folgende Explorationsexperimente unterschieden werden:

1. Bestimmen der 3D-Koordinaten eines Punktes
2. Bestimmen der 3D-Länge und des Typs (gekrümmt, gerade, konkav, konvex) einer Kante
3. Bestimmen der Orientierung und Krümmung einer Fläche und deren 3D-Typ

Zusätzlich ist eine Typisierung des Gebiets zwischen zwei Kanten in Mulde, Hügel oder ebene Fläche möglich. Diese Exploration ist dann wichtig, wenn nur Kanteninformationen aber keine Flächeninformationen vorliegen.

Aufgabe der Explorationsstrategie ist es, geeignete Explorationsexperimente durchzuführen, wenn keine gezielten Vorgaben vorhanden sind. Jeder Typ von Explorationsexperiment ist mit Kosten verbunden, die widerspiegeln, daß die zu seiner Durchführung nötigen Antastungen eine bestimmte Zeit beanspruchen. Alle möglichen Explorationsexperimente eines Typs werden entweder mittels einer Heuristik oder zufällig geordnet und anschließend zu einer Prioritätsliste zusammengefaßt. Für die Auswahl des nächsten Explorationsexperiments wird dann entweder das erste Element der Prioritätsliste ausgewählt oder mit einer bestimmten Wahrscheinlichkeit ein zufälliges Element.

Algorithmus 8.1 (Explorationsstrategie)

Eingabe: · Objekt-2D-Ansicht OA
· *Art* der Exploration (Punkt, Kante, Fläche oder Menge davon)
· Guthaben G
Ausgabe: · Als Seiteneffekt werden Informationen für OA erzeugt
Variablen: · $\mathcal{P}$ ist die Prioritätsliste

funct *Explorationsstrategie* (OA, Art, G)
  $\mathcal{P}$:= Erzeuge Prioritätsliste für die Menge aller möglichen Explorationsexperimente der gegebenen *Art* für OA
  while $G \geq 0 \wedge \mathcal{P} \neq \{\}$ **do**
    $\mathcal{P} := \mathcal{P} \setminus \{\text{Experimente } E \mid Kosten(E) > G\}$
    if Zufallsauswahl **then**
      $E :=$ Wähle zufällig ein Explorationsexperiment aus $\mathcal{P}$ aus
    else
      $E :=$ Erstes Explorationsexperiment aus $\mathcal{P}$
    Führe das Explorationsexperiment E aus
    $\mathcal{P} := \mathcal{P} \setminus \{E\} \setminus \{\text{Experimente, die als Nebeneffekt ausgeführt wurden}\}$
    $G := G - Kosten(E)$

Die Explorationsexperimente, die entweder direkt von anderen Systemkomponenten oder von der Explorationsstrategie aus aufgerufen werden, enthalten die für die oben genannten Explorationen notwendigen Verfahren. Der Ablauf erfolgt immer nach dem gleichen Muster: Zunächst werden vorhandene 2D- und 3D-Informationen verarbeitet, dann Basisexplorationen durchgeführt für 3D-Informationen, die benötigt werden, aber noch nicht vorhanden sind, und anschließend wird die eigentliche Folge von Basisexplorationen ausgeführt und die Ergebnisse entsprechend ausgewertet. In [Wei93] sind die Verfahren für die Bestimmung von 3D-Kantenlänge, Kantenhöhe, Kantentyp, Typisierung des Gebiets zwischen zwei Kanten, Flächentyp, Flächennormale und Flächenkrümmung detailliert beschrieben.

Explorationsexperimente bauen ihrerseits auf sogenannten Basisexplorationen auf. Ziel einer Basisexploration ist die Bestimmung der 3D-Koordinaten eines Punktes. Dabei werden drei Arten von Punkten unterschieden: 1. Eckpunkte, 2. Punkte auf einer Kante und 3. Punkte auf einer Fläche. Grund für diese Unterscheidung ist, daß die Antastung im wesentlichen auf der Basis einer 2D-Ansicht vorgenommen wird, die vom Objektagenten erzeugt wird. Aufgrund der Ungenauigkeit von Sensorik und Aktorik ist es nicht möglich, einen Punkt exakt anzutasten. Daher müssen lokale Strategien entwickelt werden, um die Umgebung des Punktes auszuwerten und dadurch eine möglichst exakte 3D-Position zu bestimmen. Bei der Berechnung eines Flächenpunktes wird tatsächlich nur der gewünschte Punkt angefahren. Bei einem Kantenpunkt wird senkrecht zur Kante eine Folge von Antastpunkten angefahren und aus den Knickstellen die Lage der Kante berechnet. Ein Eckpunkt entsteht durch den Schnitt zweier oder mehrerer Kanten. Bei zwei sich schneidenden Kanten wird zunächst entlang der Winkelhalbierenden angetastet und geprüft, ob tatsächlich eine Ecke vorliegt. Anschließend werden in der Nähe des Eckpunktes Antastsequenzen für Kantenpunkte durchgeführt und daraus die Ecke berechnet. Treffen mehr als zwei Kanten im Eckpunkt zusammen, werden Kantenantastsequenzen in mehreren Winkeln zum Eckpunkt hin durchgeführt.

8.4 Aktionsexperimente

Ziel der Aktionsexperimente ist die Bestimmung des Verhaltens eines Objektes bei einer Manipulationsaktion, z.B. Greifen oder Verschieben. Ein Aktionsexperiment ist hier immer

gerichtet, d.h. es wird eine gewünschte Reaktion der Objekte vorgegeben und versucht, diese durch Manipulationen zu erreichen.

Die *Aktionsstrategie* kann – wenn nötig – dazu eingesetzt werden, verschiedene Positionen zu ermitteln, von denen aus eine Objektansicht aufgenommen werden soll und basierend darauf, welche der möglichen Aktionsexperimente durchgeführt werden sollen. Falls eine Objektansicht durch den Objektagenten einem Modell zugeordnet werden kann, wird eventuell vorhandenes Erfahrungswissen für diese Objektklasse für die Planung der Aktionsexperimente verwendet.

Innerhalb eines Aktionsexperiments werden ein oder mehrere Experimente einer Aktionsart für eine Objektansicht durchgeführt. Zunächst wird in der Aktionsplanung der Parametervektor, der für die konkrete Aktionsdurchführung benötigt wird, sukzessive belegt. Nach der Durchführung der Aktion wird die Reaktionsklasse bestimmt und Erfahrungswissen aus mehreren Experimenten aufgebaut. Damit ergibt sich der folgende prinzipielle Ablauf eines Aktionsexperiments:

Algorithmus 8.2 (Aktionsexperiment)

Eingabe:
- Objekt-2D-Ansicht OA
- *Art* der Aktion
- Vorgegebene Zielreaktion R
- *Anzahl* der durchzuführenden Experimente
- Guthaben G für eventuell durchzuführende Explorationsexperimente

Ausgabe:
- Als Seiteneffekt werden Informationen und Erfahrungswissen für OA bzw. das zugehörige Modell erzeugt

funct *Aktionsexperiment*(*OA*, *Art*, *R*, *Anzahl*, *G*)
  while *Anzahl* > 0 **do**
    Parametervektor := *Aktionsplanung*(*OA*, *Art*, *R*, *G*)
    Basisaktion(*OA*, *Parametervektor*)
    Bestimme Reaktionsklasse und speichere das Ergebnis in *OA*
    Baue Erfahrungswissen auf, wenn mehrere Experimentergebnisse vorliegen
    Anzahl := *Anzahl* − 1

Die Bestimmung des Parametervektors erfolgt sukzessive und abhängig von der Art der durchzuführenden Aktion. Beispiel: Für die Aktion Greifen werden fünf Parameter belegt (s. Abbildung 8.3): 1. Grifftyp (Innen-/Außengriff), 2. Anrückfläche, 3. Kontaktkanten, 4. Kantenintervall (Anfang, Mitte, Ende), 5. Greifkraft (klein, mittel, groß). Die Greifaktion wird dann in Richtung der Kontaktkanten im Bereich der Kantenintervalle durchgeführt bis eine Kraftschwelle überschritten wird. Anschließend wird der Greifer mit der vorgegebenen Greifkraft je nach Grifftyp entweder geöffnet oder geschlossen.

Die Bestimmung der einzelnen Parameter kann durch vier verschiedene Methoden erreicht werden, die, wie in Abbildung 8.4 gezeigt, zusammenspielen können:

1. Auswahl durch Erfahrung: Wurden in früheren Experimenten verläßliche Parameterbelegungen bestimmt, können diese übernommen werden.

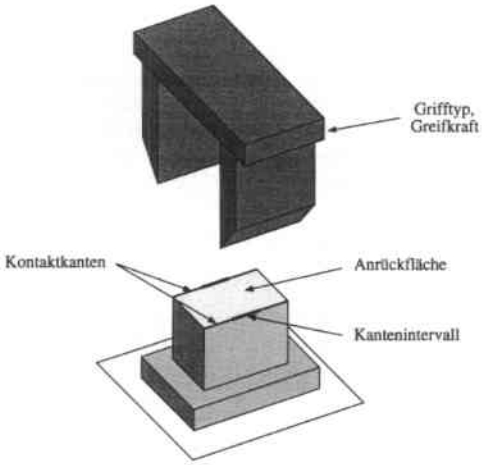

Abbildung 8.3: Parameter einer Greifaktion

2. Wissensbasierte Auswahl: Für die einzelnen Parameter können Regeln angegeben werden, wie in Abhängigkeit von bestimmten Objekteigenschaften der Parameter gewählt werden sollte. Für das Greifen sind z.B. Bewertungsregeln vorhanden, die für die Anrückfläche Orientierung, Flächenkrümmung und 2D-Fläche und für die Kontaktkanten die Parallelität, Kantenkrümmung, 2D-Länge, 3D-Länge und 2D-Distanz bewerten. Damit können beispielsweise lange, gerade, parallele Kanten als Kontaktkanten bevorzugt werden.

3. Deterministische Auswahl: Falls ein funktionaler Zusammenhang eines Parameters von vorher bestimmten Parameterwerten angegeben werden kann, ist eine direkte Berechnung möglich.

4. Zufallsauswahl: Neben den oben genannten Möglichkeiten zur Parameterbestimmung sollte mit einer gewissen Wahrscheinlichkeit auch eine zufällige Belegung des Parameters vorgenommen werden, um so auch an sich schlechter bewertete Parameterbelegungen durch ein Experiment zu testen.

Für die wissensbasierte Auswahl und die Zufallsauswahl können Filter definiert werden, die in Abhängigkeit von bestimmten Objekteigenschaften Parameterwerte ausschließen. Im Gegensatz zu den Bewertungsregeln wird dabei keine Bewertung vorgenommen, sondern der Parameter endgültig aus der Menge der möglichen Belegungen gelöscht. Ein weiterer Filter berücksichtigt die Ergebnisse erfolgloser Experimente, um so nicht wiederholt die Parameterbelegungen zu erzeugen, die laut wissensbasierter Auswahl die besten sind, die aber wiederholt im Experiment nicht erfolgreich waren.

Nach der Ausführung einer Basisaktion unter Verwendung des Parametervektors muß die Reaktionsklasse durch Interaktion mit dem Benutzer oder automatisch bestimmt werden.

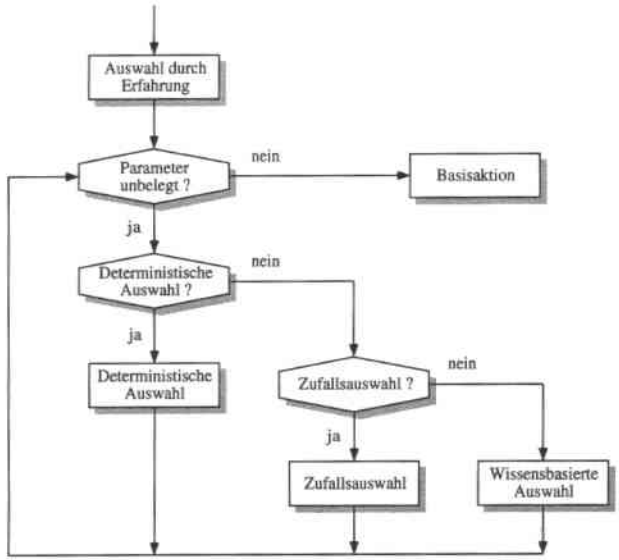

Abbildung 8.4: Belegung des Parametervektors

Als Reaktionsklassen werden beim Greifen z.B. 'gegriffen', 'verloren' und 'unbekannt' unterschieden.

Aus einer Menge von Experimentergebnissen kann Erfahrungswissen aufgebaut werden, in dem für ein Objekt oder eine Objektklasse die Erfolgswahrscheinlichkeit in den bisherigen Experimenten für bestimmte Parametervektoren und eine bestimmte Reaktionsklasse bestimmt wird.

8.5 Zusammenfassung

Die Experimentierkomponente des Systems ermöglicht die Bestimmung von im wesentlichen geometrischen Objekteigenschaften und von Reaktionen eines Objektes auf eine bestimmte Manipulationsaktion. Basierend auf den drei Ebenen Strategie, Experiment und Basisexperiment können entweder Explorations- oder Aktionsexperimente durchgeführt werden. Als Ausgangspunkt werden Objektansichten verwendet, wie sie vom Objektagenten erzeugt werden. Ergebnis der Experimente sind entweder zusätzliche Informationen über das Objekt oder Erfahrungswissen für den Zusammenhang von Objektklasse, Parameterauswahl und Reaktionsklasse.

Kapitel 9

Beispiele und Evaluierung

In diesem Kapitel werden die wichtigsten Teilkomponenten des Systems experimentell untersucht. Dazu wurden mit der in Anhang A beschriebenen Realisierung eine Reihe von Tests durchgeführt. Zunächst wird in Kapitel 9.1 die Systemarchitektur und das Verhalten des Systems bei einer Problemlösung analysiert. Die Möglichkeiten des Lernens von Regelwissen und die Induktion von Gültigkeitsbereichen werden in Kapitel 9.2 untersucht. Kapitel 9.3 liefert eine Betrachtung des Objektagenten, d.h. des Einsatzes von Lernverfahren zum Erwerb von Objektbeschreibungen. Schließlich werden in Kapitel 9.4 experimentelle Ergebnisse zum automatischen Erwerb von Objekteigenschaften dargestellt.

9.1 Einsatz der Systemarchitektur

Die Systemarchitektur selbst wurde in mehreren Szenarien getestet, die die unterschiedlichen Fähigkeiten des Systems aufzeigen sollen. In diesem Abschnitt wird anhand einiger Beispielaufgaben das Verhalten des Systems bei der Lösung von Problemen analysiert. Dabei werden zunächst keine Lernvorgänge berücksichtigt.

Als eine Aufgabe wird das Greifen und Bewegen eines Objektes betrachtet. Diese Aufgabe ist prinzipiell durch die Einnahme von sechs bzw. sieben Teilzielen (Relationen) lösbar (s. Abbildung 9.1). Zunächst wird die Relation „oberhalb" zum Gesamtobjekt eingenommen, anschließend wird abhängig von Objektmerkmalen entweder das Objekt außen oder an dem aufgesetzten Teilobjekt gegriffen. Im ersten Fall wird eine Ausrichtung zu einer Kante des Gesamtobjektes durchgeführt, der Greifer geöffnet und ein Teilziel „direkt oberhalb" des Objektes angefahren. Im zweiten Fall wird zunächst zwei Kanten des Teilobjektes die Rolle der Ausrichtkanten zugewiesen, anschließend eine Ausrichtung dazu vorgenommen, der Greifer geöffnet, die Relation „oberhalb" zu den beiden Kanten und anschließend die Relation „direkt oberhalb" zu den beiden Kanten eingenommen. In beiden Fällen soll anschließend ein Schließen des Greifers folgen und zum Schluß eine Position oberhalb der Arbeitsfläche eingenommen werden.

Im Wissen des Planagenten sind zum einen die Teilziele codiert, die jeweils eingenommen werden sollten, zum anderen die Rollenzuweisungen an einzelne Objektwissenseinheiten. Für die genannte Aufgabe werden im Planagenten 20 Regeln verwendet. Der Relationsagent verfügt über Beschreibungen der einzelnen Teilziele, über allgemeingültige Regeln

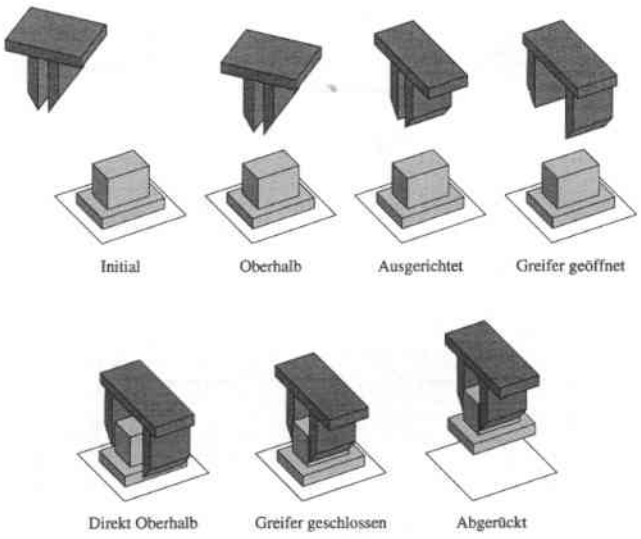

Abbildung 9.1: Teilzielfolge der Greifaufgabe

zur Ableitung von Bewegungshypothesen und zur Feststellung der Gültigkeit einer Relation. Für jede der sieben Relationstypen sind für jede der sechs Koordinaten und die Greiferposition eigene Regeln angegeben, damit ergeben sich 49 Regeln. Zusätzlich sind 20 allgemeingültige Regeln angegeben, so daß der Relationsagent insgesamt 69 Regeln besitzt.

Der Objektagent besitzt Regeln, die in Abhängigkeit von der Objektbreite und der Objektlänge bestimmen, ob das Objekt insgesamt, d.h. außen gegriffen werden kann, oder ob der Griff an dem aufgesetzten Quader erfolgen soll. Der Aktoragent enthält allgemeingültige Regeln zur Umsetzung von Bewegungshypothesen in Robotersteuerungskommandos.

Die externe Sensorik ist noch nicht vollständig in den Ablauf integriert, so daß einige Sensorhypothesen über ein Skript in das System eingebracht werden. Dazu zählen die Objektposition sowie einige Informationen über Objektmerkmale, z.B. die Lage der Ausrichtkanten. Über das Skript können stellvertretend für den Benutzer neue Hypothesen in Form von Nachrichten an die einzelnen Wissenseinheiten geschickt werden.

Die Ausführung geschieht wie in Kapitel 5.3.4 angegeben. Da hier angenommen wird, daß weder ein interner noch ein externer Lernschritt durchgeführt wird, läuft das System in einer ständigen Schleife der Aktivierung einer Wissenseinheit und einer Produktion und der Ausführung dieser Produktion.

Zunächst wird vom Benutzer die Aufgabenstellung „Greife das Objekt" vorgegeben und auf die initiale Planwissenseinheit weitergeleitet. Zu diesem Zeitpunkt sind keine Objektwissenseinheit und damit keine Objektinformationen vorhanden. Der Planagent erzeugt für

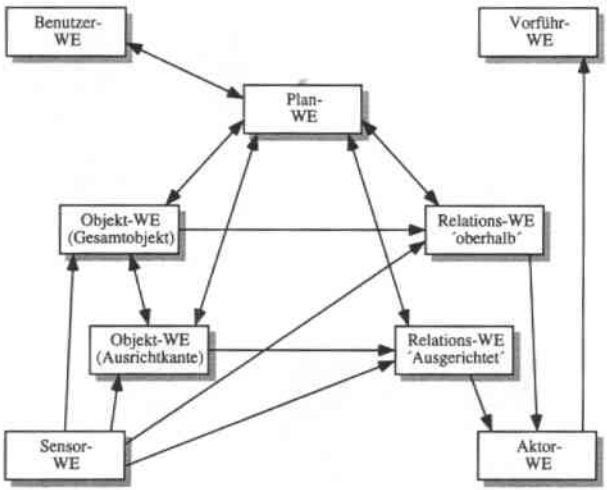

Abbildung 9.2: Struktur des Systemzustands während der Greifaufgabe

die Produktionen mit höchstem Aktivierungspotential, die teilweise erfüllte Bedingungsteile haben, entsprechende Anfragehypothesen. Auf die Sensorwissenseinheit gelangt die Anfangsstellung des Roboters in Form von Koordinaten des TCP, d.h. des Mittelpunkts des Greifers. Analog wird visuell ein Objekt erkannt und eine zugehörige Objektwissenseinheit erzeugt. Auf diese gelangen bereits die Anfragen des Planagenten. Auf der Basis von Objektinformationen, die an den Planagenten weitergeleitet werden, kann dort eine Produktion angewendet werden, die der Objektwissenseinheit die Rolle „Greifobjekt" zuweist. Damit sind alle Vorbedingungen für die Produktion bekannt, die als erstes Teilziel die Relation „oberhalb" mit Bezug zum Greifobjekt herleitet. Die Aktion dieser Produktion erzeugt eine neue Relationswissenseinheit, auf die die Informationen über TCP-Position und bereits bekannte Objektinformationen über die entsprechenden Kanäle gelangen. Damit können Produktionen angewendet werden, die Gültigkeitsbereiche für die einzelnen Koordinaten ableiten und zur Erzeugung einer Bewegungshypothese für die einzelnen Koordinaten führen. Der Aktoragent wählt die stärkste Hypothese aus und setzt ein entsprechendes Kommando an die Robotersteuerung ab. Die neuen Werte über die TCP-Position werden über die Sensor-WE an die Relations-WE geliefert. Sobald alle Koordinaten in ihrem Gültigkeitsbereich liegen, wird eine Produktion anwendbar, die eine Hypothese erzeugt, daß die Relation „oberhalb" zum Objekt mit der Rolle Greifobjekt erfolgreich eingenommen wurde.

Auf der Basis weiterer Objektinformationen wird die Hypothese abgeleitet, daß der beste Griff für das vorliegende Objekt am Gesamtobjekt stattfinden sollte. Mit dieser Information wird eine Produktion anwendbar, die als Teilobjekt des Objektes eine Ausrichtkante, d.h. eine zu der Kante gehörige Objekt-WE erzeugt. Diese erhält wieder eine Rolle zugewiesen, eine Relations-WE kann erzeugt werden zur Ausrichtung des Greifers parallel zu

diesem Teilobjekt, die notwendigen Aktionen werden abgeleitet, usw. Nach der Einnahme dieser Relation ergibt sich der in Abbildung 9.2 gezeigte Systemzustand. Nach insgesamt 276 Zyklen des Systems, d.h. Entscheidungen für die Aktivierung einer Wissenseinheit, Auswahl einer möglichen Produktion und Anwendung dieser Produktion ist die Aufgabe, bestehend aus allen oben angegebenen Teilrelationen, erfolgreich gelöst.

In weiteren Experimenten wurde die Abhängigkeit des Systemablaufs von der Reihenfolge der eintreffenden Hypothesen untersucht. Dazu werden durch direkte Benutzerinteraktion die notwendigen Hypothesen über Aufgabenziele und externe Informationen zu unterschiedlichen Zeitpunkten in das System eingebracht. Aufgrund der Ereignisorientiertheit der Aktivierung einzelner Wissenseinheiten und Produktionen und der On-line-Entscheidung für die geeignete Weiterverarbeitung kann adäquat auf das unterschiedliche Eintreffen von Informationen reagiert werden. Sobald in einem Zustand wichtige Informationen fehlen, werden Anfragehypothesen erzeugt, die z.B. bei einer Information über die Umwelt auch aktiv zu einer gezielten Sensormessung führen könnten - dieser Teilaspekt wurde bisher allerdings nicht vertieft.

Die Reaktivität des Systems bei der Vorgabe neuer Informationen und dadurch eventuell neuer Aufgaben wurde auf zwei Ebenen getestet. Zunächst wurde während der Einnahme einer Relation durch Angabe neuer Sensorinformationen für das betrachtete Objekt simuliert, daß sich das Objekt zwischenzeitlich, z.B. durch einen externen Einfluß, bewegt hat. Daraufhin gelangen die neuen Objektinformationen über die Kanäle an die vorliegenden Relationswissenseinheiten, die Produktionen zur Bestimmung der Gültigkeitsbereiche können aufgrund der Aktualität der neuen Hypothesen wieder angewendet werden, und es werden entsprechend geänderte Zielpunkte berechnet und Bewegungshypothesen für die Aktorwissenseinheit erzeugt. Das gilt auch für Koordinaten, die bereits betrachtet wurden.

Als zweite Form der Reaktivität wurde untersucht, daß eine Sensormessung, eine daraus abgeleitete Information oder eventuell sogar eine neue Aufgabenstellung zu neuen Teilzielen führen kann, die zunächst oder alternativ abgearbeitet werden müssen. Im konkreten Experiment wurde eine Produktion zur Reaktion auf eine Hypothese „Not-Aus" eingebaut, die zu einem neuen Teilziel führen sollte, bei dem der Manipulator eine bestimmte Position anfährt. Falls die Stärke der auslösenden Hypothesen groß genug ist, kann sie zu einem entsprechend hohen Aktivierungspotential der zugehörigen Produktion führen, die Produktion wird angewendet, die gewünschte Manipulatoraktion abgeleitet, und anschließend werden die alten Pläne wieder aktuell. Für diese wird eine entsprechende Fortsetzung durchgeführt. Dazu gehört auch, daß Teilziele, die bereits eingenommen waren und für die weitere Lösung notwendig sind, erneut betrachtet werden müssen. In diesem Zusammenhang muß eine erweiterte Betrachtung des Aktivierungspotentials einer Wissenseinheit durchgeführt werden, wie sie in [Tu94] untersucht wird, um sowohl Aktualität, Aktivität und die Strategie für eine Wissenseinheit betrachten zu können.

Die hierarchische Zerlegung eines Planes in Teilpläne wurde ebenfalls untersucht. Dabei müssen die in Kapitel 5.4 aufgeführten Fälle zur zeitlichen Verkettung der Teilpläne unterschieden werden. Der Fall der zeitlichen Aufeinanderfolge von Teilplänen wird analog zu dem geschilderten Vorgehen bei der Aufeinanderfolge von Teilzielen, d.h. Relationen, durch Hypothesenverknüpfungen in den Vorbedingungen erfolgreich gelöst. Die beiden Fälle der Und- und Oder-Verknüpfung entsprechen den beiden bei der Reaktivität betrachteten Möglichkeiten, daß ein Teilplan den anderen zunächst dominiert, dieser Teil-

plan abgearbeitet wird und dadurch entweder der Plan an sich bereits durchgeführt wurde (Oder-Verknüpfung) oder, daß anschließend die übrigen Teilpläne erneut um die Aktivierung konkurrieren.

9.2 Lernen von Regelwissen

Für das Lernen von Regelwissen wurden mehrere Tests durchgeführt, um die unterschiedlichen Lernalternativen analysieren zu können (s. auch [Bre93]). Als einzelne Untersuchungspunkte wurden dabei die folgenden Fälle berücksichtigt:

1. Lernen einer Folge von unbekannten Teilzielen
2. Lernen von Aufgaben, die in bestimmten Teilzielen überlappen
3. Lernen einer Aufgabe für Objekte gleichen oder unterschiedlichen Typs mit gleichen oder unterschiedlichen Merkmalen
4. Lernen einer Aufgabe mit Varianten in Abhängigkeit von bestimmten Merkmalen
5. Lernen der Relationsbeschreibung selbst
6. Auswirkung von fehlerhaften Vorführungen auf das Systemwissen

Dabei ist bisher kein Verfahren bekannt, Kenngrößen wie z.B. die Klassifikationsgenauigkeit beim Lernen eines Klassifikators für das aufgebaute Wissen zu bestimmen. Aufgrund der vielfältigen möglichen Aufgabenstellungen, für die es keine „typischen" Repräsentanten gibt, ist es auch nicht sinnvoll, zu bestimmen, welche Aufgabentypen bereits erfolgreich gelöst werden können.

Die Ergebnisse der Lernschritte sind bewußt nach wenigen vorgegebenen Beispielen aufgeführt. Damit soll gezeigt werden, daß durch die definierte Vorgehensweise bereits nach einigen Beispielen ein sinnvolles Verhalten des System erreicht werden kann und der Benutzer nicht zunächst Dutzende von Beispielen erzeugen muß, bevor eine Aufgabe vom System selbständig gelöst werden kann.

Ausgangspunkt für das Produktionswissen der einzelnen Agenten sind die aufgabenunabhängigen Produktionen wie z.B. zur Erzeugung von Anfragen, zur Verarbeitung von Nachrichten, zur Einnahme von Teilzielen und zur Ableitung von Aktorkommandos.

Im ersten Typ Lernaufgabe soll eine Folge von unbekannten Teilzielen gelernt werden. Für ein Objekt vom Typ Box mit den Merkmalen Min-x=60, Max-x=110, Min-y=80, Max-y=140, Höhe=40 wird zunächst die neue Relation „oberhalb" vorgeführt – durch Angabe des Zielpunktes (85, 100, 80, 0, 0, 0, 0). Das Objekt erhält eine neue Rolle und damit wird eine neue Rollenproduktion erzeugt. Zusätzlich wird eine Produktion zur Einnahme des Teilziels „oberhalb" für die vorliegende Situation generiert:

```
WENN
  ((Plan-WE (Var:5198)) (Rolle5182 'beliebig'))
  ∧ ((Plan-WE (Var:5198)) (Plan-Name Oberhalb-und-Rechts))
  ∧ ((Objekt-WE (Var:5197)) (Objekt-Typ Box))
  ∧ ((Objekt-WE (Var:5197)) (Max-x 110))
  ∧ ((Objekt-WE (Var:5197)) (Min-x 60))
  ∧ ((Objekt-WE (Var:5197)) (Max-y 140))
  ∧ ((Objekt-WE (Var:5197)) (Min-y 80))
  ∧ ((Objekt-WE (Var:5197)) (Objekt-Höhe 40))
DANN
  Neue Relation Oberhalb mit Bezug zu (Rolle5182)
```

Für den Relationsagenten wird für jede Koordinate eine Produktion zur Bestimmung des Gültigkeitsbereichs erzeugt. Da diese Vorführung das erste Beispiel für diese Relation ist, ergibt die funktionale Induktion zwar den gewünschten Wert – die tatsächliche Funktion ist aber noch relativ willkürlich:

```
WENN
  ((Relations-WE 'beliebig') (Relations-Name Oberhalb))
  ∧ ((Objekt-WE (Var:5199)) (Objekt-Typ Box))
  ∧ ((Objekt-WE (Var:5199)) (Objekt-Höhe 'beliebig'))
  ∧ ((Objekt-WE (Var:5199)) (Min-y 'beliebig'))
  ∧ ((Objekt-WE (Var:5199)) (Max-x 'beliebig'))
DANN
  Berechne Gültigkeitsbereich
  X muß im Intervall von (-0,5 · Max-x + 2 · Min-y - 0,5 · Objekt-Höhe)
  bis (-0,5 · Max-x + 2 · Min-y - 0,5 · Objekt-Höhe) liegen
```

Die Vorbedingung der neuen Produktion setzt sich zusammen aus dem Relationsnamen, den beteiligten Objekttypen und den Hypothesen, die in die Berechnung des Gültigkeitsintervalls eingehen. Eine genauere Untersuchung des Lernens von Relationsbeschreibungen folgt am Ende dieses Abschnitts.

Im nächsten Schritt soll das Teilziel „rechts-oberhalb" eingenommen werden. Der Benutzer gibt im Dialog den gewünschten Zielpunkt (120, 105, 75, 0, 0, 0, 0), den Relationsnamen, das Bezugsobjekt und die im ersten Schritt erzeugte Rolle an. Damit wird eine neue Produktion des Planagenten generiert, die zusätzlich zu der oben angegebenen Produktion noch folgende Vorbedingung und die angegebene geänderte Aktion besitzt:

```
WENN ...
  ∧ ((Relations-WE (Var:5212))
        (Aktuelle-Relation (Oberhalb (Rolle5182))))
DANN
  Neue Relation Rechts-Oberhalb mit Bezug zu (Rolle5182)
```

Aufbauend auf das gelernte Wissen wird eine zweite Aufgabe eingelernt, die zunächst das Teilziel „oberhalb" und dann das Teilziel „links-oberhalb" einnehmen muß. Der Benutzer

zeigt als erstes ein neues einzunehmendes Teilziel bei (80, 90, 60, 0, 0, 0, 0), das in keiner der beiden vorliegenden Gültigkeitsbereiche für Relationen liegt. Der Benutzer wählt den Relationsnamen „oberhalb" und die oben erzeugte Rolle aus. Die Rollenproduktion kann erfolgreich generalisiert werden, indem die Bedingung über den Plannamen verallgemeinert wird.

```
WENN ...
  ∧ ((Plan-WE (Var:5196))
        (Plan-Name {Oberhalb-und-Rechts, Oberhalb-und-Links}))
...
```

Ebenso kann die Produktion für die Erzeugung des Teilziels „oberhalb" generalisiert werden. Für den Relationsagenten muß jeweils eine Grenze der Koordinaten x, y und z neu bestimmt werden. Als zweites Teilziel soll die Relation „links-oberhalb" eingenommen werden. Analog zu Beispiel 1 werden Regeln zur Teilzielerzeugung und zur Gültigkeitsberechnung generiert.

Als nächster Fall wird das Lernen von gleichen Aufgaben für unterschiedliche Objekte untersucht. Zunächst werden Objekte gleichen Typs mit unterschiedlichen Merkmalsausprägungen betrachtet. Gegegeben seien eine Box mit Min-x = 100, Max-x = 130, Min-y = 50, Max-y = 80, Höhe = 300 und eine Box mit Min-x = 100, Max-x = 250, Min-y = 50, Max-y = 420 und Höhe = 40. Ziel der Aufgabe „Erreiche-Greifpunkt" ist das Erreichen der Teilziele „oberhalb" und „direkt-oberhalb". Analog zum ersten Beispiel wird ein Beispiel für die Relation „oberhalb" durch Angabe des Zielpunktes (115, 65, 400, 0, 0, 0, 0) für das erste Objekt eingelernt. Anschließend wird als Beispiel für „direkt-oberhalb" der Punkt (115, 65, 320, 0, 0, 0, 0) angefahren. Für das zweite Objekt sind die erzeugten Produktionen zunächst nicht anwendbar, da sie die spezielle Situation bei Vorliegen des ersten Objektes repräsentieren. An dieser Stelle kann entweder eine interne Generalisierung untersucht werden oder, der Benutzer gibt, wie in diesem Beispiel einen Zielpunkt (175, 235, 140, 0, 0, 0, 0) an und wählt die bekannte Relation „oberhalb" als derzeit gültig aus. Die Produktionen zur Einnahme des Teilziels und zur Erzeugung der Rolle werden entsprechend generalisiert:

```
WENN
  ((Plan-WE (Var:6781)) (Rolle6777 'beliebig'))
  ∧ ((Plan-WE (Var:6781)) (Plan-Name Erreiche-Greifpunkt))
  ∧ ((Objekt-WE (Var:6780)) (Objekt-Typ Box))
  ∧ ((Objekt-WE (Var:6780)) (Max-x [130 250]))
  ∧ ((Objekt-WE (Var:6780)) (Min-x 100))
  ∧ ((Objekt-WE (Var:6780)) (Max-y [80 420]))
  ∧ ((Objekt-WE (Var:6780)) (Min-y 50))
  ∧ ((Objekt-WE (Var:6780)) (Objekt-Höhe [40 300]))
DANN
  Neue Relation Oberhalb mit Bezug zu (Rolle6777)
```

Für die Relationsbeschreibung wird eine neue funktionale Induktion durchgeführt. Die Vorführung und Verarbeitung der Relation „direkt-oberhalb" erfolgt analog.

In ähnlicher Form werden Vorführungen für Objekte unterschiedlichen Typs, die gemeinsame Merkmale besitzen, integriert. Dabei kann auch die Produktion zur Erzeugung des Gültigkeitsbereichs durch Generalisierung der Vorbedingung auf den neuen Objekttyp übertragen werden. Im Anschluß an diese Lernvorgänge kann das gesamte Spektrum der Aufgabenausprägungen, die durch die generalisierten Beschreibungen abgedeckt werden, direkt durchgeführt werden. Aufgaben, die mit dem bisher aufgebauten Wissen falsch gelöst werden, führen zu einer Spezialisierung des Wissens.

In einem weiteren Beispiel soll wie oben normalerweise die Folge „oberhalb" und „direkt-oberhalb" eingenommen werden, für bestimmte Objekte aber „oberhalb" und „rechts-oberhalb". Nach Einnahme der Relation „oberhalb" gibt der Benutzer die Relation „rechts-oberhalb" vor. Das System würde als Alternative "direkt-oberhalb" erzeugen. Als bestes Unterscheidungskriterium wird die Höhe des Objektes bestimmt und damit eine Spezialisierung des Wertebereichs dieses Merkmals bei der Produktion zur Erzeugung der Relation „direkt-oberhalb" durchgeführt.

Im folgenden wird die Induktion von Gültigkeitsbereichen genauer untersucht. Dazu werden für die Relation „oberhalb" und „rechts-von" eine größere Zahl von Lernvorgängen der oben geschilderten Art für verschiedene Objekte und Objektpositionen durchgeführt.

Als Objekte werden betrachtet ein Quader (20 × 30 × 35), ein Stab (10 × 15 × 70), eine Platte (50 × 65 × 20) und ein Balken (60 × 8 × 40). Als Positionen werden für die vordere, linke, untere Ecke der Objekte die Punkte (10, 20, 0) und (200, 80, 0) betrachtet. Die Einheit ist jeweils mm. Die Beispiele sind in Tabelle 9.1 aufgeführt.

Nr.	Objekt	Min-x	Max-x	Min-y	Max-y	Höhe	Z_x	Z_y	Z_z
1.	Quader_{Pos1}	10	30	20	50	35	20	35	55
2.	Quader_{Pos2}	200	220	80	110	35	210	95	55
3.	Stab_{Pos1}	10	20	20	35	70	15	28	90
4.	Stab_{Pos2}	200	210	80	95	70	205	88	90
5.	Platte_{Pos1}	10	60	20	85	20	35	53	40
6.	Platte_{Pos2}	200	250	80	145	20	225	113	40
7.	Quader_{Pos1}	10	30	20	50	35	25	43	58
8.	Quader_{Pos2}	200	220	80	110	35	205	87	52
9.	Stab_{Pos1}	10	20	20	50	70	18	24	95
10.	Stab_{Pos2}	200	210	80	95	70	202	92	85
11.	Platte_{Pos1}	10	60	20	85	20	15	78	46
12.	Platte_{Pos2}	200	250	80	145	20	245	88	34

Tabelle 9.1: Beispiele für Relationsvorführungen

In den ersten sechs Beispielen wird zunächst der Mittelpunkt oberhalb des Objektes als Zielpunkt (Z_x, Z_y, Z_z) angefahren. Daraus ergeben sich als Bereichsbeschreibungen für die Koordinaten x, y und z die folgenden Intervalle. Für die rotatorischen Koordinaten und die Greiferstellung werden gemäß der Beispiele ebenfalls Funktionen abgeleitet.

```
x = [ 0,5 · Min-x + 0,5 · Max-x ; 0,5 · Max-y + 2 · Min-y ]
```

```
y = [ 0,5 · Min-y + 0,5 · Max-y ; Min-y + 0,5 · Max-y ]
z = [ Höhe ; Max-y + 2 · Min-y - Höhe ]
```

Für x und y werden also für einen Grenzwert bereits die exakten Mittelpunktsfunktionen abgeleitet. In den nachfolgenden Beispielen werden Beispiele angegeben, die verschiedene Punkte der Relation oberhalb des Objektes abdecken. Nach allen 12 Beispielen ist dann die Relation „oberhalb" folgendermaßen beschrieben:

```
x = [ Min-x ; Max-x ]
y = [ Min-y ; Max-y ]
z = [ Höhe ; Höhe + 2,0 · Min-y - 0,5 · Min-x ]
```

In ähnlicher Weise wurden Beispiele für die Relation „rechts-neben" generiert und entsprechende Funktionen erzeugt. Das Ergebnis waren folgende Beschreibungsintervalle:

```
x = [ Max-x ; Max-x + Höhe ]
y = [ Min-y ; Max-y ]
z = [ 0,5 · Höhe - 0,5 · Min-y ; Höhe ]
```

Die induzierten Funktionen beschreiben sehr gut die Gültigkeitsbereiche der gewünschten Relationen. In manchen Teilfunktionen werden Merkmale hinzugenommen, die im wesentlichen als Konstante dienen. Um die Möglichkeiten bei Verwendung von Konstanten zu testen, wurden zwei weitere „Objektmerkmale" zu den Beschreibungen hinzugenommen, die immer konstant den Wert 2 bzw. 10 hatten. Auf diese Weise konnten Konstanten in der Funktion verarbeitet werden, obwohl das Verfahren an sich dies noch nicht explizit unterstützt. Als Relation wurde eine eingeschränkte Relation „oberhalb" betrachtet, die nicht bis an den Rand des Objektes hin gültig war. Als Beschreibungsintervalle wurden dann nach zehn Beispielen die folgenden abgeleitet:

```
x = [ Min-x + Konst1 ; Max-x - Konst1 ]
y = [ Max-y - 0,5 · Min-y - 0,5 · Konst2 ; Max-y - Konst1 ]
z = [ Höhe + 0,5 · Konst1 + 0,5 · Konst2 ; 2,0 · Höhe - Konst1 - 0,5 · Konst2 ]
```

Im folgenden soll nun das Verhalten des System bei Fehleingaben des Benutzers analysiert werden. Bei der Vorführung eines fehlerhaften Zielpunktes sind die beiden Fälle zu unterscheiden, daß der Benutzer entweder den Namen der eigentlich für die Aufgabe richtigen Relation angibt, oder daß er die Aufgabe an sich falsch löst, d.h. eine fehlerhafte Relation mit zugehörigem Zielpunkt angibt.

Der erste Fall wirkt sich nur auf die Induktion der Relationsbeschreibung aus. Wird für eine Relation ein neuer Zielpunkt vorgeführt, für den die Relation explizit als gültig angegeben wird, so führt das dazu, daß die Relationsbeschreibung erweitert wird, falls die Bewertungsfunktion eine vorgegebene Grenze nicht überschreitet. Ist die Bewertung zu schlecht, wird für das neue fehlerhafte Beispiel eine eigene Produktion zur Gültigkeitsbereichsbeschreibung erzeugt mit initial niedrigerer Stärke. Wie in Kapitel 6.3.2 gezeigt,

werden zur Zeit keine expliziten Negativbeispiele berücksichtigt. Daher können bereits bestehende Beschreibungen nicht durch Angabe eines Zielpunktes, der nicht enthalten sein sollte, wieder eingeschränkt werden. Eine Lösung wäre wie bereits erwähnt eine andere Einbeziehung von „Ausreißern", indem die Wahrscheinlichkeitsverteilung über der Menge der vorliegenden Beispiele analysiert wird.

Im zweiten Fall führt der Benutzer für eine Aufgabe eine fehlerhafte Aktion vor. Anhand der obigen eingelernten Produktionen zur Lösung der Aufgaben „Oberhalb-und-Rechts" und „Oberhalb-und-Links" soll das Systemverhalten analysiert werden. Im ersten Test führt der Benutzer bei sonst gleicher vorliegender Situation für die Aufgabe „Oberhalb-und-Links" als zweites zu erreichendes Teilziel die Relation „rechts-oberhalb" vor, indem er als Zielpunkt (120, 105, 75, 0, 0, 0, 0) angibt. Das System bestimmt die Relation, die es selbst auf der Basis des vorhandenen Wissens eingenommen hätte, fragt den Benutzer, ob es sich bei dem angegebenen Zielpunkt um diese Relation handelt, und würde dann die Relationsbeschreibung entsprechend aktualisieren. In diesem Fall wählt der Benutzer aber aus der Liste der gültigen Relationen die Relation „rechts-oberhalb" aus. Die Gültigkeitsbedingungen sind erfüllt, aber das Wissen des Planagenten muß entsprechend geändert werden.

Eine Spezialisierung der Produktion zur Erzeugung der Relation „rechts-oberhalb" ist nicht möglich, da die gleichen Bedingungen vorliegen wie in den vorher eingelernten Beispielen. Aus diesem Grund wird die entsprechende Produktion gelöscht. Die Produktion, die zur Erzeugung der Relation „rechts-oberhalb" im Anschluß an die Einnahme der Relation „oberhalb" führt, wird entsprechend generalisiert. Die Aufgabe wird nachfolgend entsprechend der „fehlerhaften" Vorgabe des Benutzers durchgeführt. Gibt der Benutzer in einer weiteren Vorführung als nächste Relation wieder das korrekte Teilziel „links-oberhalb" vor (45, 125, 60, 0, 0, 0, 0) wird eine Spezialisierung durchgeführt, indem die falsche neu erzeugte Produktion wieder gelöscht wird und eine neue Produktion erzeugt wird, die wieder zur Einnahme des Teilziels „links-oberhalb" führt. Dabei wird exakt die Produktion neu erzeugt, die oben gelöscht wurde.

In einem weiteren Test wird ein ähnlicher Fehler des Benutzers angenommen, allerdings wurde die vorliegende Beispielsituation geändert. In diesem Fall ist die Objekthöhe mit 40 mm statt mit 50 mm angegeben, so daß es sein könnte, daß der Benutzer aus diesem Grund tatsächlich eine andere Aktionsfolge vorführen möchte. Zunächst werden die bestehenden Produktionen zur Rollenzuweisung und zur Erzeugung der Relation „oberhalb" intern generalisiert, um die aktuelle Situation ebenfalls abzudecken. Bei der Vorführung des Teilziels „rechts-oberhalb" muß die vorhandene Produktion nicht spezialisiert werden, da sie den neuen Fall nicht abdeckt. Daher wird eine neue Produktion zur Wissensbasis des Planagenten hinzugenommen, die eine Verallgemeinerung der Produktion zur Ableitung des Teilziels „rechts-oberhalb" darstellt. Die Relationsbeschreibungen werden wie im vorigen Teil beschrieben aktualisiert. Wird die ursprüngliche Aufgabe (Objekthöhe = 40 mm) erneut an das System gestellt, so hat die alte Produktion eine höhere Stärke als die neu hinzugenommene generalisierte, so daß die Aufgabe korrekt abgearbeitet wird. Die Aufgabe mit der Objekthöhe von 50 mm wird, wie vom Benutzer vorgeführt, gelöst. Gibt der Benutzer für diese Aufgabe bei einer weiteren Vorführung wieder das ursprüngliche Teilziel „links-oberhalb" an, so wird die ursprünglich generalisierte Produktion wieder spezialisiert auf den Bereich `Objekt-Höhe [40,40]`, da 40 und 50 die einzigen Ausprägungen für die Objekthöhe im Wissen des Planagenten sind. Die Produktionen zur Erzeugung der Rela-

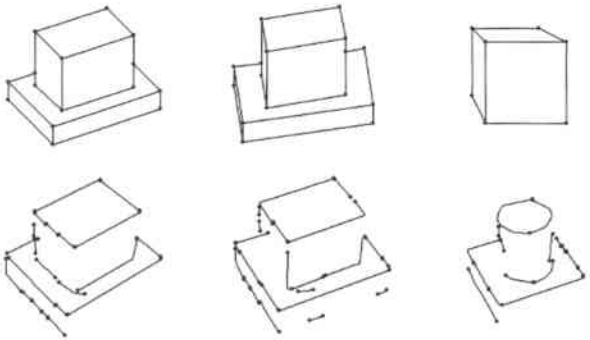

Abbildung 9.3: Verschiedene ideale und reale Objekt-2D-Ansichten

tion „links-oberhalb" werden generalisiert, um in der vorliegenden Situation angewendet werden zu können.

9.3 Lernen von generischen Objektbeschreibungen

Anhand mehrerer unterschiedlicher Tests wurde die Leistungsfähigkeit des Objektagenten, d.h. des Lernens von generischen Objektbeschreibungen, experimentell überprüft (s. auch [Sto93]). Für diese Tests wurden sowohl synthetisch erzeugte als auch reale Objektansichten verwendet. Abbildung 9.3 gibt eine Übersicht über einige der verwendeten Objektansichten.

Die realen Objekte wurden mit Hilfe einer Handkamera aufgenommen und in einem Bildverarbeitungssystem weiterverarbeitet. Als Ergebnis lieferte dieses Bildverarbeitungssystem eine Liste von Polygonzügen, die die im Originalbild gefundenen Konturen repräsentieren.

Die Schwierigkeit, eine automatische Bewertung der Leistungsfähigkeit des Systems anhand von einfachen Kenngrößen durchzuführen, liegt darin, daß das Lernverfahren an sich unüberwacht ist, d.h. es liegt keine Information vor, zu welcher Klasse ein Objekt zugeordnet werden soll. Damit ist aber auch nicht einfach eine Klassifikationsgenauigkeit bestimmbar.

Aus diesem Grund wurden zwei Arten von Experimenten durchgeführt. Der erste Typ von Experimenten diente der Untersuchung des Einflußes verschiedener Systemparameter auf die Struktur der entstehenden Modellhierarchie. Im zweiten Teil der Experimente wurde die Generalisierungsfähigkeit der Methode untersucht, indem die Modellhierarchie auf einer Trainingsmenge eingelernt wurde und dann das Verhalten auf einer unabhängigen Testmenge bestimmt wurde. Zu diesem Zweck wurden den einzelnen Ansichten Namen zugeordnet, die jedoch nicht zum Lernen oder zur Klassifikation verwendet wurden. Nach

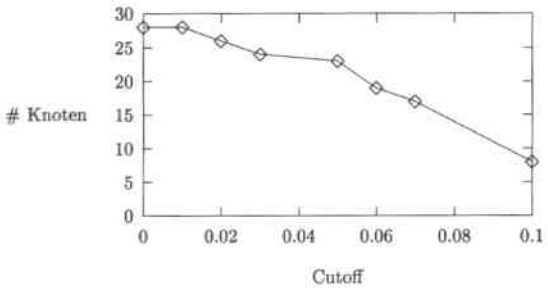

Abbildung 9.4: Test 1: Hierarchiegröße über den Cutoff

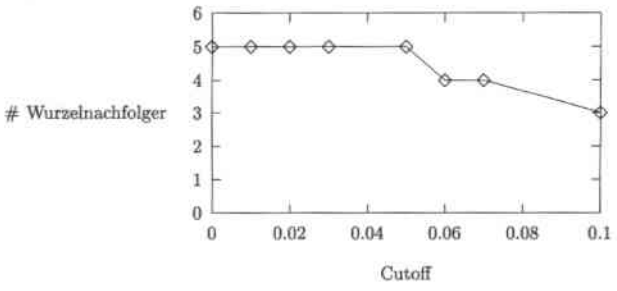

Abbildung 9.5: Test 1: Kategorien über den Cutoff

erfolgter Klassifikation einer neuen Objektansicht der Testmenge entschied der Benutzer, ob sie korrekt oder falsch klassifiziert wurde. Diese Entscheidung konnte er auf der Basis der gespeicherten Namen treffen. Auf diese Weise konnte für das unüberwachte Lernverfahren eine Klassifikationsgenauigkeit für neue Objektansichten, die nicht zum Lernen verwendet wurden, berechnet werden.

Im ersten Test wurden 20 aus idealen Kantenlisten erzeugte Objektansichten für acht verschiedene Werte des Cutoff eingelernt. Der Cutoff wirkt sich direkt auf die Tiefe der Hierarchie und damit auf die Hierarchiegröße aus (s. Abbildung 9.4). Je größer der Cutoff gewählt wird, um so kleiner wird die entstehende Hierarchie, so daß in der Modellhierarchie eine Übergeneralisierung in den einzelnen Modellen bewirkt wird. Andererseits wird die Hierarchie unnötig groß, je kleiner der Cutoff gewählt wird. Es entsteht der Effekt einer unnötigen Spezialisierung. Im Extremfall des Wertes 0.0 für den Cutoff wird jeder einzelnen Objektansicht ein eigenes Blatt zugeordnet.

Der Cutoff wirkt sich außerdem auf die Anzahl der Teilmodelle (Kategorien) auf oberster

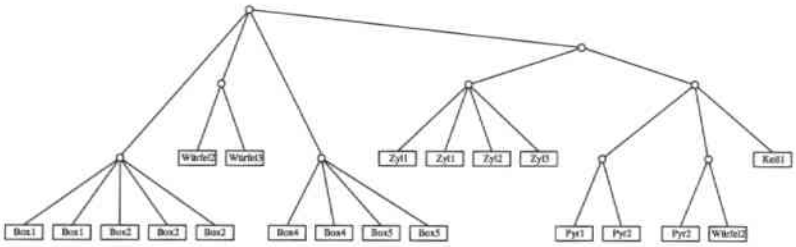

Abbildung 9.6: Beispiel einer Modellhierarchie

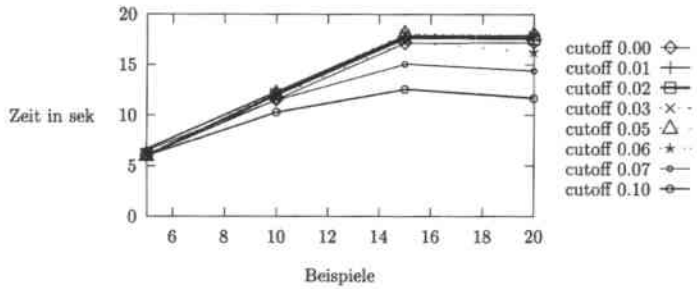

Abbildung 9.7: Test 1: Lernzeit über die Anzahl eingelernter Instanzen

Ebene aus (s. Abbildung 9.5). Bei den eingelernten Ansichten handelt es sich um Ansichten von Würfeln, Zylindern, Pyramiden, Keilen und perspektivisch verzerrten Würfeln. Als Mensch würde man die Ansichten in vier oder fünf Klassen einteilen. Der beste Cutoff-Wert, der Hierarchietiefe und Kategorienanzahl gleichzeitig möglichst optimiert, liegt bei 0.05. Die gleiche Analyse bei der Verwendung von realen Ansichten in der Testgruppe 2 liefert einen besten Wert des Cutoffs zwischen 0.05 und 0.06.

Abbildung 9.6 zeigt die Modellhierarchie, die bei Verwendung des Cutoff 0.05 entsteht. Während die ersten vier Klassen eine exakte Ballung enthalten, befinden sich in der Ballung, die Pyramiden repräsentieren sollte, auch eine Würfelansicht. Der Grund liegt darin, daß die quadratische Pyramidengrundseite auch einer Würfelseite zugeordnet werden kann, und sich dadurch eine genügend hohe Ähnlichkeit ergibt.

In den Abbildungen 9.7 und 9.8 sind schließlich noch die Lern- und Klassifikationszeit je nach Anzahl der bereits gelernten Instanzen eingetragen. Es zeigt sich, daß sich durch die hierarchische Anordnung der gelernten Modelle ein Anstieg der Laufzeit ergibt, der weniger als linear ist. Ein weiterer Grund für die Reduktion des Zeitbedarfs pro Einlernen einer Objektansicht liegt in der Lernfähigkeit des Zuordnungsalgorithmus, so daß bei weiteren Klassifikations- und Lernvorgängen immer schneller eine bessere Zuordnung gefunden werden kann (s. Test 3).

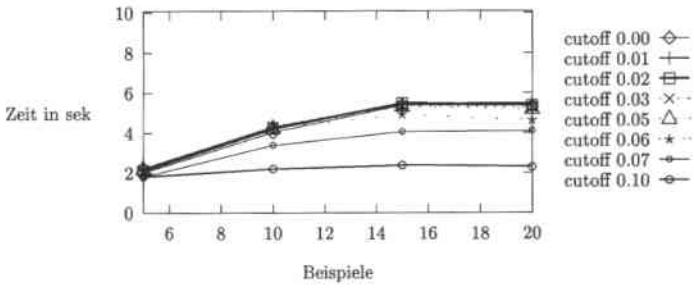

Abbildung 9.8: Test 1: Klassifikationszeit über die Anzahl eingelernter Instanzen

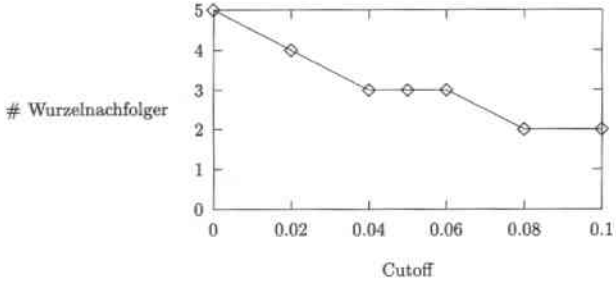

Abbildung 9.9: Test 2: Kategorien über den Cutoff

In einem zweiten Test wurden 20 reale Objektansichten gelernt, die aus Kameraaufnahmen von drei verschiedenen Objekten entstanden sind. Die Anzahl der entstehenden Kategorien in Abhängigkeit vom Cutoff sind in Abbildung 9.9 aufgetragen. Die Fehler, die sich bei einer genauen Analyse der Zuordnung von realen Objektansichten zu Objektmodellen ergeben, sind wesentlich auf die schlechte Qualität des in Hardware realisierten Kantendetektors zurückzuführen. Eine Rekombination kleiner Kantenstücke oder die Aufsplittung von Polygonzügen mit übermäßig spitzen Ecken würde zu besseren Ergebnissen führen. Eine bessere Vorverarbeitung der Bildinformationen wurde im Rahmen dieser Arbeit allerdings nicht untersucht.

Um die Lernfähigkeit des Zuordnungsalgorithmus zu untersuchen, wurde in einem dritten Test eine Hierarchie aus 20 idealen Objektansichten aufgebaut und anschließend wurden diese 20 Ansichten zweimal darin klassifiziert, eine Veränderung der Modellhierarchie wurde dabei nicht vorgenommen. Es ergibt sich eine im Mittel niedrigere Klassifikationszeit im zweiten Durchlauf im Vergleich zum ersten (s. Abbildung 9.10).

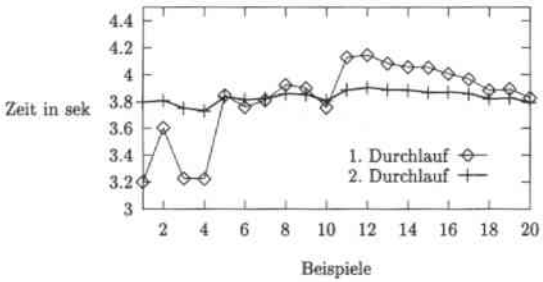

Abbildung 9.10: Test 3: Klassifikationszeit

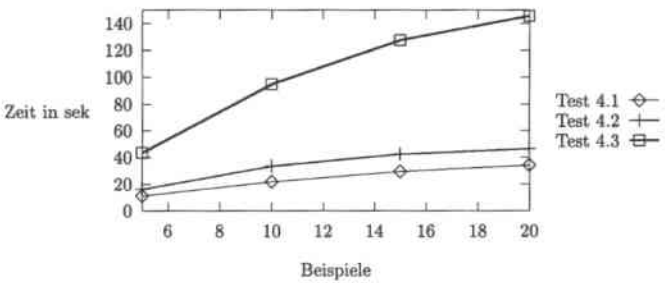

Abbildung 9.11: Test 4: Lernzeit über die Anzahl eingelernter Instanzen

In einer weiteren Testserie wurde der Einfluß verschiedener Größen des Zuordnungsalgorithmus untersucht, wie die maximale Anzahl von Objekt-n-Tupeln, die für die Zuordnung zu einem Modell-n-Tupel bewertet werden und die maximale Rekursionstiefe des Zuordnungsschrittes (Test 4.1: Breite = 2, Tiefe = 2, Test 4.2: Breite = 3, Tiefe = 2, Test 4.3: Breite = 3, Tiefe = 3). Als Testobjekte wurden wieder reale Objekte verwendet und zwar neun Würfelansichten, vier Zylinderansichten und sieben Ansichten eines Klebebandabrollers. Die Auswirkung dieser Parameter auf den Zeitbedarf der Lernphase ist vergleichend in Abbildung 9.11 aufgetragen.

In einer zweiten Reihe von Experimenten wurde die Generalisierungsfähigkeit des Verfahrens getestet. Dazu wurde die Fehlerrate bei der Klassifikation neuer Objektansichten aus einer Testmenge, die nicht zum Lernen verwendet wurde, mit einer bestehenden, gelernten Modellhierarchie bestimmt. Aufgrund des unüberwachten Charakters des Lernverfahrens konnten diese Experimente, wie bereits in der Einleitung dieses Abschnitts erwähnt, nur mit Eingriff eines Benutzers durchgeführt werden. Nach der Klassifikation einer Ansicht durch das System wurde durch einen Benutzer bewertet, ob die Klassifikation *korrekt*

Rotation	Translation	Streckung	Anzahl korrekt	Prozentual korrekt
nein	nein	nein	50	100.0
ja	nein	nein	35	70.0
nein	ja	nein	50	100.0
nein	nein	ja	50	100.0
ja	ja	ja	37	74.0

Tabelle 9.2: Test 5: Klassifikation von Ansichten von gedrehten, verschobenen und gestreckten Objekten auf der Basis einer Modellhierarchie, die nur mit den Ansichten der Originalobjekte unüberwacht eingelernt wurde

erfolgte oder nicht. Zur Unterstützung wurden allen eingelernten Ansichten Namen zugeordnet, so daß eine genaue Analyse des Teilbaumes möglich war, in den einklassifiziert wurde.

In einer ersten Testserie wurde die in Test 1 gelernte Modellhierarchie (s. Abbildung 9.6) als Grundlage für die Klassifikation verwendet. Diese Modellhierarchie wurde aus zwanzig Ansichten von Objektmodellen erzeugt, bei denen das Modell in der Ursprungslage, d.h. weder rotiert noch verschoben, vorlag. Diese Ansichten bildeten die Trainingsmenge. Der Cutoff wurde auf 0.05 gesetzt. Die jeweils fünfzig neuen Objektansichten der Testmengen wurden durch zufällige Rotationen von 0 bis 2π, Translationen von 0.0 bis 50.0 mm in der Ebene senkrecht zur optischen Achse und Streckungen um einen Faktor zwischen 0.0 und 10.0 der Objektmodelle erzeugt. Die Modellhierarchie selbst wurde während des Klassifikationsvorgangs auf der Testmenge nicht geändert. Das bedeutet auch, daß die zugrundeliegende Modellhierarchie für alle fünf Experimente identisch war. Die Ergebnisse sind in Tabelle 9.2 zusammengefaßt.

Es zeigt sich, daß das Verfahren mit translatorisch verschobenen oder skalierten Objekten sehr gut funktioniert. Die Klassifikationsgenauigkeit bei rotierten Objekten liegt bei 70%. Dieser Verlust an Genauigkeit ist vor allem im Zuordnungsverfahren begründet. Da beim Einlernen bestimmte Attribute und Relationen nur in einer einzigen Ausprägung vorkamen und die Modellhierarchie entsprechend aufgebaut wurde, sind die Bewertungen für bestimmte Zuordnungen entsprechend schlechter. Aufgrund des Greedy-Algorithmus, der bei der Klassifikation eingesetzt wird, werden daher bei rotierten Objekten nicht die „richtigen" Zuordnungen gefunden und es findet entweder eine enstprechende Fehlentscheidung für einen der Nachfolger statt oder die Gesamtbewertung liegt unterhalb des Cutoff-Wertes, so daß keine tiefere Einklassifizierung vorgenommen wird.

Um den Einfluß der nicht repräsentativen Lernbeispiele auf das Klassifikationsergebnis zu messen, wurden in einer zweiten Testserie auch zum Einlernen Ansichten von rotierten, verschobenen und skalierten Objekten verwendet. Zu diesem Zweck wurden wie oben beschrieben zwanzig Objektansichten für zufällig rotierte, verschobene und skalierte Objekte bestimmt. Diese wurden als Trainingsmenge für den Aufbau der Modellhierarchie verwendet. Damit ergab sich eine bessere Verteilung der Lerninstanzen im Raum aller

Rotation	Translation	Streckung	Anzahl korrekt	Prozentual korrekt
ja	nein	nein	43	86.0
nein	ja	nein	44	88.0
nein	nein	ja	44	88.0
ja	ja	ja	48	96.0

Tabelle 9.3: Test 6: Klassifikation von Ansichten von gedrehten, verschobenen und gestreckten Objekten auf der Basis einer Modellhierarchie, die auch mit Ansichten von entsprechend veränderten Objekten unüberwacht eingelernt wurde

möglichen Instanzen. Weiterhin wurden für jedes Experiment zufällig 50 Ansichten von je nach Experiment rotierten, verschobenen und/oder gestreckten Objekten erzeugt. Diese Objektansichten dienten jeweils als Testmenge und wurden nicht für den Lernvorgang eingesetzt. Auch hier wurde die Modellhierarchie während der Klassifikation nicht verändert und war für alle Tests identisch.

Die Klassifikationsgenauigkeit der gelernten Modellhierarchie auf den Testmengen ist in Tabelle 9.3 angegeben. Die Ergebnisse waren im Durchschnitt besser als im vorhergehenden Test, auch rotierte Objekte wurden mit 86% Genauigkeit erkannt. Allerdings wurden die scharfen Verteilungen, die sich oben positiv bei Translation und Streckungen auswirkten, entsprechend unschärfer. Eine weitere Verbesserung der Klassifikationsgenauigkeit ist durch die Verwendung topologischer Relationen bei der Objektbeschreibung zu erwarten. Bisher werden als Relationen Beziehungen wie Parallelität, Linearität und ähnliches eingesetzt, eine Angabe beispielsweise über die Nachbarschaftsbeziehungen der Kanten an einem Eckpunkt liegt nicht vor.

In einem weiteren Test wurden die Auswirkungen von fehlenden Kanten in einer Objektansicht untersucht. Dazu wurden 80 Objektansichten erzeugt, die auf zufällig rotierten, verschobenen und gestreckten Objekten basierten. Zusätzlich konnten jeweils zufällig Kanten fehlen. 30 von diesen Objektansichten bildeten die Trainingsmenge, für die eine entsprechende Modellhierarchie erzeugt wurde. Die 50 restlichen Ansichten dienten als Testmenge. Auf dieser wurde eine Klassifikationsgenauigkeit von 84% erzielt.

Insgesamt haben die Experimente mit dem Objektagenten gezeigt, daß das Verfahren tatsächlich in der Lage ist, selbständig Ballungen in den vorgelegten Objektansichten zu bestimmen und diese hierarchisch anzuordnen. Eine wesentliche Verbesserung der Qualität bei realen Objektansichten ist zu erwarten, wenn die Vorverarbeitungsverfahren zur Extraktion von Kanten verbessert werden und damit weitere strukturelle Merkmale abgeleitet werden können, die entscheidend für die Qualität der Zuordnung sind.

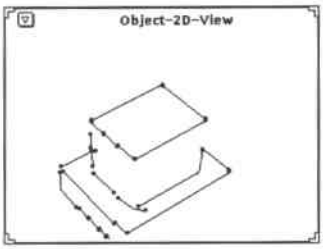

Abbildung 9.12: Beispiel einer Objekt-2D-Ansicht

9.4 Lernen von Objekteigenschaften

Der Experimentieragent wurde anhand verschiedener Objekt-2D-Ansichten des Objektagenten getestet. Eine vollautomatische Durchführung der Experimentierzyklen inklusive Auswertung war noch nicht möglich, so daß der Benutzer eingekoppelt wurde, um die konkrete Aktion am System durchzuführen bzw. die Reaktionsklasse bei einem Aktionsexperiment zu bestimmen.

Die Explorationskomponente wurde anhand von einzelnen Experimenten getestet, die an verschiedenen Objekten durchgegeführt wurden. Für das in Abbildung 9.12 gezeigte Objekt sollte die Flächennormale der Deckfläche und anschließend die Objekthöhe, d.h. die Höhe dieser Fläche bestimmt werden. Für die Fläche wurde in der Planung des Explorationsexperiments ein Punkt bestimmt, der angetastet werden soll, auf diesen hin wurde der Manipulator bewegt, bis von der Kraftmeßdose die Überschreitung einer maximalen Kraft festgestellt wurde. Daraufhin wurde die Bewegung des Manipulators unterbrochen (was aufgrund der Hardwarestruktur frühestens nach 32 ms möglich ist). Die Position des Manipulators zu diesem Zeitpunkt kann aus den Gelenkwinkelstellungen durch die Lösung des direkten kinematischen Problems bestimmt werden.

Auf der Basis des Explorationsexperiments zur Bestimmung einer Flächennormalen wurden vier Punkte auf der Fläche angetastet. Die 3D-Koordinaten der Flächenpunkte wurden wie folgt gemessen (in mm zum Referenzkoordinatensystem) :

$$P_1 = \begin{pmatrix} 120,16 \\ 160,17 \\ 421,45 \end{pmatrix}, P_2 = \begin{pmatrix} 119,91 \\ 159,12 \\ 421,46 \end{pmatrix}, P_3 = \begin{pmatrix} 121,54 \\ 160,43 \\ 421,42 \end{pmatrix}, P_4 = \begin{pmatrix} 119,69 \\ 160,76 \\ 421,45 \end{pmatrix}$$

Der daraus berechnete Normalenvektor zeigt, daß die Fläche parallel zur xy-Ebene liegt: $\vec{n} = (0,0173 \quad 0,0051 \quad 0,9871)$. Als Höhe des Objektes wurde experimentell 45,3 mm bestimmt, die tatsächliche Höhe betrug 49,0 mm.

Auf einem Objekt mit einer Dachkante wurde eine Antastsequenz zur Bestimmung eines Kantenpunktes durchgeführt. Der erste Punkt der bestimmten Sequenz wurde angefahren und anschließend lokal eine Abrück- und erneute Anrückbewegung durchgeführt, d.h. die weiteren Antastpunkte werden nicht aus der Initialstellung des Roboters heraus angefahren. Die erhaltene Sequenz von Punkten lautet in Weltkoordinaten:

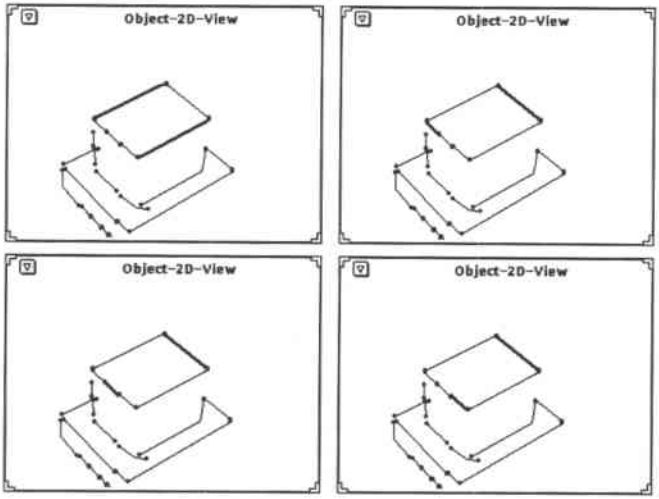

Abbildung 9.13: Erfolgreiche Kantenpaare für einen Griff

$$P_1 = \begin{pmatrix} 120,16 \\ 163,95 \\ 109,99 \end{pmatrix}, P_2 = \begin{pmatrix} 120,22 \\ 168,46 \\ 114,54 \end{pmatrix}, P_3 = \begin{pmatrix} 120,30 \\ 172,71 \\ 118,53 \end{pmatrix}, P_4 = \begin{pmatrix} 120,18 \\ 179,84 \\ 115,93 \end{pmatrix}, P_5 = \begin{pmatrix} 120,18 \\ 185,86 \\ 109,01 \end{pmatrix}$$

Als Kantenpunkt wurde der Punkt mit den Koordinaten $P_K = (120,26 \quad 175,38 \quad 121,05)$ bestimmt. Tatsächlich lag der Kantenpunkt bei $P = (120,20 \quad 174,10 \quad 120,00)$.

Die Aktionsexperimente wurden am Beispiel der Aktion Greifen untersucht. Grundlage für die Bestimmung des Parametervektors innerhalb der Aktionsplanung war eine Objektbeschreibung, wie sie vom Objektagenten erzeugt wurde. Dabei wurden keine Flächeninformationen verwendet. In den Experimenten wurde untersucht, wie die einzelnen Parameter bestimmt und aus erfolgreichen Experimenten Erfahrungswissen aufgebaut wird. Als Beispiel wird zunächst das in Abbildung 9.12 dargestellte Objekt betrachtet. Durch die Hardware werden 21 Kanten bzw. Konturstücke identifiziert, so daß theoretisch 210 Kantenpaare für einen Griff in Frage kommen. Durch einen Filter, der lineare Kantenpaare herausfiltert, können 38 Paare entfernt werden. Unter der Annahme, daß das Greifexperiment nur in den in Abbildung 9.13 gezeigten vier Fällen erfolgreich wäre, würde bei einer reinen Zufallsauswahl im Mittel nur einer von 43 Versuchen zum Erfolg führen. Wird eine wissensbasierte Auswahl des Parameters 'Kontaktkanten' eingesetzt, so wird sehr viel schneller ein erfolgreiches Experiment geplant.

In zwei Tests wurde die Menge der durch Explorationsexperimente vorhandenen 3D-Informationen variiert. Im ersten Versuch liegen keinerlei 3D-Informationen über das Objekt vor, beim zweiten Versuch wird angenommen, daß die Ergebnisse aller gewünschten

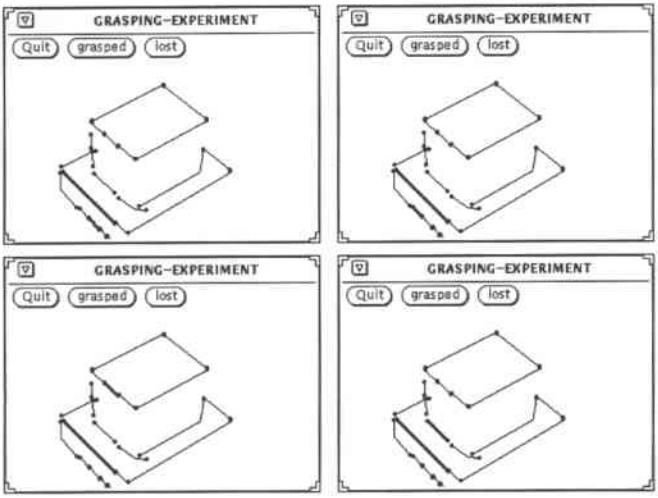

Abbildung 9.14: Kantenpaare mit höchster Bewertungspunktzahl

Explorationsexperimente vorliegen. In beiden Versuchen wurden dieselben Bewertungsregeln und Filter eingesetzt. Der Filter für die Betrachtung erfolgloser Experimente wurde so definiert, daß ein Parameterwert ausgefiltert wird, wenn mit ihm vier Aktionen geplant wurden, die alle erfolglos waren.

Die in Abbildung 9.14 markierten vier Kantenpaare erhalten alle die höchste Bewertungspunktzahl. Daher werden zunächst Experimente mit einem dieser vier Kantenpaare durchgeführt. Durch jedes erfolglose Experiment erhöht sich die Wahrscheinlichkeit, daß das Kantenpaar herausgefiltert wird. Nach maximal 16 Versuchen wird das Kantenpaar mit der nächstbesten Bewertung eingesetzt, das allerdings auch nicht erfolgreich ist. In der Menge der beiden nächstbesten Bewertungen ist ein erfolgreiches Kantenpaar enthalten – und zwar das in Abbildung 9.13 rechts unten gezeigte, so daß insgesamt nach maximal 24 erfolglosen Experimenten ein erfolgreiches Kantenpaar gefunden wird . Tatsächlich werden in den Versuchen allerdings sehr viel weniger erfolglose Experimente durchgeführt, da die Wahrscheinlichkeit für die Ausfilterung eines solchen Kantenpaares kontinuierlich steigt. Ebenso könnte die prinzipielle Ausfilterung auch bereits nach weniger als vier Fehlversuchen geschehen. Minimal wäre bereits das sechste Experiment erfolgreich. Aus mehreren erfolgreichen Experimenten für dasselbe Kantenpaar wird dann Erfahrungswissen in Form des verwendeten Parametervektors aufgebaut.

Im zweiten Versuch wurde angenommen, daß bereits Explorationsexperimente durchgeführt wurden, und damit z.B. bekannt ist, ob eine Kante direkten Kontakt mit der Tischoberfläche hat oder nicht. Dadurch kann die Zahl möglicher Kantenpaare für die wissensbasierte Auswahl durch die Filterung auf 96 reduziert werden. Die beste Bewertung erhalten die beiden unteren in Abbildung 9.14 gezeigten Kantenpaare. Die nächstbeste Bewertung erhalten zwei Kantenpaare, von denen eines erfolgreich ist. Somit wird durch

Ausführung von einigen Explorationsexperimenten bereits nach maximal 12 Fehlversuchen ein erfolgreiches Experiment durchgeführt.

An Aufnahmen von vier Objekten, einem Vierkant, einem Seitenschneider, einem Schraubenzieher und einer Abisolierzange wurden weitere Aktionsexperimente durchgeführt. Ein Kantenpaar wurde dabei herausgefiltert, wenn zwei Aktionen durchgeführt wurden, die beide erfolglos waren. Für ein Kantenpaar wurde Erfahrungswissen aufgebaut, d.h. es wurde gelernt, sobald zwei erfolgreiche Aktionen mit ihm durchgeführt wurden. In der nachfolgenden Tabelle ist angegeben, wieviele Kanten aus dem Grauwertbild extrahiert wurden, wieviele potentielle Kantenpaare es damit gab, wieviele Kantenpaare nach der Anwendung des Filters übrig blieben, wieviele Versuche durchgeführt wurden, bis das erste Mal ein erfolgreiches Paar geplant wurde und wieviele Versuche insgesamt benötigt wurden, bis ein erfolgreiches Kantenpaar gelernt wurde. In keinem der Versuche wurden Explorationsexperimente durchgeführt, um weitere Objekteigenschaften zu bestimmen.

	Vierkant	Seitenschn.	Schraubenz.	Abisolierz.
# Kanten	15	8	17	31
# Kantenpaare	105	28	136	465
# Kantenpaare nach Filter	88	26	98	446
# Fehlversuche	2	10	37	21
# Versuche bis gelernt	3	14	50	28

Da die Abisolierzange aus sehr vielen kurzen Kanten bestand, die zueinander parallel waren, die aber als nicht erfolgreich bewertet wurden, wurde die Wichtigkeit der Parallelität zweier Kanten in diesem Fall herabgesetzt.

Insgesamt haben die Versuche mit dem Experimentieragenten gezeigt, daß die automatische Planung und Durchführung von Explorations- und Aktionsexperimenten erfolgreich möglich ist. Je mehr Informationen über ein Objekt vorhanden sind, um so schneller kann ein erfolgreiches Aktionsexperiment durchgeführt werden. Daneben ist die richtige Auswahl der Bewertungen für bestimmte Eigenschaften von Objektmerkmalen zur Bestimmung des Parametervektors wichtig. Eine automatische Adaption dieser Bewertungen ist eine sinnvolle Erweiterung dieses Ansatzes, wurde aber bisher nicht durchgeführt.

9.5 Zusammenfassung

Die experimentellen Untersuchungen der Systemarchitektur und ihrer Teilkomponenten haben gezeigt, daß mit den entwickelten Methoden die angestrebten Ziele erreicht wurden. Die Verfahren ermöglichen für die Lösung von Manipulationsaufgaben neuartige Vorgehensweisen für die „Programmierung", die Ausführung und die sukzessive Verbesserung des zugrundeliegenden Wissens durch Lernvorgänge. Die vollständige Integration aller Teilkomponenten des Systems ist in dieser Arbeit noch nicht durchgeführt worden. Aufbauend darauf könnte dann das langfristig angestrebte Ziel des Programmierens durch Vorführen erreicht und eine entsprechende Untersuchung des Einsatzes dieses Systems auch für Laien durchgeführt werden.

Kapitel 10

Zusammenfassung und Ausblick

Die Fähigkeit, automatisch oder benutzergestützt Wissen initial zu erwerben und sukzessive zu korrigieren, ist eine wesentliche Voraussetzung für die Erschließung neuer Anwendungsbereiche für Manipulatoren und mobile Systeme. Die vorliegende Arbeit befaßt sich mit der Anwendung und Integration von Lernverfahren zur Realisierung dieser Fähigkeiten in fortgeschrittenen, autonomen Robotersystemen.

Für die Forschung auf diesem Gebiet liefert die vorliegende Arbeit fünf wichtige Beiträge:

1. Die erstmalige detaillierte Analyse der Anwendbarkeit und der Anwendungen von Techniken des Maschinellen Lernens in der Robotik.
2. Die erste Systemarchitektur für ein autonomes Manipulationssystem, die eine Integration verschiedener Lerntechniken unterstützt und es erlaubt, flexibel auf neue Informationen zu reagieren und gleichzeitig strategisch ein Ziel zu verfolgen.
3. Eine Methode zur automatischen und benutzergestützten Generalisierung und Spezialisierung von Regelwissen, mit der auch zum ersten Mal der Aufbau von Systemwissen eines autonomen Robotersystems durch die Methodik des Programmierens durch Vorführen möglich ist.
4. Eine Methode zum unüberwachten und gegebenenfalls überwachten Lernen von generischen Objektbeschreibungen, die den Aufbau einer Modellhierarchie aus einer Menge von Objektansichten ermöglicht.
5. Ein Konzept zur Durchführung von taktilen Explorationsexperimenten und von Aktionsexperimenten zur Bestimmung von Objekteigenschaften.

Die prinzipielle Analyse der Anwendbarkeit von Lernverfahren in der Robotik hat die großen Differenzen aufgezeigt, die zwischen Robotikproblemen und typischen Anwendungen von ML-Techniken insbesondere bezüglich der Wissensrepräsentation, den Anforderungen an Daten oder Beispiele und der Betrachtung von Planung und Ausführung bestehen. Darin liegt mit Sicherheit ein Hauptgrund für die geringe Zahl der veröffentlichten Arbeiten zu lernenden Robotersystemen. Auf der Basis der herausgearbeiteten Lernaufgaben bzw. der Lernziele in der Robotik wurden die existierenden Arbeiten erstmals klassifiziert und zusammenfassend beschrieben. Dabei zeigte sich, daß in den meisten Arbeiten

versucht wurde, eine sehr spezielle Teilaufgabe zu lösen. Eine Betrachtung der Einbindung in ein Gesamtsystem oder der möglichen Erweiterbarkeit wurde nur in ICARUS, SOAR, THEO und WISMO zum Teil durchgeführt.

Aus diesem Grund wurde in dieser Arbeit ein Top-Down-Ansatz verfolgt: Aufbauend auf einer Anforderungsanalyse an Komponenten für lernende, autonome Robotersysteme wurde zunächst eine Architektur entwickelt. In diesem Kontext wurden dann einige der wichtigsten Lernaufgaben gelöst.

Die Architektur zeichnet sich durch eine homogene Struktur der einzelnen Komponenten (Agenten und Wissenseinheiten) aus, die gemeinsam die gestellten Aufgaben bearbeiten und gleichzeitig auf neue Informationen über die Umwelt reagieren können. Die gesamten zugrundeliegenden Wissensrepräsentationen wurden dahingehend entwickelt, daß sie den Einsatz von Lernverfahren ermöglichen und unterstützen. Für die Anwendung des Wissens während des Systemablaufs wurde eine Aktivierungsfunktion definiert, die zur Auswahl geeigneter Wissenseinheiten und Produktionen führt. Durch eine Regelanwendung entstehen entweder neue Wissenseinheiten oder neue Informationen – in Form von Hypothesen –, die über die einzelnen Kommunikationskanäle an interessierte Einheiten weitergeleitet werden. Durch diese neue Definition des dynamischen Ablaufs innerhalb des Systems ist eine integrierte Betrachtung von reaktivem und strategischem Verhalten möglich geworden.

Für die allgemein eingesetzte Repräsentationsform der Produktionsregeln wurden Generalisierungs- und Spezialisierungsoperatoren definiert. Durch diese ist es möglich, automatisch eine Verallgemeinerung oder, benutzergestützt, eine Verallgemeinerung oder Spezialisierung des Systemwissens durchzuführen. Für diesen Vorgang wurde ein Hypothesenvorzugskriterium definiert, das auf der Signifikanz einer einzelnen Elementarbedingung in der Menge der bestehenden Produktionen basiert. Die Integration von Benutzervorführungen wurde dadurch ermöglicht, daß die Ebene der Teilziele in Form von Relationen explizit gemacht wurde. Beim Programmieren durch Vorführen kann der Benutzer ein nächstes Teilziel vorführen oder das zuletzt eingenommene korrigieren. In beiden Fällen wird – wenn nötig – entweder das bestehende Wissen des Systems modifiziert oder neues Wissen aufgebaut. Dazu gehört neben dem Planungswissen insbesondere auch das Wissen zur Beschreibung von Teilzielen. Dafür wurde ein spezielles Lernverfahren zur funktionalen Induktion von Intervallbeschreibungen entwickelt. Sämtliche entwickelte Lernmethoden sind prinzipiell unabhängig vom konkreten Inhalt der manipulierten Repräsentationen, so daß die Erkenntnisse auch auf andere Anwendungsbereiche übertragen werden können.

Für den Aufbau generischer Objektbeschreibungen wurde die Kernidee des CLASSIT-Algorithmus verwendet. Während dieses Verfahren auf einer numerischen Attribut-Wert-Liste arbeitet, kann die erweiterte Methode erstmalig nominale, numerische und vor allem strukturelle (relationale) Attribute gemeinsam betrachten. Außerdem wurden weitere Strukturierungen in der Repräsentation vorgenommen, um den Anforderungen der Objekterkennung gerecht zu werden. Für die entwickelte Repräsentation wurde ein Zuordnungsverfahren entwickelt, da die Korrespondenzen zwischen Elementen der Objektansichten nicht – wie allgemein bei ML-Ansätzen zur Klassifikation üblich – durch gleiche Attributnamen bestimmt werden können. Für die entwickelte Modellrepräsentation wurde die Bewertungsfunktion der Ballungsnützlichkeit entsprechend erweitert, die als Grundlage für das unüberwachte und das zur Unterstützung dienende überwachte Lernen der Mo-

dellhierarchie verwendet wird. Die einzelnen Lernoperatoren wurden auf die gleichzeitige Betrachtung von Modell- und Ansichtenhierachie erweitert. Während die Lernoperatoren und die oberen Ebenen der Beschreibung, bestehend aus Modell und 2D-Ansicht, spezifisch für die Aufgaben des Objektagenten sind, ist mit der Untersuchung struktureller Attribute, der Weiterentwicklung der Bewertungsfunktion und dem Zuordnungsverfahren ein Fortschritt erreicht worden, der unabhängig von der konkreten Anwendung ist.

Weitere Informationen über ein Objekt können durch die entwickelte Methode zum Lernen von Objekteigenschaften gewonnen werden. Zu diesen Eigenschaften zählen im wesentlichen statische Eigenschaften wie Geometrie, Gewicht etc. Daneben können Aktionsexperimente geplant werden, durch die das Verhalten des untersuchten Objektes bei bestimmten Aktionsfolgen getestet werden kann.

Die entwickelten Verfahren wurden in einer Reihe von Experimenten mit simulierten Daten und mit realer Hardware getestet. Diese Versuche haben die Leistungsfähigkeit der entwickelten Methoden gezeigt und den neuen Ansatz zur Akquisition und Modifikation von unterschiedlichem Wissen für autonome Robotersysteme validiert.

Aufgrund des großen Umfangs der Problematik konnten in dieser Arbeit nicht sämtliche identifizierte Teilprobleme bearbeitet werden. Die wichtigsten möglichen Weiterentwicklungen dieser Arbeit sind:

- Integration von mächtigeren Elementaroperationen zur Realisierung anderer Relationsklassen - z.B. Während-Relationen mit Kraftregelung. Diese Regler können z.B. durch Neuronale Netze oder Fuzzyregeln repräsentiert werden. Entsprechend sind dafür geeignete Lernverfahren zu entwickeln und zu untersuchen, ob auch dort die Analyse von Benutzervorführungen hilfreich ist.

- Integration weiterer Sensorik, durch die insbesondere dreidimensionale Informationen über die Objekte gewonnen werden können, ohne eine taktile Exploration durchzuführen. Entsprechend sind dann die Verfahren für den Objektagenten zu erweitern. Eine Möglichkeit der Realisierung ist die Verwendung der in Kapitel 7.1 angesprochenen dreidimensionalen Geons als Basiselemente und die Betrachtung von Relationen zwischen diesen, wie in der vorliegenden Arbeit geschehen.

- Fortführung der entwickelten Methoden zur Benutzerinteraktion, um langfristig das Programmieren durch Vorführen auch für Laien zu ermöglichen.

- Lösung weiterer Lernaufgaben im Rahmen der entwickelten Architektur.

- Einsatz der Architektur für Roboter zur Lösung komplexerer Aufgabenstellungen wie z.B. allgemeine Servicearbeiten, Demontage, Bearbeitung von Kleinserien.

Durch diese Erweiterungen kann diese Arbeit zu einem völlig neuen Umgang mit Robotersystemen führen und die Grundlage für den Einsatz von Robotern in neuen Anwendungsgebieten bilden. Die Untersuchung der Anforderungen von lernenden Robotersystemen haben zudem zu neuen Fragestellungen und Verfahren geführt, die auch für das Gebiet des Maschinellen Lernens allgemein von großem Interesse sind.

Anhang A

Realisierung

In diesem Kapitel soll das System demonstriert werden, das auf der Basis dieser Arbeit entwickelt wurde, und mit dem die gezeigten Beispiele erarbeitet wurden. Das gesamte System wurde in Common Lisp auf einer SUN Workstation programmiert. Als Programmierparadigma wurde die objektorientierte Programmierung in Form der Lisp-Erweiterung CLOS (Common Lisp Object System) verwendet. Die Kopplung zur Robotersteuerung wurde in C realisiert.

Der Kern des Systems besteht aus den gesamten Definitionen zur Repräsentation des agentenunabhängigen Wissens, dem Ablaufmechanismus, den Kommunikationsmöglichkeiten und der Verwaltung des Systemzustandes. Abbildung A.1 zeigt die zentrale Systemoberfläche. Das obere Fenster dient der Steuerung für alle Teilkomponenten. Die Graphendarstellung darunter zeigt den Systemzustand bei der Problemlösung, d.h. die einzelnen Wissenseinheiten und die zwischen ihnen definierten Kanäle. Über ein Menü der einzelnen sensitiven Knoten können die aktuellen Hypothesen einer Wissenseinheit oder die Produktionen des zugehörigen Agenten angezeigt werden. Für ausgewählte Wissenseinheiten kann in einem zugehörigen Fenster der Hypothesenfluß, die Anwendung von Produktionen und die Verarbeitung des Ergebnisses mitverfolgt werden. Im Beispiel ist ein Ausschnitt des Protokolls einer Relationswissenseinheit gezeigt.

Das zentrale Steuerungsfenster dient zum Start einer Problemlösung, zur Unterbrechung nach einem Zyklus, zur schrittweisen oder kontinuierlichen Abarbeitung oder zur Abarbeitung bis zu einem gegebenen Problemlösezyklus. Im Textfenster wird, wenn gewünscht, ein Protokoll der Abarbeitung eines Problems dargestellt. Über die Menüs der zentralen Steuerung kann das Gesamtsystem und jeder einzelne Agent konfiguriert werden. Die Module für die Agenten können je nach Bedarf hinzugeladen werden. Für jeden Agenten einzeln oder für alle gleichzeitig können die aktuellen Produktionen gespeichert oder ein Satz bestehender Produktionen geladen werden.

Im Menüpunkt zur Hardwaresteuerung können die einzelnen Kommunikationsprozesse zur Robotersteuerung und zur Bildverarbeitung gestartet bzw. beendet werden. Über die Menüs für Agenten und Wissenseinheiten kann ausgewählt werden, welche Teile wie visualisiert werden und wie die einzelnen Fenster auf dem Bildschirm plaziert werden. Über die Demo-Funktion kann eine Konfigurationsdatei eingeladen werden, die die Produktionsdateien der einzelnen Agenten und eventuell zusätzlich eine vorhandene Skriptdatei

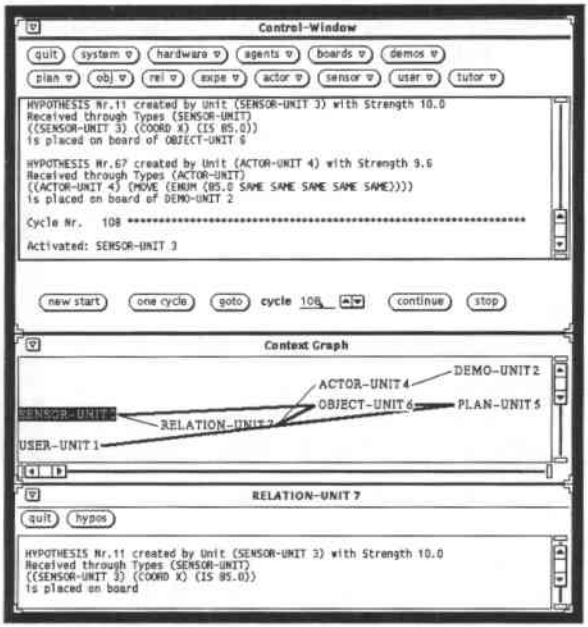

Abbildung A.1: Systemumgebung

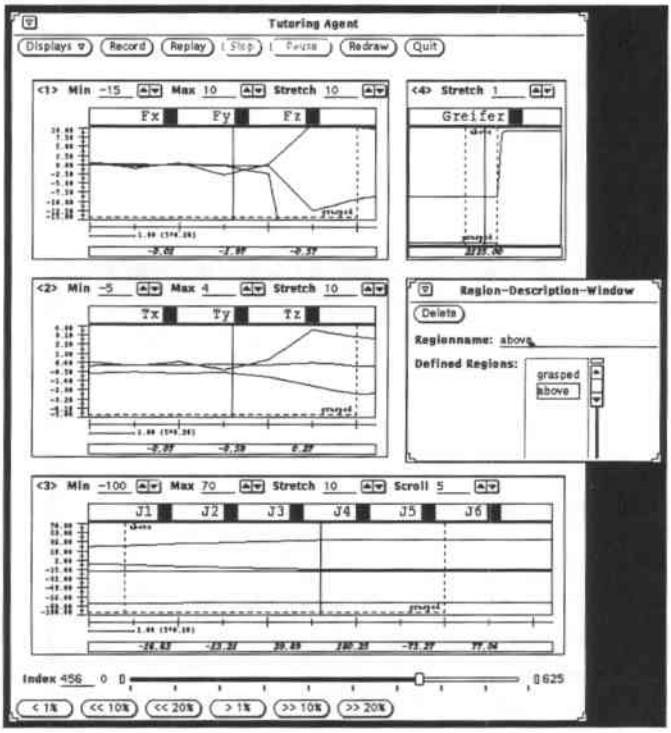

Abbildung A.2: Vorführagent

spezifiziert. In der unteren Zeile sind Menüs für die einzelnen Agenten eingerichtet. Über diese kann die Funktionalität jedes Agenten wie unten angegeben angesteuert werden.

Während des Systemablaufs kann der Benutzer einen externen Lernvorgang initiieren. Dazu gibt er an, ob er die letzte Relation korrigieren oder den nächsten Zielpunkt vorgeben will. In beiden Fällen gibt er die Koordinate des Zielpunktes an und trifft dann je nach vorliegender Situation eine Auswahl, welchen Relationstyp zu welchen Objektmerkmalen er dadurch vorgeführt hat bzw. welche Relationstypen oder Merkmale neu erzeugt werden sollen. Die einzelnen Lernschritte und Lernergebnisse können dann protokolliert werden.

Als zukünftige Eingabe für die externen Lernschritte wurde der Vorführagent entwickelt. Mit Hilfe einer 6D-Kugel kann der Benutzer die gewünschte Aufgabe am realen Roboter vorführen. Dabei werden interne und externe Sensorwerte aufgezeichnet und graphisch visualisiert. Dazu kann entweder eine punktuelle Darstellung als Balken- oder Zeigerdiagramm oder eine kontinuierliche Darstellung als Graph verwendet werden (s. Abbildung A.2).

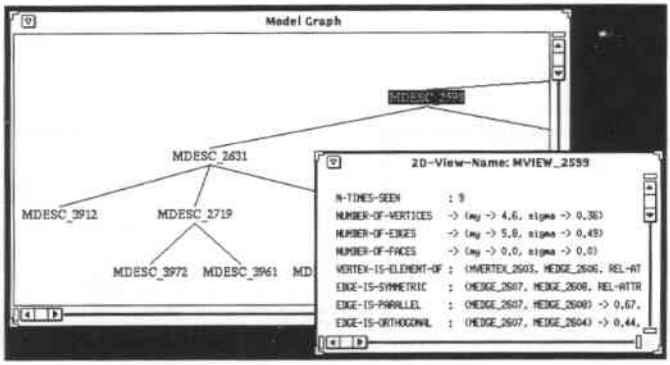

Abbildung A.3: Darstellung einer Modellhierarchie und eines Modellknotens

Die Ansteuerung eines simulierten Roboters ist durch die Ankopplung an das MIG-System [Sma91] ebenfalls möglich. In der gezeigten Abbildung A.2 sind von oben nach unten die von der Kraftmeßdose gemessenen Kräfte, die Momente und die sechs Gelenkwinkel dargestellt. Das rechte obere Display zeigt die Greiferöffnung. In diesen Darstellungen kann der Benutzer nachträglich navigieren und sich die einzelnen Teile seiner Vorführung anzeigen lassen. In einer Konfiguration für den Vorführagenten werden die gewünschten Displays, die Funktionen, die jeweils dargestellt werden sollen, die Wertebereiche, die Zeitskalen und die Farbzuordnungen spezifiziert. Das mittlere Fenster auf der rechten Seite dient der interaktiven Segmentierung der Vorführung. Der Benutzer kann Punkte oder Bereiche benennen und dann in einem der Displays kennzeichnen. Diese Kennzeichnungen dienen der Angabe von Relationen, die in einem bestimmten Punkt erreicht waren bzw. über eine Zeitspanne galten. Die einzelnen Regionen werden in den verschiedenen Displays markiert. In einer Weiterentwicklung des Systems sollen diese Regionen und Punkte als direkte Eingabe für den externen Lernschritt eingesetzt werden.

Der Objektagent besteht neben den in Kapitel 7 angegebenen Repräsentationsmethoden und Lernverfahren aus einer Reihe von Verarbeitungsroutinen zur Berechnung der Merkmale einer Objektansicht. Über das zentrale Menü kann die aktuelle Modellhierarchie und einzelne Modellbeschreibungen dargestellt werden (s. Abbildung A.3). Außerdem können Hierarchien und Objektansichten geladen und gespeichert, Objekte visualisiert, eingelernt und klassifiziert werden.

Die Planung und Bewertung von Experimenten zur Akquisition von Objekteigenschaften wird im Experimentieragenten durchgeführt. Für die Aktionsexperimente wird die ausgewählte Parameterbelegung textuell und graphisch dargestellt. Der Benutzer spezifiziert dann die Reaktionsklasse bei diesem Experiment (s. Abbildung A.4). Anschließend wird, wenn möglich, Erfahrungswissen für die Objektklasse aufgebaut und weitere Experimente geplant.

Die Ankopplung der Systemarchitektur an die reale Hardware geschieht über eine Menge von Elementaroperationen, die in Tabelle A.1 kurz beschrieben sind. Sensor- und Aktor-

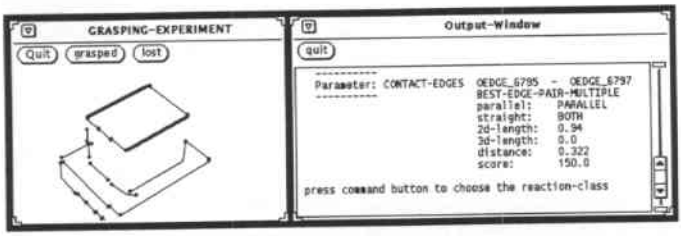

Abbildung A.4: Visualisierung eines Experiments zur Angabe der Reaktionsklasse

Elementaroperation	**Beschreibung**
movew-abs(x, y, z, ω, α, θ)	Absolute Bewegung in Weltkoordinaten
movew-rel(Δx, Δy, Δz, $\Delta\omega$, $\Delta\alpha$, $\Delta\theta$)	Relative Bewegung in Weltkoordinaten
movej-rel(Δj_1, Δj_2, Δj_3, Δj_4, Δj_5, Δj_6)	Relative Bewegung in Gelenkkoordinaten
movet-rel(Δx, Δy, Δz, $\Delta\omega$, $\Delta\alpha$, $\Delta\theta$)	Relative Bewegung in Toolkoordinaten
set-gripper(Pos, Kraft)	Setzen der Öffnungsweite der Greiferbacken
get-world-pos()	Auslesen der TCP-Position in Weltkoordinaten
get-joint-pos()	Auslesen der Gelenkwinkelstellungen
get-gripper-pos()	Auslesen der Greiferöffnung
get-force-torque()	Auslesen der Kräfte und Momente

Tabelle A.1: Elementaroperationen zur Kommunikation mit der Roboterhardware

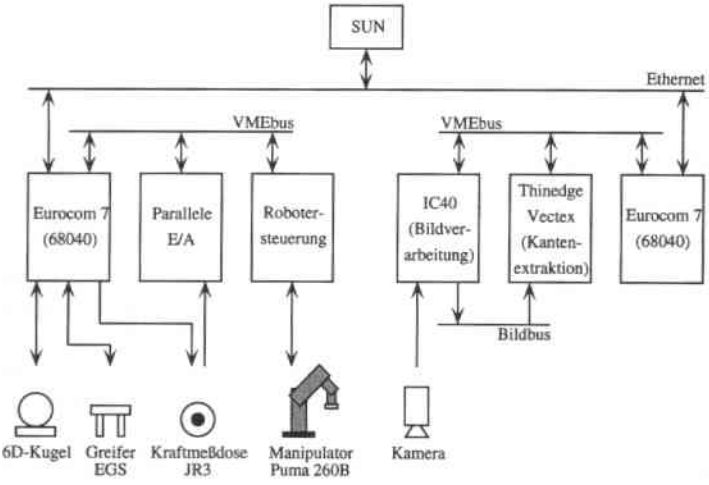

Abbildung A.5: Aufgebaute Hardware

agent führen von Lisp aus Aufrufe von elementaren Sensor- und Bewegungskommandos in C durch. Diese werden über eine Socketkommunikation an die Hauptrechner von Robotersteuerung und Bildverarbeitung gesandt. Dort werden sie ausgewertet und führen zu entsprechenden Verarbeitungsschritten.

Die Struktur des Hardwareaufbaus ist in Abbildung A.5 gezeigt. Das gesamte System wird auf einer SUN-Workstation ausgeführt und kommuniziert über Ethernet mit den beiden 68040-Karten der Robotersteuerung und der Bildverarbeitung. Die Bildverarbeitungshardware besteht aus einer Miniaturkamera, einem Framegrabber und zwei Karten zur Echtzeit-Kantenextraktion der Firma Eltec. Die Robotersteuerung basiert auf einer UNIVAL-Steuerung der Firma Stäubli-Unimation, die über ein Dual-Ported Ram mit der eigenen CPU-Karte über einen VMEbus kommuniziert. An diese CPU-Karte sind über eine Parallel-I/O-Karte die Kraftmeßdose sowie über V24-Schnittstellen die Greifersteuerung und die 6D-Kugel angekoppelt.

Literaturverzeichnis

[AAMR87] C. G. Atkeson, E. W. Aboaf, J. McIntyre und D. J. Reinkensmeyer. Model-Based Robot Learning. In *Proceedings of Robotics Research - The Fourth International Symposium*, 1987.

[Abr91] B. Abramson. An Analysis of Error Recovery and Sensory Integration for Dynamic Planners. In *Proceedings of the 9th National Conference on Artificial Intelligence, Anaheim*, Seiten 744–749, 1991.

[AC87] P. E. Agre und D. Chapman. Pengi: An Implementation of a Theory of Activity. In *Proceedings of the 6th National Conference on Artificial Intelligence, Seattle*, Seiten 268–272, 1987.

[AG90] Z. Ahmad und A. Guez. On the Solution to the Inverse Kinematic Problem. In *Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati*, Seiten 1692–1697, 1990.

[AI89] H. Asada und H. Izumi. Automatic Program Generation from Teaching Data for the Hybrid Control of Robots. *IEEE Transactions on Robotics and Automation*, 5(2):166–173, April 1989.

[AK89] D. W. Aha und D. Kibler. Noise-Tolerant Instance-Based Learning Algorithms. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, Detroit*, Seiten 794–799, 1989.

[AKA91] D. W. Aha, D. Kibler und M. K. Albert. Instance-Based Learning Algorithms. *Machine Learning*, 6(1):37–66, Januar 1991.

[Alb84] J. S. Albus. Robotics. In M. Brady und al., Hrsg., *Robotics and Artificial Intelligence*, Seiten 65–93. Springer-Verlag, 1984.

[All87] P. Allen. *Robotic Object Recognition Using Vision and Touch*. Kluwer Academic Publishers, 1987.

[And85] P. M. Andreae. Justified Generalization: Acquiring Procedures from Examples. Technical Report AI-TR-834, Artificial Intelligence Laboratory, MIT, 1985.

[And89] J. R. Anderson. A Theory of the Origins of Human Knowledge. *Artificial Intelligence*, 40:315–351, 1989.

[AR89] C. G. Atkeson und D. J. Reinkensmeyer. Using Associative Content-Addressable Memories to Control Robots. In *Proceedings of the IEEE International Conference on Robotics and Automation, Scottsdale*, Seiten 1859–1864, 1989.

[AR90] C. G. Atkeson und D. J. Reinkensmeyer. Using Associative Content-Addressable Memories to Control Robots. In P. H. Winston und S. A. Shellard, Hrsg., *Artificial Intelligence at MIT, Volume 2*, Kapitel 28, Seiten 102–127. The MIT Press, 1990.

[Asa90] H. Asada. Teaching and Learning of Compliance Using Neural Nets: Representation and Generation of Nonlinear Compliance. In *Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati*, Seiten 1237–1244, 1990.

[AT91] J. A. Allen und K. Thompson. Probabilistic Concept Formation in Relational Domains. In *Proceedings of the 8th International Workshop on Machine Learning, Evanston*, Seiten 375–379, 1991.

[AY89] H. Asada und B.-H. Yang. Skill Acquisition From Human Experts Through Pattern Processing of Teaching Data. In *Proceedings of the IEEE International Conference on Robotics and Automation, Scottsdale*, Seiten 1302–1307, 1989.

[Baj89] R. Bajcsy. Active Perception and Exploratory Robotics. In *Proceedings of Artificial Intelligence and Information-Control Systems of Robots*, Seiten 1–9. North Holland, 1989.

[BD91] S. Bennett und G. DeJong. Comparing Stochastic Planning to the Acquisition of Increasingly Permissive Plans for Complex, Uncertain Domains. In *Proceedings of the 8th International Workshop on Machine Learning, Evanston*, Seiten 586–590, 1991.

[Ben89] S. W. Bennett. Learning Uncertainty Tolerant Plans Through Approximation in Complex Domains. Technical Report UILU-ENG-89-2204, Univ. of Illinois at Urbana-Champaign, 1989.

[Ben90] S. W. Bennett. Reducing Real-world Failures of Approximate Explanation-based Rules. In *Proceedings of the 7th International Conference on Machine Learning, Austin*, Seiten 226–234, 1990.

[Ber91] K. Berns. Anwendungen neuronaler Netze in der Robotik. *Robotersysteme*, Seiten 23–32, 1991.

[BGGS89] F. Bergadano, R. Gemello, A. Giordana und L. Saitta. ML-SMART: A Problem Solver for Learning From Examples. *Fundamenta Informaticae*, 12:29–50, 1989.

[BGS88] F. Bergadano, A. Giordana und L. Saitta. Automated Concept Acquisition in Noisy Environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(4):555–578, Juli 1988.

[BGSDM90] S. Basu, A. Gupta, N. Sarkar und D. Dutta Majumder. Knowledge Representation for Vision: an Associative Network for Single Object Representation and Recognition. In *Proceedings of the 10th International Conference on Pattern Recognition, Atlantic City*, Seiten 297–299, 1990.

[BH91] M. Barbehenn und S. Hutchinson. Learning Conditional Effects of Actions for Robot Navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento*, Seiten 260–265, 1991.

[BI87] O. Bartenstein und H. Inoue. Learning-Assisted Robot Programming. In *Proceedings of Robotics Research - The Fourth International Symposium*, 1987.

[Bie85] I. Biederman. Human Image Understanding: Recent Research and a Theory. *Computer Vision, Graphics, and Image Processing*, 32:29–73, 1985.

[BL90] R. Bergevin und M. D. Levine. Extraction of Line Drawing Features for Object Recognition. In *Proceedings of the 10th International Conference on Pattern Recognition, Atlantic City*, Seiten 496–501, 1990.

[BL92] R. Bergevin und M. Levine. Part Decomposition of Objects from Single View Line Drawings. *Image Understanding*, 55(1):73–82, 1992.

[BM89] J. Blythe und T. M. Mitchell. On becoming reactive. In *Proceedings of the 6th International Workshop on Machine Learning, Ithaca*, Seiten 255–257, 1989.

[Bre93] A. Bredehöft. *Lernen von Roboteraktionen aus Benutzervorführungen*. Diplomarbeit, Institut für Prozeßrechentechnik und Robotik, Universität Karlsruhe, 1993.

[Bro82] R. A. Brooks. Symbolic Error Analysis and Robot Planning. *The International Journal of Robotics Research*, 1(4):29–68, 1982.

[Bro85] R. A. Brooks. A Layered Intelligent Control System for a Mobile Robot. In *Proceedings of the 3rd International Symposium on Robotic Research*, Seiten 365–372, 1985.

[Bro90] R. A. Brooks. A Robust Layered Control System for a Mobile Robot. In P. H. Winston und S. A. Shellard, Hrsg., *Artificial Intelligence at MIT, Volume 2*, Kapitel 24, Seiten 2–27. The MIT Press, 1990.

[Bro91] R. A. Brooks. Intelligence Without Reason. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence, Sydney*, Seiten 569–595, 1991.

[BSP85] A. Bundy, B. Silver und D. Plummer. An Analytical Comparison of Some Rule-Learning Programs. *Artificial Intelligence*, 27:137–181, 1985.

[BT87] R. Bajcsy und C. Tsikos. Perception via Manipulation. In *Proceedings of Robotics Research - The Fourth International Symposium*, Seiten 237–244, 1987.

[Buc88] S. J. Buckley. Compliance Viewed as Programming a Damped Spring. In *Proceedings of the 7th National Conference on Artificial Intelligence, Saint Paul*, Seiten 762–767, 1988.

[Buc89] S. J. Buckley. Teaching Compliant Motion Strategies. *IEEE Transactions on Robotics and Automation*, 5(1):112–118, Februar 1989.

[Byl91] T. Bylander. A Simple Model of Knowledge Compilation. *IEEE Expert*, 6(2):73–74, 1991.

[Cai90] T. Cain. A comparative survey of integrated learning systems. Technical Report 90-11, Information and Computer Science, University of California, Irvine, 1990.

[Car83] J. G. Carbonell. Learning by Analogy: Formulating and Generalizing Plans from Past Experience. In R. Michalski, J. Carbonell und T. Mitchell, Hrsg., *Machine Learning - An Artificial Intelligence Approach, Vol. I*, Seiten 137–161. Morgan Kaufmann, 1983.

[Car86] J. G. Carbonell. Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition. In R. Michalski, J. Carbonell und T. Mitchell, Hrsg., *Machine Learning - An Artificial Intelligence Approach, Vol. II*, Seiten 371–392. Morgan Kaufmann, 1986.

[Car89] J. G. Carbonell. Introduction: Paradigms for Machine Learning. *Artificial Intelligence*, 40:1–9, 1989.

[CB87] J. H. Connell und M. Brady. Generating and Generalizing Models of Visual Objects. *Artificial Intelligence*, 31:159–183, 1987.

[CB91] P. Clark und R. Boswell. Rule Induction with CN2: Some Recent Improvements. In *Proceedings of the European Working Session on Learning, Porto*, Seiten 151–163, 1991.

[CG87] J. G. Carbonell und Y. Gil. Learning by Experimentation. In *Proceedings of the 4th International Workshop on Machine Learning, Irvine*, Seiten 256–266, 1987.

[CG90] J. Carbonell und Y. Gil. Learning by Experimentation: The Operator Refinement Method. In Y. Kodratoff und R. Michalski, Hrsg., *Machine Learning - An Artificial Intelligence Approach, Vol. III*, Seiten 191–213. Morgan Kaufmann, 1990.

[Che86] K. Chen. Smooth Path Tracking through Symbolic Computations. Technical Report ISG 86-13, AI Laboratory, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, 1986.

[Chr92a] A. D. Christiansen. Automatic Acquisition of Task Theories for Robotic Manipulation. Technical Report CMU-CS-92-111, Carnegie Mellon University, 1992.

[Chr92b] A. D. Christiansen. Learning to Predict in Uncertain Continuous Tasks. In *Proceedings of the 9th International Workshop on Machine Learning, Aberdeen*, Seiten 72–81, 1992.

[CK91] R. Cromwell und A. Kak. Automatic Generation of Object Class Descriptions using Symbolic Learning Techniques. In *Proceedings of the 9th National Conference on Artificial Intelligence, Anaheim*, Seiten 710–717, 1991.

[CL88] E. Cheung und V. Lumelsky. Motion Planning For Robot Arm Manipulators With Proximity Sensing. In *Proceedings of the IEEE International Conference on Robotics and Automation, Philadelphia*, Seiten 740–745, 1988.

[CL89] E. Cheung und V. J. Lumelsky. Proximity Sensing in Robot Manipulator Motion Planning: System and Implementation Issues. *IEEE Transactions on Robotics and Automation*, 5(6):740–751, Dezember 1989.

[CMM83] J. G. Carbonell, R. Michalski und T. M. Mitchell. An overview of machine learning. In R. Michalski, J. Carbonell und T. Mitchell, Hrsg., *Machine Learning - An Artificial Intelligence Approach, Vol. II*, Seiten 3–24. Morgan Kaufmann, 1983.

[CN89] P. Clark und T. Niblett. The CN2 Induction Algorithm. *Machine Learning*, 3:261–283, 1989.

[Con89a] J. Connell. A Colony Architecture for an Artificial Creature. Technical Report AI-TR 1151, MIT AI Laboratory, 1989.

[Con89b] J. H. Connell. A Behavior-Based Arm Controller. *IEEE Transactions on Robotics and Automation*, 5(6):784–791, Dezember 1989.

[Con90] J. H. Connell. *Minimalist Mobile Robotics - A Colony-Style Architecture for an Artificial Creature.* Academic Press, Inc., 1990.

[Cor92] S. Cord. *Symbolische Lernverfahren für Robotikanwendungen.* Studienarbeit, Institut für Prozeßrechentechnik und Robotik, Universität Karlsruhe, 1992.

[CTM90] A. D. Christiansen, M. M. T. und T. M. Mitchell. Learning Reliable Manipulation Strategies without Initial Physical Models. In *Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati*, Seiten 1224–1230, 1990.

[CWD+91] S. Chien, B. Whitehall, T. Dietterich, R. Doyle, B. Falkenhainer, J. Garrett und S. Lu. Machine Learning in Engineering Automation. In *Proceedings of the 8th International Workshop on Machine Learning, Evanston*, Seiten 577–580, 1991.

[Dan87] A. Danyluk. The use of explanations for similarity-based learning. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, Seiten 274–276, 1987.

[DH73] R. Duda und P. Hart. *Pattern Classification and Scene Analysis.* J. Wiley, 1973.

[Die92] T. Diederich. *Untersuchung des Erwerbs von Objektwissen durch aktive Experimente eines Robotersystems.* Diplomarbeit, Institut für Prozeßrechentechnik und Robotik, Universität Karlsruhe, 1992.

[Dil88] R. Dillmann. *Lernende Roboter - Aspekte maschinellen Lernens.* Nr. 15 in Fachberichte Messen, Steuern, Regeln. Springer, 1988.

[Dil91] R. Dillmann. Strategies for Learning Elementary Mobile Robot Operations. In G. Schmidt, Hrsg., *Proceedings of the International Workshop on Information Processing in Autonomous Mobile Robots*, Seiten 279–292. Springer, 1991.

[DJ90] K. De Jong. Genetic-Algorithm-Based Learning. In Y. Kodratoff und R. Michalski, Hrsg., *Machine Learning - An Artificial Intelligence Approach, Vol. III*, Seiten 611–638. Morgan Kaufmann, 1990.

[DKW93] R. Dillmann, J. Kreuziger und F. Wallner. PRIAMOS: An experimental platform for reflexive navigation. *Robotics and Autonomous Systems*, 11:195–203, 1993.

[DL84] B. Dufay und J.-C. Latombe. An Approach to Automatic Robot Programming Based on Inductive Learning. In M. Brady und R. Paul, Hrsg., *Proceedings of Robotics Research: The 1st International Symposium*, Seiten 97–115. The MIT Press, 1984.

[DM86] G. DeJong und R. Mooney. Explanation-Based Learning: An Alternative View. *Machine Learning*, 1:145–176, 1986.

[DS88] G. B. Dunn und J. Segen. Automatic Discovery of Robotic Grasp Configurations. In *Proceedings of the IEEE International Conference on Robotics and Automation, Philadelphia*, Seiten 396–401, 1988.

[Etz91] O. Etzioni. STATIC - A Problem-Space Compiler for PRODIGY. In *Proceedings of the 9th National Conference on Artificial Intelligence, Anaheim*, Seiten 533–540, 1991.

[EW91] R. Elio und L. Watanabe. An Incremental Deductive Strategy for Controlling Constructive Induction in Learning from Examples. *Machine Learning*, 7(1):7–44, Juli 1991.

[Fah74] S. E. Fahlman. A Planning System for Robot Construction Tasks. *Artificial Intelligence*, 5:1–49, 1974.

[Fan93] M. Fankanowsky. *Ankopplung eines lernenden Planungssystems an ein reales Robotersystem.* Studienarbeit, Institut für Prozeßrechentechnik und Robotik, Universität Karlsruhe, 1993.

[FI87] N. V. Findler und L. H. Ihrig. Analogical Reasoning By Intelligent Robots. In A. Wong und A. Pugh, Hrsg., *Machine Intelligence and Knowledge Engineering for Robotic Applications*, F33, NATO ASI, Seiten 269–282. Springer, 1987.

[Fir87] R. J. Firby. An Investigation into Reactive Planning in Complex Domains. In *Proceedings of the 6th National Conference on Artificial Intelligence, Seattle*, Seiten 202–206, 1987.

[Fis87] D. Fisher. Knowledge Acquisition Via Incremental Conceptual Clustering. *Machine Learning*, 2:139–172, 1987.

[Fis89] D. H. Fisher. Noise-Tolerant Conceptual Clustering. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, Detroit*, Seiten 825–830, 1989.

[FK85] B. Fox und K. Kempf. A Representation for Opportunistic Scheduling. In *Proceedings of the 3rd International Symposium on Robotic Research*, Seiten 109–115, 1985.

[FM89] D. H. Fisher und K. B. McKusick. An Empirical Comparison of ID3 and Back-propagation. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, Detroit*, Seiten 788–793, 1989.

[FM90] B. C. Falkenhainer und R. S. Michalski. Integrating Quantitative and Qualitative Discovery in the Abacus System. In Y. Kodratoff und R. S. Michalski, Hrsg., *Machine Learning - An Artificial Intelligence Approach, Vol. III*, Seiten 153–190. Morgan Kaufmann, 1990.

[FN71] R. E. Fikes und N. J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, Vol. 2:189–208, 1971.

[FS84] E. Feigenbaum und H. Simon. EPAM-like models of recognition and learning. *Cognitive Science*, 8:305–336, 1984.

[FW90] B. Frommherz und G. Werling. Generating Robot Action Plans by means of an Heuristic Search. In *Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati*, Seiten 884–889, 1990.

[FW93] D. Fensel und M. Wiese. Refinement of Rule Sets with JoJo. In P. B. Brazdil, Hrsg., *Proceedings of the European Conference on Machine Learning, Wien*, Seiten 378–383, 1993.

[GC85] M. A. Gluck und J. E. Corter. Information, Uncertainty, and the Utility of Concepts. In *Proceedings of the 7th Annual Conference of the Cognitive Science Society, Irvine*, Seiten 283–287, 1985.

[Gen90] J. H. Gennari. *An Experimental Study of Concept Formation.* Technical report 90-26, Dept. of Information and Computer Science, University of California, Irvine, 1990.

[Ger90] M. T. Gervasio. Learning General Completable Reactive Plans. In *Proceedings of the 8th National Conference on Artificial Intelligence, Boston*, Seiten 1016–1021, 1990.

[Gin87] M. Gini. Symbolic and Qualitative Reasoning for Error Recovery in Robot Programs. In U. Rembold und K. Hörmann, Hrsg., *Languages for Sensor-Based Control in Robotics*, Seiten 147–167. Springer, 1987.

[Gin90] M. Gini. Automatic Error Detection and Recovery. In U. Rembold, Hrsg., *Robot technology and applications*, Seiten 445–483. Marcel Dekker, 1990.

[GL87] M. P. Georgeff und A. L. Lansky. Reactive Reasoning and Planning. In *Proceedings of the 6th National Conference on Artificial Intelligence, Seattle*, Seiten 677–682, 1987.

[GLF89] J. Gennari, P. Langley und D. Fisher. Models of Incremental Concept Formation. *Artificial Intelligence*, 40:11–61, 1989.

[GS87] Y. Goto und A. Stentz. Mobile Robot Navigation: The CMU System. *IEEE Expert*, Seiten 44–54, 1987.

[Hal89] R. Hall. Computational approaches to analogical reasoning: A comparative analysis. *Artificial Intelligence*, 39:39–120, 1989.

[Hau92] M. Hauser. *Entwurf und Implementierung eines Verfahrens zum Lernen von Roboteraktionsplänen*. Diplomarbeit, Institut für Prozeßrechentechnik und Robotik, Universität Karlsruhe, 1992.

[HdMS89] L. Homem de Mello und A. Sanderson. A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences. In *Proceedings of the IEEE International Conference on Robotics and Automation, Scottsdale*, Seiten 56–61, 1989.

[HdMS91a] L. S. Homem de Mello und A. C. Sanderson. A Correct and Complete Algorithm for the Generation of Mechanical Assembly Sequences. *IEEE Transactions on Robotics and Automation*, 7(2):228–240, April 1991.

[HdMS91b] L. S. Homem de Mello und A. C. Sanderson. Representations of Mechanical Assembly Sequences. *IEEE Transactions on Robotics and Automation*, 7(2):211–227, April 1991.

[Hei89] R. Heise. Demonstration Instead of Programming: Focussing Attention in Robot Task Acquisition. Technical Report 89/360/22, Department of Computer Science, The University of Calgary, Canada, 1989.

[HH83] G. Hirzinger und J. Heindl. Sensor programming - a new way for teaching a robot paths and forces/torques simultaneously. In *Proceedings of the 3rd International Conference on Robot Vision and Sensory Controls, Cambridge*, Seiten 399–408, 1983.

[HHM88] A. Hörmann, T. Hugel und W. Meier. Ein Ansatz zur Realisierung intelligenter fehlertoleranter Robotersysteme. *Robotersysteme*, Seiten 223–231, 1988.

[Hir89] H. Hirsh. Incremental Version-Space Merging: A General Framework for Concept Learning. Technical Report STAN-CS-89-1274, Stanford University, Department of Computer Science, Juli 1989.

[HK90] S. Hutchinson und A. Kak. Extending the Classical AI Planning Paradigm to Robotic Assembly Planning. In *Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati*, Seiten 182–189, 1990.

[HKT87] H.-W. Hein, G. Kellermann und C. Thomas. X-AiD: A Shell for Building Highly Interactive And Adaptive User Interfaces. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence, Mailand*, Seiten 97–99, 1987.

[HKW92] R. Hamann, J. Kreuziger und W. Wenzel. Dokumentation der Machine Learning Library. Technical report, Institut für Prozeßrechentechnik und Robotik, Universität Karlsruhe, 1992.

[HL90] Y. Huang und C. Lee. An Automatic Assembly Planning System. In *Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati*, Seiten 1594–1599, 1990.

[HL91] Y. Huang und C. Lee. A Framework of Knowledge-Based Assembly Planning. In *Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento*, Seiten 599–604, 1991.

[Hör92a] A. Hörmann. *Begleitende Montageablaufplanung für ein sensorgestütztes Zweiarm-Manipulatorsystem.* DISKI 11. Infix, Sankt Augustin, 1992.

[Hör92b] A. Hörmann. On-line Planning of Action Sequences for a Two-Arm Manipulator System. In *Proceedings of the IEEE International Conference on Robotics and Automation, Nice*, Seiten 1109–1114, 1992.

[HR85] B. Hayes-Roth. A Blackboard Architecture for Control. *Artificial Intelligence*, 26:251–321, 1985.

[HR90a] L. O. Hall und S. G. Romaniuk. A Hybrid Connectionist, Symbolic Learning System. In *Proceedings of the 8th National Conference on Artificial Intelligence, Boston*, Seiten 783–788, 1990.

[HR90b] A. Hörmann und U. Rembold. The KAMRO System - An Advanced Robot for Autonomous Assembly. In U. Rembold, R. Dillmann und P. Levi, Hrsg., *Proceedings Autonome Mobile Systeme, 6. Fachgespräch*, Seiten 3–19, 1990.

[HR91] A. Hörmann und U. Rembold. Development of an Advanced Robot for Autonomous Assembly. In *Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento*, Seiten 2452–2457, 1991.

[HS88] S. Hirai und T. Sato. Object Model for Telerobot. In *Proceedings of the IEEE International Workshop on Intelligent Robots and Systems*, Seiten 701–706, 1988.

[HS89] S. Hirai und T. Sato. Multi-Level Manual and Autonomous Control Superposition for Intelligent Telerobot. In *Proceedings of the NASA Conference on Space Telerobotics, Pasadena*, Seiten 131–140, 1989.

[HS90] S. Hirai und T. Sato. Integration of a Task Knowledge Base and a Cooperative Maneuvering System for the Telerobot 'MEISTER'. In *Proceedings of the IEEE International Workshop on Intelligent Robots and Systems, Tsuchiura*, Seiten 349–354, 1990.

[HST91a] T. Hasegawa, T. Suehiro und K. Takase. A Model-Based Manipulation System with Skill-Based Execution in Unstructured Environment. In *Proceedings of the 5th International Conference on Advanced Robotics, Pisa*, Seiten 970–975, 1991.

[HST91b] T. Hasegawa, T. Suehiro und K. Takase. A Robot System for Unstructured Environments Based on an Environment Model and Manipulation Skills. In *Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento*, Seiten 916–923, 1991.

[Hus91] Z. Hussain. *Digital Image Processing - Practical Applications of Parallel Processing Techniques.* Ellis Horwood, 1991.

[HW90] W. Hättich und H. Wandres. Automatic Learning of Structural Models for Workpiece Recognition Systems. In *Proceedings of the 10th International Conference on Pattern Recognition, Atlantic City*, Seiten 279–281, 1990.

[IKS93] K. Ikeuchi, M. Kawade und T. Suehiro. Toward Assembly Plan from Observation: Task Recognition with Planar, Curved and Mechanical Contacts. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Yokohama*, Seiten 2294–2301, 1993.

[IS91] K. Ikeuchi und T. Suehiro. Towards an Assembly Plan from Observation: Task recognition with polyhedral objects. Technical Report CMU-CS-91-167, School of Computer Science, Carnegie Mellon University, 1991.

[IS92] K. Ikeuchi und T. Suehiro. Towards an Assembly Plan from Observation. In *Proceedings of the IEEE International Conference on Robotics and Automation, Nice*, Seiten 2171–2177, 1992.

[ISMK86] M. Ishii, S. Sakane, Y. Mikami und M. Kakikura. Teaching Robot Operations and Environments by Using a 3-D Visual Sensor System. In *Proceedings of the International Conference on Intelligent Autonomous Systems, Amsterdam*, Seiten 283–289, 1986.

[Jan88] L.-E. Janlert. Modeling Change - The Frame Problem. In Z. W. Pylyshyn, Hrsg., *The Robot's Dilemma*, Seiten 1–40. ABLEX Publishing Corporation, 1988.

[JL90] M. Johnson und M. Leahy, Jr. Adaptive Model-Based Neural Network Control. In *Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati*, Seiten 1704–1709, 1990.

[JLP90] J. Jones und T. Lozano-Pérez. Planning Two-Fingered Grasps for Pick-and-Place Operations on Polyhedra. In *Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati*, Seiten 683–688, 1990.

[Kad88] C. M. Kadie. Diffy-S: Learning Robot Operator Schemata from Examples. In *Proceedings of the 5th International Conference on Machine Learning*, Seiten 430–436, 1988.

[KC86] S. T. Kedar-Cabelli. Analogy - From a Unified Perspective. Technical Report ML-TR 3, Laboratory for Computer Science Research, Rutgers University, 1986.

[KC92] J. Kreuziger und S. Cord. Anwendungen symbolischer Lernverfahren in der Robotik. Technical Report 23/92, Fakultät für Informatik, Universität Karlsruhe, 1992.

[KGN94] M. Kaiser, A. Giordana und M. Nuttin. Integrated acquisition, execution, evaluation and tuning of elementary skills for intelligent robots. In *Proceedings of the IFAC Symposium on Artificial Intelligence in Real Time Control, Valencia*, 1994.

[KH89] S.-Y. Kung und J.-N. Hwang. Neural Network Architectures for Robotic Applications. *IEEE Transactions on Robotics and Automation*, 5(5):641–657, October 1989.

[KH93a] J. Kreuziger und M. Hauser. The Application of Symbolic Learning Techniques in a Robot Manipulation System. In A. Giordana, Hrsg., *Proceedings of the Workshop on Learning Robots, European Conference on Machine Learning, Vienna*, 1993.

[KH93b] J. Kreuziger und M. Hauser. A New System Architecture for Applying Symbolic Learning Techniques to Robot Manipulation Tasks. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Yokohama*, Seiten 1441–1448, 1993.

[KHW91] J. Kreuziger, R. Hamann und W. Wenzel. Comparison of Inductive Learning Programs. In S. Thrun, J. Bala, E. Bloedorn, I. Bratko und al., Hrsg., *The MONK's problems - A Performance Comparison of Different Learning Algorithms*. Technical Report CMU-CS-91-197, Carnegie Mellon University, 1991.

[KII92] Y. Kuniyoshi, M. Inaba und H. Inoue. Seeing, Understanding and Doing Human Task. In *Proceedings of the IEEE International Conference on Robotics and Automation, Nice*, Seiten 2–9, 1992.

[KLL84] T. Kodratoff und R. Lemerle-Loisel. Learning Complex Structural Descriptions from Examples. *Computer Vision, Graphics, and Image Processing*, 27:266–290, 1984.

[KME91] C. A. Knoblock, S. Minton und O. Etzioni. Integrating Abstraction and Explanation-Based Learning in PRODIGY. In *Proceedings of the 9th National Conference on Artificial Intelligence, Anaheim*, Seiten 541–546, 1991.

[Kre92] J. Kreuziger. Application of Machine Learning to Robotics - An Analysis. In *Proceedings of the 2nd International Conference on Automation, Robotics and Computer Vision, Singapore*, Seiten RO–15.4.1–RO–15.4.5, 1992.

[KRTD90] Y. Kodratoff, C. Rouveirol, G. Tecucci und B. Duval. Symbolic approaches to uncertainty. In Z. W. Ras und M. Zemankova, Hrsg., *Intelligent Systems - State of the art and future directions*, Seiten 259–291. Ellis Horwood, 1990.

[Kup88] M. Kuperstein. Generalized Neural Model For Adaptive Sensory-Motor Control Of Single Postures. In *Proceedings of the IEEE International Conference on Robotics and Automation, Philadelphia*, Seiten 140–144, 1988.

[Kur90] A. Kurz. Selbstorganisierende Entwicklung von zielgerichtetem Verhalten eines durch einen lernenden Regelkreis kontrollierten Fahrzeuges. In U. Rembold, R. Dillmann und P. Levi, Hrsg., *Proceedings Autonome Mobile Systeme, 6. Fachgespräch*, Seiten 135–150, 1990.

[KW94] J. Kreuziger und W. Wenzel. Learning Generic Object Descriptions for Autonomous Robot Systems. In *Proceedings of the 5th International Symposium on Robotics and Manufacturing*, 1994.

[LC91] V. Lumelsky und E. Cheung. Towards Safe Real-Time Robot Teleoperation: Automatic Whole-Sensitive Arm Collision Avoidance Frees the Operator for Global Control. In *Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento*, Seiten 797–802, 1991.

[LE92] H. Legewie und W. Ehlers. *Knaurs Moderne Psychologie*. Droemer Knaur, 1992.

[Leb86] M. Lebowitz. Integrated Learning: Controlling Explanation. *Cognitive Science*, 10:219–240, 1986.

[Leb87] M. Lebowitz. Experiments with Incremental Concept Formation: UNIMEM. *Machine Learning*, 2:103–138, 1987.

[Lee89] M. H. Lee. *Intelligent Robotics*. Open University Press, 1989.

[Lew93] A. Lewark. *Entwicklung eines Tools zur Vorführung von Roboteraktionen*. Studienarbeit, Institut für Prozeßrechentechnik und Robotik, Universität Karlsruhe, 1993.

[LGCS89] S. Leake, T. Green, S. Cofer und T. Sauerwein. Hierarchical Ada Robot Programming System (HARPS). In *Proceedings of the IEEE International Conference on Robotics and Automation*, Seiten 1022–1028, 1989.

[LHM91] D. Lyons, A. Hendriks und S. Mehta. Achieving Robustness by Casting Planning as Adaption of a Reactive System. In *Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento*, Seiten 198–203, 1991.

[LHYT90] J. Laird, M. Hucka, E. Yager und C. Tuck. Correcting and Extending Domain Knowledge using Outside Guidance. In *Proceedings of the 7th International Conference on Machine Learning, Austin*, Seiten 235–243, 1990.

[Lin91] L.-J. Lin. Programming Robots Using Reinforcement Learning and Teaching. In *Proceedings of the 9th National Conference on Artificial Intelligence, Anaheim*, Seiten 781–786, 1991.

[LM92] L.-J. Lin und T. M. Mitchell. Memory Approaches To Reinforcement Learning In Non-Markovian Domains. Technical Report CMU-CS-92-138, School of Computer Science, Carnegie Mellon University, Mai 1992.

[LNR87] J. Laird, A. Newell und P. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(3), 1987.

[LPJM+87] T. Lozano-Pérez, J. Jones, E. Mazer, P. O'Donnell, W. Grimson, R. Tournassoud und A. Lanusse. Handey: A Robot system that recognizes, plans and manipulates. In *Proceedings of the IEEE International Conference on Robotics and Automation, Raleigh*, Seiten 843–849, 1987.

[LPT85] C. Laugier und J. Pertin-Troccaz. SHARP: A System for Automatic Programming of Manipulation Robots. In *Proceedings of the 3rd International Symposium on Robotic Research*, Seiten 125–132, 1985.

[LPW77] T. Lozano-Perez und P. H. Winston. LAMA: A Language for Automatic Mechanical Assembly. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, Seiten 710–716, 1977.

[LR90] J. E. Laird und P. S. Rosenbloom. Integrating Execution, Planning, and Learning in Soar for External Environments. In *Proceedings of the 8th National Conference on Artificial Intelligence, Boston*, Seiten 1022–1029, 1990.

[LS84] A. Levas und M. Selfridge. A User-Friendly High-Level Robot Teaching System. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1984.

[LS90] S. Lee und Y. G. Shin. Assembly Planning Based on Subassembly Extraction. In *Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati*, Seiten 1606–1611, 1990.

[LTI+89] P. Langley, K. Thompson, W. Iba, J. H. Gennari und J. A. Allen. An Integrated Cognitive Architecture for Autonomous Agents. Technical Report 89-28, University of California, Irvine, Department of Information and Computer Science, September 1989.

[LW77] L. Lieberman und M. Wesley. AUTOPASS: Am Automatic Programming System for Computer Controlled Mechanical Assembly. *IBM Journal on Research Development*, Seiten 321–333, 1977.

[LZSB86] P. Langley, J. M. Zytkow, H. A. Simon und G. L. Bradshaw. The Search for Regularity: Four Aspects of Scientific Discovery. In R. S. Michalski,

J. G. Carbonell und T. M. Mitchell, Hrsg., *Machine Learning - An Artificial Intelligence Approach, Vol. II*, Seiten 425–469. Morgan Kaufman, 1986.

[Mae90] P. Maes, Hrsg. *Designing Autonomous Agents.* MIT/Elsevier, 1990.

[MB90] P. Maes und R. A. Brooks. Learning to Coordinate Behaviors. In *Proceedings of the 8th National Conference on Artificial Intelligence, Boston*, Seiten 796–802, 1990.

[MCK+89] S. Minton, J. Carbonell, C. Knoblock, D. Kuoka, O. Etzioni und Y. Gil. Explanation-Based Learning - A Problem Solving Perspective. *Artificial Intelligence*, 40:63–118, 1989.

[MCM89] M. T. Mason, A. D. Christiansen und T. M. Mitchell. Experiments in Robot Learning. In *Proceedings of the 6th International Workshop on Machine Learning, Ithaca*, Seiten 141–145, 1989.

[Mel88] B. W. Mel. Building and Using Mental Models in a Sensory-Motor Domain: A Connectionist Approach. In *Proceedings of the 5th International Conference on Machine Learning*, Seiten 207–213, 1988.

[Meu88] R. Meunier. Lernen von Handlungsplänen mit Pattern-Matching-Verfahren. Technical Report Arbeitspapiere der GMD, Nr. 324, Gesellschaft für Mathematik und Datenverarbeitung mbH, 1988.

[MHM+89] G. Meijer, L. Hertzberger, T. Mai, E. Gaussens und F. Arlabosse. Exception handling system for autonomous robots based on PES. In *Proceedings of the International Conference on Intelligent Autonomous Systems, Amsterdam*, Seiten 65–77, 1989.

[Mic83] R. S. Michalski. A Theory and Methodology of Inductive Learning. In R. Michalski, J. Carbonell und T. Mitchell, Hrsg., *Machine Learning - An Artificial Intelligence Approach, Vol. I*, Seiten 83–134. Morgan Kaufmann, 1983.

[Mic86] R. M. Michalski. Understanding The Nature Of Learning: Issues And Research Directions. In R. S. Michalski, J. G. Carbonell und T. M. Mitchell, Hrsg., *Machine Learning - An Artificial Intelligence Approach, Vol. II*, Seiten 3–25. Morgan Kaufmann, 1986.

[Mic91] R. S. Michalski. Toward a Unifying Theory of Learning: An Outline of Basic Ideas. In *Proceedings of the 1st World Conference on the Fundamentals of Artificial Intelligence, Paris*, 1991.

[Mil87] W. T. Miller, III. Sensor-Based Control of Robotic Manipulators Using a General Learning Algorithm. *IEEE Journal of Robotics and Automation*, Ra-3(2):157–165, April 1987.

[Min75] M. Minsky. A framework for representing knowledge. In P. Winston, Hrsg., *The Psychology of Computer Vision.* McGraw-Hill, 1975.

[Min88] S. Minton. Learning Effective Search Control Knowledge: An Explanation-Based Approach. Technical Report CMU-CS-88-133, Carnegie Mellon University, 1988.

[Mit82] T. M. Mitchell. Generalization as Search. *Artificial Intelligence*, 18:203–226, 1982.

[Mit89] T. M. Mitchell. Theo: A Framework for Self-Improving Systems. In K. VanLehn, Hrsg., *Architectures for Intelligence*. Erlbaum, 1989.

[Mit90] T. M. Mitchell. Becoming Increasingly Reactive. In *Proceedings of the 8th National Conference on Artificial Intelligence, Boston*, Seiten 1051–1058, 1990.

[Mit91] T. M. Mitchell. Personal communication. European Summer School on Machine Learning, Belgium, 1991.

[MK90] R. S. Michalski und Y. Kodratoff. Research in Machine Learning: Recent Progress, Classification of Methods, and Future Directions. In Y. Kodratoff und R. S. Michalski, Hrsg., *Machine Learning - An Artificial Intelligence Approach, Vol. III*, Kapitel 1, Seiten 3–30. Morgan Kaufmann, 1990.

[MKKC86] T. M. Mitchell, R. Keller und S. Kedar-Cabelli. Explanation-based Generalization: A Unifying View. *Machine Learning*, 1:47–80, 1986.

[MKKD94] S. Münch, J. Kreuziger, M. Kaiser und R. Dillmann. Robot Programming by Demonstration (RPD) - Using Machine Learning and User Interaction Methods for the Development of Easy and Comfortable Robot Programming Systems. In *Proceedings of the 25th International Symposium on Industrial Robots, Hannover*, 1994.

[MMHL86] R. Michalski, I. Mozetic, J. Hong und N. Lavrac. The multipurpose incremental learning system AQ15 and its application to three medical domains. In *Proceedings of the 5th National Conference on Artificial Intelligence, Philadelphia*, Seiten 1041–1045, 1986.

[MMS85] T. Mitchell, S. Mahadevan und L. Steinberg. LEAP: A learning apprentice for VLSI design. In *Proceedings of the 9th International Conference on Artificial Intelligence, Los Angeles*, Seiten 573–580, 1985.

[Moo90] A. Moore. Acquisition of Dynamic Control Knowledge for a Robotic Manipulator. In *Proceedings of the 7th International Conference on Machine Learning, Austin*, Seiten 244–252, 1990.

[MR93] K. Morik und A. Rieger. Learning Action-oriented Perceptual Features for Robot Navigation. In A. Giordana, Hrsg., *Proceedings of the Workshop on Learning Robots, European Conference on Machine Learning, Vienna*, 1993.

[MS83] R. Michalski und R. Stepp. Learning from observation: Conceptual Clustering. In R. Michalski, J. Carbonell und T. Mitchell, Hrsg., *Machine Learning - An Artificial Intelligence Approach, Vol. I*, Seiten 331–363. Morgan Kaufmann, 1983.

[MS91] R. Maclin und J. Shavlik. Refining Domain Theories Expressed as Finite-State Automata. In *Proceedings of the 8th International Workshop on Machine Learning, Evanston*, Seiten 524–528, 1991.

[MSTG89] R. Mooney, J. Shavlik, G. Towell und A. Gove. An Experimental Comparison of Symbolic and Connectionist Learning Algorithms. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, Detroit*, Seiten 775–780, 1989.

[Mug93] S. Muggleton. Inductive Logic Programming: derivations, successes and shortcomings. In *Proceedings of the European Conference on Machine Learning*, Seiten 21–37, 1993.

[Nag84] H.-H. Nagel. Digitisierung und Klassifikation von Signalen. Skript der Fakultät für Informatik, Universität Karlsruhe, 1984.

[NB87] H. Niemann und H. Bunke. *Künstliche Intelligenz in Bild- und Sprachanalyse.* Teubner, 1987.

[New82] A. Newell. The Knowledge Level. *Artificial Intelligence*, 18(1):87–127, 1982.

[PAB80] R. Popplestone, A. A.P. und I. Bellos. An Interpreter for a Language for Describing Assemblies. *Artificial Intelligence*, 14:79–107, 1980.

[PG88] J. Pabon und D. Gossard. Connectionist Networks for Learning Coordinated Motion in Autonomous Systems. In *Proceedings of the 7th National Conference on Artificial Intelligence, Saint Paul*, Seiten 791–795, 1988.

[PK90] M. Pazzani und D. Kibler. The Utility of Knowledge in Inductive Learning. Technical Report 90-18, Information and Computer Science, University of California, Irvine, 1990.

[Qui86] J. Quinlan. Induction of Decision Trees. *Machine Learning*, 1:81–106, 1986.

[RD89] U. Rembold und R. Dillmann. The Control System of the Autonomous Mobile Robot KAMRO of the University of Karlsruhe. In *Proceedings of the International Conference on Intelligent Autonomous Systems*, Seiten 565–575, 1989.

[RMLB88] K. Rao, G. Medioni, H. Liu und G. Bekey. Robot Hand-Eye Coordination: Shape Description and Grasping. In *Proceedings of the IEEE International Conference on Robotics and Automation, Philadelphia*, Seiten 407–411, 1988.

[RS87] R. Rivest und R. Schapire. A new approach to unsupervised learning in deterministic environments. In *Proceedings of the 4th International Workshop on Machine Learning, Irvine*, Seiten 364–375, 1987.

[Sac74] E. D. Sacerdoti. Planning in a Hierarchy of Abstraction Spaces. *Artificial Intelligence*, 5:115–135, 1974.

[SB91] L. Stark und K. Bowyer. Achieving Generalized Object Recognition through Reasoning about Association of Function to Structure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(10):1097–1103, 1991.

[SB92] M. Sassin und S. Bocionek. Programming by Demonstration: A Basis for Auto-Customizable Workstation Software. In *Proceedings of the Workshop on Intelligent Workstations for Professionals*. Springer, 1992.

[Sch87] M. Schoppers. Universal Plans for Reactive Robots in Unpredictable Environments. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence, Mailand*, Seiten 1039–1046, 1987.

[Sch94] J. Schloen. Wissensbasierte Bewegungsausführung für die Montageautomatisierung mit Industrierobotern. Institut für Prozeßrechentechnik und Robotik, Universität Karlsruhe, 1994.

[SD90] J. W. Shavlik und T. G. Dietterich, Hrsg. *Readings in Machine Learning*. Morgan Kaufmann, 1990.

[Seg88a] J. Segen. Learning Graph Models of Shape. In *Proceedings of the 5th International Conference on Machine Learning, Ann Arbor*, Seiten 29–35, 1988.

[Seg88b] A. Segre. *Machine Learning of Robot Assembly Plans*. Kluwer Academic Publishers, 1988.

[SH86] C. Sammut und D. Hume. Learning Concepts in a Complex Robot World. In T. Mitchell, J. Carbonell und R. Michalski, Hrsg., *Machine Learning - A Guide To Current Research*, Seiten 291–294. Kluwer Academic Publishers, 1986.

[SH87] T. Sato und S. Hirai. MEISTER: A Model Enhanced Intelligent and Skillful Teleoperational Robot System. In *Proceedings of Robotics Research - The Fourth International Symposium*, Seiten 155–162, 1987.

[Sha90] R. Shadmer. Learning Virtual Equilibrium Trajectories for Control of a Robot Arm. *Neural Computation*, 2(4):436–446, 1990.

[Sim83] H. A. Simon. Why Should Machines Learn? In R. S. Michalski, J. G. Carbonell und M. T. M., Hrsg., *Machine Learning - An Artificial Intelligence Approach, Vol. I*, Seiten 25–38. Tioga Publishing Company, 1983.

[Sim90] W. Simon. Lern- und Selbstanpassungsfähigkeit eines Expertensystems zur intelligenten Prozeßablaufsteuerung von robotergestützten Montagevorgängen. In *Proceedings GMA-Kongreß 'Meß - und Automatisierungstechnik'*, 1990.

[Sim91] W. Simon. *Untersuchungen zu einer intelligenten und lernfähigen Robotersteuerung für die Montageautomatisierung*. Nr. 47 in Reihe 20: Rechnerunterstützte Verfahren. VDI Verlag, 1991.

[SK91] P. v. d. Smagt und B. J. Kröse. A real-time learning neural robot controller. In T. Kohonen, K. Mäkisara, O. Simul und J. Kangas, Hrsg., *Artificial Neural Networks*, Seiten 351–356. Elsevier Science Publishers B.V., 1991.

[SL83] M. Selfridge und A. Levas. Teaching Robots by Example. In *Proceedings of the 13th International Symposium on Industrial Robots and Robots 7, Chicago*, Seiten 13–160–13–165, 1983.

[SL89] R. P. Sobek und J.-P. Laumond. Using Learning to Recover Side-Effects of Operators in Robotics. In *Proceedings of the 6th International Workshop on Machine Learning, Ithaca*, Seiten 205–208, 1989.

[SM86] R. E. Stepp und R. S. Michalski. Conceptual Clustering: Inventing Goal-Oriented Classifications of Structured Objects. In R. S. Michalski, J. G. Carbonell und T. M. Mitchell, Hrsg., *Machine Learning - An Artificial Intelligence Approach, Vol. II*, Seiten 471–498. Morgan Kaufmann, 1986.

[SM90] W. Simon und J. Matthiesen. Automated Generation of Strategy Rules From User Instructions Within an Expert System for Robot-Based and Sensory-Controlled Assembly Automation. In *Proceedings of the International Conference on Applications of Artificial Intelligence in Engineering*, 1990.

[Sma91] M. Smaluhn. *Single Input Graphics Module for the X Window Environment.* Diplomarbeit, Institut für Prozeßrechentechnik und Robotik, Universität Karlsruhe, 1991.

[ST89a] J. W. Shavlik und G. G. Towell. Combining Explanation-Based Learning and Artificial Neural Networks. In *Proceedings of the 6th International Workshop on Machine Learning, Ithaca*, Seiten 90–92, 1989.

[ST89b] J. Shavlik und G. Towell. An Approach to Combining Explanation-based and Neural Learning Algorithms. *Connection Science*, 1(3), 1989.

[Sta88a] S. Stansfield. Reasoning about Grasping. In *Proceedings of the 7th National Conference on Artificial Intelligence, Saint Paul*, Seiten 768–773, 1988.

[Sta88b] S. Stansfield. Representing Generic Objects for Exploration and Recognition. In *Proceedings of the IEEE International Conference on Robotics and Automation, Philadelphia*, Seiten 1090–1095, 1988.

[Sta88c] S. Stansfield. Representing Generic Objects for Exploration and Recognition. In *Proceedings of the IEEE International Conference on Robotics and Automation, Philadelphia*, Seiten 1090–1095, 1988.

[Sta90] S. Stansfield. Knowledge-Based Robotic Grasping. In *Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati*, Seiten 1270–1275, 1990.

[Sta91] S. Stansfield. Robotic Grasping of Unknown Objects: A Knowledge-based Approach. *The International Journal of Robotics Research*, 10(4):314–326, 1991.

[Sto93] M. Stockwald. *Evaluierung und Erweiterung eines lernfähigen Objekterkennungsmoduls.* Studienarbeit, Institut für Prozeßrechentechnik und Robotik, Universität Karlsruhe, 1993.

[Tal88] H. Tallis. Tuning Rule-Based Systems to Their Environments. In *Proceedings of the 5th International Conference on Machine Learning*, Seiten 8–14, 1988.

[Tan90] M. Tan. CSL: A Cost-Sensitive Learning System for Sensing and Grasping Objects. In *Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati*, Seiten 858–863, 1990.

[TBB+91] S. Thrun, J. Bala, E. Bloedorn, I. Bratko und al. The MONK's problems - A Performance Comparison of Different Learning Algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University, 1991.

[Tec91] G. Tecuci. A Multistrategy Learning Approach to Domain Modeling and Knowledge Acquisition. In *Proceedings of the European Working Session on Learning, Porto*, Seiten 14–32, 1991.

[TK90] G. Tecuci und Y. Kodratoff. Apprenticeship Learning in Imperfect Domain Theories. In Y. Kodratoff und R. S. Michalski, Hrsg., *Machine Learning - An Artificial Intelligence Approach, Vol. III*, Seiten 514–551. Morgan Kaufmann, 1990.

[TLI91] K. Thompson, P. Langley und W. Iba. Using Background Knowledge in Concept Formation. In *Proceedings of the 8th International Workshop on Machine Learning, Evanston*, Seiten 554–558, 1991.

[TO92] T. Takahashi und H. Ogata. Robotic Assembly Operation based on Task-Level Teaching in Virtual Reality. In *Proceedings of the IEEE International Conference on Robotics and Automation, Nice*, Seiten 1083–1088, 1992.

[TS90a] M. Tan und J. C. Schlimmer. Two Case Studies in Cost-Sensitive Concept Acquisition. In *Proceedings of the 8th National Conference on Artificial Intelligence, Boston*, Seiten 854–860, 1990.

[TS90b] M. Tan und J. C. Schlimmer. Two Case Studies in Cost-Sensitive Concept Acquisition. In *Proceedings of the 8th National Conference on Artificial Intelligence, Boston*, Seiten 854–860, 1990.

[TS92] G. G. Towell und J. W. Shavlik. Refining Symbolic Knowledge Using Neural Networks. In *Proc. of the Workshop on Multi-Strategy Learning Systems*, Seiten 257–272, 1992.

[TSN90] G. G. Towell, J. W. Shavlik und M. O. Noordwier. Refinement of Approximate Domain Theories by Knowledge-Based Neural Networks. In *Proceedings of the 8th National Conference on Artificial Intelligence, Boston*, Seiten 861–866, 1990.

[Tu94] N. L. Tu. *Aktivierungsmechanismen und Lernstrategien für Planungssysteme*. Diplomarbeit, Institut für Prozeßrechentechnik und Robotik, Universität Karlsruhe, 1994.

[Utg89] P. Utgoff. Incremental Induction of Decision Trees. *Machine Learning*, 4:161–186, 1989.

[VC90] M. M. Veloso und J. G. Carbonell. Integrating analogy into a general problem-solving architecture. In Z. W. Ras und M. Zemankova, Hrsg., *Intelligent Systems - State of the art and future directions*, Seiten 29–51. Ellis Horwood, 1990.

[VC91] M. M. Veloso und J. G. Carbonell. Learning by Analogical Replay in Prodigy: First Results. In *Proceedings of the European Working Session on Learning, Porto*, Seiten 375–390, 1991.

[VS91] E. G. Vaaler und W. P. Seering. A Machine Learning Algorithm for Automated Assembly. In *Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento*, Seiten 2231–2237, 1991.

[WBKL83] P. H. Winston, T. A. Binford, B. Katz und M. Lowry. Learning Physical Descriptions from Function Definitions, Examples, and Precedents. M.I.T. AI Memo 679, AI Laboratory, MIT, 1983.

[WBKL84] P. H. Winston, T. O. Binford, B. Katz und M. Lowry. Learning Physical Descriptions from Functional Definitions, Examples, and Precedents. In M. Brady und R. Paul, Hrsg., *Proceedings of Robotics Research - The 1st International Symposium*, Seiten 117–135, 1984.

[WE87] L. Watanabe und R. Elio. Guiding Constructive Induction for Incremental Learning from Examples. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence, Mailand*, Seiten 293–296, 1987.

[Wei93] B. Weismann. *Einsatz von aktiven Experimenten eines Roboters zur Akquisition von statischen und dynamischen Objekteigenschaften.* Diplomarbeit, Institut für Prozeßrechentechnik und Robotik, Universität Karlsruhe, 1993.

[Wen92] W. Wenzel. *Einsatz von symbolischen Lernverfahren zum Aufbau von prototypischen Objektbeschreibungen.* Diplomarbeit, Institut für Prozeßrechentechnik und Robotik, Universität Karlsruhe, 1992.

[Wer93] G. Werling. *Produktorientierte automatische Planung von Prüfoperationen bei der robotergestützten Montage.* DISKI 46. Infix, Sankt Augustin, 1993.

[Wil88] D. E. Wilkins. *Practical Planning.* Morgan Kaufmann, 1988.

[Win80] P. H. Winston. Learning and Reasoning by Analogy. *Communications of the ACM*, 23(12):689–703, Dezember 1980.

[Win81] P. H. Winston. Learning new Principles from Precedents and Exercises: The Details. AI Memo 632, AI Laboratory, MIT, 1981.

[WK89] S. M. Weiss und I. Kapouleas. An Empirical Comparison of Pattern Recognition, Neural Nets, and Machine Learning Classification Methods. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence, Detroit*, Seiten 781–787, 1989.

[Wol89] J. D. Wolter. On the Automatic Generation of Assembly Plans. In *Proceedings of the IEEE International Conference on Robotics and Automation, Scottsdale*, Seiten 62–68, 1989.

[Won91] A. K. Wong. Robotic Vision Knowledge System. In C. G. Lee, Hrsg., *Sensor-Based Robots: Algorithms and Architectures*, Seiten 61–92. Springer, 1991.

[XB90] X. Xia und G. Bekey. Integrating Robotic Assembly Planning and Scheduling: A Two-Dimensional View. In *Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati*, Seiten 1956–1961, 1990.

[YY90] T. Yabuta und T. Yamada. Possibility of Neural Networks Controller for Robot Manipulators. In *Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati*, Seiten 1686–1691, 1990.

[ZD91] Y. Zheng und L. K. Daneshmend. Learning Error-Recovery Strategies in Telerobotic Systems. In *Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento*, Seiten 252–259, 1991.

[ZM93] T. Zrimec und P. Mowforth. Learning in the Domain of Robot Pushing. In A. Giordana, Hrsg., *Proceedings of the Workshop on Learning Robots, European Conference on Machine Learning, Wien*, 1993.

Index

DISKI – Dissertationen zur Künstlichen Intelligenz

Band 1: Norbert Reithinger (Universität Saarbrücken): Eine parallele Architektur zur inkrementellen Generierung multimodaler Dialogbeiträge.
ISBN 3-929037-01-7

Band 2: Claus Müller (Universität Karlsruhe): Verwendung von Bildauswertungsmethoden zur Erkennung und Lagebestimmung von generischen polyedrischen Objekten im Raum.
ISBN 3-929037-02-5

Band 3: Andreas Günter (Universität Hamburg): Flexible Kontrolle in Expertensystemen zur Planung und Konfigurierung in technischen Domänen.
ISBN 3-929037-03-3

Band 4: Ingo Syska (Universität Hamburg): Modulare Problemlösungsarchitekturen für Konstruktionssysteme.
ISBN 3-929037-04-1

Band 5: Wolfgang Tank (TU Berlin): Modellierung von Expertise über Konfigurierungsaufgaben. ISBN 3-929037-05-X

Band 6: Günter Wöhlke (Universität Karlsruhe): Wissensbasierte Greifplanung für Mehrfinger-Roboterhände.
ISBN 3-929037-06-8

Band 7: Knut Möller (Universität Bonn): Adaptive Roboterkontrolle mit konnektionistischen Systemen.
ISBN 3-929037-07-6

Band 8: Andreas Strasser (TU München): Generierung domänenspezifischer Wissensrepräsentationssysteme und Transformation von Wissensbasen mit einer Anwendung in der Rechtsinformatik.
ISBN 3-929037-08-4

Band 9: Longin Latecki (Universität Hamburg): Digitale und Allgemeine Topologie in der bildhaften Wissensrepräsentation.
ISBN 3-929037-09-2

Band 10: Jutta Eusterbrock (Universität Stuttgart): Wissensbasierte Verfahren zur Synthese mathematischer Beweise: Eine kombinatorische Anwendung.
ISBN 3-929037-10-6

Band 11: Andreas Hörmann (Universität Karlsruhe): Begleitende Montageablaufplanung für ein sensorgestütztes Zweiarm-Manipulatorsystem.
ISBN 3-929037-11-4

Band 12: Franz Kummert (Universität Erlangen-Nürnberg): Flexible Steuerung eines sprachverstehenden Systems mit homogener Wissensbasis.
ISBN 3-929037-12-2

Band 13: Dieter Koller (Universität Karlsruhe): Detektion, Verfolgung und Klassifikation bewegter Objekte in monokularen Bildfolgen am Beispiel von Straßenverkehrsszenen.
ISBN 3-929037-13-0

Band 14: Thomas Martinetz (TU München): Selbstorganisierende neuronale Netzwerkmodelle zur Bewegungssteuerung.
ISBN 3-929037-14-9

Band 15: Roman Cunis (Universität Hamburg): Das 3-stufige Frame-Repräsentationsschema – eine mehrdimensional modulare Basis für die Entwicklung von Expertensystemkernen.
ISBN 3-929037-15-7

Band 16: Byoung-Tak Zhang (Universität Bonn): Lernen durch Genetisch-Neuronale Evolution: Aktive Anpassung an unbekannte Umgebungen mit selbstentwickelten parallelen Netzwerken.
ISBN 3-929037-16-5

Band 17: Kai Zercher (TU München): Wissensintensives Lernen für zeitkritische technische Diagnoseaufgaben.
ISBN 3-929037-17-3

Band 18: Gerd Neugebauer (TH Darmstadt): Pragmatische Programmsynthese.
ISBN 3-929037-18-1

Band 19: Manfred Jeusfeld (Universität Passau): Änderungskontrolle in deduktiven Datenbanken.
ISBN 3-929037-19-X

Band 20: Adelinde Uhrmacher (Universität Koblenz-Landau): EMSY – Ein Modellierungskonzept für ökologische und biologische Systeme unter besonderer Berücksichtigung ihrer dynamischen Veränderung.
ISBN 3-929037-20-3

Band 21: Ulrich Aßmann (Universität Bonn): Parallele Modelle für Deduktionssysteme.
ISBN 3-929037-21-1

Band 22: Tilo Messer (TU München): Wissensbasierte Synthese von Bildanalyseprogrammen.
ISBN 3-929037-22-X

Band 23: Klaus-Dieter Althoff (Universität Kaiserslautern): Eine fallbasierte Lernkomponente als integraler Bestandteil der MOLTKE-Werkbank zur Diagnose technischer Systeme.
ISBN 3-929037-23-8

Band 24: Klaus Obermayer (TU München): Adaptive Neuronale Netze und ihre Anwendung als Modelle der Entwicklung kortikaler Karten.
ISBN 3-929037-24-6

Band 25: Wolfgang Ertel (TU München): Parallele Suche mit randomisiertem Wettbewerb in Inferenzsystemen.
ISBN 3-929037-25-4

Band 26: Klaus Fischer (Universität Saarbrücken): Verteiltes und kooperatives Planen in einer flexiblen Fertigungsumgebung.
ISBN 3-929037-26-2

Band 27: Josef Pauli (TU München): Erklärungsbasiertes Computer-Sehen von Bildfolgen.
ISBN 3-929037-27-0

Band 28: Reinhard Prechtel (Universität Erlangen-Nürnberg): Erklärungen für komplexe Wissensbasen.
ISBN 3-929037-28-9

Band 29: Christoph Lehner (Universität München): Grammatikentwicklung mit Constraint Logik Programmierung. Implementierung einer Grammatik für das Deutsche mit PROLOG III.
ISBN 3-929037-29-7

Band 30: Petra Ludewig (Universität Osnabrück): Inkrementelle wörterbuchbasierte Wortschatzerweiterungen in sprachverarbeitenden Systemen. Entwurf einer konstruktiven Lexikonkonzeption.
ISBN 3-929037-30-0

Band 31: Wolfgang Eckstein (TU München): Die Bildanalysesprache TRIAS.
ISBN 3-929037-31-9

Band 32: Claudia Sommer (Universität Erlangen-Nürnberg): MoKon – Ein Ansatz zur Wissensbasierten Konfiguration von Variantenerzeugnissen.
ISBN 3-929037-32-7

Band 33: Yong Cao (Universität Hamburg): Zur Darstellung und Verarbeitung von Wissen über Himmelsrichtungen.
ISBN 3-929037-33-5

Band 34: Michael Spreng (Universität Karlsruhe): Situationsanalyse bei Kontakten während der Ausführung von Roboterbewegungen in unsicheren Umgebungen.
ISBN 3-929037-34-3

Band 35: Hans-Christian Brüning (TU München): Erweiterung der Wissensbasierten CAD-Konstruktion um Restriktionsnetztechniken.
ISBN 3-929037-35-1

Band 36: Haibin Liu (Universität Erlangen-Nürnberg): Oberflächenbasierte Segmentierung von Tiefenbildern.
ISBN 3-929037-36-X

Band 37: Jürgen Sauer (Universität Oldenburg): Wissensbasiertes Lösen von Ablaufplanungsproblemen durch explizite Heuristiken.
ISBN 3-929037-37-8

Band 38: Axel Köhne (Universität Karlsruhe): Integration von Aktionsplanung und Konfigurierung.
ISBN 3-929037-38-6

Band 39: Karlhorst Klotz (TU München): Eine mehrschichtige Architektur zur Fehlerdiagnose und Fehlerbehebung bei der Entwicklung logischer Programme.
ISBN 3-929037-39-4

Band 40: Barbara Hemforth (Universität Bochum): Kognitives Parsing: Repräsentation und Verarbeitung sprachlichen Wissens.
ISBN 3-929037-40-8

Band 41: Michael Ley (Universität Trier): Ein Datenbankkern zur Speicherung variabel strukturierter Feature-Terme. Implementierungstechniken.
ISBN 3-929037-41-6

Band 42: Günther Specht (TU München): Source-to-Source Transformationen zur Erklärung des Programmverhaltens bei deduktiven Datenbanken.
ISBN 3-929037-42-4

Band 43: Josef Ingenerf (RWTH Aachen): Benutzeranpaßbare semantische Sprachanalyse und Begriffsrepräsentation für die medizinische Dokumentation.
ISBN 3-929037-43-2

Band 44: Thomas Bayer (Universität Erlangen-Nürnberg): Ein modellgestütztes Analysesystem zum Bildverstehen strukturierter Dokumente.
ISBN 3-929037-44-0

Band 45: Volker Steinhage (Universität Bonn): Verdeckungen und spezielle Sichten bei der Polyederrekonstruktion, viii+207 S., kart. 48,– DM.
ISBN 3-929037-45-9

Band 46: Gerhard Werling (Universität Karlsruhe): Produkorientierte automatische Planung von Prüfoperationen bei der robotergestützten Montage.
ISBN 3-929037-46-7

Band 47: Albert Maier (Universität Trier): Einbettung von Konzepthierarchien in ein Deduktives Datenbanksystem.
ISBN 3-929037-47-5

Band 48: Frank Maurer (Universität Kaiserslautern): Hypermediabasiertes Knowledge Engineering für verteilte wissensbasierte Systeme.
ISBN 3-929037-48-3

Band 49: Lothar Simon (Universität Erlangen-Nürnberg): Dynamische, situationsbezogene Hypertext-Handbücher für komplexe Tätigkeiten.
ISBN 3-929037-49-1

Band 50: Marion Mast (Universität Erlangen-Nürnberg): Ein Dialogmodul für ein Spracherkennungs- und Dialogsystem.
ISBN 3-929037-50-5

Band 51: Joachim Posegga (Universität Karlsruhe): Deduktion mit Shannongraphen für Prädikatenlogik erster Stufe.
ISBN 3-929037-51-3

Band 52: Stephan Mehl (Universiät Koblenz-Landau): Dynamische semantische Netze – Zur Kontextabhängigkeit von Wortbedeutungen.
ISBN 3-929037-52-1

Band 53: Jürgen Angele (Universität Karlsruhe): Operationalisierung des Modells der Expertise mit KARL.
ISBN 3-929037-53-X

Band 54: Martin Schröder (Universität Hamburg): Erwartungsgestützte Analyse medizinischer Befundungstexte. Ein wissensbasiertes Modell zur Sprachverarbeitung.
ISBN 3-929037-54-8

Band 55: Jochen Heinsohn (Universität Saarbrücken): ALCP – Ein hybrider Ansatz zur Modellierung von Unsicherheit in terminologischen Logiken.
ISBN 3-929037-55-6

Band 56: Andrea Neufeld (Universität Karlsruhe): Validierung konzeptueller Schemata
ISBN 3-929037-56-4

Band 57: Werner Karbach (Universität Bielefeld): MODEL-K: Modellierung und Operationalsierung von Selbsteinschätzung und -steuerung durch Reflexion und Metawissen.
ISBN 3-929037-57-2

Band 58: Andreas Winklhofer (TU München): Zeitrepräsentation und merkmalsgesteuerte Suche zur Terminplanung.
ISBN 3-929037-58-0

Band 59: Jun Zhao (Universität Bremen): Qualitative Analyse im Rahmen qualitativen und modellbasierten Schließens.
ISBN 3-929037-59-9

Band 60: Susanne Neubert (Universität Karlsruhe): Modellkonstruktion in MIKE. Methoden und Werkzeuge.
ISBN 3-929037-60-2

Band 61: Karsten Berns (Universität Karlsruhe): Steuerungsansätze auf der Basis Neuronaler Netze für sechsbeinige Laufmaschinen.
ISBN 3-929037-61-0

Band 62: Thorsten von Stein (Universität Hamburg): Wissensbasierte Analyse medizinischer Bilder – das Biotop-Verfahren.
ISBN 3-929037-62-9

Band 63: Michael Otte (Universität Karlsruhe): Extraktion von linienförmigen Merkmalen und Ermittlung des optischen Flusses mit seinen Ableitungen aus Voll- und Halbbilsfolgen.
ISBN 3-929037-63-7

Band 64: Klaus Grebner (Universität Bielefeld): Wissensbasierte Entwicklungsumgebung für Bildanalysesysteme aus dem industriellen Bereich.
ISBN 3-929037-64-5

Band 65: Jana Köhler (Universität Saarbrücken): Wiederverwendung von Plänen in deduktiven Planungssystemen.
ISBN 3-929037-65-3

Band 66: Christoph Klauck (Universität Bremen): Eine Graphgrammatik zur Repräsentation und Erkennung von Features in CAD/CAM.
ISBN 3-929037-66-1

Band 67: Markus A. Thies (Universität Saarbrücken): Planbasierte Hilfeverfahren für direktmanipulative Systeme. Erkennung, Vervollständigung und Visualisierung von Interaktionsplänen.
ISBN 3-929037-67-X

Band 68: Jürgen Kreuziger (Universität Karlsruhe): Eine Architektur zur Anwendung symbolischer Lernverfahren in der Robotik.
ISBN 3-929037-68-8

Band 69: Bernd Müller (Universität Oldenburg): PPO – Eine objektorientierte Prolog-Erweiterung zur Entwicklung wissensbasierter Anwendungssysteme.
ISBN 3-929037-69-6

Band 70: Ute Schmid (Technische Universität Berlin): Erwerb rekursiver Programmiertechniken als Induktion von Konzepten und Regeln.
ISBN 3-929037-70-X

Band 71: Jörg R. J. Schirra (Universität Saarbrücken): Bildbeschreibung als Verbindung von visuellem und sprachlichem Raum. Eine interdisziplinäre Untersuchung von Bildvorstellungen in einem Hörermodell.
ISBN 3-929037-71-8

Band 72: Volker Gengenbach (Universität Karlsruhe): Einsatz von Rückkopplungen in der Bildauswertung bei einem Hand-Auge-System zur automatischen Demontage.
ISBN 3-929037-72-6

Band 73: Jan Schloen (Universität Karlsruhe): Wissensbasierte Bewegungsausführung für die Montageautomatisierung mit Industrierobotern.
ISBN 3-929037-73-4

Band 74: Friedrich Mädler (Universität Osnabrück): Problemzerlegung durch Nadelöhrmengen. Ein modellbasierter Ansatz zur Akquisition von Kontrollwissen für Planungssysteme.
ISBN 3-929037-74-2